ProtoDUNE Prompt Processing System (p3s)

When you start testing

Meaningful testing requires that at least one datatype is defined in the system which should match whatever datatype(s) you use in your workflow template (DAG). If your files all have extension '.tst' you may define the datatype "test" (which again must be consistent with you DAG) and use the following command to add the datatype to the server:

./dataset.py -R -j '{"name":"test", "ext":".tst", "comment":"testing"}'

If you no longer need the datatype you created (perhaps as the result of testing), it can be removed from the server as folllows:

./dataset.py -D test

Embedded documentation

Many directories contain .md files that are easy to read in the browser on GitHub by just clicking on a file. Some of the more important instructions are converted to PDF format and kept in the "documentation" folder in this repo.

If you want to convert any of the .md files to PDF at will, the following utility may be used which is easy to install on Linux - see this example:

pandoc -s -o README.pdf README.md

Learning about the clients

All clients have the '-h' or the equivalent '–help' option which summarizes the command line syntax, in addition some have the 'usage' option which may provide more info.

Design Paper and Motivation

Supporting documents and an outline of the design can be found in the FNAL DocDB 1861 (authorization required for access).

The p3s is the computing platform for protoDUNE to support Data Quality Management (DQM). Its requirements and mode of operation are different from a typical production system in the following: * there are stringent ETA requirements for processing jobs since for DQM purposes the results become stale (not actionable) very fast * only a portion of the data (configurable) needs to be processed * in any stage of processing a portion of the data unknown apriory can be dropped in order to optimize throughput * there is no retry mechanism since any substantial delay in processing a unit of data makes the result less relevant (again, the focus is on ETA) * processing streams are initiated purely automatically and in real time by the data arriving from DAQ * there is no distinct data handling system for two reasons. First, the cluster which runs p3s is either literally local or can access data through a POSIX-like interface with some modest development and deployment effort. Second, a data handling system would introduce additional complexity, latency and potentially failure modes. Instead, p3s relies on federated storage such as provided by an instance of XRootD. A high-performance NAS could be an alternative. In either case, for purposes of access and processing the data is essentially local on the cluster.

Job dispatch

Pilots

To minimize latency and provide the ability to run transparently on a few local resources (e.g. the cluster at EHN1, CERN Tier-0 and perhaps some other facilities on or around CERN campus) any reliance on the flavor of the underlying batch system needs to be eliminated. In addition, latencies inherent in any batch system should be optimally mitigated. Both problems are addressed by utilizing a pilot-based job dispatch, where the pilots (agents) contact a central service and only receive jobs in case the batch slot is secured and the environment validated. This also combats a few failure modes.

Pilot States

An example of what states a pilot can go through during its lifecycle is given below: * *active*: registered on the server, no attempt at brokerage yet * *no jobs*: no jobs matched this pilot * *dispatched*: got a job and preparing its execution (may still fail) * *running*: running the payload job * *finished*: job has completed * *stopped*: stopped after exhausting all brokerage attempts.

Workflow

Workflow as a DAG

While workflow in p3s will be simple compared to a typical production system, it still includes a few steps and can be modeled as a simple DAG. Different stages in the workflow may need to be dynamically prioritized in order to get deliverables in a timely manner.

A workflow is instantiated based on a DAG template. DAG templates are persistent in the database and can be added or deleted at will. A convenient external representation of a DAG is a XML file describing the corresponding graph, which can be readily parsed and used for both import and export of DAGs.

XML Schema

At the time of writing, the GraphML schema is used as the "input language" describing graphs. It's one of standard schemas for describing graphs and editors and other tools exist for manipulating data in this format, altough it's human readable and can be easily edited by hand

Pairing Jobs to Data

Setting the environment variables to supply information about I/O is preferred due to flexibility of such method. Dependencies between interfaces of jobs in a workflow need to be minimized.

Once a WF is defined (based on a DAG), so are dataset characteristics (i.e. file names) in each edge of the graph. At this point the environment of each node can be updated to include references to file, via environment variables.

Phased development

- Phase I: service supporting individual jobs, triggered by incoming data
- Phase II: simple workflows
- Phase III: complex workflows

Location of the input raw data

The protoDUNE DAQ writes the data to its own "online buffer" from which it is transferred to CERN EOS (centralized distributed high-performance disk storage). In order to continue operation in case of a network outage which could make EOS inaccessible, the system mush be able to optionally feed from the online buffer without putting too much extra I/O load on it.

Just-in-time job execution

The near-time nature of prompt processing requires "just-in-time" job submission and not be subjected to the often unpredictable latencies found in batch systems. An efficient and tried way to achieve this is the pilot-based job submission.

Components

- Web service:
- workflows and their templates (dags)
- jobs
- data
- handling of pilots' requests for registration and payload
- Clients
- The *pilot* - submission and management of pilot data on the server
- The *job* - submission of job definitions to the server and management of job data on the server

Software dependencies

- Python3+
- Django 1.10+
- django-tables2
- RDBMS (TBD but most likely PostgreSQL; sqlite used for development puprposes only)
- Apache
- NetworkX (some versions may present compatibility issues)
- GraphML (optional but very helpful, doesn't need to be installed as it's a schema)

TODO

Models

No outstansing issues

Time limits

- loss of the pilot heartbeat
- wall clock limit on the job execution

Brokerage

- better policy management

Installation and Integration

- PosgeSQL
- Apache - issue: different configuration features on Ubuntu and CentOS
- XRootD

Storage management

- Cleanup after job execution
- Flushing of obsolete data
- Clearing older DB entries

Workflow

- Implement true dag traversal (i.e. for multiple edges coming to a node)
- in the job class
- in the GraphML source