Created by: Maxim Potekhin *potekhin@bnl.gov*

February 2018

Version 1.02 (release notes: added information on EOS and other access, directory locations etc)

Introduction

Purpose and content of this document

This document explains how to set up and run the job submission client. These instructions are not generic as they are tailored to operation of the protoDUNE experiment at CERN in 2018. There are references to certain directory locations, environment variables and scripts that are application-specific.

There is a separate "overview" document which contains a general description of how p3s works and what its components are. For the end user a lot of this detail won't matter since they are typically interested in just running a number of jobs on resources provided by the system and following their progress, consulting the log files if necessary. That's the extent of the instructions found below. These instructions were validated by a few new users. If anything does not work as expected, please consult the author of this document.

Please keep track of the version number located on top of this document. Once incremental changes become significant the version number will be bumped up. It is important to refer to the right set of instructions as p3s is gradually enters operations period and adjustments are made. This document also exists in PDF format and can be printed out for your convenience, in the documents/pdf folder of the p3s repository (as explained below).

Preparing to run

These instructions apply to the **lxplus** interactive Linux facility at CERN. To set up access to p3s on that platform one needs to follow a few simple steps as described below. To use p3s for the needs of the protoDUNE experiment it is strongly recommended that you ensure that your account

- is attributed to **np-comp** Linux group at CERN
- has membership in the following CERN e-groups:

1. eos-experiment-cenf-np04-readers
2. eos-experiment-cenf-np04-writers

This will allow you to access, move and delete if necessary the data produced by p3s. To get these memberships sorted out you may file a ticket on the CERN services pages and/or contact Nectarios Benekos and the author of this manual.

Get the p3s client software

At a minimum you need a client script (which is written in Python) to submit job descriptions to p3s for execution. You will also benefit from looking at job templates stored as JSON files in the p3s repository. If not already done so, install p3s software at the location of your choice by *cloning the content from GitHub*. For the purposes of this writeup, you are assumed to be on an interactive node located at CERN such as lxplus. Run the command

```
git clone https://github.com/DUNE/p3s.git
```

This step is done in two cases only (so not often):

- you are just starting to use the system
- you are informed that there was an update and you should switch to a newer version

After you run "git clone" your current directory will contain a subdirectory **p3s**. Of immediate interest to you are the following folders within p3s:

- **p3s/clients** containing multiple client scripts with different functions; of particular interest to you right now is the script **job.py**
- **p3s/configuration** containing a few bash scripts which simplify setup for individual sites (such as CERN)
- **p3s/documents** with documentation (such as this writeup) in both Markdown (md) and PDF format
- **p3s/inputs** and its subdirectories such as jobs/larsfot with job definition and wrapper script templates

Set up and verify the Python environment

The next step is also CERN-specific and its purpose is to set up the Python environment for p3s clients without having to install anything yourself. This needs to be done every time you have a fresh shell session which you plan to use for p3s interaction. It may be added to your log-in profile to save some typing line but can also be done manually. Either way, please activate the "Python virtual environment" by running this command:

```
source ~np04dqm/public/vp3s/bin/activate
```

To verify that this actually worked, please change the directory to **p3s/clients** which is mentioned above. Run the script:

```
./verifyImport.py
```

In the output you should see version of Python which is 3.5+, a couple of "OK" messages and finally the word "Success". If anything is amiss, contact the author of this document.

Verify access to p3s server

While you can specify the server URL and other parameters the p3s clients need on the command line it is often more convenient to just run a script which will set a few environment variables to be used by default. For example, if running at CERN you could simply use the command (assuming you cloned the p3s directory as described above)

```
source p3s/configuration/lxvm_np04dqm.sh
```

Then you don't need to worry about the server URL etc. Other environment variables contained in this file will be explained later in this document where necessary. Now, you can switch to the "clients" directory and try to run the command:

```
./summary.py -P
```

If it connects to the server sucessfully, it will print a few current stats for p3s. Example of the output:

```
Domain: p3s-web:80, hostname p3s-web.cern.ch, uptime 22 days, 14:37:23.850000
Pilots: total 200, idle 200, running 0, stopped 0, TO 0
Jobs: total 22, defined 0, running 0, finished 6, pilotTO 16
Workflows 0
Datasets 3
```

If you see a failure, please contact the author of this document.

The Web monitor

The next step is to make sure that you can also see the Web pages served by p3s so you have monitoring functionality. Try to access **http://p3s-web.cern.ch** in your browser (if you are at CERN).

Currently the server **p3s-web.cern.ch** is only accessible within the confines of the CERN firewall. If you want to access it from an external machine, please use a ssh tunnel like in the command below

```
ssh -4 -L 8008:p3s-web.cern.ch:80 myCERNaccount@lxplus.cern.ch
```

. . . in which case pointing your browser to localhost:8008 will result in you seeing the p3s server (which is on port 80 at CERN).

Submitting a job

The mechanics of the job submission

Keep in mind that when you "submit a job" all you are doing is sending a record containing all the info necessary for running a particular executable, to the p3s database. The system (p3s) will then match this job with a live and available **pilot** occupying a batch slot in CERN Tier-0 facility and deploy the payload for execution in that batch slot (i.e. on one of the Worker Nodes at CERN). The pilot will monitor the state of the job under its management and send periodic "heartbeats" to the server to tell it that it's still alive. Once the job completes, the pilot closes logs, optionally performs other "close out" functions and resumes querying the p3s server for the next job.

This is a typical case of a pilot-based framework which entails the following

- typically very low latency of the start of job execution since you are not waiting for a HTConfor or other queue; in some cases such as busy HT Condor queues gains can be quite substantial
- you don't have to run batch commands yourself
- ease of automation since same template can be used in automated submission
- easy to read tabulated view of all of your jobs in the p3s monitor which is a Web application
- the identity under which jobs are executed is not your identity but the production identity... This can be helpful or not helpful depending on situation, and path permissions need to be thought through. The NP04 group at CERN has access to various EOS and some AFS directories so this can be made to work

An example of the job description

The following dummy example (with rather arbitrary attributes, file names and variables) demonstrates how JSON is used to describe jobs. Let us assume that we created a file named "myjob.json" with the following contents:

```
[
    {
        "name":         "p3s_test",
        "timeout":      "100",
        "jobtype":      "print_date",
        "payload":      "/home/userXYZ/my_executable.sh",
        "env":          {"P3S_MODE":"COPY","MYFILE":"/tmp/myfile.txt"},
        "priority":     "1",
```

```
        "state":        "defined"
    }
]
```

Note that this format corresponds to a *list* of objects i.e. such file can naturally contain a number of jobs; however having just one element in this list is absolutely fine.

For the job to be eligible for execution the "state" attribute needs to be set to "defined" as showed above. Other possible states will be discussed later. The other two attributes that need t be set are the **payload** and **env**. They are explained below. The remaining attributes of the job are less relevant for initial testing.

The script referenced in the **payload** attribute must be readable and executable by members of the same computing group as the pilot (in our case that's "np-comp") and if it's located in AFS the relevant permissions come on top of that. The "public" directory in your AFS-based directory tree is a good choice to place your own scripts.

The payload and the environment

The **payload** attribute of the job definition (such as in the example above) is the path of the script that will run. It is strongly recommended that this is a shell wrapper, and the *bash* shell is most commonly used. If the `env` attribute contains `"P3S_MODE":"COPY"` then the script will be copied into a sandbox *by the pilot* at execution time. Otherwise an attempt will be made to execute it *in situ* which may or may not work depending on the permissions (including both AFS permissions if that's what you are using, and also Linux flags which should be at least +rx).

The **env** attribute in the JSON snippet above defines the job environment in the Linux sense. It can be used for most anything but in particular, it can be used to communicate to the running job the names of input and output files. This is typically done in the wrapper script itself, i.e. within the wrapper ("my_executable.sh" in the current example) we may find:

```
foo -i $MYFILE
```

In this example, the binary executable *foo* will read input data from the file whose name is stored in the environment variable *MYFILE*. The name of this environment variable does not matter as long as it is consistent with what's in the JSON file such as shown above.

Please note that the user has complete freedom as to how to factor the information between the **env** attribute JSON file and the script. There are few limits in desingning job descriptions.

Running your first job

We will use a prefab example for your first run. Inspect the **p3s/inputs/jobs** directory of the repo that you cloned from GitHub. Find and examine the file **printEnv.json**.

```
[
     {
      "name":       "printEnv",
      "timeout":    "100",
      "jobtype":    "printEnv",
      "payload":    "/bin/env",
      "priority":   "1",
      "state":      "defined",
      "env":        {
              "P3S_COMMENT":"we are not using a wrapper since we are just running the standa
                    }
     }
]
```

Two things to note here:

- the payload is just the standard built-in "env" command which will print the environment to stdout. We are not using a wrapper in this example for simplicity's sake.
- for same reason, we are not using the P3S_MODE environment variable as the excutable does not need to be copied into a sandbox

Now we can submit this job to the server. Assuming the p3s client software is installed, and **we changed the current path to the "p3s/clients" directory**, let's make sure we set the correct environment, so unless already done so use the command

```
source ../configuration/lxvm_np04dqm.sh
```

After that the following command can be used (once again form the "clients" directory)

```
./job.py -j ../inputs/jobs/printEnv.json
```

And that's it. The path to the JSON file used with this client can be anything, it just has to be readable for your user identity. When looking at the monitoring pages of p3s this job will be marked with your userID on the system from which you submitted it e.g. if you work on lxplus this will be your lxplus userID. If you are at CERN, the URL where you can check your jobs will be:

```
http://p3s-web.cern.ch/monitor/jobs
```

Entries in the jobs table are clickable and will reveal more infrormation if you access the link. Note that while in this example the execution time is negligible

it may take a while for the state of the job in the monitor to be displayed as "finished" due to the way polling period and hearbeats are set in the pilot. Pay attention to the UUID of the job since this will allow you to locate the stdout and stderr of the job when it finishes, which will be contained in the directory

`/eos/experiment/neutplatform/protodune/np04tier0/p3s/joblog`

In this test case, the *.out file will contain a printout of the environment found on the worker node. The* .err file should be empty if everything worked well.

Since running a functioning wrapper script (as opposed to Linux built-in commands) is a more realistic use case note that it's entirely up to the author of the wrapper script to define the location of the output files other than stdout. Please see the "p3s/inputs" directory for examples of job descriptions which access different directories. In many cases, ROOT and other files produced by the current version of p3s will move the files they produce to the following location in EOS:

`/eos/experiment/neutplatform/protodune/np04tier0/p3s/outputs`

. . . but once again this depends on what's in the wrapper script.

The **job** client script we use here and all clients in p3s suite support the "-h" command line option which prints an annotated list of all command line options. Take a look, it's helpful. If you need more verbose output, you can add "-v 2" to the command line to change the verbosity level.

Creating and editing your wrapper scripts

It is important to ensure that the wrapper you are testing or using is located in a storage area that's accessible to the members of the np-comp group (see note on group attribution in the beginning of this document). When working in AFS, this also requires that you use a directory under your "public" directory branch otherwise the file won't be readable by p3s regardless.

As a simple exercise, you may want to try the following:

- create and rename a copy of the JSON file used in the previous example
- edit the JSON file by replacing the reference to /bin/env with something like `/afs/cern.ch/user/f/foo/public/myTest.sh`
- put something simple in the bash script myTest.sh located in your public folder, like /bin/env or /bin/date
- submit this job to p3s by specifying the path to your modified JSON file on the command line for the "job.py" client
- watch the job execute and look for the output in the "joblog" directory as stated above

In case something is amiss please contact the author of this document. If all worked well, you may try to create a more realistic and complex job description.

Your wrapper script is pretty free form so is you need to set up an environment or fetch a piece of input from somewhere, please do so. Look at the examples in the "inputs" directory.

"Test Wrapper"

There is a script which allows the user to test the setup of the JSON file and the payload working together by running everything on an interactive node such as lxplus as opposed to submitting it to lxbatch, which is often more convenient for basic debugging. Similar to the "job" client presented above, the test wrapper can be invoked from the p3s/clients directory as follows (assuming you have created a file "myOwnExample.json"):

```
./testwrapper.py -j myOwnExample.json
```

which is quite similar to normal submission. Like in the previous case, the "-h" option will output helpful information. For example:

- -p option allows to override the path of the payload script i.e. in principle you can run anything (i.e. any executable) using the same skeleton template
- -f and -F respectively overwrite the path defined by the P3S_INPUT_FILE and P3S_OUTPUT_FILE

Note on environment variables

With rare exceptions such as *P3S_MODE* (which may change in future versions) there is no semantic importance to the exact names of the environment variables used in formulating your job. The only thing that matters is that the payload script contained correct references to the environment.

Location of the data and log files

Directories in EOS

For storage of data and log files p3s can use *any* storage area (e.g. a directory) which is read/write accessible to its pilots. This means the user id of the pilots running in p3s must be a member of requisite groups and/or have correct permission for the directory used for log and data storage in p3s. At CERN we are taking advantage of the distributed **EOS** storage facility that is accessible from both interactive and worker nodes. The top level directory for all sorts of p3s data has been defined as

```
/eos/experiment/neutplatform/protodune/np04tier0/p3s
```

Log files

The p3s configuration file mentioned above (such as lxvm_np04dqm.sh) is not used just by client scripts, but also by the server. As such, it contains other useful info such as pointers to directories to keep log files for HTCondor submission of pilot jobs, pilot logs and job logs. The latter will be of most interest for the end user. Consider the following example:

```
[np04dqm@lxplus056 src]$ env | grep P3
P3S_PILOTLOG=/eos/experiment/neutplatform/protodune/np04tier0/p3s/pilotlog
P3S_JOBLOG=/eos/experiment/neutplatform/protodune/np04tier0/p3s/joblog
P3S_SERVER=http://p3s-web:80/
P3S_CONDOR_ERROR=/afs/cern.ch/work/n/np04dqm/condor
P3S_PILOTS=100
P3S_CONDOR_LOG=/afs/cern.ch/work/n/np04dqm/condor
P3S_PILOT_MAXRUNTIME=90000
P3S_PILOT_TO=1000
P3S_VERBOSITY=0
P3S_SITE=lxvm
P3S_CONDOR_BASE=/afs/cern.ch/work/n/np04dqm/condor
P3S_DIRPATH=/eos/experiment/neutplatform/protodune/np04tier0/p3s
P3S_CONDOR_OUTPUT=/afs/cern.ch/work/n/np04dqm/condor
```

We are taking advantage of availability of distributed file systems at CERN with are visible from most nodes, i.e. all logs are kept in a single directory structure which can be browsed by the user (as opposed to local disks on a few dedicated nodes). The reason HTCondor logs are keps in AFS as opposed to EOS is a current CERN policy.

Moving on to “real” payloads

Please take a look at examples in p3s/inputs/larsoft for realistic examples. JSON files used to submit LArSoft jobs are fairly standard in that they typically specify locattion of the FCL file etc. The wrapper script sets up the execution environment from CVMFS and/or local AFS build if necessary.