January 2017

User guide to p3s Workflow Interface and Client

Intro

The protoDUNE prompt processing system (p3s) consists of a central web service (named *promptproc*) and a number of clients. At the time of writing, the "workflow client" in p3s exists in the form of a Python script appropriately named *workflow.py*. For brief summary of command line options run the script with "-h" option.

The function of the client is to create, manage, adjust and delete workflows in p3s. The client does not provide much of monitoring functionality, which instead mainly exists in the Web interface of the p3s system.

In p3s — just like in many workflow management systems — the Directed Acyclic Graph (DAG) is an abstraction of a workflow. In the graph, vertices represent jobs and edges represent data, since it is data that defines logic of the workflow and relationship among the constituent jobs. Vertex and Edge objects have attributes necessary to describe properties and behaviors of jobs and data elements.

In p3s a DAG describes the topology and general characteristics of edges and vertices of a class of workflows but does not correspond to a running process or processes nor does it have a complete information that would be necessary to create a functional workflow. In fact, workflows are created by adding enough information to DAGs such that that a proper execution context is created (i.e. the environment, paths to executable, location and name of files etc). Workflows therefore are created based on DAGs serving as templates (abstractions) by adding necessary parameters.

Describing DAGs

DAGs and Workflows are stored on the central server of p3s and are persisted using the nbackend database.

There are many ways to describe a DAG in a way suitable for creating a record in p3s. One convenient method it to leverage an existing XML schema for desribing graphs, *GraphML* — which is supported in a number of software packages and editing and visualization tools.

Option "-g" of the workflow client allows to use a graphML file containing a DAG description and send it to the server. A name can be suppplied via a command line argument, and if absent it is derived from the name of the file containing

the graph. DAG names are unique in p3s, and that property is enforced in the database by making the name attribute the primary key. Example of creating a DAG:

```
workflow.py -g myDAG.graphml
```

Defining Workflows

Workflows are created based on templates (DAGs) which must be exist on the server by the time a request for a new workflow is sent. A name can be optionally set for a workflow but it's not expected to be unique. Worflows are identified in the system by their UUIDs which are automatically generated.

Example of creating a workflow based on a DAG:

```
workflow.py -a myDAG -n my Workflow -f myFileInfo.json
```

Object Deletion

The "-d" option accompanied by the object key (e.g. UUID for workflows) will cause the client to delete the corresponding object from the database. For both DAGs and workflows, this will cause the deletion not only of the object itself as a record, but also of its vertices and edges.

Until serious testing has been completed, please consult the experts about this, such as the author of this software - especially if the system is in production.

Supplying File and Job Info during creation of a workflow

File Info

By default, p3s will create filenames for a workflow dynamically utilizing UUID and a predefined extension as per the declared data type. This can be changed by using The option "-f" which is is overloaded and can be:

- a stirng not formatted in JSON and taking values such as:
- "sticky", in which case a workflow inherits the file names from its parent DAG
- "inherit:name", in which case filenames will be automatically generated based on the supplied name and DAG topology
- a JSON-formatted string which can specify the filenames for any of the DAG's edges if desired
- a name of a JSON file containing same information

Job Info

It works similar to "file info" explained above. Information supplied in JSON format (either on the command line or in a file supplied with -j option and having json extension will be included in the job object on the server side.

Example: `./workflow.py -v 2 -a source1 -n sourceTest -s defined -f ../inputs/fileinfo1.json -j '{"filter":{"payload":"env"}}'`