Created by: Maxim Potekhin

January 2018

Version 1.01

Introduction

Purpose 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 and it's important to refer to the right set of instructions.

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 p3s client software

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.

```
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. Of immediate interest to you are the following:

- **p3s/clients** containing multiple client scripts with different functions
- **p3s/documents** with documentation (such as thiswriteup)

- **p3s/inputs** and it's subdirectories such as jobs/larsfot with job definition and wrapper script templates

Set up and verify the Python environment

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

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

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

```
./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 developer.

Verify access to p3s server

While you can specify the server address and other parameters the p3s clients need on the command line it is often more convenient just to run a script which will set a few environment variables which to be used by default. For example, if running at CERN you would simply use the command

```
source p3s/configuration/lxvm.sh
```

Then you don't need to worry about the server URL etc. For example, 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. 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).

Running a job

Resources

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 batch slot in CERN Tier-0 facility and deploy the payload to it, which means

- typically very low latency 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 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 payload and the environment

The **payload** 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 at execution time. Otherwise an attempt will be made to execute it *in situ* which may or may not work depending on the permissions.

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.

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.

"Hello, World!"

Use the template

Consider the following example which is in p3s/inputs/jobs directory of the repo that you cloned from GitHub. The file name is "simplejob1.json".

```
[
    {
    "name":     "simple p3s job, type 1",
    "timeout":  "100",
        "jobtype":  "type1",
        "payload":  "/home/maxim/projects/p3s/inputs/jobs/simplejob1.sh",
        "priority": "1",
```

```
    "state":    "defined",
    "env":      {
        "P3S_TEST":"TRUE",
        "P3S_MODE":"COPY",
        "P3S_INPUT_FILE":"/home/maxim/p3s.in",
        "P3S_OUTPUT_FILE":"/home/maxim/p3s.out"
             }
    }
]
```

To use this template for testing

- please copy this file and edit your copy so that it points to the actual location of the script (which must be readable by the np04-comp group)
- make sure that the input file location is readable to members of *np04-comp* group at CERN (or just globally readable) and the path to the output file can likewise be used (i.e. must be writeable).

This is the corresponding "payload script" named in this case "simpljob1.sh". Note that the exact name is unimportant, that's just what it is in the JSON template

```
#!/bin/bash

echo pid, ppid: $$ $PPID

if [ -z ${P3S_INPUT_FILE+x} ];
then
    echo No input file specified, entering sleep mode
    /bin/sleep 10
    exit
fi


echo Using input file $P3S_INPUT_FILE

wc -l $P3S_INPUT_FILE > $P3S_OUTPUT_FILE
```

It is important that the path to simplejob1.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.

Submit the job

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 (assuming the JSON file is in the current path)

```
./job.py -j ./simplejob1.json
```

And that's it. The path to the JSON file 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.

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.

"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:

```
./testwrapper.py -j ./simplejob1.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
- -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* (likely to change) 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.