

---

# Chapter 1. Student notes for Pegasus WMS tutorial

## 1. Introduction

These are the student notes for the Pegasus WMS tutorial. They are designed to be used in conjunction with instructor presentation and support.

You will see two styles of machine text here:

```
Text like this is input that you should type.
```

```
Text like this is the output you should get.
```

For example:

```
$ date
```

```
Mon June 1 11:54:58 BST 2008
```

You will need to log into the tutorial machine, using an ssh client and the login name and password supplied separately.

On Linux or Mac OS X, open a terminal window and type:

For the purpose of this tutorial replace any instance of trainXX with your viz-login username.

```
$ ssh trainXX@viz-login.isi.edu
```

```
[welcome message]
trainXX@viz-login:~$
```

You will need to generate a grid proxy to run workflows on the Grid. We have installed X509 user certificates that you can use to generate the proxy

```
$ trainXX@viz-login:~/pegasus-wms$ grid-proxy-init
```

```
Your identity: /O=edu/OU=ISI/OU=isi.edu/CN=Tutorial User 01
Creating proxy ..... Done
Your proxy is valid until: Thu Jul 3 13:14:29 2008
```

Check your proxy using grid-proxy-info.

```
$ grid-proxy-info
```

```
subject  : /O=edu/OU=ISI/OU=isi.edu/CN=Tutorial User 01/CN=1106574642
issuer   : /O=edu/OU=ISI/OU=isi.edu/CN=Tutorial User 01
identity : /O=edu/OU=ISI/OU=isi.edu/CN=Tutorial User 01
type     : Proxy draft (pre-RFC) compliant impersonation proxy
strength : 512 bits
path     : /tmp/x509up_u1045
timeleft : 11:57:17
```

## 2. Mapping and executing workflows using Pegasus WMS

In this chapter you will be introduced to planning and executing a workflow through Pegasus WMS locally. You will then plan and execute a larger Montage workflow on the GRID.

All the exercises in this Chapter will be run from the \$HOME/pegasus-wms/ directory. All the files that are required reside in this directory

```
$ cd $HOME/pegasus-wms
```

Files for the exercise are stored in subdirectories:

```
$ ls
config dags dax
```

You may also see some other files here.

## 2.1. Creating a DIAMOND DAX

We generate a 4 node diamond dax. There is a small piece of java code that uses the DAX API to generate the DAX. Open the file \$HOME/pegasus-wms/dax/CreateDAX.java in a file editor:

```
$ emacs -nw dax/CreateDAX.java
```

There is a function constructDAX( String daxFile ) that constructs the DAX. Towards the end of the function there is some commented out code.

```
$ //create fourth job //analyze
//To be uncommented for exercise 2.1
/*
String id4="ID0000004";
job=new Job (NAMESPACE,ANALYZE,VERSION,id4);
//add the arguments to the job
job.addArgument(new PseudoText("-a analyze "));
job.addArgument(new PseudoText("-T60 "));
job.addArgument(new PseudoText("-i "));
job.addArgument(new Filename(FC1));
job.addArgument(new PseudoText(" "));
job.addArgument(new Filename(FC2));
job.addArgument(new PseudoText(" -o "));
job.addArgument(new Filename(FD));

//add the files used by the job
job.addUses(new Filename(FC1,LFN.INPUT));
job.addUses(new Filename(FC2,LFN.INPUT));
job.addUses(new Filename(FD,LFN.OUTPUT));

//add the job to the dax
dax.addJob(job);

//the job with id4 is a child to both jobs id2 and id3
dax.addChild(id4,id2);
dax.addChild(id4,id3);
*/
//End of commented out code for Exercise 2.1
```

The above snippet of code, adds a job with the ID0000004 to the DAX. It illustrates how to specify

1. the arguments for the job
2. the logical files used by the job
3. the dependencies to other jobs
4. adding the job to the dax

After uncommenting the code, compile and run the CreateDAX program.

```
$ cd dax
$ javac CreatedAX.java
```

```
$ java -classpath .:$CLASSPATH CreatedAX ./diamond.dax
```

Let us view the generated diamond.dax.

```
$ cat diamond.dax
```

Inside the DAX, you should see three sections.

1. list of all the files used in the workflow
2. definition of all jobs - each job in the workflow. 4 jobs in total.
3. list of control-flow dependencies - this section specifies a partial order in which jobs are to executed.

## 2.2. Setting up the Replica Catalog

First lets change to the tutorial base directory.

```
$ cd $HOME/pegasus-wms
```

In this exercise you will insert entries into the Replica Catalog. The replica catalog that we will use today is a simple file based catalog. We also support and recommend GLOBUS RLS or a JDBC implementation for production runs.

A Replica Catalog maintains the LFN to PFN mapping for the input files of your workflow. Pegasus queries it to determine the locations of the raw input data files required by the workflow. Additionally, all the materialized data is registered into Replica Catalog for data reuse later on.

You can use the **rc-client** command to insert , query and delete from the replica catalog.

To execute the diamond dax created in **exercise 2.1**, we will need to register input file f.a in the replica catalog. The file f.a resides at /scratch/tutorial/inputdata/diamond/f.a . Let us insert a single entry into the replica catalog.

```
$ rc-client -Dpegasus.user.properties=config/properties insert f.a \
  gsiftp://viz-login.isi.edu//scratch/tutorial/inputdata/diamond/f.a pool=local
```

Let us know verify if f.a has been registered successfully by querying the replica catalog using rc-client

```
$ rc-client -Dpegasus.user.properties=config/properties lookup f.a
f.a gsiftp://viz-login.isi.edu//scratch/tutorial/inputdata/diamond/f.a pool="local"
```

The **rc-client** also allows for bulk insertion of entries. We will be inserting the entries for montage workflow using the bulk mode. The input data to be used for the montage workflow resides in the /scratch/tutorial/inputdata/0.2degree directory. We are going to insert entries into the replica catalog that point to the files in this directory.

The instructors have provided:

- A file replicas.in, the input data file for the rc-client that contains the mappings that need to be populated in the Replica Catalog. The file is inside the config directory

Instructions:

- Let us see what the file looks like.

```
$ cat config/rc.in
# file-based replica catalog: 2007-06-02T13:11:35.954-07:00
statfile_20070529_153243_22618.tbl
  gsiftp://viz-login.isi.edu/scratch/tutorial/inputdata/0.2degree/statfile.tbl
  pool="local"
2mass-atlas-990502s-j1440198.fits
  gsiftp://viz-login.isi.edu/scratch/0.2degree/2mass-atlas-990502s-j1440198.fits
  pool="local"
2mass-atlas-990502s-j1440186.fits
```

```
gsiftp://viz-login.isi.edu/scratch/0.2degree/2mass-atlas-990502s-j1440186.fits
pool="local"
2mass-atlas-990502s-j1430092.fits
gsiftp://viz-login.isi.edu/scratch/0.2degree/2mass-atlas-990502s-j1430092.fits
pool="local"
2mass-atlas-990502s-j1420198.fits
gsiftp://viz-login.isi.edu/scratch/0.2degree/2mass-atlas-990502s-j1420198.fits
pool="local"
2mass-atlas-990502s-j1420186.fits
gsiftp://viz-login.isi.edu/scratch/0.2degree/2mass-atlas-990502s-j1420186.fits
pool="local"
cimages_20070529_153243_22618.tbl
gsiftp://viz-login.isi.edu/scratch/0.2degree/cimages.tbl pool="local"
pimages_20070529_153243_22618.tbl
gsiftp://viz-login.isi.edu/scratch/0.2degree/pimages.tbl pool="local"
region_20070529_153243_22618.hdr
gsiftp://viz-login.isi.edu/scratch/0.2degree/region.hdr pool="local"
2mass-atlas-990502s-j1430080.fits
gsiftp://viz-login.isi.edu/scratch/0.2degree/2mass-atlas-990502s-j1430080.fits
pool="local"
```

- Now we are ready to run rc-client and populate the data. Since each of you have an individual file replica catalog, all the 10 entries should be successfully registered.

```
$ rc-client -Dpegasus.user.properties=config/properties --insert config/rc.in

#Successfully worked on : 10 lines
#Worked on total number of : 11 lines.
```

- Now the entries have been successfully inserted into the Replica Catalog. We should query the replica catalog for a particular lfn.

```
$ rc-client -Dpegasus.user.properties=config/properties \
            lookup pimages_20080505_143233_14944.tbl

pimages_20080505_143233_14944.tbl
gsiftp://viz-login.isi.edu/nfs/shared-scratch/tutorial/pegasus-wms/inputdata/montage/0.2degree
pool="local"
```

Congratulations!! You have the replica catalog setup correctly for use. This is the catalog which you will tinker with most, while running Pegasus.

## 2.3. Setting up the Site Catalog and Transformation Catalog

In this exercise you will setup your Site Catalog and the Transformation Catalog.

The instructors have provided:

- A ready transformation catalog (tc.data) in the \$HOME/pegasus-wms/config directory.
- A semi ready properties file in the \$HOME/pegasus-wms/config directory.
- The site catalog contains information about the layout of your grid where you want to run your workflows. For each site information like workdirectory, jobmanagers to use, gridftp servers to use and other site wide information like environment variables to be set is maintained.

You can use the sc-client command to generate a site catalog from a hand written sites.txt file.

```
$ sc-client -f config/sites.txt -o config/sites.xml

2007.06.02 17:06:12.215 PDT: [INFO] Reading config/sites.txt
2007.06.02 17:06:11.262 PDT: [INFO] Reading config/sites.txt (completed)
2007.06.02 17:06:11.276 PDT: [INFO] Written xml output to file : config/sites.xml
```

- The transformation catalog maintains information about where the application code resides on the grid. In our case, it contains the locations where the Diamond or Montage code is installed on the various grid sites. We will use the **tc-client** to add the entry for the transformation analyze into the transformation catalog.

```
$ tc-client -Dpegasus.user.properties=config/properties -a -l diamond::analyze:2.0 \
-p /nfs/software/pegasus/default/bin/keg -r local -t INSTALLED -s INTEL32::LINUX

2008.04.30 15:11:59.313 PDT: [INFO] Added tc entry sucessfully
```

Let us try and query for the entry we inserted

```
$ tc-client -Dpegasus.user.properties=config/properties -q -P -l diamond::analyze:2.0

#RESID      LTX              PFN              TYPE            SYSINFO
local      diamond::analyze:2.0  /nfs/software/pegasus/default/bin/keg  INSTALLED      INTEL32::LINUX
```

We can also query the transformation catalog for all the entries in it. Let us see what our transformation catalog looks like

```
$ tc-client -Dpegasus.user.properties=config/properties -q -B

viz  bin/mDiff
      gsiftp://viz-login.isi.edu/nfs/software/montage/default/bin/mDiff
      STATIC_BINARY  INTEL32::LINUX  ENV::MONTAGE_HOME="."
viz  bin/mFitplane
      gsiftp://viz-login.isi.edu/nfs/software/montage/default/bin/mFitplane
      STATIC_BINARY  INTEL32::LINUX  ENV::MONTAGE_HOME="."
viz  mAdd:3.0
      gsiftp://viz-login.isi.edu/nfs/software/montage/default/bin/mAdd
      STATIC_BINARY  INTEL32::LINUX  ENV::MONTAGE_HOME="."
viz  mBackground:3.0
      gsiftp://viz-login.isi.edu/nfs/software/montage/default/bin/mBackground
      STATIC_BINARY  INTEL32::LINUX  ENV::MONTAGE_HOME="."
viz  mBgModel:3.0
      gsiftp://viz-login.isi.edu/nfs/software/montage/default/bin/mBgModel
      STATIC_BINARY  INTEL32::LINUX  ENV::MONTAGE_HOME="."
viz  mConcatFit:3.0
      gsiftp://viz-login.isi.edu/nfs/software/montage/default/bin/mConcatFit
      STATIC_BINARY  INTEL32::LINUX  ENV::MONTAGE_HOME="."
viz  mDiffFit:3.0
      gsiftp://viz-login.isi.edu/nfs/software/montage/default/bin/mDiffFit
      STATIC_BINARY  INTEL32::LINUX  ENV::MONTAGE_HOME="."
viz  mImgtbl:3.0
      gsiftp://viz-login.isi.edu/nfs/software/montage/default/bin/mImgtbl
      STATIC_BINARY  INTEL32::LINUX  ENV::MONTAGE_HOME="."
viz  mJPEG:3.0
      gsiftp://viz-login.isi.edu/nfs/software/montage/default/bin/mJPEG
      STATIC_BINARY  INTEL32::LINUX  ENV::MONTAGE_HOME="."
viz  mProject:3.0
      gsiftp://viz-login.isi.edu/nfs/software/montage/default/bin/mProjectPP
      STATIC_BINARY  INTEL32::LINUX  ENV::MONTAGE_HOME="."
viz  mProjectPP:3.0
      gsiftp://viz-login.isi.edu/nfs/software/montage/default/bin/mProjectPP
      STATIC_BINARY  INTEL32::LINUX  ENV::MONTAGE_HOME="."
viz  mShrink:3.0
      gsiftp://viz-login.isi.edu/nfs/software/montage/default/bin/mShrink
      STATIC_BINARY  INTEL32::LINUX  NULL
```

- Open the properties file and check a few properties.

```
$ cat config/properties

## SELECT THE REPLICAT CATALOG MODE AND URL
pegasus.catalog.replica = SimpleFile
pegasus.catalog.replica.file = ${user.home}/pegasus-wms/config/rc.data

## SELECT THE SITE CATALOG MODE AND FILE
pegasus.catalog.site = XML
pegasus.catalog.site.file = ${user.home}/pegasus-wms/config/sites.xml

## SELECT THE TRANSFORMATION CATALOG MODE AND FILE
pegasus.catalog.transformation = File
```

```
pegasus.catalog.transformation.file = ${user.home}/pegasus-wms/config/tc.data

## USE DAGMAN RETRY FEATURE FOR FAILURES
pegasus.dagman.retry=2

## STAGE ALL OUR EXECUTABLES
pegasus.catalog.transformation.mapper = Staged

## CHECK JOB EXIT CODES FOR FAILURE
pegasus.exitcode.scope=all

## OPTIMIZE DATA AND EXECUTABLE TRANSFERS
pegasus.transfer.refiner=Bundle

#STAGE DATA AND EXECUTABLES USING GRIDFTP 3rd PARTY MODE
pegasus.transfer.*.thirdparty.sites=*

## WORK AND STORAGE DIR

pegasus.dir.storage = ${user.home}/storage
pegasus.dir.exec = ${user.home}/exec
```

## 2.4. Planning workflow using pegasus-plan and running locally using pegasus-run.

In this exercise we are going to run pegasus-plan to generate a concrete workflow from the abstract workflow (diamond.dax). The Concrete workflow generated, are condor submit files that are submitted locally using pegasus-run

The instructors have provided:

- A dax (diamond.dax) in the \$HOME/pegasus-wms/dax directory.

You will need to write some things yourself, by following the instructions below:

- Run pegasus-plan to generate the condor submit files out of the dax.
- Run pegasus-run to submit the workflow locally.

Instructions:

- Let us run pegasus-plan on the diamond dax.

```
$ cd ~/pegasus-wms
$ pegasus-plan -Dpegasus.user.properties=`pwd`/config/properties \
  --dax `pwd`/dax/diamond.dax --force\
  --dir dags -s local -o local --nocleanup
```

The above command says that we need to plan the diamond dax locally. The condor submit files are to be generated in a directory structure whose base is dags. We also are requesting that no cleanup jobs be added as we require the intermediate data to be saved. Here is the output of pegasus-plan.

```
2008.01.28 15:00:49.536 PST: [INFO] Parsing the DAX
2008.01.28 15:00:50.063 PST: [INFO] Parsing the DAX (completed)
2008.01.28 15:00:50.174 PST: [INFO] Parsing the site catalog
2008.01.28 15:00:50.327 PST: [INFO] Parsing the site catalog (completed)
2008.01.28 15:00:50.394 PST: [INFO] Doing site selection
2008.01.28 15:00:50.436 PST: [INFO] Doing site selection (completed)
2008.01.28 15:00:50.437 PST: [INFO] Grafting transfer nodes in the workflow
2008.01.28 15:00:50.508 PST: [INFO] Grafting transfer nodes in the workflow (completed)
2008.01.28 15:00:50.523 PST: [INFO] Grafting the remote workdirectory creation jobs
  in the workflow
2008.01.28 15:00:50.537 PST: [INFO] Grafting the remote workdirectory creation jobs
  in the workflow (completed)
2008.01.28 15:00:50.538 PST: [INFO] Generating the cleanup workflow
2008.01.28 15:00:50.542 PST: [INFO] Generating the cleanup workflow (completed)
```

```
2008.01.28 15:00:50.563 PST: [INFO] Generating codes for the concrete workflow
2008.01.28 15:00:50.684 PST: [INFO] Generating codes for the concrete workflow (completed)
2008.01.28 15:00:50.684 PST: [INFO] Generating code for the cleanup workflow
2008.01.28 15:00:50.718 PST: [INFO] Generating code for the cleanup workflow (completed)
2008.01.28 15:00:51.087 PST: [INFO] I have concretized your abstract workflow.
```

The workflow has been entered into the workflow database with a state of "planned".  
The next step is to start or execute your workflow. The invocation required is

```
pegasus-run -Dpegasus.user.properties=/home/trainXX/pegasus-wms/dags/trainXX/pegasus\
/diamond/run0001/pegasus.7543.properties \
/home/trainXX/pegasus-wms/dags/trainXX/pegasus/diamond/run0001
```

- **Now run pegasus-run as mentioned in the output of pegasus-plan. Do not copy the command below it is just for illustration purpose.**

```
$ pegasus-run -Dpegasus.user.properties=/nfs/home/trainXX/pegasus-wms/dags/trainXX\
/pegasus/diamond/run0001/pegasus.7543.properties \
/nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/diamond/run0001
```

Checking all your submit files for log file names.  
This might take a while... Done.

```
-----
File for submitting this DAG to Condor : diamond-0.dag.condor.sub
Log of DAGMan debugging messages : diamond-0.dag.dagman.out
Log of Condor library output : diamond-0.dag.lib.out
Log of Condor library error messages : diamond-0.dag.lib.err
Log of the life of condor_dagman itself : diamond-0.dag.dagman.log
Condor Log file for all jobs of this DAG : /tmp/diamond-07544.log
-no_submit given, not submitting DAG to Condor.
You can do this with: "condor_submit diamond-0.dag.condor.sub"
-----
```

Submitting job(s). Logging submit event(s).  
1 job(s) submitted to cluster 20068.  
I have started your workflow, committed it to DAGMan,  
and updated its state in the work database.  
A separate daemon was started to collect information about the progress of the workflow.  
The job state will soon be visible. Your workflow runs in base directory.

```
cd /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/diamond/run0001
```

```
*** To monitor the workflow you can run ***
pegasus-status -w diamond-0 -t 20080128T150049-0800
or
pegasus-status /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/diamond/run0001

*** To remove your workflow run ***
pegasus-remove -d 20068.0
or
pegasus-remove /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/diamond/run0001
```

## 2.5. Tracking the progress of the workflow and debugging the workflows.

In this exercise we are going to list ways to track your workflow, and give some debugging hints when something goes wrong.

We will change into the directory, that was mentioned by the output of pegasus-run command.

```
$ cd /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/diamond/runXXXX
```

In this directory you will see a whole lot of files. That should not scare you. Unless things go wrong, you need to look at just a very few number of files to track the progress of the workflow

- **Run the command pegasus-status as mentioned by pegasus-run above to check the status of your jobs. Use the watch command to auto repeat the command every 2 seconds.**

```
$ watch pegasus-status /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/diamond/runXXXX
-- Submitter: viz-login.isi.edu : <128.9.72.178:46426> : viz-login.isi.edu
ID      OWNER/NODENAME  SUBMITTED RUN_TIME ST PRI SIZE CMD
19982.0 train01 1/28 13:58 0+00:03:20 R 0 9.8 condor_dagman -f -
19986.0 |-findrange 1/28 14:02 0+00:00:00 I 0 9.8 kickstart -n pegas
19987.0 |-findrange 1/28 14:02 0+00:00:00 I 0 9.8 kickstart -n pegas
```

The above output shows that a couple of jobs are running under the main dagman process. Keep a lookout to track whether a workflow is running or not. If you do not see any of your job in the output for sometime (say 30 seconds), we know the workflow has finished. We need to wait, as there might be delay in Condor DAGMan releasing the next job into the queue after a job has finished successfully.

If output of pegasus-status is empty, then either your workflow has - successfully completed - stopped midway due to non recoverable error

- Another way to monitor the workflow is to check the jobstate.log file. This is the output file of the monitoring daemon that is parsing all the condor log files to determine the status of the jobs. It logs the events seen by Condor into a more readable form for us.

```
$ more jobstate.log
1201557528 INTERNAL *** TAILSTATD_STARTED ***
1201557528 INTERNAL *** DAGMAN_STARTED ***
1201557528 generate_ID000001_0 UN_READY - - -
1201557528 findrange_ID000003_0 UN_READY - - -
1201557528 findrange_ID000002_0 UN_READY - - -
1201557528 analyze_ID000004_0 UN_READY - - -
[...]
```

In the starting of the jobstate.log, when the workflow has just started running you will see a lot of entries with status UN\_READY. That designates that DAGMan has just parsed in the .dag file and has not started working on any job as yet. Initially all the jobs in the workflow are listed as UN\_READY. After sometime you will see entries in jobstate.log, that shows a job is being executed etc.

```
1201557747 generate_ID000001_0 EXECUTE 19996.0 local -
1201557747 generate_ID000001_0 GLOBUS_SUBMIT 19996.0 local -
1201557812 generate_ID000001_0 JOB_TERMINATED 19996.0 local -
1201557812 generate_ID000001_0 POST_SCRIPT_STARTED - local -
1201557817 generate_ID000001_0 POST_SCRIPT_TERMINATED 19996.0 local -
1201557817 generate_ID000001_0 POST_SCRIPT_SUCCESS - local -
```

The above shows the being submitted and then executed on the grid. In addition it lists that job is being run on the grid site local (which is your submit machine). The various states of the job while it goes through submission to execution to post processing are in UPPERCASE.

- Successfully Completed : Let us again look at the jobstate.log. This time we need to look at the last few lines of jobstate.log

```
$ tail jobstate.log
1201559232 analyze_ID000004 JOB_TERMINATED 20023.0 local -
1201559232 analyze_ID000004 POST_SCRIPT_STARTED - local -
1201559238 analyze_ID000004 POST_SCRIPT_TERMINATED 20023.0 local -
1201559238 analyze_ID000004 POST_SCRIPT_SUCCESS - local -
1201559238 INTERNAL *** DAGMAN_FINISHED ***
1201559239 INTERNAL *** TAILSTATD_FINISHED 0 ***
```

Looking at the last two lines we see that DAGMan finished, and tailstatd finished successfully with a status 0. This means workflow ran successfully. Congratulations you ran your workflow on the local site successfully. The workflow generates a single output file montage.jpg that resides in the directory **/scratch/trainXX/storage/f.d** where trainXX is your user id.

To view the file, you can copy f.d to your viz-login webspace, and view it in your web browser:

```
$ cp /scratch/trainXX/storage/f.d ~/public_html
```

Point your web browser to: <http://viz-login.isi.edu/~trainXX/f.d> where trainXX is your viz-login user id.

- Unsuccessfully Completed (Workflow execution stopped midway) : Let us again look at the jobstate.log. Again we need to look at the last few lines of jobstate.log

```
$ tail jobstate.log
1180840233 analyze_ID000004_0 JOB_TERMINATED 2787.0 local -
1180840233 analyze_ID000004_0 POST_SCRIPT_STARTED - local -
1180840238 analyze_ID000004_0 POST_SCRIPT_TERMINATED 2787.0 local -
1180840238 analyze_ID000004_0 POST_SCRIPT_FAILURE 1 local -
1180840373 INTERNAL *** DAGMAN_FINISHED ***
1180840373 INTERNAL *** TAILSTATD_FINISHED 1 ***
```

Looking at the last two lines we see that DAGMan finished, and tailstatd finished unsuccessfully with a status 1. We can easily determine which job failed. It is inter\_tx\_mDiffFit\_ID000007\_0 in this case. To determine the reason for failure we need to look at its kickstart output file which is JOBNAME.out.NNN. where NNN is 000 - NNN

## 2.6. Condor DAGMan format and log files etc.

In this exercise we will learn about the DAG file format and some of the log files generated when the DAG runs.

- Now take a look at the DAG file...

```
$ cat dags/trainXX/pegasus/diamond/run0001/diamond-0.dag
#####
# PEGASUS GENERATED SUBMIT FILE
# DAG diamond
# Index = 0, Count = 1
#####
JOB generate_ID000001 generate_ID000001.sub
RETRY generate_ID000001 2

JOB findrange_ID000002 findrange_ID000002.sub
RETRY findrange_ID000002 2

JOB findrange_ID000003 findrange_ID000003.sub
RETRY findrange_ID000003 2

JOB analyze_ID000004 analyze_ID000004.sub
RETRY analyze_ID000004 2

JOB diamond_0_pegasus_concat diamond_0_pegasus_concat.sub

JOB diamond_0_local_cdir diamond_0_local_cdir.sub
SCRIPT POST diamond_0_local_cdir /nfs/software/pegasus/default/bin/exitpost
-Dpegasus.user.properties=/nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus
/diamond/run0001/pegasus.31433.properties
/home/trainXX/pegasus-wms/dags/trainXX/pegasus/diamond/run0001/diamond_0_local_cdir.out
RETRY diamond_0_local_cdir 2

PARENT generate_ID000001 CHILD findrange_ID000002
PARENT generate_ID000001 CHILD findrange_ID000003
PARENT findrange_ID000002 CHILD analyze_ID000004
PARENT findrange_ID000003 CHILD analyze_ID000004
PARENT diamond_0_pegasus_concat CHILD generate_ID000001
PARENT diamond_0_local_cdir CHILD diamond_0_pegasus_concat
#####
# End of DAG
#####
```

- ... and the dagman.out file.

```
$ cat dags/trainXX/pegasus/diamond/run0001/diamond-0.dag.dagman.out
```

```
1/29 10:32:14 *****
1/29 10:32:14 ** condor_scheduniv_exec.20133.0 (CONDOR_DAGMAN) STARTING UP
1/29 10:32:14 ** /nfs/software/condor/6.9.4/bin/condor_dagman
1/29 10:32:14 ** $CondorVersion: 6.9.4 Aug 30 2007 $
1/29 10:32:14 ** $CondorPlatform: I386-LINUX_RHEL3 $
1/29 10:32:14 ** PID = 3202
1/29 10:32:14 ** Log last touched time unavailable (No such file or directory)
1/29 10:32:14 ***** [....]
1/29 10:32:27 Submitting Condor Node diamond_0_local_cdir job(s)...
1/29 10:32:27 submitting: condor_submit
-a dag_node_name' '=' 'diamond_0_local_cdir
-a +DAGManJobId' '=' '20133
-a DAGManJobId' '=' '20133
-a submit_event_notes' '=' 'DAG' 'Node:' 'diamond_0_local_cdir
-a +DAGParentNodeNames' '=' '""diamond_0_local_cdir.sub
1/29 10:32:27 From submit: Submitting job(s).
1/29 10:32:27 From submit: Logging submit event(s).
1/29 10:32:27 From submit: 1 job(s) submitted to cluster 20134.
1/29 10:32:27 assigned Condor ID (20134.0)
1/29 10:32:27 Just submitted 1 job this cycle...
1/29 10:32:27 Event: ULOG_SUBMIT for Condor Node diamond_0_local_cdir (20134.0)
1/29 10:32:27 Number of idle job procs: 1
1/29 10:32:27 Of 6 nodes total: 1/29 10:32:27 Done Pre Queued Post Ready Un-Ready Failed
1/29 10:32:27 === === === === === ===
1/29 10:32:27 0 0 1 0 0 5 0 [....]
1/29 10:33:24 Done Pre Queued Post Ready Un-Ready Failed
1/29 10:33:24 === === === === ===
1/29 10:33:24 6 0 0 0 0 0 1/29 10:33:24 All jobs Completed!
1/29 10:33:24 Note: 0 total job deferrals because of -MaxJobs limit (0)
1/29 10:33:24 Note: 0 total job deferrals because of -MaxIdle limit (0)
1/29 10:33:24 Note: 0 total PRE script deferrals because of -MaxPre limit (20)
1/29 10:33:24 Note: 0 total POST script deferrals because of -MaxPost limit (20)
1/29 10:33:24 **** condor_scheduniv_exec.20133.0 (condor_DAGMAN) EXITING WITH STATUS 0
```

## 2.7. Removing a running workflow

Sometimes you may want to halt the execution of the workflow or just permanently remove it. You can stop/halt a workflow by running the pegasus-remove command mentioned in the output of pegasus-run

```
$ pegasus-remove /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/diamond/runXXXX
```

```
Job 2788.0 marked for removal
```

## 2.8. Planning workflow using pegasus-plan and Running pegasus-run to submit the workflow to a grid resource.

In this exercise we are going to run pegasus-plan to generate a concrete workflow from the abstract workflow (montage.dax). The Concrete workflow generated, are condor submit files that are submitted to remote grid resources using pegasus-run

The instructors have provided:

- A dax (montage.dax) in the \$HOME/pegasus-wms/dax/ directory.

You will need to write some things yourself, by following the instructions below:

- Run pegasus-plan to generate the condor submit files out of the dax.
- Run pegasus-run to submit the workflow to the grid.

Instructions:

- Let us run pegasus-plan on the montage dax on the skynet cluster. If multiple sites are available you could provide the sites using a comma "," separated list like skynet,viz etc.

```
$ cd $HOME/pegasus-wms
$ pegasus-plan -Dpegasus.user.properties=`pwd`/config/properties \
  --dir dags --sites skynet --output local --force \
  --nocleanup --dax `pwd`/dax/montage.dax
```

The above command says that we need to plan the montage dax on the skynet site. The output data needs to be transferred back to the local host. The condor submit files are to be generated in a directory structure whose base is dags. We also are requesting that no cleanup jobs be added as we require the intermediate data on the remote host. Here is the output of pegasus-plan.

```
2008.04.27 12:59:15.171 PDT: [INFO] Parsing the DAX
2008.04.27 12:59:15.972 PDT: [INFO] Parsing the DAX (completed)
2008.04.27 12:59:16.085 PDT: [INFO] Parsing the site catalog
2008.04.27 12:59:16.257 PDT: [INFO] Parsing the site catalog (completed)
2008.04.27 12:59:16.323 PDT: [INFO] Doing site selection
2008.04.27 12:59:16.427 PDT: [INFO] Doing site selection (completed)
2008.04.27 12:59:16.428 PDT: [INFO] Grafting transfer nodes in the workflow
2008.04.27 12:59:16.621 PDT: [INFO] Grafting transfer nodes in the workflow (completed)
2008.04.27 12:59:16.628 PDT: [INFO] Grafting the remote workdirectory creation jobs
  in the workflow
2008.04.27 12:59:16.644 PDT: [INFO] Grafting the remote workdirectory creation jobs
  in the workflow (completed)
2008.04.27 12:59:16.645 PDT: [INFO] Generating the cleanup workflow
2008.04.27 12:59:16.649 PDT: [INFO] Generating the cleanup workflow (completed)
2008.04.27 12:59:16.668 PDT: [INFO] Generating codes for the concrete workflow
2008.04.27 12:59:17.330 PDT: [INFO] Generating codes for the concrete workflow (completed)
2008.04.27 12:59:17.331 PDT: [INFO] Generating code for the cleanup workflow
2008.04.27 12:59:17.414 PDT: [INFO] Generating code for the cleanup workflow (completed)
2008.04.27 12:59:17.421 PDT: [INFO]
```

I have concretized your abstract workflow. The workflow has been entered into the workflow database with a state of "planned". The next step is to start or execute your workflow. The invocation required is

```
pegasus-run -Dpegasus.user.properties=/nfs/home/trainXX/pegasus-wms/dags/trainXX\
/pegasus/montage/run0001/pegasus.57010.properties \
  --nodatabase /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/montage/run0001
```

- If you get any errors above while running pegasus-plan you can add -vvvvv to enable maximum verbosity on pegasus-run.
- **Now run pegasus-run as mentioned in the output of pegasus-plan. Do not run the command below it is only for illustration purposes.**

```
$ pegasus-run -Dpegasus.user.properties=/nfs/home/trainXX/pegasus-wms/dags/trainXX\
/pegasus/montage/run0001/pegasus.51773.properties \
  /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/montage/run0001
```

Checking all your submit files for log file names.  
This might take a while... Done.

```
-----
File for submitting this DAG to Condor : montage-0.dag.condor.sub
Log of DAGMan debugging messages : montage-0.dag.dagman.out
Log of Condor library output : montage-0.dag.lib.out
Log of Condor library error messages : montage-0.dag.lib.err
Log of the life of condor_dagman itself : montage-0.dag.dagman.log
Condor Log file for all jobs of this DAG : /tmp/montage-051774.log -no_submit given, not
submitting DAG to Condor.
You can do this with: "condor_submit montage-0.dag.condor.sub"
-----
```

Submitting job(s). Logging submit event(s).  
1 job(s) submitted to cluster 19968.

I have started your workflow, committed it to DAGMan,  
and updated its state in the work database.

```
A separate daemon was started to collect information about the progress of the workflow.
The job state will soon be visible. Your workflow runs in base directory.

cd /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/montage/run0001

*** To monitor the workflow you can run ***

pegasus-status -w montage-0 -t 20080128T114525-0800
or
pegasus-status /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/montage/run0001

*** To remove your workflow run ***

pegasus-remove -d 19968.0
or
pegasus-remove /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/montage/run0001
```

The above command submits the workflow to Condor DAGMan/CondorG. After submitting it starts a monitoring daemon tailstatd that parses the condor log files to update the status of the jobs and push it in a work database.

Monitor the workflow using the commands provided in the output of the pegasus-run command and other commands explained earlier.

The workflow generates a single output file montage.jpg that resides in the directory **/scratch/trainXX/storage/montage.jpg** where trainXX is your user id if it runs successfully

To view the image, you can copy montage.jpg to your viz-login webspace, and view it in your web browser:

```
$ cp /scratch/trainXX/storage/montage.jpg ~/public_html
```

Point your web browser to: <http://viz-login.isi.edu/~trainXX/montage.jpg> where trainXX is your viz-login user id.

## 3. Advanced Exercises

### 3.1. Optimizing a workflow by clustering small jobs (To Be Done offline)

Sometimes a workflow may have too many jobs whose execution time is a few seconds long. In such instances the overhead of scheduling each job on a grid is too large and the runtime of the entire workflow can be optimized by using Pegasus clustering techniques. One such technique is to cluster jobs horizontally on the same level into one or more sequential jobs.

```
$ cd $HOME/pegasus-wms
$ pegasus-plan -Dpegasus.user.properties=`pwd`/config/properties \
  --dir `pwd`/dags --sites skynet --output local --nocleanup --force\
  --cluster horizontal --dax `pwd`/dax/montage.dax
```

### 3.2. Data Reuse

In the DAX you can specify what output data products you want to track in the replica catalog. This is done by setting the register flags with the output files for a job. For our tutorial, we only register the final output data products. So if you were able to execute the diamond or the montage workflow successfully, we can do data reuse. Let us run **pegasus-plan** on the diamond workflow again. However, this time we will remove the **--force** option.

```
$ cd $HOME/pegasus-wms
$ pegasus-plan -Dpegasus.user.properties=`pwd`/config/properties --dax `pwd`/dax/diamond.dax \
  --dir `pwd`/dags -s local -o local --nocleanup

2008.04.30 17:12:09.851 PDT: [INFO] Parsing the DAX
2008.04.30 17:12:10.299 PDT: [INFO] Parsing the DAX (completed)
2008.04.30 17:12:10.344 PDT: [INFO] Parsing the site catalog
2008.04.30 17:12:10.480 PDT: [INFO] Parsing the site catalog (completed)
```

```
2008.04.30 17:12:10.545 PDT: [INFO] Doing site selection
2008.04.30 17:12:10.568 PDT: [INFO] Doing site selection (completed)
2008.04.30 17:12:10.569 PDT: [INFO] Grafting transfer nodes in the workflow
2008.04.30 17:12:10.618 PDT: [INFO] The leaf file f.d is already at the output pool local
2008.04.30 17:12:10.618 PDT: [INFO] Grafting transfer nodes in the workflow (completed)
2008.04.30 17:12:10.627 PDT: [INFO] Grafting the remote workdirectory creation jobs in
the workflow
2008.04.30 17:12:10.632 PDT: [INFO] Grafting the remote workdirectory creation jobs in
the workflow (completed)
2008.04.30 17:12:10.632 PDT: [INFO] Generating the cleanup workflow
2008.04.30 17:12:10.636 PDT: [INFO] Generating the cleanup workflow (completed)
2008.04.30 17:12:10.662 PDT: [INFO]
```

The executable workflow generated contains no nodes.  
It seems that the output files are already at the output site.  
To regenerate the output data from scratch specify --force option.

```
2008.04.30 17:12:10.663 PDT: [INFO] Time taken to execute is 1.081 seconds
```

You can increase the debug level to see how pegasus deletes the jobs bottom up of the workflow. Pass -vvvvv to pegasus-plan command.

### 3.3. Nested DAGs and Deferred Planning

DAGMan allows the users to submit a DAG of DAG's. A user can have a workflow in which one of the jobs maybe another DAG. Pegasus leverages this feature of DAGMan for partitioning workflows and interleaving the planning and execution of subsequent sub workflows. Let us partition the montage workflows using the **partitiondax** command.

```
$ cd $HOME/pegasus-wms
$ partitiondax -Dpegasus.user.properties=./config/properties --dax dax/montage.dax \
--dir ./pdags/ --type horizontal

2008.04.30 16:15:36.755 PDT: [INFO] Parsing the DAX
2008.04.30 16:15:37.325 PDT: [INFO] Parsing the DAX (completed)
2008.04.30 16:15:37.805 PDT: [INFO] Determining relations between partitions
2008.04.30 16:15:37.807 PDT: [INFO] Determining relations between partitions (completed)
2008.04.30 16:15:37.810 PDT: [INFO] Time taken to execute is 1.314 seconds
```

Now let us look at how this partitioned workflow looks like. In the ./pdags directory you will see a collection of dax files and a pdax file.

```
$ ls pdags/
montage.pdax          partition_montage_2.dax  partition_montage_4.dax
partition_montage_6.dax  partition_montage_8.dax  partition_montage_1.dax
partition_montage_3.dax  partition_montage_5.dax  partition_montage_7.dax
partition_montage_9.dax
```

Let us look at the generated pdax file. It is an XML file that links the smaller dax files together. We partitioned the workflows using horizontal partitioning type. It means all the jobs on a particular level of the original workflow become a sub workflow.

```
$ train02@viz-login:~/pegasus-wms$ cat pdags/montage.pdax

<?xml version="1.0" encoding="UTF-8"?>
<!-- generated: 2008-04-30T16:15:37-07:00-->
<pdag xmlns="http://pegasus.isi.edu/schema/PDAX"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://pegasus.isi.edu/schema/PDAX
http://pegasus.isi.edu/schema/pdax-2.0.xsd" name="montage"
  index="0" count="1" version="2.0" >

  <partition name="partition_montage" index="1" id="ID1">
    <job name="mProjectPP" id="ID000003"/>
  </partition>
</pdag>
```

```
<job name="mProjectPP" id="ID000004"/>
<job name="mProjectPP" id="ID000006"/>
<job name="mProjectPP" id="ID000005"/>
<job name="mProjectPP" id="ID000001"/>
<job name="mProjectPP" id="ID000002"/>
</partition>
....

<!-- Relations between the partitions -->
<child ref="ID8">
  <parent ref="ID7"/>
</child>
<child ref="ID4">
  <parent ref="ID3"/>
</child>
<child ref="ID6">
  <parent ref="ID5"/>
</child>
<child ref="ID3">
  <parent ref="ID2"/>
</child>
<child ref="ID2">
  <parent ref="ID1"/>
</child>
<child ref="ID9">
  <parent ref="ID8"/>
</child>
<child ref="ID7">
  <parent ref="ID6"/>
</child>
<child ref="ID5">
  <parent ref="ID1"/>
</child>
<parent ref="ID4"/>
</child>
</pdag>
```

Now let us submit this partitioned workflow . The only difference is instead of invoking **pegasus-plan** with **--dax** option, we invoke with **--pdax** option

```
$ pegasus-plan -Dpegasus.user.properties=`pwd`/config/properties \
  --pdax `pwd`/pdags/montage.pdax --dir `pwd`/dags -s skynet -o local \
  --nocleanup --force
```

```
2008.04.30 16:40:16.158 PDT: [INFO] Parsing the site catalog
2008.04.30 16:40:16.641 PDT: [INFO] Parsing the site catalog (completed)
2008.04.30 16:40:17.272 PDT: [INFO]
```

I have concretized your abstract workflow. The workflow has been entered into the workflow database with a state of "planned". The next step is to start or execute your workflow. The invocation required is

```
pegasus-run -Dpegasus.user.properties=/nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus\
/montage/run0003/pegasus.23677.properties \
  --nodatabase /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/montage/run0003
```

```
2008.04.30 16:40:17.272 PDT: [INFO] Time taken to execute is 1.429 seconds
```

Now submit the outer level workflow that is the workflow of workflows ( dag of dags ). **Submit the command that is mentioned in the output of your pegasus-plan invocation.**

```
$ pegasus-run -Dpegasus.user.properties=/nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus\
/montage/run0003/pegasus.23677.properties \
  --nodatabase /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/montage/run0003
```

```
Checking all your submit files for log file names.
```

## Student notes for Pegasus WMS tutorial

```
This might take a while...
Done.
```

```
-----
File for submitting this DAG to Condor           : montage-0.dag.condor.sub
Log of DAGMan debugging messages               : montage-0.dag.dagman.out
Log of Condor library output                   : montage-0.dag.lib.out
Log of Condor library error messages           : montage-0.dag.lib.err
Log of the life of condor_dagman itself        : montage-0.dag.dagman.log
```

```
Condor Log file for all jobs of this DAG       : /tmp/montage-023678.log
-no_submit given, not submitting DAG to Condor. You can do this with:
"condor_submit montage-0.dag.condor.sub"
-----
```

```
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 21970.
```

I have started your workflow, committed it to DAGMan, and updated its state in the work database. A separate daemon was started to collect information about the progress of the workflow. The job state will soon be visible. Your workflow runs in base directory.

```
cd /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/montage/run0003
```

```
*** To monitor the workflow you can run ***
```

```
pegasus-status -w montage-0 -t 20080430T164015-0700
```

```
or
```

```
pegasus-status /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/montage/run0003
```

```
*** To remove your workflow run ***
```

```
pegasus-remove -d 21970.0
```

```
or
```

```
pegasus-remove /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/montage/run0003
```

Now let us look how the outer level dag looks like. **Let's cd in to the directory that is mentioned in the output of your invocation of pegasus-run.** This is where the the outer - level workflow resides.

```
$ cd /nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/montage/run0003
$ train02@viz-login:~/pegasus-wms/dags/trainXX/pegasus/montage/run0003$ more montage-0.dag
```

```
#####
# PEGASUS WMS GENERATED DAG FILE
# DAG montage
# Index = 0, Count = 1
#####
JOB 00/ID1 00/ID1.sub
SCRIPT PRE 00/ID1 /nfs/software/pegasus/default/bin/pegasus-plan
-Dpegasus.user.properties=/nfs/home/trainXX/pegasus-wms/dags/trainXX/pegasus/montage/
run0003/pegasus.23677.properties -Dpegasus.log.*/=/nfs/home/trainXX/pegasus-wms/
dags/trainXX/pegasus/montage/run0003/00/ID1.pre.log --dax /nfs/home/train02/
pegasus-wms/pdags/partition_montage_1.dax --dir /nfs/home/trainXX/pegasus-wms/dags/
trainXX/pegasus/montage/run0003/00/PID1 --basename PID1 --sites tg_ncsa
--output local --nocleanup --monitor --deferred
--randomdir=trainXX/pegasus/montage/run0003/00/PID1 --group pegasus

RETRY 00/ID1 2

---
```

The prescript invoke the planning operation on the sub workflows. The 00/ID1.sub files are condor dagman submit files, that execute condor\_dagman for the subworkflow. Each subworkflow runs in it's own submit directory, like **00/PIDX** where **X** is the partition number

```
$ train02@viz-login:~/pegasus-wms/dags/trainXX/pegasus/montage/run0003$ ls 00/PID1/
PID1.cache                               mProjectPP_ID000001.err.000  mProjectPP_ID000004.sub
partition_montage_1_tg_ncsa_cdir.err.000
```

```
PID1.dag          mProjectPP_ID000001.out.000  mProjectPP_ID000005.err.000
                  partition_montage_1_tg_ncsa_cdir.out.000
PID1.dag.dagman.out  mProjectPP_ID000001.sub      mProjectPP_ID000005.out.000
                  partition_montage_1_tg_ncsa_cdir.sub
PID1.dot          mProjectPP_ID000002.err.000  mProjectPP_ID000005.sub
                  pegasus.37039.properties
PID1.log          mProjectPP_ID000002.out.000  mProjectPP_ID000006.err.000
                  rc_tx_tg_ncsa_0.err.000
braindump.txt     mProjectPP_ID000002.sub      mProjectPP_ID000006.out.000
                  rc_tx_tg_ncsa_0.in
chmod_mProjectPP_ID000005_0.err.000  mProjectPP_ID000003.err.000  mProjectPP_ID000006.sub
                  rc_tx_tg_ncsa_0.out.000
chmod_mProjectPP_ID000005_0.out.000  mProjectPP_ID000003.out.000  partition_montage_1.dax
                  rc_tx_tg_ncsa_0.sub
chmod_mProjectPP_ID000005_0.sub      mProjectPP_ID000003.sub
partition_montage_1_pegasus_concat.err  remove.db
cleanup          mProjectPP_ID000004.err.000
partition_montage_1_pegasus_concat.out  tailstatd.log
jobstate.log     mProjectPP_ID000004.out.000
partition_montage_1_pegasus_concat.sub
```

### 3.4. Executing the workflow in a non shared filesystem environment. (To Be Done offline)

All the jobs till now have been executed on the shared filesystem on the `tg_ncsa` cluster. The input data required by the Montage workflow was staged to a directory on the shared filesystem. All the jobs were then executed in that directory.

A recent feature addition to Pegasus ( still in testing phase ), allows you to execute each of the jobs in a `tmp` directory on the worker nodes filesystem. For this to happen, a Second Level Staging (SLS) needs to occur, that transfers the data from the directory on the shared filesystem to a directory on the local filesystem of the worker node.

```
Set the property
pegasus.execute.*.filesystem.local=true

in your properties file.

Repeat Exercise 2.8.
```