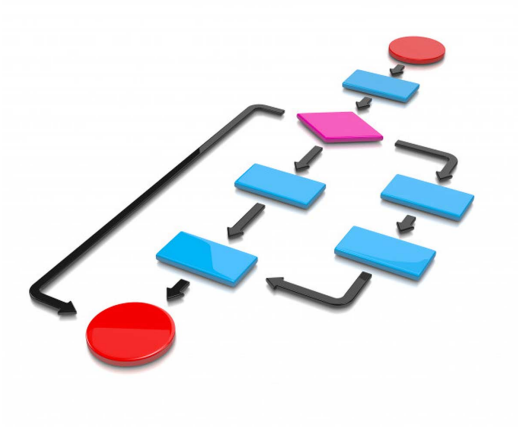
APS User Guide

Rafael Larrosa Jiménez, Pablo Orozco Guerrero

2020



Contents

1 Introduction

APS is a set of shell scripts developed by Rafael Larrosa Jimenez that allow the user create pipeline of jobs using an integrated queue system, SLURM, in a easily way and only in a few steps. In this user guide you will learn how to create and use workflows with APS, the requirements that are needed before starting to create a workflow, how to define each step and it dependencies, how to run and check a complete flow, etc.
The idea of this project was born years ago when a lot of researches wanted to use the HPC for their own jobs but a lot of them had no knowledge about computing or how to use the HPC.
Due to this situation, Rafael started to work in APS project in order to create a easy way for the researches to get introduced in computing and usage of HPC. Thanks to this set of scripts, any user with or without knowledge of computing can access to the HPC and start to work.
In this document the user will find all the information and requirements(Github links, SLURM Queue Sytem information, HPC details, Computing guidelines) needed before using APS and a step by step guide of how the project works, beside a explanation of how the workflow works in the system. The user has at his disposal an example experiment that can be executed for the user and check how APS works and the structure of the project.

2 Environment

HPC (High Performance Computer) is the ideal environment for the project due to its characteristics. An HPC is a set of computing nodes interconnected between them in order to get a higher computing capacity.
Hundreds or thousands of nodes working at the same time to complete a task and thanks to the high number of nodes, the store and network, the HPC can achieves high performance ratio and operations per second.
APS uses SLURM Queye System during its execution so it has to be installed in the environment before use it.
Slurm is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. The goal of this guide is not to teach how to install and configure SLRUM, the user can consult that process in the next link:
https://slurm.schedmd.com/download.html

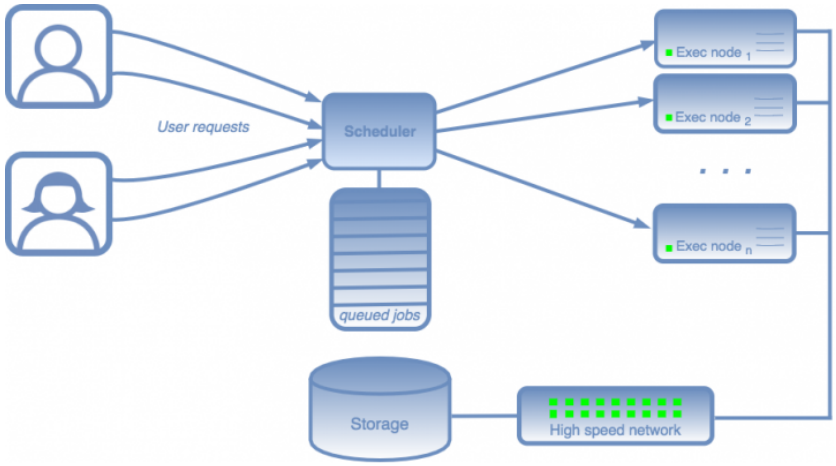


Figure 1: Simplified scheme of SLURM.

3 Requirements

In this section we are going to show what the user needs to configure before to start an experiment.

3.1 Downloading APS

First step is getting the APS scripts set. You can get them directly from the GitHub repository:
https://github.com/rlarrosa1/aps.git
Once we got the scripts a new folder with the name ***aps*** should appears in the location where we the ***git clone*** command was executed. If we move to the folder we will find the set of scripts, we can access to aps folder using ***cd aps/*** command and ***ls -l*** command to see a list of the elements within that folder.

- aps_check_all.sh
- aps_clean_ouput.sh
- aps_create_skeleton.sh
- aps_delete_steps_output_for_replay.sh
- create_example_exp.sh
- aps_calcualte_benchmarks.sh
- aps_flow_check.sh
- aps_send_flow.sh
- create_example_exp.sh
- helper.sh
- worksteps

It is highly recommended to add the direction of the aps folder to the environment variable *$PATH*, in this way we could use the scripts from a different location. To do that first we need to know the current value of our variable *$PATH*. Do the next:

1. Introduce ***echo $PATH*** command in the prompt and you will get something like:

   /home/user

2. Move to the aps folder created in the previous step using ***cd aps/*** command.

3. Introduce ***pwd*** and the full route of the aps directory will appear in the screen, something like :

4. Now we are going to update $PATH variable content by adding the new aps route. For that we execute the next command:
   export PATH=PATH: result of pwd command execution.
   The aps scripts are available now from every location and are ready to be used.

3.2 Templates

It is necessary a template files before the process starts. In those templates the information about required resources will be declared.

An example of those templates comes within the repository content. In ***worksteps*** folder there are six example job template that can be use as example or as base to a new script.

3.3 Input Data

In order to provide the data to the process what we need to do it´s create a text file and fill it with the information in a certain format. In order to create a input file you can choose between a text editor, Nano or Vi for example, and create the file using the text editor as follows:
nano samplesToProcess.txt
With the previous command the text editor nano will be opened and the file created, after that we have to introduce the input data following the next structure:

SAMPLE 1	**SAMPLE 2**	**SAMPLE 3**
12	26	32
49	45	61

To ensure a good performance and output of the process the input data has to follow the previous structure.

3.4 Variables declaration

During the execution of the process, several environment variables will be used and it is necessary configure some of them before start the experiment. In Linux, environment variables are placeholders for information stored within the system that passes data to programs started in shells.
The environment variables that we need to set before start should be declared in a sh file which should starts with prefix *init*, for example:
initiateExperiment.sh.
It is highly recommended to create a separated folder for each experiment, we can create a folder using the command ***mkdir***, for example:
mkdir exampleExperiment

We move to the folder using ***cd*** command and once we are in the new folder we are going to create de init.sh file. To do that select your favorite text editor like Vi or Nano to create the file: ***nano initiateExperiment.sh***
In the text editor, the way to set a value to an environment variable is using ***export*** command as we did before to update the *PATH* variable. The minimum variables that a process needs to run are:

- EXPERIMENT: The name of the experiment
- INPUTDIR: Route of the experiment folder
- INPUTFILE: Reference to the input text file
- JOB_SOURCE: Route of the templates folder

An example of a variable set that have been already initialized:

- EXPERIMENT=Experiment1
- INPUTDIR=/home/user/FirstExperiment
- INPUTFILE=/home/user/FirstExperiment/Samples.txt
- JOB_SOURCE=home/user/FirstExperiment/workflows

4 Creating an experiment

Once we followed all the previous steps and we made all the necessary requirements we can start a experiment. Let´s summarise the steps:

1. Download aps and update *$PATH* variable.
2. Create a separate folder for the new experiment using ***mkdir*** command.
3. Copy the templates from the example experiment to the new folder or create new ones
4. Create a text file with the input data.
5. Create a .sh file with all the environment variables declaration.

We are ready to start an experiment, firt thing we need to do is execute the init.sh file. To do that you can execute it typing the next in the console: ***./init.sh*** or ***. init.sh***

Once ***init.sh*** file has been executed it is needed to create the jobs for the flow. There are two options for this, create the jobs manually or use the examples template. Example templates are within the GitHub repository, they can be copied to a different route using ***cp*** command, for example:
cp -R aps/worksteps/ targetDir/
With this, the directory and the content will be copied in ***targetDir***. The other option is create the job manually. For this option a bigger knowledge about scripting is needed because it depends in the user needs. If manual option is chosen, SLURM configuration parameters have to be declared within the script. The parameters are the next:

1. cpus-per-task : Number of cpus that the job use
2. task : Number of task selected for the job
3. nodes : Number of nodes selected for the job
4. mem : Memory selected for the job
5. error file : Path and name for the error file
6. output file : Path and name for the output file

At this point we are going to start to use the scripts, the first script to be used is ***aps_create_skelleton.sh***, this script will create all the necessary elements for the process. It will create a sub folder per every sample in the input file based on the templates.
Let´s see the execution for an input file as follows:

SAMPLE 1	**SAMPLE 2**
1	2
4	5
7	8

As we can see in the image, new folders have been created, one per every sample in the input file. Use ***ls -l*** command to see all the files and folder in your current location.
Other files that have been created are *works.log* and *commands_to_clean_errors.txt*.
works.log is a log file used by the process but it can give information about what is happening during the execution of the different steps in the experiment. You can use ***cat works.log*** and the content of the file will be displayed in the screen.
commands_to_clean_errors.txt is a text file which contains the commands that the user needs to execute in case there is an error in some of the steps of the experiment.

```
export EXPERIMENT=experimentM
~/experimentM ~/experimentM
Mon Mar 29 22:21:10 CEST 2021 file for 1_seqtrim.sh in 1 created
Mon Mar 29 22:21:10 CEST 2021 file for 1_seqtrim.sh in 2 created
Mon Mar 29 22:21:10 CEST 2021 file for 1_seqtrim.sh in 4 created
Mon Mar 29 22:21:10 CEST 2021 file for 1_seqtrim.sh in 5 created
Mon Mar 29 22:21:10 CEST 2021 file for 1_seqtrim.sh in 7 created
Mon Mar 29 22:21:10 CEST 2021 file for 1_seqtrim.sh in 8 created
Mon Mar 29 22:21:10 CEST 2021 file for 2_STAR.sh in 1 created
Mon Mar 29 22:21:10 CEST 2021 file for 2_STAR.sh in 2 created
Mon Mar 29 22:21:10 CEST 2021 file for 2_STAR.sh in 4 created
Mon Mar 29 22:21:10 CEST 2021 file for 2_STAR.sh in 5 created
Mon Mar 29 22:21:11 CEST 2021 file for 2_STAR.sh in 7 created
Mon Mar 29 22:21:11 CEST 2021 file for 2_STAR.sh in 8 created
Mon Mar 29 22:21:11 CEST 2021 file for 3_cufflink.sh in 1 created
Mon Mar 29 22:21:11 CEST 2021 file for 3_cufflink.sh in 2 created
Mon Mar 29 22:21:11 CEST 2021 file for 3_cufflink.sh in 4 created
Mon Mar 29 22:21:11 CEST 2021 file for 3_cufflink.sh in 5 created
Mon Mar 29 22:21:11 CEST 2021 file for 3_cufflink.sh in 7 created
Mon Mar 29 22:21:11 CEST 2021 file for 3_cufflink.sh in 8 created
Mon Mar 29 22:21:11 CEST 2021 file for 4_cuffquant.sh in 1 created
Mon Mar 29 22:21:11 CEST 2021 file for 4_cuffquant.sh in 2 created
Mon Mar 29 22:21:11 CEST 2021 file for 4_cuffquant.sh in 4 created
Mon Mar 29 22:21:11 CEST 2021 file for 4_cuffquant.sh in 5 created
Mon Mar 29 22:21:11 CEST 2021 file for 4_cuffquant.sh in 7 created
Mon Mar 29 22:21:11 CEST 2021 file for 4_cuffquant.sh in 8 created
Mon Mar 29 22:21:11 CEST 2021 sync job file for 5_unify_cuffdiff.sh
in 5_unify_cuffdiff created
Mon Mar 29 22:21:11 CEST 2021 sync job file for
6_unify_create_cuff_db.sh in 6_unify_create_cuff_db created
```

Figure 2: Execution of aps_create_skeleton.sh

```
Mar 29 22:33 1
Mar 29 22:33 2
Mar 29 22:33 4
Mar 29 22:34 5
Mar 29 22:21 5_unify_cuffdiff
Mar 29 22:21 6_unify_create_cuff_db
Mar 29 22:34 7
Mar 29 22:34 8
Mar 29 22:21 commands_to_clean_errors.txt
Mar 29 22:32 error_sending.txt
Mar 29 22:33 estadisticas
Mar 29 22:17 inicia_experimento.sh
Mar 29 22:16 samples_to_process.lis
Mar 29 22:16 workflows
Mar 29 22:32 works.log
Mar 29 22:32 works_trace.err
```

Figure 3: Directories and files created after the execution of the first script

So if you see an error message in the screen during the execution of some step, check the previous file and execute the commands within it to solve the errors.
As we can see we have a folder per every sample, inside of every folder there are others auto-generated scripts based on the templates where instructions and information about the required resources are specified. Those folders and scripts which have been created with the execution of the first aps script are called jobs.
Next step is send those jobs to the queue system to be processed, in order to do that we need to execute the next aps script: ***aps_send_flow.sh***
The queue system used in Picasso HPC is SLURM, is a free and open-source job scheduler for Linux and Unix-like kernels. It provides three key functions. First, it allocates exclusive and/or non-exclusive access to resources (computer nodes) to users for some duration of time so they can perform work.
Second, it provides a framework for starting, executing, and monitoring work (typically a parallel job such as MPI) on a set of allocated nodes.
Finally, it arbitrates contention for resources by managing a queue of pending jobs. For more information about SLURM check the next link
https://slurm.schedmd.com/quickstart.html

```
list_jobs 1 2 3 4 5 6
Launching step 1 . In step 1 : jobs sent: 6, jobs already finished: 0, already queued: 0
Launching step 2 . In step 2 : jobs sent: 6, jobs already finished: 0, already queued: 0
Launching step 3 . In step 3 : jobs sent: 6, jobs already finished: 0, already queued: 0
Launching step 4 . In step 4 : jobs sent: 6, jobs already finished: 0, already queued: 0
Launching step 5 . In step 5 : jobs sent: 1, jobs already finished: 0, already queued: 0
Launching step 6 . In step 6 : jobs sent: 1, jobs already finished: 0, already queued: 0
In total: jobs sent: 26, jobs already queued: 0, already finished: 0
```

Figure 4: Execution of aps_send_flow.sh

Now that we have a better understanding about the queue system we can execute the script and start the process. For the next step we will execute ***aps_send_flow.sh*** script, this script will analyze the jobs and will send them to the queue system.
After the execution we will see the next message in the screen:
Where we can see the list of jobs to be processed and the launching steps.
During the execution of this script the jobs are being enqueued in the queue system, we can see it if we execute ***squeue*** SLURM command in the terminal, we can see the result of the command execution in **Figure 7**.
This command will display the list of jobs that are being processed in the queue system.

```
JOBID   NAME                              USER        STATUS     CPUS/NODES      REASON/NODES
=============================================================================================
9412699  1_envia_seqtrim_1.sh              user         [R]         16/2      None/cn23,fat02
9412700  1_envia_seqtrim_2.sh              user         [PD]        16/1      Resources/
9412701  1_envia_seqtrim_4.sh              user         [PD]        16/1      Priority/
9412702  1_envia_seqtrim_5.sh              user         [PD]        16/1      Priority/
9412703  1_envia_seqtrim_7.sh              user         [PD]        16/1      Priority/
9412704  1_envia_seqtrim_8.sh              user         [PD]        16/1      Priority/
9412705  2_STAR_hg_1.sh                    user         [PD]        16/1      Dependenc/
9412706  2_STAR_hg_2.sh                    user         [PD]        16/1      Dependenc/
9412707  2_STAR_hg_4.sh                    user         [PD]        16/1      Dependenc/
9412708  2_STAR_hg_5.sh                    user         [PD]        16/1      Dependenc/
9412709  2_STAR_hg_7.sh                    user         [PD]        16/1      Dependenc/
9412710  2_STAR_hg_8.sh                    user         [PD]        16/1      Dependenc/
9412711  3_pasa_cufflink_1.sh              user         [PD]        4/1      Dependenc/
9412712  3_pasa_cufflink_2.sh              user         [PD]        4/1      Dependenc/
9412713  3_pasa_cufflink_4.sh              user         [PD]        4/1      Dependenc/
9412714  3_pasa_cufflink_5.sh              user         [PD]        4/1      Dependenc/
9412715  3_pasa_cufflink_7.sh              user         [PD]        4/1      Dependenc/
9412716  3_pasa_cufflink_8.sh              user         [PD]        4/1      Dependenc/
9412717  4_cuffquant_1.sh                  user         [PD]        16/1      Dependenc/
9412718  4_cuffquant_2.sh                  user         [PD]        16/1      Dependenc/
9412719  4_cuffquant_4.sh                  user         [PD]        16/1      Dependenc/
9412720  4_cuffquant_5.sh                  user         [PD]        16/1      Dependenc/
9412721  4_cuffquant_7.sh                  user         [PD]        16/1      Dependenc/
9412722  4_cuffquant_8.sh                  user         [PD]        16/1      Dependenc/
9412723  5_pasa_cuffdiff_5_unify_cuffdi    user         [PD]        32/1      Dependenc/
9412724  6_crea_db_from_cufflink.sh        user         [PD]        1/1      Dependenc/
.
.
.
you have submitted 26 jobs, there are other 1472 jobs in the queue
---------------------------------------------------------------------
TOTAL                          JOBS: 1498                      USERS: 24
[R:104/PD:1394] CPUS: 5620
IDLE NODES:
```

Figure 5: Execution of squeue SLURM command

After the execution, new files have been created, the files created are:

- ***error_sending.txt***: A text file where we can see if some job had an issue when it was being sent to the queue system.
- ***works_trace.err***: Other error log that save the error information during the execution of the process.

We can check ***works.log*** in every moment to see the information of our process, remember that you can see the information executing ***cat works.log*** command.
Now that the jobs have been processed by the system we need to check what happened with our samples and jobs, to do that we need to execute the next aps script: ***aps_check_all.sh***.
The function of this script is check the current status of the process, that means that the script checks the number of jobs created, how many are running, finished successfully, error cases, enqueued jobs and so on.
Besides that, it creates a file indicating the folders and file that need to be removed due to errors during the process in order to create them again and repeat the process.

4.1 Optimization

A new feature was recently added to APS, optimization feature. This optimization feature allows the user to select different configuration option at the creation point of the flow in order to compare the metrics of efficiency, execution time and amount of resources requested and get the best combination. For example, for an experiment that analyze a protein chain and compares it with another one, it probably has a lot of input data and different jobs which goal can be different, some of them focused to find matches between the date and other one looking for consecutive sequences for example.
If a wrong parameters configuration is selected, the performance of the flow could be affected giving a bad execution time, waste of resources and bad user experience as result. Due to this the optimization feature was added.
To use optimization feature a new option is introduced for ***aps_create_skelleton.sh***, ***-o*** option. With this option, the script will execute the optimization feature, this means that now the user can pass the configuration options as parameters for the script.

The parameters accepted for the script are:

1. name of the job
2. cpus-per-task : Number of cpus that the job use
3. ntask : Number of task selected for the job, similar to threads
4. nodes : Number of nodes selected for the job
5. mem : Memory selected for the job
6. time : Set a limit on the total run time of the job
7. constraint : sd or cal (cal recommended)
8. error file : Path and name for the error file
9. output file : Path and name for the output file

And this is an example of how to declare those parameters in a script:

```
#SBATCH -J g16.SAMPLE.sh
#SBATCH --cpus-per-task=32
#SBATCH --ntask=1
#SBATCH --nodes=1
#SBATCH --mem=50gb
#SBATCH --time=7-0:0
#SBATCH --constraint=cal
#SBATCH --error=job.1_step1.%J.err
#SBATCH --output=job.1_step1.%J.out
```

Figure 6: Example of configuration parameters

The only mandatory requirement for optimization option is file naming.
It is required naming output file and error file as follows:
job.*step name*.%J.out or .err
For example, if the current execution step is named: 1_step_calculation,
then the output and error file should take the name as:
job.1_step_calculation.%J.out or .err
And %J will be replaced for the job id when this job is executed.

An example of how to use optimization options is execute ***aps_create_skelleton.sh*** with the previous options.

aps_create_skelleton.sh -o -c 12,24,32 -m 5gb -t 1,2

With the previous example, a set of jobs will be created with different parameters combinations based in the inputs received. The previous execution will not create all the jobs for the experiment, it will create as much jobs as combinations of parameters in the input. Doing this, only a small number of jobs are created saving time and resources.
Second step is execute ***aps_send_flow.sh*** to send the jobs to the queue system and then execute them. When the jobs are finished, SLURM has a set of information of the jobs taken directly from the operating system and SLURM software itself as cpu time, execution time or efficiency. With this information, the new optimization feature is able to find the best combination of parameters for the jobs.
When the jobs are done, ***aps_calculate_benchmarks.sh*** has to be executed and the script will consult the metrics for the finished jobs and get the best option for every job.

```
. aps_calculate_benchmarks.sh

Starting the analysis to get the best performance

1_step.sh has been updated with the best
option to optimize the resources

2_step.sh has been updated with the best
option to optimize the resources
```

Figure 7: Example of aps_calculate_benchmarks.sh

Finally, recently a new script has been added to APS, this new script is ***create_example_exp.sh***. This script creates an example experiment using a mock inputfile and job templates within APS repository.
With this script an example experiment is created in home directory as ***example_experiment***. Inside of that directory the user can find the necesary elements to start a process. Basic input file is added, job templates are copied from APS repository and init file Before use it ensure that the APS repository has been clone in user´s home directory.
For further questions or problem with APS contact here: soporte@scbi.uma.es.