artdaq: DAQ Software Development Made Simple

John Freeman
CHEP 2016
10 October 2016
The “art” in artdaq

- “art” is an application developed in Fermilab’s Scientific Computing Division which performs event-based processing for an experiment’s offline analysis
- This processing is done using pluggable modules; modules can perform event filtering, analysis, reconstruction and output
- A standard set of modules is available + experiments can write their own
- The choice of modules is referred to as an art “workflow”, and is communicated to art via a FHiCL (*.fcl) document (Fermilab Hierarchical Command Language – think JSON, or XML).
- An example of running art at the command line:

  art –s ExperimentsInputFile.root –c ExperimentSpecificWorkflow.fcl

  For more, see art.fnal.gov
Motivation

• *art’s* features would be very useful for online running as well – events being produced in real time:
  – Filtering can reduce the data initially stored
  – Analysis allows for online monitoring
  – Modules could be written in common for offline and online
  – A DAQ could take advantage of existing modules

• This is where *artdaq* comes in!
artdaq Is

- A set of processes, which provide “hooks” for experiments to embed code (primarily art modules + communication with upstream hardware)
- Additionally, infrastructure for
  - State-machine DAQ transitions (“start”, “stop”, etc.)
  - Transport + assembly of data fragments
  - DAQ metrics reporting (event rate, etc.)
- FHiCL-configurable, like art – very flexible
- Supported for most major Linux variants
- A simple “toy” artdaq-based DAQ system will be described on the next few slides
  - Keep in mind when the system is described that a real-life system will have more of each type of process
  - Also keep in mind that the processes can (and probably will) run on different hosts
BoardReaders call objects (here, ExperimentSpecificClass1 and 2) which implement the `artdaq::CommandableFragmentGenerator` base class’s functions – `start`, `getNext`, `stop`.

- `getNext` reads data in according to the experiment’s protocol and returns it wrapped in `artdaq::Fragment` objects (data stamped with a fragment ID and sequence ID).
**EventBuilders: Assembly and Filtering/Compression**

- **BoardReader process #1**
  - Assemble fragments with even numbered sequence IDs into events
  - Filter/compress events in embedded art workflow

- **BoardReader process #2**
  - Assemble fragments with odd numbered sequence IDs into events
  - Filter/compress events in embedded art workflow

- “Round Robin” fragment sending:
  - Each BoardReader sends fragments with a fixed fragment ID, all sequence IDs
  - Each EventBuilder is in charge of assembling all fragment IDs for 1/N sequence IDs
Diskwriting

- Events are saved in *art*-readable *.root files
- The FHiCL documents used to configure the *artdaq* processes (and hence the DAQ) can also be saved in the *.root files

Data Logger process
- Non-blocking event sends to Dispatcher process downstream (next slide)
- Writes all events to storage
Online Physics Monitoring

- *artdaq* provides a plugin whereby standalone *art* processes can read events passing through the system.
- Can configure fraction of events sent to a process, or even apply experiment-specific cuts!
Online Physics Monitoring

- **artdaq** provides a plugin whereby standalone art processes can read events passing through the system.
- Can configure fraction of events sent to a process, or even apply experiment-specific cuts!

**Dispatcher process**
- Separate transport lines to each online monitoring art process
- Allows data logger to focus only on writing to storage

**Data Logger process**
- Run ExperimentModule2 on every Nth event
- Run ExperimentModule1 on every event

**art process**
- 1/N
DAQ Monitoring and More

- **artdaq** provides the MessageViewer app, which prints messages from both **artdaq** and experiment-specific code with severity level indicated by color.

- Plugins are provided so that the metrics reported by **artdaq** processes can be displayed in different formats (Ganglia, Graphite, etc. – FHiCL configurable).

- TRACE debugging
# Experiments Which Use *artdaq*

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Peak Incoming Data Rate (GB/s)</th>
<th># BoardReaders</th>
<th># EventBuilders</th>
<th>EventBuilder data reduction factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUNE 35ton</td>
<td>0.1</td>
<td>24</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>Darkside-50</td>
<td>0.5</td>
<td>12</td>
<td>16</td>
<td>~5</td>
</tr>
<tr>
<td>LArIAT</td>
<td>0.3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Mu2e</td>
<td>33</td>
<td>36</td>
<td>~500</td>
<td>~100</td>
</tr>
<tr>
<td>protoDUNE-SP</td>
<td>3</td>
<td>~80</td>
<td>10-20</td>
<td>1</td>
</tr>
<tr>
<td>SBND</td>
<td>0.4</td>
<td>~20</td>
<td>10-20</td>
<td>1</td>
</tr>
<tr>
<td>ICARUS</td>
<td>0.4</td>
<td>~20</td>
<td>10-20</td>
<td>1</td>
</tr>
</tbody>
</table>
Mu2e Planned Layout

- "Distro" is "event distributor" (artdaq BoardReader)
- "Filters" runs art filter algorithms (artdaq EventBuilder)
- EVB is TRK+CALO+CRV event builder (artdaq EventBuilder)
- "Broker" is CRV request & fragment broker (artdaq BoardReader)
- "Logger" is data logger (artdaq Aggregator)

- Complete TRK/CALO Events
- Accepted TRK/CALO Events
- CRV Fragment Request (specific event window)
- CRV Data Request (specific event window)
- CRV Data Fragment (single event window)
- Accepted TRK/CALO/CRV Events
Upcoming Developments

• Convenience and choice
• Ability to configure FHiCL parameters via a GUI rather than through editing ASCII files
  – Can save/retrieve parameters in DB
• Run control / process management
  – Experiments won’t need to develop software to control when artdaq processes are created, destroyed, and sent state transitions
• Data transport flexibility via plugins
  – Current data transport done via MPI
  – We’d like the transport layer to be something you could choose
• *artdaq*-based DAQ toolkit
• Goal is to provide “off-the-shelf” DAQ components
• Designed for small lead-time experiments – get a DAQ up and running in a matter of hours
• Provides Run Control GUI, firmware for supported boards and configuration management system

[http://otsdaq.fnal.gov/beta](http://otsdaq.fnal.gov/beta)
Conclusions

• Developed by Fermilab’s RSI (Real-Time Software Infrastructure) group, *artdaq* is used by many experiments

• Designed to provide online users the benefits of the *art* package, it also provides numerous useful features which experimenters won’t need to build from the ground up

• *artdaq* was created to make experimenter’s lives easier, and is constantly being improved with that goal in mind—*reusability and flexibility*

• To learn how to begin running a simple *artdaq*-based system within minutes, go to [https://cdcvs.fnal.gov/redmine/projects/artdaq-demo/wiki](https://cdcvs.fnal.gov/redmine/projects/artdaq-demo/wiki)
  – Works on most major Linux distributions (Scientific Linux, Ubuntu 14, …)
  – Can also run it out of VirtualBox, using this file:  [https://goo.gl/OoU6vJ](https://goo.gl/OoU6vJ)