PALFA Pipeline2.0 Documentation

Patrick Lazarus, ...
plazar@physics.mcgill.ca

February 11, 2011

Abstract

1 Installation

1.1 Dependencies

1.2 Getting Started

Here we will present basic step-by-step instructions for setting up the pipeline.

Step 1 Create a directory where you want the pipeline to be installed.

Step 2 Download the pipeline source files. This should be done by cloning the git repository on github.com: https://github.com/plazar/pipeline2.0.

```
$ git clone git://github.com/plazar/pipeline2.0.git
```

NOTE: This will create a sub-directory called "pipeline2.0".

Step 3 Add the pipeline directory to your PYTHONPATH environment variable.

Step 4 Create configuration files using the examples provided. Modify the settings so they are appropriate for your system. Run the `sanity_check.py` script to ensure all your configurations have the correct types, the neccessary directories exist and you have the correct privileges for files that must be read/written.

Step 5 Set up the database. This is done using the `create_database.py` script.

Step 6 Start the downloader using `StartDownloader.py`.

NOTE: The downloader does not need to be run on the same computer as the job pooler. However, the directory where the downloader saves files must be accessible from the computer where the job pooler is being run.

Alternatively, it is possible to add files to the job-tracker database without using the downloader. To do this use `add_files.py`. Be sure to set the `delete_rawdata` configuration to `False` if you do not want raw data files to be deleted when the pipeline no longer requires them.

Step 7 Start the job pooler using `StartJobPool.py`. The job pooler will start submitting jobs when data files are finished downloading.

Step 8 Start the uploader using `StartJobUploader.py`. The uploader will upload results to the common database when jobs are successfully processed.

2 The Pipeline

An overview of the pipeline.

2.1 Features

- Sanity check of configurations
- Dynamic zapping
- Automatic download of data files
- File size checks when downloading
- Error emails
- Automatic (configurable) retry of failed jobs

- Check of results before uploading
- Automatic (configurable) deletion of data files (upon success or terminal failure)

2.2 Components

2.2.1 Job-tracking Database

2.2.2 Downloader

2.2.3 Job Pooler

2.2.4 Uploader

Diagnostics Diagnostics stored in the common database are split into two types: Numeric values and binary data. Some of the "binary data" diagnostics are actually plain text files.

The following per-beam diagnostics are uploaded to the common database:

- *RFI mask percentage* (Numeric value)
  Percentage of data masked due to RFI.
- *Num cands folded* (Numeric Value)
  The number of candidates folded.
- *Num cands produced* (Numeric Value)
  The total number of candidates produced, including those with sigma lower than the folding threshold.
- *Min sigma folded* (Numeric value)
  The smallest sigma value of all folded candidates from this beam.
- *Num cands above threshold* (Numeric value)
  The number of candidates produced (but not necessarily folded) that are above the desired sigma threshold.
- *RFIfind png* (Binary data)
  Output image produced by rfifind in png format.
- *Accelcands list* (Binary data)
  The combined and sifted list of candidates produced by accelsearch.

- *Zaplist used* (Binary data)
  The list of frequencies and ranges zapped from the power spectrum before searching this beam.

3 Reference

3.1 Executables

`StartDownloader.py` A script to start the downloader background process.
usage: `StartDownloader.py`

`StartJobPool.py` A script to start the job pooler background process.
usage: `StartJobPool.py`

`StartJobUploader.py` A script to start the results uploader background process.
usage: `StartJobUploader.py`

3.2 Generic Queue Manager Interface

3.3 Configurations