Chapter 1. Pegasus Quick Start Guide

1. Getting Started

This document is structured to get you and up running with the Pegasus planner with minimal effort.

Download the latest release of Pegasus from http://pegasus.isi.edu

Pegasus can alternatively be found in the latest version of VDT at http://vdt.cs.wisc.edu

To install Untar the distribution

```
$ gtar zxvf pegasus-binary-<version>.tar.gz
```

This guide assumes you are using a Bourne Shell derived shell. e.g. /bin/sh or /bin/bash.

Set PEGASUS_HOME environment variable

```
$ unset PEGASUS_HOME

$ export PEGASUS_HOME=<path to pegasus directory>
```

Set up the PEGASUS user environment

```
$ unset CLASSPATH
$ source $PEGASUS_HOME/setup.sh
```

Check Java version and ensure it is JAVA 1.5.x

```
$ java -version

java version "1.5.0_07"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_07-164)
Java HotSpot(TM) Client VM (build 1.5.0_07-87, mixed mode, sharing)
```

This guide assumes that you have both Globus http://www.globus.org and Condor http://cs.wisc.edu/condor installed and available on the submit machine. You only need the Globus proxy clients if you plan to run locally on your machine. Pegasus does require a personal Condor installation with a condor queue on the submit machine.

Check Globus Installation

```
$ echo $GLOBUS_LOCATION

/path/to/globus/install
```

Make sure you source the Globus environment

```
$ GLOBUS_LOCATION/etc/globus-user-env.sh
```

Now generate a proxy certificate from you X509 credentials. If you don't have any Grid credentials you can try to ask your local grid expert or look on the Globus website for instructions on how to use SimpleCA to generate Grid credentials.

```
$ grid-proxy-init
```

Make sure you have condor installed and can submit jobs into condor. Try checking the status of the condor queue.

```
$ condor_q
```

If the above tests work all is fine, otherwise contact your friendly neighborhood administrator.

2. Abstract Workflow (DAX)

Pegasus uses an XML format to describe the input abstract workflow (DAX). The DAX is divided into 3 sections.

- **File List:** This section lists all the files used in the workflow. They can be of type input, output, inout or executable
- **Tasks:** This section lists all the tasks or jobs in the workflow. Along with the tasks are arguments required to invoke the tasks, any profiles that may be associated with the tasks and any files used as input or output by the task.
- **Dependencies:** This section lists all the dependencies betweeen the tasks in the workflow.

For an example DAX see the file $PEGASUS_HOME/examples/blackdiamond.dax

A document describing various ways to generate a DAX and the DAX schema can be found on the Pegaus website at http://pegasus.isi.edu/doc.php

3. Catalogs

Pegasus uses several catalogs to help in its planning. These catalogs need to be setup before Pegasus can plan a workflow. The catalogs used by Pegasus are :

- **Replica Catalog:** This catalog is used to lookup the location of input data as well as any existing output data referenced in the abstract workflow.
- **Site Catalog:** This catalog is used to track information about the various sites and their layout on the grid.
- **Transformation Catalog:** This catalog is used to lookup information about the location of executables (transformation) installed or available for staging on the Grid.

3.1. Replica Catalog (RC)

The Replica Catalog keeps mappings of logical file ids/names (LFN's) to physical file ids/names (PFN's). A single LFN can map to several PFN's. A PFN consists of a URL with protocol, host and port information and a path to a file. Along with the PFN one can also store additional key/value attributes to be associated with a PFN.

Pegasus supports 3 different implemenations of the Replica Catalog.

1. Simple File
2. Database via JDBC
3. Replica Location Service (RLS)

In this guide we will only use the Simple File. The same client will however work for all 3 implementations. The implementation that the client talks to is configured using Pegasus properties.

The black diamond example mentioned in Section 2 takes one input file, which we will have to tell the replica catalog. Create a file f.a by running the date command as shown

```
$ date > $HOME/f.a
```

We now need to register this file in the replica catalog using the rc-client. Replace the gsiftp://url with the appropriate parameters for your grid site.

```
$ rc-client -Dpegasus.catalog.replica=SimpleFile -Dpegasus.catalog.replica.file=$HOME/rc \
insert f.a gsiftp://somehost:port/path/to/file/f.a pool=local
```

You may first want to check, if the file registeration made it into the replica catalog. Since we are using a Simple File catalog we can just go look at the file $HOME/rc to see if there are any entries in there.

```
$ cat $HOME/rc

# file-based replica catalog: 2007-07-10T17:52:53.405-07:00
f.a gsiftp://smarty.isi.edu/nfs/asd2/gmehta/f.a pool="local"
```

The above line shows that entry for file f.a was made correctly.

If you are using some other mode of RC you can also use the rc-client to look for entries.

```
$ rc-client -Dpegasus.catalog.replica=SimpleFile -Dpegasus.catalog.replica.file=$HOME/rc \
lookup LFN f.a

f.a gsiftp://smarty.isi.edu/nfs/asd2/gmehta/f.a pool="local"
```

Your Replica Catalog is now set.

3.2. Site Catalog (SC)

The Site Catalog describes the compute resources (which are often clusters) that we intend to run the workflow upon. A site is a homogeneous part of a cluster that has at least a single GRAM gatekeeper with a jobmanager-fork and jobmanager-<scheduler> interface and at least one gridftp server along with a shared file system. The GRAM gatekeeper can be either WS GRAM or Pre-WS GRAM. A site can also be a condor pool or glidein pool with a shared file system.

Pegasus currently supports two implementation of the Site Catalog

1. XML
2. File

The format for the File is as follows

```
site site_id {
  #required. Can be a dummy value if using Simple File RC
  lrc "rls://someurl"

  #required on a shared file system
  workdir "path/to/a/tmp/shared/file/sytem/"

  #required one or more entries
  gridftp "gsiftp://hostname/mountpoint” "GLOBUS VERSION"

  #required one or more entries
  universe transfer "hostname/jobmanager-<scheduler>" "GLOBUS VERSION"

  #reqired one or more entries
  universe vanilla "hostname/jobmanager-<scheduler>" "GLOBUS VERSION"

  #optional
  sysinfo  "ARCH::OS:OSVER:GLIBC"

  #optional
  gridlaunch "/path/to/gridlaunch/executable"

  #optional zero or more entries
  profile namespace "key" "value"
}
```

The gridlaunch and profile entries are optional. All the rest are required for each pool. Also the transfer and vanilla universe are mandatory. You can add multiple transfer and vanilla universe if you have more then one head node on the cluster. The entries in the Site Catalog have the following meaning:

1. site - A site identifier.
2. lrc - URL for a local replica catalog (LRC) to register your files in. Only used for RLS implementation of the RC
3. workdir - A remote working directory (Should be on a shared file system)
4. gridftp - A URL prefix for a remote storage location.
5. universe - Different universes are supported which map to different batch jobmanagers.

"vanilla" for compute jobs and "transfer" for transfer jobs are mandatory. Generally a transfer universe should map to the fork jobmanager.

6. gridlaunch - Path to the remote kickstart tool (provenance tracking)
7. sysinfo - The arch/os/osversion/glibc of the site. The format is ARCH::OS:OSVER:GLIBC where OSVERSION and GLIBC are optiona.

   ARCH can have one of the following values INTEL32, INTEL64, SPARCV7, SPARCV9, AIX, AMD64. OS can have one of the following values LINUX,SUNOS. The default value for sysinfo if none specified is INTEL32::LINUX
8. Profiles - One or many profiles can be attached to a pool.

   One example is the environments to be set on a remote pool.

Create a file called sites.txt and edit the contents. Replace '$HOME' in the example below to your home directory.

```
$ emacs $HOME/sites.txt
```

Let's say you have one cluster available to run your jobs called clus1. You need to add 2 sections in the pool.config.txt file, one for the cluster and one for the local machine. The local site entry is mandatory. Change all the entries below to reflect your local host and cluster setting including all environment variables

```
site local {
lrc "rlsn://localhost"
workdir "$HOME/workdir"
gridftp "gsiftp://localhost/$HOME/storage" "4.0.5"
universe transfer "localhost/jobmanager-fork" "4.0.5"
universe vanilla  "localhost/jobmanager-fork" "4.0.5"
gridlaunch "/nfs/vdt/pegasus/bin/kickstart"
sysinfo "INTEL32::LINUX"
profile env "PEGASUS_HOME" "/nfs/vdt/pegasus"
profile env "GLOBUS_LOCATION" "/vdt/globus"
profile env "LD_LIBRARY_PATH" "/vdt/globus/lib"
profile env "JAVA_HOME" /vdt/java
}

site clus1 {
lrc "rlsn://clus1.com"
workdir "$HOME/workdir-clus1"
gridftp "gsiftp://clus1.com/jobmanager-fork" "4.0.3"
universe transfer "clus1.com/jobmanager-fork" "4.0.3"
universe vanilla  "clus1.com/jobmanager-pbs" "4.0.3"
sysinfo "INTEL32::LINUX"
gridlaunch "/opt/nfs/vdt/pegasus/bin/kickstart"
profile env "PEGASUS_HOME" "/opt/nfs/vdt/pegasus"
profile env "GLOBUS_LOCATION" "/opt/vdt/globus"
profile env "LD_LIBRARY_PATH" "/opt/vdt/globus/lib"
}
```

This file can be used as is by Pegasus but we prefer the XML version of the Site Catalog because it allows for a richer description of the site. The tool to convert the text site catalog to XML is called sc-client

```
$ sc-client --files $HOME/sites.txt --output $HOME/sites.xml


2007.07.10 19:04:34.799 PDT: [INFO] Reading $HOME/sites.txt
2007.07.10 19:04:34.844 PDT: [INFO] Reading $HOME/sites.txt (completed)
2007.07.10 19:04:34.851 PDT: [INFO] Written xml output to file : $HOME/sites.xml
```

Cat the sites.xml file and just take a look.

```
$ cat $HOME/sites.xml

<sitecatalog xmlns="http://pegasus.isi.edu/schema/sitecatalog"
```

```
  xsi:schemaLocation="http://pegasus.isi.edu/schema/sitecatalog
  http://pegasus.isi.edu/schema/sc-2.0.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.0">
  <site handle="local" gridlaunch="/nfs/vdt/pegasus/bin/kickstart"
   sysinfo="INTEL32::LINUX">
    <profile namespace="env" key="PEGASUS_HOME" >/nfs/vdt/pegasus</profile>
    <profile namespace="env" key="GLOBUS_LOCATION" >/vdt/globus</profile>
    <profile namespace="env" key="LD_LIBRARY_PATH" >/vdt/globus/lib</profile>
    <profile namespace="env" key="JAVA_HOME" >/vdt/java</profile>
    <lrc url="rlsn://localhost" />
    <gridftp  url="gsiftp://localhost" storage="/$HOME/storage" major="4" minor="0"
     patch="5">
    </gridftp>
    <jobmanager universe="transfer" url="localhost/jobmanager-fork" major="4" minor="0"
     patch="5" />
    <jobmanager universe="vanilla" url="localhost/jobmanager-fork" major="4" minor="0"
     patch="5" />
    <workdirectory >$HOME/workdir</workdirectory>
  </site>
  <site handle="clus1" gridlaunch="/opt/nfs/vdt/pegasus/bin/kickstart"
   sysinfo="INTEL32::LINUX">
    <profile namespace="env" key="PEGASUS_HOME" >/opt/nfs/vdt/pegasus</profile>
    <profile namespace="env" key="GLOBUS_LOCATION" >/opt/vdt/globus</profile>
    <profile namespace="env" key="LD_LIBRARY_PATH" >/opt/vdt/globus/lib</profile>
    <lrc url="rlsn://clus1.com" />
    <gridftp  url="gsiftp://clus1.com" storage="/jobmanager-fork" major="4" minor="0"
     patch="3">
    </gridftp>
    <jobmanager universe="transfer" url="clus1.com/jobmanager-fork" major="4" minor="0"
     patch="3" />
    <jobmanager universe="vanilla" url="clus1.com/jobmanager-pbs" major="4" minor="0"
     patch="3" />
    <workdirectory >$HOME/workdir-clus1</workdirectory>
  </site>
</sitecatalog>
```

The site catalog can also be generated using the pegasus-get-sites command for the OSG grid using the VORS catalog.

To configure Pegasus to pick up this site catalog file we will add some entries in the Pegasus properties file in Section 4.

3.3. Transformation Catalog (TC)

The Transformation Catalog maps logical transformations to physical executables on the system. It also provides additional information about the transformation as to what system they are compiled for, what profiles or environment variables need to be set when the transformation is invoked etc.

Pegasus currently supports two implementations of the Transformation Catalog

1. **File:** A simple multi column text file

2. **Database:** A database backend (MySQL or PostgreSQL) via JDBC

In this guide we will look at the format of the File based TC. The File based TC is also used as a bulk input for populating the Database implementation.

The format of the File is as follows

```
#site  logicaltr   physicaltr   type  system  profiles(NS::KEY="VALUE")

site1 sys::date:1.0 /usr/bin/date  INSTALLED INTEL32::LINUX:FC4.2:3.6
                   ENV::PATH="/usr/bin";PEGASUS_HOME="/usr/local/pegasus"
```

The system and profile entries are optional and will use default values if not specified. The entries in the file format have the following meaning:

1. site - A site identifier.
2. logicaltr - The logical transformation name. The format is NAMESPACE::NAME:VERSION where NAMESPACE and NAME are optional.
3. physicaltr - The physical transformation path or URL.

   If the transformation type is INSTALLED then it needs to be an absolute path to the executable. If the type is STATIC_BINARY then the path needs to be a HTTP, FTP or gsiftp URL
4. type - The type of transformation. Can have on of two values
   - INSTALLED: This means that the transformation is installed on the remote site
   - STATIC_BINARY: This means that the transformation is available as a static binary and can be staged to a remote site.
5. system - The system for which the transformation is compiled.

   The formation of the sytem is ARCH::OS:OSVERSION:GLIBC where the GLIBC and OS VERSION are optional. ARCH can have one of the following values INTEL32, INTEL64, SPARCV7, SPARCV9, AIX, AMD64. OS can have one of the following values LINUX,SUNOS. The default value for system if none specified is INTEL32::LINUX
6. Profiles - The profiles associated with the transformation. For indepth information about profiles and their priorities read the Profile Guide.

   The format for profiles is NS::KEY="VALUE" where NS is the namespace of the profile e.g. Pegasus,condor,DAGMan,env,globus. The key and value can be any strings. Remember to quote the value with double quotes. If you need to specify several profiles you can do it in several ways
   - NS1::KEY1="VALUE1",KEY2="VALUE2";NS2::KEY3="VALUE3",KEY4="VALUE4"

     This is the most optimized form. Multiple key values for the same namespace are separated by a comma "," and different namespaces are separated by a semicolon ";"
   - NS1::KEY1="VALUE1";NS1::KEY2="VALUE2";NS2::KEY3="VALUE3";NS2::KEY4="VALUE4"

     You can also just repeat the triple of NS::KEY="VALUE" separated by semicolons for a simple format;

We need to map our declared transformations (preprocess, findranage, and analyze) from the example DAX above to a simple "mock application" name "keg" ("canonical example for the grid") which reads input files designated by arguments, writes them back onto output files, and produces on STDOUT a summary of where and when it was run. Keg ships with Pegasus in the bin directory. Run keg on the command line to see how it works.

```
$ keg -o /dev/fd/1

Timestamp Today: 20040624T054607-05:00 (1088073967.418;0.022)
Applicationname: keg @ 10.10.0.11 (VPN)
Current Workdir: /home/unique-name
Systemenvironm.: i686-Linux 2.4.18-3
Processor Info.: 1 x Pentium III (Coppermine) @ 797.425
Output Filename: /dev/fd/1
```

Now we need to map all 3 transformations onto the "keg" executable. We place these mappings in our File transformation catalog for site clus1. In earlier version of Pegasus one had to define entries for Pegasus executables like transfer, replica client, dirmanager etc on each site as well as site "local". This is no longer required. Pegasus 2.0 and later automatically picks up the paths for these binaries from the environment profile PEGASUS_HOME set in the site catalog for each site.

Create a file called tc.data and edit the contents

```
$ vim $HOME/tc
```

```
# Site  LFN  PFN  TYPE  SYSTEM PROFILE

clus1 black::preprocess:1.0  gsiftp://clus1.com/opt/nfs/vdt/pegasus/bin/keg
                     STATIC_BINARY INTEL32::LINUX ENV::KEY1="VALUE1"
clus1 black::findrange:1.0  gsiftp://clus1.com/opt/nfs/vdt/pegasus/bin/keg
                     STATIC_BINARY INTEL32::LINUX ENV::KEY2="VALUE2"
```

Note: A single entry needs to be on one line. The above example is just formatted for convenience.

Alternatively you can also use the tc-client to add entries to any implementation of the transformation catalog. The following eg: shows us adding the last entry in the File based transformation catalog.

```
$ tc-client -Dpegasus.catalog.transformation=File \
-Dpegasus.catalog.transformation.file=$HOME/tc -a -r clus1 -l black::analyze:1.0 \
-p gsiftp://clus1.com/opt/nfs/vdt/pegasus/bin/keg  -t STATIC_BINARY -s INTEL32::LINUX \
-e ENV::KEY3="VALUE3"

2007.07.11 16:12:03.712 PDT: [INFO] Added tc entry sucessfully
```

To verify if the entry was correctly added to the transformation catalog you can use the tc-client to query.

```
$ tc-client -Dpegasus.catalog.transformation=File \
-Dpegasus.catalog.transformation.file=$HOME/tc -q -P -l black::analyze:1.0

#RESID     LTX          PFN                 TYPE              SYSINFO

clus1    black::analyze:1.0    gsiftp://clus1.com/opt/nfs/vdt/pegasus/bin/keg
                STATIC_BINARY    INTEL32::LINUX
```

We are now done with setting up all the catalogs.

4. Pegasus Properties

Properties allow you to configure Pegasus in various ways by selection various catalog implementations, enabling and disabling various features or just additionally tuning them. Pegasus utilizes Java properties technology. Pegasus properties can be defined in several ways

1. System Property File - Found in $PEGASUS_HOME/etc/properties. Lowest Priority

   These properties are generally set by the administrator to be shared by all users of the installation. They default location of $PEGASUS_HOME/etc/properties can be overridden by using the property pegasus.system.properties=<path to a file> on the command line or in the user properties file.

2. User Property File - Found in the $HOME/.pegasusrc. Higher Priority

   Thse properties are generally set by a user to augment existing properties from the system property file or override a few setting. The default location of $HOME/.pegasusrc can be overridden by using the property pegasus.user.properties on the command line or in the system property file.

3. Command Line Properties - Provided on the command line of any Pegasus executable. Highest Priority

   The command line properties are specified on the command line of any Pegasus clients by using -D<property>=<value>. These properties need to be defined before any arguments to the clients.

   Note: There is no space between -D and the property name

There is a priority to the order of reading and evaluating properties from these sources. The properties in the system file have the lowest priority, user properties override system properties and command line properties have the highest priority. Note that the values rely on proper capitalization, unless explicitly noted otherwise. Some properties if not specified may have default values. You can use variable substitution to use the value of one property in a second property. For instance, ${pegasus.home} means that the value depends on the value of the pegasus.home property plus any noted additions. You can use this notation to refer to other properties, though the extent of the substitutions are limited. Usually, you want to refer to a set of the standard system properties. Nesting is not allowed. Substitutions will only be done once.

If you are in doubt which properties are actually visible, a sample application called show-properties dumps all properties after reading and prioritizing them.

For running the above example we shall create a user properties file in $HOME/.pegasusrc and specify properties for Pegasus to find various catalogs

```
$ vim $HOME/.pegasusrc


pegasus.catalog.replica=SimpleFile
pegasus.catalog.replica.file=$HOME/rc
pegasus.catalog.site=XML
pegasus.catalog.site.file=$HOME/sites.xml
pegasus.catalog.transformation=File
pegasus.catalog.transformation.file=$HOME/tc
pegasus.catalog.transformation.mapper=Staged
pegasus.dir.storage=pegasusstorage
pegasus.dir.exec=pegasusexec
pegasus.exitcode.scope=all
```

Substitute all $HOME and $USER variables above with actual values for your site.

For a complete list of Pegasus properties refer to the properties.pdf in $PEGASUS_HOME/doc or on the Pegasus webiste at http://pegasus.isi.edu/doc.php

5. Running Pegasus

Pegasus plans an abstract workflow into a concrete/executable workflow by querying various catalogs and performing several refinement steps. We have already setup all the catalogs required by Pegasus in the previous sections and also configured properties that will affect how Pegasus plans the workflow.

Pegasus planner is invoked on the command line by running the pegasus-plan command. The client takes several parameters including one or more sites where to compute the workflow, an optional single site if the output data needs to be staged and stored and the input DAX file.

5.1. Running Locally

In this section we will be planning the blackdiamond workflow to run locally on the submit machine and store the data on the submit machine also. Invoke the pegasus-plan command with the --sites option set to local and also the --output option set to local. The -D dags option specifies to Pegasus to create the Dag and submit files in a directory called dags inside the current directory. Lastly specify the blackdiamond.dax file as the input DAX to the client.

```
$ pegasus-plan --sites local --output local -D dags \
--dax $PEGASUS_HOME/examples/blackdiamond.dax

2007.07.11 17:55:28.118 PDT: [INFO] Parsing the DAX
2007.07.11 17:55:28.668 PDT: [INFO] Parsing the DAX (completed)
2007.07.11 17:55:28.725 PDT: [INFO] Parsing the site catalog
2007.07.11 17:55:28.926 PDT: [INFO] Parsing the site catalog (completed)
2007.07.11 17:55:28.985 PDT: [INFO] Doing site selection
2007.07.11 17:55:29.025 PDT: [INFO] Doing site selection (completed)
2007.07.11 17:55:29.025 PDT: [INFO] Grafting transfer nodes in the workflow
2007.07.11 17:55:29.137 PDT: [INFO] Grafting transfer nodes in the workflow (completed)
2007.07.11 17:55:29.142 PDT: [INFO] Grafting the remote workdirectory creation jobs
                                    in the workflow
2007.07.11 17:55:29.159 PDT: [INFO] Grafting the remote workdirectory creation jobs
                                    in the workflow (completed)
2007.07.11 17:55:29.159 PDT: [INFO] Generating the cleanup workflow
2007.07.11 17:55:29.164 PDT: [INFO] Generating the cleanup workflow (completed)
2007.07.11 17:55:29.164 PDT: [INFO] Adding cleanup jobs in the workflow
```

```
2007.07.11 17:55:29.183 PDT: [INFO] Parsing the site catalog
2007.07.11 17:55:29.223 PDT: [INFO] Parsing the site catalog (completed)
2007.07.11 17:55:29.248 PDT: [INFO] Adding cleanup jobs in the workflow (completed)
2007.07.11 17:55:29.280 PDT: [INFO] Generating codes for the concrete workflow
2007.07.11 17:55:29.476 PDT: [INFO] Generating codes for the concrete workflow(completed)
2007.07.11 17:55:29.477 PDT: [INFO] Generating code for the cleanup workflow
2007.07.11 17:55:29.515 PDT: [INFO] Generating code for the cleanup workflow (completed)
2007.07.11 17:55:29.520 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/asd2/gmehta/PEGASUS\
/dags/gmehta/pegasus/black-diamond/run0001/pegasus.61698.properties \
--nodatabase /nfs/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0001


2007.07.11 17:55:29.521 PDT: [INFO] Time taken to execute is 1.71 seconds
```

Looks like the planning worked. pegasus-plan tells you what command to use next to submit your workflow. Just copy the pegasus-run line from your output and run the command. pegasus-plan submits your planned workflow to Condor-G for execution either locally or on the grid.

```
$ pegasus-run -Dpegasus.user.properties=/nfs/asd2/gmehta/PEGASUS/dags\
/gmehta/pegasus/black-diamond/run0001/pegasus.61698.properties \
--nodatabase /nfs/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0001

Checking all your submit files for log file names.
This might take a while...
Done.
-----------------------------------------------------------------------
File for submitting this DAG to Condor           : black-diamond-0.dag.condor.sub
Log of DAGMan debugging messages                 : black-diamond-0.dag.dagman.out
Log of Condor library output                     : black-diamond-0.dag.lib.out
Log of Condor library error messages             : black-diamond-0.dag.lib.err
Log of the life of condor_dagman itself          : black-diamond-0.dag.dagman.log

Condor Log file for all jobs of this DAG         : /tmp/black-diamond-061699.log
-no_submit given, not submitting DAG to Condor.  You can do this with:
"condor_submit black-diamond-0.dag.condor.sub"
-----------------------------------------------------------------------
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 22402.

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/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0001

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

pegasus-status -w black-diamond-0 -t 20070711T175528-0700
or
pegasus-status /nfs/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0001

*** To remove your workflow run ***

pegasus-remove /nfs/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0001
```

The workflow was successfully submitted for execution to the local submit host. The planned workflow Dag and submit files reside in the directory /nfs/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0001 as mentioned by the output of pegasus-run. Pegasus run also prints out a few monitoring commands that we will use in Section 6.

If you have a grid site configured the proceed to Section 5.2 else proceed to Section 6.

5.2. Running on the Grid

While the local workflow is executing let us submit another workflow to run on the Grid. We have a grid site clus1 set up for use. Replace the --sites local option in the previous run with --sites clus1. We will also need to add a a --force flag because we already ran the same workflow earlier locally and some out of the output products of the workflow may be registered in the Replica Catalog. Pegasus does workflow reduction based on existence of data products in the Replica Catalog. The --force option is a way to tell Pegasus to ignore existing data products and compute the entire workflow again.

```
$ pegasus-plan --sites clus1 --output local -D dags \
--dax $PEGASUS_HOME/examples/blackdiamond.dax --force
2007.07.11 18:12:14.541 PDT: [INFO] Parsing the DAX
2007.07.11 18:12:15.287 PDT: [INFO] Parsing the DAX (completed)

...
...

2007.07.11 18:12:15.847 PDT: [INFO] Generating codes for the concrete workflow
2007.07.11 18:12:16.008 PDT: [INFO] Generating codes for the concrete workflow(completed)
2007.07.11 18:12:16.008 PDT: [INFO] Generating code for the cleanup workflow
2007.07.11 18:12:16.049 PDT: [INFO] Generating code for the cleanup workflow (completed)
2007.07.11 18:12:16.054 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/asd2/gmehta/PEGASUS/dags\
/gmehta/pegasus/black-diamond/run0002/pegasus.7398.properties \
--nodatabase /nfs/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0002


2007.07.11 18:12:16.055 PDT: [INFO] Time taken to execute is 1.918 seconds
```

Submit the planned workflow also to Condor-G using the pegasus-run command printed by the previous step.

```
$ pegasus-run -Dpegasus.user.properties=/nfs/asd2/gmehta/PEGASUS/dags\
/gmehta/pegasus/black-diamond/run0002/pegasus.7398.properties \
--nodatabase /nfs/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0002

...
...


cd /nfs/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0002

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

pegasus-status -w black-diamond-0 -t 20070711T181215-0700
or
pegasus-status /nfs/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0002

*** To remove your workflow run ***

pegasus-remove -d 22417.0
or
pegasus-remove /nfs/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0002
```

6. Monitoring and Debugging

The output in the previous section showed that the workflow was submitted for execution. The output also gave the relevant commands to use to monitor the progress of the workflow or to terminate it. Lets go to the directory where

Pegasus generated the Condor Dag and submit files. Use the directory name from the output of your pegasus-run command, either the local run or the grid run.

```
$ cd /nfs/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0002
```

If you do a listing of the files in the directory you will see several .sub files where are the job submit files. The output of each job is written in the .out.* file and the error goes in the .err file.

To monitor the execution of the workflow lets run the pegasus-status command as suggested by the output of the pegasus-run command above.

```
$ pegasus-status /nfs/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0002


-- Submitter: smarty.isi.edu : <128.9.72.26:53194> : smarty.isi.edu
 ID      OWNER/NODENAME   SUBMITTED     RUN_TIME ST PRI SIZE CMD
22417.0   gmehta          7/11 18:13   0+00:03:58 R  0   9.8  condor_dagman -f -
22423.0    |-rc_tx_analy  7/11 18:16   0+00:00:54 R  2   0.0  kickstart -n pegas
22424.0    |-rc_tx_findr  7/11 18:16   0+00:00:00 I  2   0.0  kickstart -n pegas
22425.0    |-rc_tx_prepr  7/11 18:16   0+00:00:00 I  2   0.0  kickstart -n pegas
```

The above output shows that several 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 CondorDAGMAN 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.

```
$ less jobstate.log

1184202818 INTERNAL *** DAGMAN_STARTED ***
1184202831 black-diamond_0_viz_cdir SUBMIT 22418.0 clus1 -
1184202846 black-diamond_0_viz_cdir EXECUTE 22418.0 clus1 -
1184202846 black-diamond_0_viz_cdir GLOBUS_SUBMIT 22418.0 clus1 -
1184202846 black-diamond_0_viz_cdir GRID_SUBMIT 22418.0 clus1 -
1184202977 black-diamond_0_viz_cdir JOB_TERMINATED 22418.0 clus1 -
1184202977 black-diamond_0_viz_cdir POST_SCRIPT_STARTED - clus1 -
1184202982 black-diamond_0_viz_cdir POST_SCRIPT_TERMINATED 22418.0 clus1 -
1184202982 black-diamond_0_viz_cdir POST_SCRIPT_SUCCESS - clus1 -

...
...

1184205172 new_rc_register_analyze_ID000004 POST_SCRIPT_SUCCESS - local -
1184205302 cln_analyze_ID000004 JOB_TERMINATED 22436.0 clus1 -
1184205302 cln_analyze_ID000004 POST_SCRIPT_STARTED - clus1 -
1184205307 cln_analyze_ID000004 POST_SCRIPT_TERMINATED 22436.0 clus1 -
1184205307 cln_analyze_ID000004 POST_SCRIPT_SUCCESS - clus1 -
1184205307 INTERNAL *** DAGMAN_FINISHED ***
1184205311 INTERNAL *** TAILSTATD_FINISHED 0 **
```

The above shows the create dir job being submitted and then executed on the grid. In addition it lists that job is being run on the grid site clus1 The various states of the job while it goes through submission to execution to postprocessing are in UPPERCASE.

At the bottom of the output we see that DAGMAN and TAILSTATD have FINISHED and with and exit code of zero "0" which signifies that the workflow ran successfully. If there were any errors then the TAILSTATD would exit with a non zero exitcode and the failed jobs would have a job state of FAILURE next to it.

If your workflow fails then you can look at the job name (second column in the output) which has failed and check the contents of the kickstart record output stored in the jobname.out.NNN file where NNN can be 000 to 999 or the jobname.err file

One can also monitor the status of the running workflow by looking at the output of the Condor DAGMan output file.

```
$ tail -f black-diamond-0.dag.dagman.out

7/11 18:52:27 Node new_rc_tx_analyze_ID000004_0 job proc (22435.0) completed suc
cessfully.
7/11 18:52:27 Node new_rc_tx_analyze_ID000004_0 job completed
7/11 18:52:27 Running POST script of Node new_rc_tx_analyze_ID000004_0...
7/11 18:52:27 Number of idle job procs: 0
7/11 18:52:27 Of 17 nodes total:
7/11 18:52:27  Done     Pre   Queued    Post   Ready   Un-Ready   Failed
7/11 18:52:27   ===     ===      ===     ===     ===        ===      ===
7/11 18:52:27    14       0        0       1       0          2        0
```

You will see lines like the one above scrolling by giving you statistics of home many jobs have been finished, failed, running or queued. It will also tell if you the workflow has finished if the DAGMan finishes and the name of any jobs that failed.

If you want to abort your workflow for any reason you can use the pegasus-remove command listed in the output of pegasus-run invocation or by specifiying the Dag directory for the workflow you want to terminate.

```
$ pegasus-remove /nfs/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0001
```

Pegasus will remove the DAGMan and all the jobs related to the DAGMan from the condor queue. A rescue DAG will be generated in case you want to resubmit the same workflow and continue execution from where it last stopped. A rescue DAG only skips jobs that have completely finished. It does not continue a partially running job unless the executable supports checkpointing.

To resubmit an aborted or failed workflow with the same submit files and rescue Dag just rerun the pegasus-run command

```
$ pegasus-run -Dpegasus.user.properties=/nfs/asd2/gmehta/PEGASUS/dags\
/gmehta/pegasus/black-diamond/run0001/pegasus.61698.properties \
--nodatabase /nfs/asd2/gmehta/PEGASUS/dags/gmehta/pegasus/black-diamond/run0001
```