Offline requirements for conditions data

DUNE Conditions DB Workshop

David Adams

BNL

July 2, 2021

This session

Session is intended to be interactive

- I will show slides to frame the discussion
- And invite comments as we go along
  - Please speak up if I forget to do so
  - (Maybe let me finish the slide or at least my sentence)
- Expect to have plenty of time after I show all the slides
  - That is the best time to raise topics I have neglected
  - Also opportunity to have additional discussion on points raised earlier

Goals for the session

- Organizers would like to leave meeting with an action plan
  - (I am not an organizer)
- An important topic for that "plenty of time" at the end

Definitions

Define some of the terms in the title

- Offline refers to the reconstruction, monitoring, calibration and analysis of event data after that data is written by DAQ (or simulation)
    - Trigger, DAQ and DQM also process event data and so this discussion is also relevant in those domains
    - I am not limiting the scope to the production framework (art event loop)
        - User frameworks in C++, Root or python will want access to conditions data
- We want to go beyond abstract requirements
    - I propose and we discuss high-level design
        - Especially the interface(s) presented to off line users
    - Action plan includes products of those discussions
- Conditions data is any additional data needed to process event data
    - Identifying and categorizing these is part of our action plan
    - We may have different implementations for different categories
    - Much more on this follows

Conditions data operations

Five operations are carried out with conditions data

- Evaluation is not a topic for this meeting
- Insertion will not be discussed further
    - Might be helpful to have interfaces as discussed for retrieval
    - Not clear if it is worth the effort since this is performed by a much smaller set of more-expert users
    - Might be the same structs used for retrieval
- Storage we will discuss a bit
    - Ideally, typical end-user would not have to know about this
        - For some categories, the user might have to pre-stage conditions data
    - Big concern for conditions DB group to ensure the data required for a job (not just art event loop) is promptly available at any location it is needed
    - Could be commercial DB, lightweight DB, Root tree, fcl, json, text, …
- Retrieval of conditions data is a major topic here
    - For given time or event ID to process the current event
- Query of conditions is needed to find data subsets for analysis
    - Often a list of runs but might also be events
    - Often need to query multiple categories

Keys

Conditions data retrieval is indexed by a key

- Typically run+event or time stamp
  - Run alone would suffice for some categories (e.g. run conditions)
  - Event condition data could provide map of run+event to time stamp
- Some categories require additional keys, e.g., APA or channel

Storage is typically indexed by time stamp IOV

- I.e., a contiguous range of time stamps
- Might use run+event range instead
  - Problems if event numbers are not guaranteed to increase with time
- Run IOV can be used where data does not change during run
- Need policy to handle overlapping IOVs
  - Return value that was entered last or
  - Return value with the latest IOV start or
  - Return error
  - Storage might be configured not to allow overlaps

Query and retrieval

Retrieval of conditions data

- Retrieval can occur at a high rate
  - Processing and analysis jobs ask for new data for each event
  - This conditions retrieval should not slow jobs significantly
- Jobs retrieving conditions data may run anywhere
  - Any supported site and perhaps user desktop or laptop
- Conditions data may be cached for retrieval
  - We know which run or runs a job will process
  - And which conditions data it needs
  - We might want to pre-stage the relevant conditions data
    - As we do with event data

Query is quite different

- May want to query all runs or events
- But don’t fetch much or anything beyond keys for selected data
- Result stored as an *event dataset*—another conditions category
  - Event dataset = list of selected event IDs (run+event numbers)
- OK if this runs slow (minutes?) and only at a few sites

Separation of retrieval and storage

For discussion

We should separate storage and retrieval

- Define an interface for retrieving conditions data
  - This can be different for each conditions category
    - However, easier for users if a similar pattern is used for most categories
- Provide implementation of the interface for relevant storage type
  - Implementation(s) could allow retrieval from multiple storage types

Pros

- Insulate user from migration to a new storage technology
- Insulate user from schema evolution
- Data can be cached in a different format
  - E.g. a job or site might hold limited data in lightweight DB or text format

Cons

- A little bit of complexity

Query is distinct

- Likely expressed in a language natural to the storage

Retrieval data format

Retrieval returns a simple data struct

- Payload could be a single value
- Or (better) a collection of values stored together
    - Same category and IOV range
    - This could reduce load on retrieval
    - Collection schema are specific to the category
- Also return some metadata
    - Status e.g., number of successfully retrieved values
    - Key
    - IOV range holding the values

Data language

- Define struct in C++
- Use Root dictionary to make this available in Root scripts and python

Retrieval interface

Natural to use C++ class to retrieve conditions data

- Use Root dictionary to access this class from Root script or python
- Class instead of free function allows configuration info
    - Hints for finding data, timeout, error handling, …

Natural to make this class an art service or tool

- So we have run-time configuration in fcl
    - Same configuration language as framework processing jobs
- Run-time discovery of class library (via name in configuration)
- My preference is to use a named tool (rather than service or module)
    - Easier to use outside art event processing loop
    - Difficult to use multiple instances of a service type in one job
        - E.g., if we want to use the same service type for two conditions categories

User access to class (tool/service) should be through an interface

- So change in retrieval implementation does not require any modification of the user code

Categories

First pass at list of categories (∼DB tables)

- Run configuration
- Event metadata
- Geometry
- Event data file metadata (now in SAM)
- Datasets
- Good run (event?) lists
- Electron lifetime
- LAr temperature
- Cathode and anode voltages and currents
- Detector status (APAs, FEMBs, …)
- Channel status
- TPC CE channel calibration
- TPC dQ/dx correction
- Beam status

Run configuration

Run configuration holds conditions for data taking

- DAQ params, active detector elements
- Trigger list
- Electronic settings: gain, shaping, pulser, …
- Called "run configuration" because we typically start a new run if any of these parameters change

Presently included in raw data file metadata

Queried to build datasets and GRLs

- So it should be in a true database

Event metadata

Event metadata might include

- Time stamp
- Trigger(s) for readout decision
- Event data summary useful for offline selection/trigger
  - E.g. trigger primitives or offline flags

Used mostly for queries

- Translate run+event key to time stamp key
- Select events to build datasets

This is big: one row for each event

Geometry

Offline has services for geometry

- Underlying storage is gdml files
- These services are not pretty but we may not want to invest the effort to change implementation and usage
- We might want something more for non-ideal geometry
  - Non-uniformities in wires spacings
  - Offsets in APA or cathode positions
  - Deviation from planarity for APAs or cathodes
  - DB issue if these change with time
- Can we connect offline channels to the results of QC tests on corresponding parts?

Event data file metadata

Map run and processing stage+version to a list of files

- Also map run+event so we don't have to read all files to find a particular event?
- To configure the processing job with a list of input files
- And perhaps to stage those files in advance of running a job
- Not used while processing events
    - We don't care which file holds the event data being processed
- This is handled by SAM now and there are plans to switch to Rucio+?
    - Data object is a string blob that can be parsed to get name-value pairs

Datasets

Dataset specifies a subset of our event data

- E.g., as a list of keys such as run or event IDs
- Plus processing stage(s): raw or reco stage+version, …
- This might carry a list of event data file IDs
    - Or the above can be used to retrieve those from the event data file category
- SAM provides limited support for datasets
    - Dataset is defined by query on file metadata
    - So we cannot specify particular events from a run

Good run lists (GRLs)

GRLs are important for physics analysis

- Which runs should be used (and which not)
- Might want to treat these as special cases of datasets
- Physics groups should decide

Electron lifetime

LAr purity is evaluated with dedicated monitors

- Important conditions data to record
- Might also want to include results from analysis of event data
    - Separate table?

Queries on electron lifetime are useful

- Select long-lifetime runs for physics analysis
- Monitoring plots

Retrieval during event processing is important

- To convert observed charge to deposited charge

LAr temperature

LAr properties depend on temperature

- Recombination, drift speed, …
- Need to retrieve temperature during reco to estimate these

Cathode and anode voltages and currents

Voltages and currents can be used in queries

- To select events good for physics processing
- Important for monitoring
- Useful for studying detector problems

Voltages may also be retrieved during event processing

- To estimate recombination and drift speed

I often wished I had this when studying protoDUNE-SP data

- Like to have something for protoDUNE 2 (and for protoDUNE 1)

Detector status: APAs, FEMBs, …

ProtoDUNE experience

- Many runs taken with fewer than all six APAs
  - This is in the run configuration
- In addition, FEMBs can be missing, dropping in and out from event to event

Detector status used in queries

- For physics require all of some minimum number of APAs be present
  - And some requirement on FEMBs/APA

Retrieval during processing

- During data preparation, we can see which data is present but downstream jobs need to know when the absence of hits or tracks is due to detector issues
- This could be stored as part of the event data (but isn't yet)

Evaluating detector status requires reading the data

- Could be done as part of DAQ?
- Changing code or configuration for decoding can change results

TPC channel status

Individual channels (sometimes whole ASICs) can have issues

- Dead, weak or noisy
- Status can change run-to-run or even event-to-event
- Observed channels go bad and then recover in protoDUNE

ProtoDUNE used fcl-based channel status service

- Only three states: good, bad, noisy
- Intermittent channels handled by flagging bad always
- Move to something better for protoDUNE 2?

TPC CE channel calibration

Calibration info for reach TPC channel

- Charge calibration
  - Area, height or both
- Shaping time
  - Used for deconvolution (signal fitting)

Read access

- Retrieval for low-level reconstruction
- Queries and retrieval for detector studies

TPC dQ/dx correction

Physics signals used to validate/refine channel calibrations

- E.g., muon dE/dx vs. range
- Historically this was use instead of the CE calibration

Retrieval of corrections

- During reconstruction to correct charge response
- In studies to compare different corrections or no correction

May want to query corrections

- Look at correlations between different sources or with CE values

Beam status

Event data holds the beam data

- Is everything we need there?

Queries needed for physics and detector studies

- Status (on/off, energy, etc.)
  - Maybe this is in run data?
- Event data indicating presence, energy and particle ID
  - Should we add this to the event metadata category?
  - Or have a separate beam event data table?

And more…

Photo detectors

Near detector

Vertical drift

Space charge

…

Summary/comments

Many conditions operations

- Evaluation, writing, storage and reading
- Latter done by many clients and includes
  - Queries over all (or much) data and
  - Retrieval all (or subset) of category data for a given key

IOV keys

- Time stamp or run/event
- May depend on category

Propose here to define interfaces for retrieval

- Expect many users at many sites to retrieve conditions data
- Dedicated interface for each category
  - Allows for multiple implementations
    - Evolution with time, site dependent (local caches)
  - Proposal here: Tool interface to retrieve C++ struct specific to each category
    - Root dict provides access from Root scripts and python
    - One alternative is common tool interface returning generic map/table
  - Easier for users if a common pattern is followed for different categories

Categories

- Starting list presented here