C++ Analyzer Update

Ole Hansen

Jefferson Lab

Joint Hall A & Hall C Data Analysis Workshop
December 18, 2013

Hall A Analysis Software: Podd (C++ Analyzer)

- Version 1.5 in production since 2008
  - Used by every experiment (except for parity in integrating mode)
  - Very stable & well debugged
  - Occasionally beginning to show its age (see later this talk)
- But: 12 GeV program right around the corner
  - Improvements desirable to support Hall A 12 GeV experiments, especially large installations (SBS, SoLID)
  - Collaboration with Hall C started to maintain common framework
  - Plenty of work ahead

Home page: `http://hallaweb.jlab.org/podd/`

Podd 1.5: Progress in 2013

Version 1.5 is in maintenance mode, *i.e.* we only apply backward-compatible bugfixes & small improvements

- Maintenance
  - Minor bugfixes
  - User-friendly messages for exceptions (usually occurring in user code)
  - Support for latest ROOT, Linux and compiler versions
- Improvements
  - Profiled & improved text database performance bottlenecks
  - Small modifications to accommodate Hall C analyzer (Steve Wood)
- Code development ▸ Tutorials this afternoon
  - Source code repository moved to github
  - Bug tracking using github's built-in issue tracker
  - Optional SCons configure/build system (Ed Brash)

Current version: 1.5.25 ▸ web

Roadmap I: Podd 1.6 ("12 GeV Startup edition")

Release 1.6 Planned Features

- EVIO unbundled, loaded from external library ✓
- Hall C changes integrated ✓
- Backlog of binary-compatibility workarounds removed ✓
- Bug-tracking system ✓ ▸ Ed Brash's talk
- Code split into core and hall-specific libraries
- "Event Type Handler" plug-ins
- Abstract database interface for all modules, similar to existing `LoadDB`
- Time-zone safe `TTimeStamp` for all dates/times
- Test & validation procedures
- (Object-oriented decoder → moved to Release 2.0) ▸ Bob Michaels's talk

ETA: Spring 2014

External EVIO Library

- Podd $\leq$ 1.5 includes bundled EVIO 1 (dating to 1998) for reading CODA data
- EVIO is now at version 4. This new version is required for reading CODA 3 data from pipelined DAQ
- Solution: Unbundle EVIO, load from external shared library
- Implications:
  - Must separately download, build and install EVIO
  - SCons is strongly recommended to build EVIO $\rightarrow$ additional requirement that is not installed everywhere
  - Must set additional environment variables for building and running Podd. Currently: `EVIO_INCDIR` and `EVIO_LIBDIR`
  - Instructions will be included with Podd
- Probably will take ROOT's approach: bundle recent version and build it if no system version detected. Splitting off EVIO library, even if sources bundled, is already beneficial

Event Type Handler Plugins

- Problem: CODA event type numbers currently hard-coded for Hall A. Hall C uses different assignments. Also, no support for alternative event handling or new event types.
- Solution: Plug-in modules for `THaAnalyzer`, replacing `PhysicsAnalysis()`, `ScalerAnalysis()`, etc.
- Default modules for standard event types (physics, scalers, EPICS) will be provided in the core library. Mapping will be defined in the database.
- Similar to object-oriented decoder, but at higher level

Event Type Handler base class

```
class THaEventTypeHandler : public TObject {

public:
  THaEventTypeHandler();
  virtual ~THaEventTypeHandler();

  virtual Int_t  Analyze( const THaEvData& evdata, Int_t prior_status ) = 0;
  ...
};
```

Metadata, Self-Documenting Output

- Goal: Add capability to determine *exactly* how replay results were created
- Solves long-standing problem of "mystery" production data files
- Save relevant metadata as a ROOT object in output file. ROOT schema evolution allows this class to grow as needed
- Some information (about input file) already saved in this way:

Example Run Object in Output

```
analyzer [3] gSystem->Load("libsolgem.so");  // ← contains TSolSimFile class
analyzer [4] TFile* f = new TFile("solout_5gem_ld2dis_muon.root","READ")
analyzer [5] .ls
TFile** solout_5gem_ld2dis_muon.root
 TFile* solout_5gem_ld2dis_muon.root
  KEY: TSolSimFile Run_Data;2 Digitized MC data
  KEY: TTree T;1 Hall A Analyzer Output DST
analyzer [6] Run_Data->Print()
OBJ: TSolSimFile solout_5gem_ld2dis_muon.root Digitized MC data
Run number: 1
Run date:   Sun Jan  1 00:00:00 2012
Requested event range: 1-4294967295
Analyzed events:       2952
...
```

- Drawback: Need ROOT dictionary for metadata class

Metadata, Self-Documenting Output (cont.)

Candidate metadata items to write to replay output (preliminary)

- Experiment name & run number
- Full paths of input & output file(s)
- Replay time, host, user, work directory
- Analyzer version (version & git commit)
- ROOT version loaded at run time
- EVIO version loaded at run time
- Build time, host, compiler, user, directory
- Build flags (CXXFLAGS, others)
- Database URL (DB_DIR, SQL host, etc.)
- All database keys & values loaded, validity time stamps
- Replay script? Copy of source?
- Output & cut definition files (full copies)
- All parsed output definitions
- Optional comment/description created at replay time
- Analyzer class, decoder class
- Analyzer runtime flags (scalers, helicity)

Database API

- Retain 1.5 API in v1.6+ for backward compatibility
- Only minimal code changes required (see code snippets)
- v1.6+ API allows for different backends.
  Examples
  - Hall A-style flat files
  - Hall C-style parameter file
  - MySQL server
- Backend can be set and/or configured from replay script

Podd 1.5 Database Access

```
UserDetector::ReadDatabase( const TDatime& date ) {
  FILE* file = OpenFile( date );
  DBRequest request[] = {
    { "planeconfig", &planeconfig, kString },
    { "MCdata",      &mc_data,     kInt,    0, 1 },
    { 0 }
  };
  Int_t err = LoadDB( file, date, request, fPrefix );
  fclose(file);
};
```

Podd 1.6+ Database Access

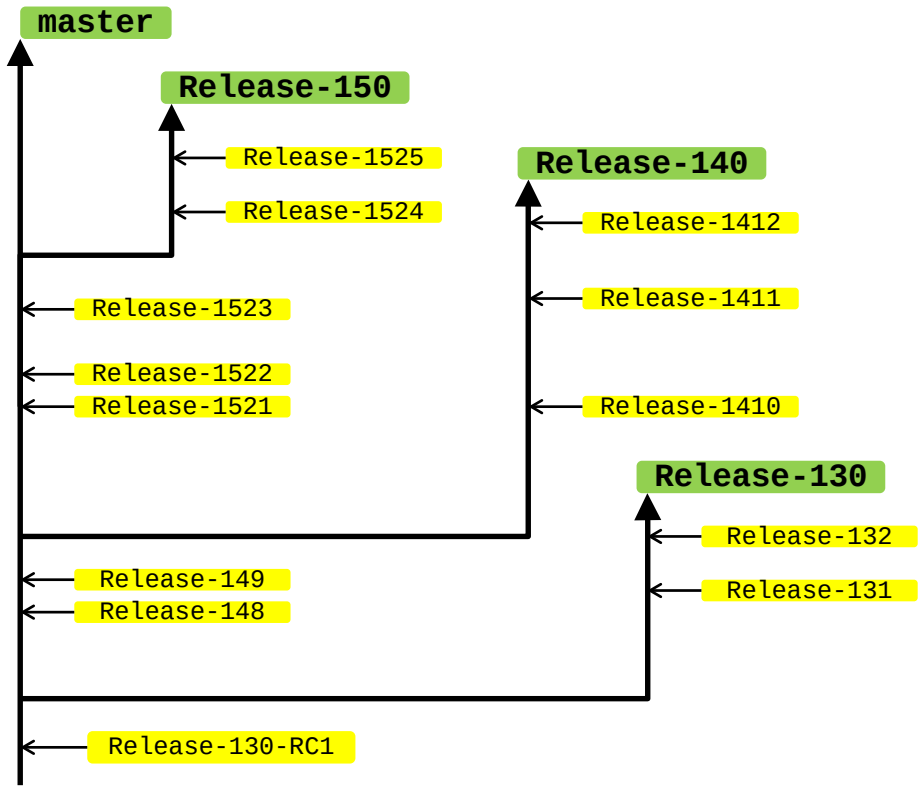
```
THaInterface.C:
  THaDB* gHaDB = new THaFileDB( DB_DIR );  // Default DB

UserDetector::ReadDatabase( const TTimeStamp& date ) {
  DBRequest request[] = {
    { "planeconfig", &planeconfig, kString },
    { "MCdata",      &mc_data,     kInt,    0, 1 },
    { 0 }
  };
  Int_t err = LoadDB( date, request, fPrefix );
  fclose(file);
};
```

Podd Git Repository Organization

URL: `https://github.com/JeffersonLab/analyzer.git`

- `master` branch holds development version (currently 1.6). Contribute new features here.
- `Release-NNN` branches are maintenance releases. Contribute bug fixes for a production version here.
- Tags represent actual releases
- Only maintainers have push access; users should contribute via github pull requests
- More in Ed Brash's talk this afternoon ▸ talk



Collaboration with Hall C

- Hall C migrating to C++ analyzer based on Hall A framework ▸ Gabi's talk
- Bi-weekly meetings
- Found that Hall C code seamlessly integrates into framework (only very minor changes needed)
- Common developments
  - SCons build system
  - Database interface (in progress)
  - Object-oriented decoder (in progress)
  - Code validation tools (in progress)
- Collaboration on code via github

Roadmap II: Podd 2.0 ("Parallelized edition")

Podd 2.0 Planned Features

- Automatic event-level parallelization → major upgrade
- ROOT file output speed improvement, if at all possible
- Object-oriented decoder ▸ Bob Michaels's talk
- Decoders for pipelined 12 GeV electronics. CODA 3 support.
- SQL database backend. (Expect databases to become big and complex.)
- Improved VDC analysis

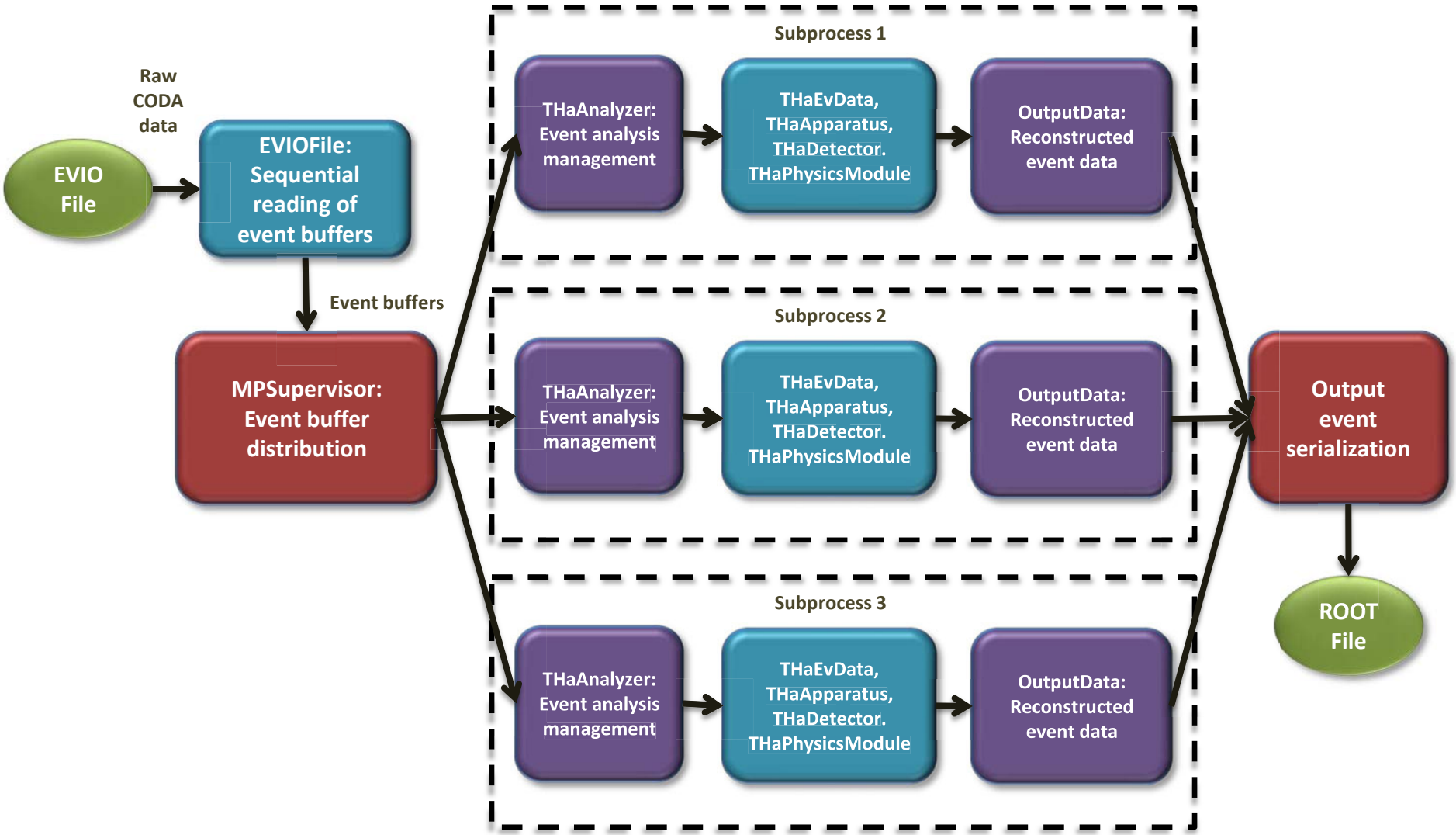
Lots of work. Help welcome!

ETA: Early 2015

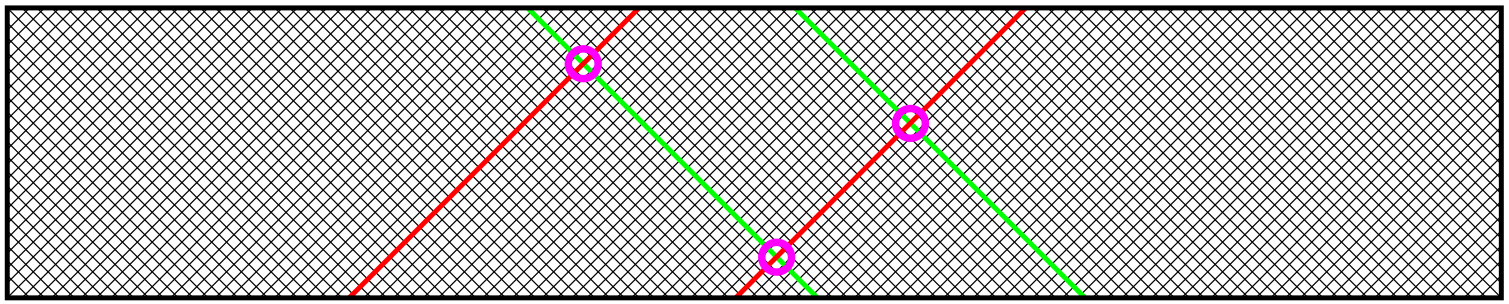
Event-Level Parallelization: Plan

- Problem: Current analyzer is single-threaded
- Workaround: Multiple replay jobs per node, one per run
  $\longrightarrow$ requires some manual setup (though relatively minor)
- Better solution: Parallelize analyzer
  - Code wasn't written for threading $\longrightarrow$ multithreading difficult
  - Support for multithreading would require rewriting all analysis object classes to implement per-thread instances of event processing methods
  - Solution: `fork()` subprocesses instead of creating threads
  - Additional memory overhead minimal
    - ★ Linux `fork()` implemented using copy-on-write
    - ★ Threads would need per-thread event data storage as well
  - In this way, keep existing code, only write new "event loop"
  - Main drawback: Need extensive inter-process communication (IPC)
- Particularly useful for SBS and beyond ($\geq$ 2017)
- Hall C analyzer will inherit parallelization feature transparently

Parallelized Analyzer Flowchart (concept)



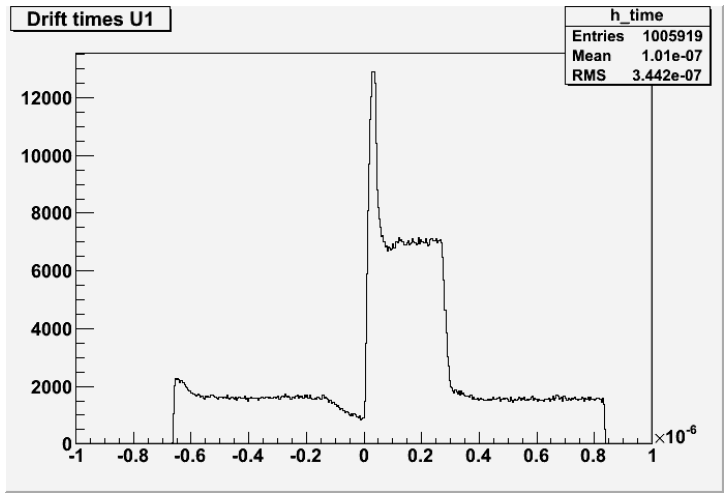
VDC Tracking Improvements I: Multi-Cluster Events

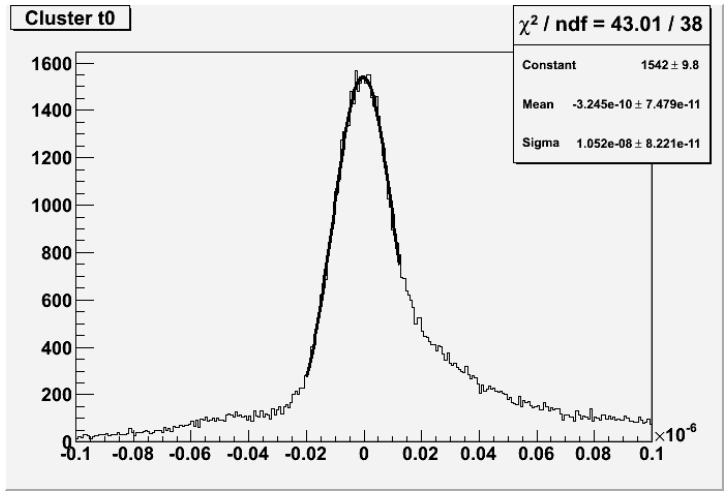


- At low rates, secondary clusters often close to primary ones and have $t_0 \approx 0$
- Probably caused by scraping and/or delta electrons
- Multiple clusters in ≥ 1 plane cause u-v matching ambiguity
  $\longrightarrow$ increased probability of mis-reconstructed track
- Cannot be significantly improved in software only
- 3rd wire direction (expensive) or 3rd tracker plane (less effective) may help
  $\longrightarrow$ upcoming G_M^p experiment will test

VDC Tracking Improvements II: Accidental Coincidences

- Accidentals occur at high singles rate (MHz)
- Cause multiple cluster topology as in previous case
- Can be largely resolved in software only by performing non-linear 3-parameter fit to cluster TDC values to extract track time offset t_0
- $\approx \times 10$–20 background rejection with APEX test data
- Code written, but needs more testing and integration
- Could likely be further improved with 3rd tracker plane, as in previous case





Software for Specific Experiments

- G_M^p: will be using an FPP chamber to assist HRS tracking
- DVCS: hopefully have all their software (?)
- APEX ($\geq$ 2016): need high-rate VDC analysis
- SBS ($\geq$ 2017)
  - GEM track reconstruction optimization
  - Coordinate detector analysis
  - GEp(5) recoil polarimetry
  - Calorimeter cluster search
  - RICH analysis & PID
- Møller is getting into early software engineering stage. Might need to optimize existing parity code for large data volume.
- SoLID
  - Heavy simulation activities, will be ongoing for years
  - Currently working on setting up full simulation framework
  - Need to develop new track reconstruction algorithm

Manpower needs to be found within the respective collaborations

Conclusions

- Hall A analysis software essentially ready for early 12 GeV experiments (through 2015)
- Some work needed to upgrade the core analysis software for the requirements of post-2015 12 GeV experiments over the next 1–2 years
- Additional manpower very desirable to keep schedule!
- Collaboration with Hall C has already proven to be very productive. Looking forward to continuing the good work.