Table of Contents

JEDI-based Analysis......1

Introduction......2

Changes to analysis workflow......3

Benefits......5

Setup Instructions......6
- AFS......6
- CVMFS......6

Additional Info for Ganga Users......7

Setup Instructions for old (non-JEDI) clients......8
- AFS......8
- CVMFS......8

Differences from user's point of view......9

JEDI-based Analysis

Introduction

JEDI is an intelligent component in the PanDA system to have capability for task-level workload management. This page describes how JEDI changes user analysis on PanDA.

Changes to analysis workflow

Figure 1 and 2 show schematic views of old and new analysis workflows, where a **task** is defined as a workload which takes one or more input datasets (or dataset containers) and produces one or more output dataset containers, and a **job** takes one or more input files and produces one or more output files. A task is composed of multiple jobs. The key change is that tasks are submitted to the system instead of jobs. Once a task is submitted to the system, JEDI works as follows:

1. JEDI runs the brokerage to choose a good site based on data and resource availability,
2. takes a minimum chunk of files from input dataset (or dataset container) to generate scout jobs,
3. sends the scout jobs to the site,
4. collects job profiles from results of scout jobs, such as I/O rate, memory consumption, output size, and execution time,
5. runs the brokerage again to find at most 5 good sites,
6. and generates a bunch of jobs for remaining files according to job profiles and each site configuration. For example, jobs are dynamically configured to run on more files if the site has longer walltime limit.
7. Finally JEDI sends an email notification once the task is done. Email notifications can be suppressed if the user prefer.

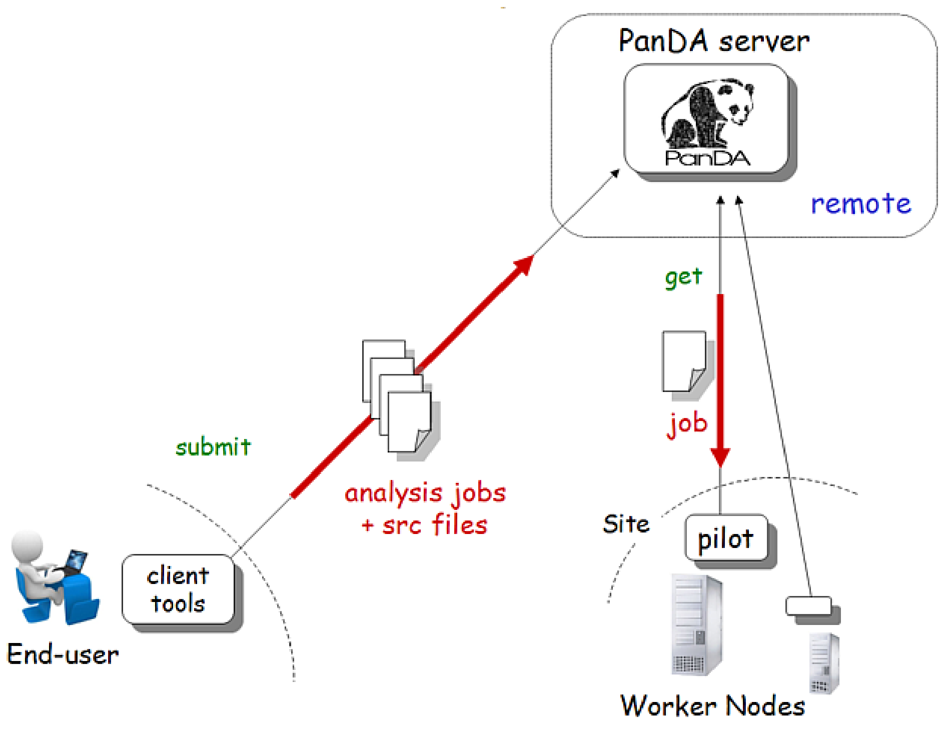
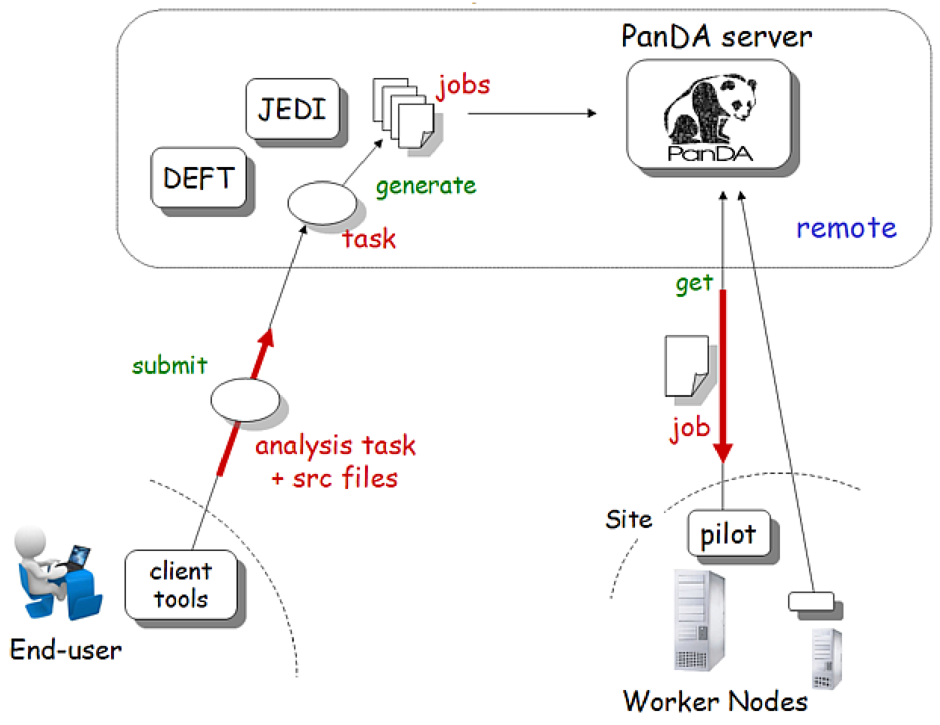


Figure 1. Old analysis workflow. Jobs were generated on the client side to be submitted to the system.



Fegure 2. New analysis workflow with JEDI. Tasks are submitted to the system and JEDI generates jobs on the server side.

Benefits

- Performance of client tools is improved since heavy procedures like file lookup are moved to the server side. Also centralization of user functions brings better maintainability for client tools. For example, hot bug fixes can immediately be propagated to users.
- Users can more efficiently exploit computing resources without detailed knowledge on the grid. JEDI dynamically optimizes job parameters on behalf of users.
- JEDI has a capability for automatic retry at the same or another site. Input is dynamically split or combined based on site configuration. For example, if jobs are reassigned to a site where longer walltime limit is available, jobs are reconfigured to run on more input.
- The overall system performance is improved since internal database access is optimized for task-level workload management. Users can get quicker response from the system.
- Scout jobs are introduced to measure job profiles before generating a bunch of jobs. If no scout jobs succeeded the task is aborted. The user can avoid filling up the system with problematic jobs. Also job parameters are optimized by using real job profiles, which is more accurate than user's manual intervention. For example, sometimes users unintentionally submit short jobs to long queues since they don't know beforehand how exactly long their jobs take.
- JEDI brokerage is aware of network. For example, when siteA has free CPUs and good network connection to siteB where input data is available, jobs can be sent to siteA even if siteA doesn't have the input data. The jobs run at siteA and remotely read data from siteB. That is, jobs can go to siteA and siteB while jobs went only to siteB in the old system. Thus, users can get more CPU resources in the new system.
- The old system forgot output files older than 30 days. When the user ran again on the same input and output, duplicated files were produced. This problem has been solved since JEDI has a permanent file table in the database.
- Duplicated files were also produced when the user ran multiple pathena/prun processes in parallel. This problem has been solved since JEDI has more robust lock mechanism.
- A capability of task chaining is available. For example, completion of an evgen task can trigger a G4 simulation task. The G4 simulation task can start processing before the evgen task is completed, if necessary. In this case, G4 jobs are generated when new EVNT files become available.
- Retry of marge jobs kept failing in the old system since merge datasets were frozen after the original jobs failed. This problem has been addressed in JEDI since internally new output datasets are created for each retry and they are added to the original output dataset container.

Setup Instructions

Here is how to setup client tools. Once they are setup, command names such as pathena/prun/pbook and command-line options are the same as before. JEDI is used by default as of 12th Aug 2014.

AFS

If you use afs to setup your user environment, after you have setup your normal environment please source the following line:

```
$ source /afs/cern.ch/atlas/offline/external/GRID/DA/panda-client/latest/etc/panda/panda_setup.sh
```

CVMFS

If you are using CVMFS to setup your user environment you can use:

```
$ export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
$ source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
$ showVersions panda
$ localSetupPandaClient
```

For ganga users replace the last step with:

```
$ localSetupGanga
```

Additional Info for Ganga Users

To properly use Jedi, please switch your backend to it via:

```
ganga --jedi ....
```

For the IPython/script interface, you will need to change:

```
j.backend = Jedi()
```

depending on how you submit. Most Panda options are still available and so won't need changing but NOTE: You cannot use a splitter with Jedi as it determines the split itself. Please remove the DQ2JobSplitter (or --split) when submitting.

For a limited time, Ganga will switch from Panda to Jedi and ignore any splitter in the background, showing an Error during submission to remind you. This feature will be removed in the future though so please take care to update your scripts!

Setup Instructions for old (non-JEDI) clients

In case of problems with JEDI that were not foreseen.

AFS

If you use afs to setup your user environment, after you have setup your normal environment please source the following line:

```
$ source /afs/cern.ch/atlas/offline/external/GRID/DA/panda-client/old/etc/panda/panda_setup.sh
```

CVMFS

If you are using CVMFS to setup your user environment you can use:

```
$ export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
$ source $ATLAS_LOCAL_ROOT_BASE/user/atlasLocalSetup.sh
$ showVersions panda
$ localSetupPandaClient 0.4.41
```

For ganga users replace the last step with:

```
$ localSetupGanga
```

To re-enable direct Panda submission (not recommended!), please set the following in your .gangarc:

[Panda] AllowDirectSubmission = True

Differences from user's point of view

Migration should be almost transparent but there are some changes.

- Output file names are not sequential as explained in this section
- One output dataset container is created per output type. Each container name is outDS_blah/. For prun, "blha" is the output file name by default, but can be changed to something by prepending "something" to the output name in `--outputs`. E.g., `--outputs=XYZ:outputFileName`, which produces outDS_XYZ/ instead of outDS_outputFileName/.
- Tasks with the same outDS cannot be submitted/executed in parallel. The error message is something like `ERROR : task submission failed. jediTaskID=xxx is in the yyy state for outDS=zzz. You can re-execute the task with the same or another input once it goes into finished/failed/done`. This is because of an internal protection in JEDI to prevent file duplication caused by parallel execution of client tools. Internally one outDS is mapped to one task. If you try to submit a task with an existing outDS even for another input dataset, it is internally regarded as "re-execution" of an old task which produced the outDS. Re-execution is prohibited to avoid race condition while the task is in active state. That is, if you submit a task running on a datasetA with an outDS, you cannot re-submit the task for another datasetB with the same outDS until the task is done for the datasetA. Note that you can submit one task running on both datasetA and datasetB by using `--inDS=datasetA,datasetB` or specifying all datasets in `--inDsTxt`.
- Scout jobs run before jobs are generated in bulk. Users can disable scouting by using `--skipScout` but that is not recommended.
- Intermediate datasets, which contained pre-merged files in the old system, are not created for `--mergeOutput`. Merged files are directly added to outDS_blah/ instead of outDS.merge_blah/.

-- TadashiMaeno - 04 Jul 2014

This topic: PanDA > PandaJediAnalysis
Topic revision: r11 - 2014-08-17 - TadashiMaeno