Created by: Maxim Potekhin

January 2018

Version 1.00

Introduction

Purpose of this document

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.

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.

Get the software

If not already done so, install p3s software at the location of your choice simply by cloning the content from GitHub (you must be on an interactive node at CERN)

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

After you run this, your current directory will contain a subdirectory **p3s**. This subdirectory will in turn contain a number of subdirectories, and the one of immediate interest now is **p3s/clients**.

Set up the Python environment

The next step is CERN-specific. Activate the "Python virtual environment" by running this command

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

Verify the Python environment

Change the directory to **p3s/clients** which is mentioned above. Run the script to verify the environment:

```
./verifyImport.py
```

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 developer.

Make sure you can access the monitoring server

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).

Running a job

Job description

The following example (with arbitrary 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 purpose of this introduction, let's say that the "state" attribute needs to be set to "defined" as showed above. Most important attributes are the **payload** and **env**. They are explained below.

The payload

This is the path of the script that will run. It is strongly recommended that this is a shell wrapper, the *bash* shell is most commonly used. Important caveats are (a) this scirpt needs to be accessible from whatever node is used for execution, fo

The Environment

The "env" attribute 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. withing the wrapper 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.

"Hello, World!"

Let's create a simple test job. For that we'll need the payload script - which can be named anything but to correlate with the JSON example above let's call it "my_executable.sh".

```
#!/bin/bash
# This script is "my_executable.sh" in the JSON example above
date > $MYFILE
```

It is important that the path /home/p3s/my_executable.sh is readable and executable for other users, otherwise the system won't be able to run it. For example, in **lxplus** it is optimally placed in the "public" subdirectory in your account which is on AFS and is open to public.

WORK IN PROGRESS BELOW

- Use a dedicated client ("job.py") to submit this job description to the server which will then orchestrate its execution
- Monitor the progress of jobs using a P3S Web page
- Browse and use the output files produced by jobs

In the following we assume that the CERN instance of P3S is used, which entails certain conventions and conviences such as sharing files and scripts via AFS and EOS.

P3S Clients

job.py

This client can be used for the following:

- send a job description (or a number of job descriptions) to the P3S server. This can be done by reading a description of job(s) contained in a JSON file.
- if necessary, adjust job attributes
- delete a job from the server

Now we can submit this job to the server. Assuming the p3s client software is installed, and we changed to the "clients" directory, the following command can be used

```
./job.py -j ./myjob.json
```

And that's it.