Pegasus WMS Tutorial

Karan Vahi, Ewa Deelman
( vahi,deelman@isi.edu )

\(^1\)Center for Grid Technologies, USC Information Sciences Institute
Outline of Tutorial

- Pegasus-WMS and Composition of a Simple Workflow In terms of DAX.
- Pegasus Internals
- Mapping and Executing Simple Workflow Locally
- Mapping and Executing Simple Workflow On the Grid
- Optimization techniques for mapping and executing Large Scale workflows

Exercise Notes and Slides online at
http://pegasus.isi.edu/tutorial/tg08/index.php
Cyberinfrastructure: Local machine, cluster, Condor pool, Grid

Workflow Generation Utilities

Abstract Workflow Description (devoid of resource bindings, Portable across resources)

Pegasus WMS

Results Delivered To user-specified location

Provenance and Performance Recorded

Tasks

Monitoring information

Results Delivered

Cyberinfrastructure:

Abstract Workflow Description (devoid of resource bindings, Portable across resources)

Pegasus WMS

Results Delivered To user-specified location

Provenance and Performance Recorded

Tasks

Monitoring information

Cyberinfrastructure: Local machine, cluster, Condor pool, Grid
Pegasus-Workflow Management System
a layered approach

A reliable, scalable workflow management system that an application or workflow composition service can depend on to get the job done

<table>
<thead>
<tr>
<th>Workflow Mapping System</th>
<th>A decision system that develops strategies for reliable and efficient execution in a variety of environments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow Execution System</td>
<td>Reliable and scalable execution of dependent tasks</td>
</tr>
<tr>
<td>Task Execution System</td>
<td>Reliable, scalable execution of independent tasks (locally, across the network), priorities, scheduling</td>
</tr>
</tbody>
</table>

Cyberinfrastructure: Local machine, cluster, Condor pool, Grid
Pegasus Workflow Management System

Abstract Workflow

- A reliable, scalable workflow management system that an application or workflow composition service can depend on to get the job done

- A decision system that develops strategies for reliable and efficient execution in a variety of environments

- Reliable and scalable execution of dependent tasks

- Reliable, scalable execution of independent tasks (locally, across the network), priorities, scheduling

Cyberinfrastructure: Local machine, cluster, Condor pool, OSG, TeraGrid
Pegasus workflow

- DAX
  - What it describes
  - How to read a DAX
  - How to generate a DAX
    - Describe the various methods
      - Direct XML
      - Wings
      - DAX API
      - Behind portals
    - Migrating from a DAG to DAX
Abstract Workflow (DAX)

Exercise: 2.1

- Pegasus workflow description—DAX
  - workflow “high-level language”
  - devoid of resource descriptions
  - devoid of data locations
  - refers to codes as logical transformations
  - refers to data as logical files

- Exercise:
  - Use CreateDAX.java to generate a diamond dax
Understanding DAX (1)

<!-- part 1: list of all files used (may be empty) -->
<filename file="f.input" link="input"/>
<filename file="f.intermediate" link="input"/>
<filename file="f.output" link="output"/>
<filename file="keg" link="input">

<!-- part 2: definition of all jobs (at least one) -->
<job id="ID000001" namespace="pegasus" name="preprocess" version="1.0" >
  <argument>-a top -T 6  -i <filename file="f.input"/>  -o <filename file="f.intermediate"/>
</argument>
<uses file="f.input" link="input" register="false" transfer="true"/>
<uses file="f.intermediate" link="output" register="false" transfer="false">
  <!-- specify any extra executables the job needs . Optional  -->
  <uses file="keg" link="input" register="false" transfer="true" type="executable">
</uses>
</job>

<job id="ID000002" namespace="pegasus" name="analyze" version="1.0" >
  <argument>-a top -T 6  -i <filename file="f.intermediate"/>  -o <filename file="f.output"/>
</argument>
<uses file="f.intermediate" link="input" register="false" transfer="true"/>
<uses file="f.output" link="output" register="true" transfer="true"/>
</job>

<!-- part 3: list of control-flow dependencies (empty for single jobs) -->
<child ref="ID0000002">
  <parent ref="ID0000001"/>
</child>
(excerpted for display)
Comparison of abstract and executable workflows

<table>
<thead>
<tr>
<th>Abstract Workflow</th>
<th>Executable Workflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Describes your workflow at a logical level</td>
<td>Describes your workflow in terms of physical files and paths</td>
</tr>
<tr>
<td>Site Independent</td>
<td>Site Specific</td>
</tr>
<tr>
<td>Captures just the computation that the user (you) want to do</td>
<td>Has additional jobs for data movement etc.</td>
</tr>
</tbody>
</table>
Outline of Tutorial

- Pegasus-WMS and Composition of a Simple Workflow In terms of DAX
- Pegasus Internals
- Mapping and Executing Simple Workflow Locally
- Mapping and Executing Simple Workflow On the Grid
- Optimization techniques for mapping and executing Large Scale workflows

Exercise Notes and Slides online at
http://pegasus.isi.edu/tutorial/tg08/index.php
Pegasus WMS restructures and optimizes the workflow, provides reliability.
Discovery

- **Data**
  - Where do the input datasets reside?

- **Executables**
  - Where are the executables installed?
  - Do binaries exist somewhere that can be staged to remote grid sites?

- **Site Layout**
  - What does a grid site look like?
Replica Catalog Overview—finding data

- Replica Catalog stores mappings between logical files and their target locations
- Used to
  - discover input files for the workflow
  - track data products created
  - data reuse
- Data is replicated for scalability, reliability and availability
Replica Catalog

- Pegasus interfaces with a variety of replica catalogs
  - File based Replica Catalog
    - useful for small datasets (like this tutorial)
    - cannot be shared across users
  - Database based Replica Catalog
    - useful for medium sized datasets
    - can be used across users
  - Globus Replica Location Service
    - useful for large scale data sets across multiple users
    - LIGO’s LDR deployment
The rc-client is a command line tool to interact with Replica Catalog

- One client talks to all types of Replica Catalog

Practical exercise (Refer Exercise 2.2):

- Use the rc-client to
  - Populate the Replica Catalog
    - Single insert of an entry
    - Bulk inserts
  - Query the Replica Catalog
  - Remove entries (Offline exercise)
Site Catalog—finding resources

- Contains information about various sites on which workflows may execute
- For each site following information is stored:
  - Installed job-managers for different types of schedulers
  - Installed GridFTP servers
  - Local Replica Catalogs where data residing in that site has to be catalogued
  - Site Wide Profiles like environment variables
  - Work and storage directories
Site Catalog Exercise (Ex 2.3 10 minutes)

- Two clients for generating a site catalog
  - pegasus-get-sites
    - Allows you to generate a site catalog
      - For OSG grid sites by querying VORS
      - For ISI skynet, Teragrid, UC SofaGrid by querying a SQLLite2 database
  - sc-client
    - Allows you to generate a site catalog
      - By specifying information about a site in a textual format in a file
      - One file per site
<site handle="isi_skynet" sysinfo="INTEL32::LINUX" gridlaunch="/nfs/software/vds/vds/bin/kickstart">

  <profile namespace="env" key="PEGASUS_HOME">/nfs/software/pegasus</profile>
  <lrc url="rlsn:// smarty.isi.edu" />
  <gridftp url="gsiftp://skynet-data.isi.edu" storage="/nfs/storage01" major="2" minor="4" patch="3" />
  <gridftp url="gsiftp://skynet-2.isi.edu" storage="/nfs/storage01" major="2" minor="4" patch="3" />
  <jobmanager universe="vanilla" url="skynet-login.isi.edu/jobmanager-pbs" major="2" minor="4" patch="3" total-nodes="93" />
  <jobmanager universe="transfer" url="skynet-login.isi.edu/jobmanager-fork" major="2" minor="4" patch="3" total-nodes="93" />
  <workdirectory>/nfs/scratch01</workdirectory>

</site>
Transformation Catalog ---- finding codes

- Transformation Catalog maps logical transformations to their physical locations

- Used to
  - Discover application codes installed on the grid sites
  - Discover statically compiled codes, that can be deployed at grid sites on demand
Transformation Catalog Overview

- For each transformation following are stored
  - Logical name of the transformation
  - Type of transformation (INSTALLED or STATIC_BINARY)
  - Architecture, OS, Glibc version
  - The resource on which the transformation is available
  - The URL for the physical transformation
  - Profiles that associate runtime parameters like environment variables, scheduler related information
Transformation Catalog Exercise 2.3

- tc-client is a command line client that is primarily used to configure the database TC
- Works even for file based transformation catalog
- Practical exercise (Refer Exercise 2.3):
  - tc-client
    - Insert an entry
    - Query for a single entry
    - Query for all the entries
Component Configuration

- Component Configuration using Properties File
- Most of the configuration of Pegasus is done by properties

- Properties can be specified
  - On the command line
  - In $HOME/.pegasusrc file
  - In $PEGASUS_HOME/etc/properties

- All properties are described in $PEGASUS_HOME/doc/properties.pdf

- For the tutorial the properties are configured in the $HOME/tutorial/config/properties file
DAGMan Configuration

- Condor configuration files
- Environment variables
  (_CONDOR_<macroname>)
- DAGMan configuration file (6.9.2+)
- Condor_submit_dag command line
Outline of Tutorial

- Pegasus-WMS and Composition of a Simple Workflow In terms of DAX.
- Pegasus Internals
- Mapping and Executing Simple Workflow Locally
- Mapping and Executing Simple Workflow On the Grid
- Optimization techniques for mapping and executing Large Scale workflows

Exercise Notes and Slides online at
http://pegasus.isi.edu/tutorial/tg08/index.php
Map and Execute Workflow Locally

- Take a 4 node diamond abstract workflow (DAX) and map it to an executable workflow that runs locally
Basic Workflow Mapping

- Select where to run the computations
  - Change task nodes into nodes with executable descriptions
    - Execution location
    - Environment variables initializes
    - Appropriate command-line parameters set

- Select which data to access
  - Add stage-in nodes to move data to computations
  - Add stage-out nodes to transfer data out of remote sites to storage
  - Add data transfer nodes between computation nodes that execute on different resources
Basic Workflow Mapping

- Add nodes that register the newly-created data products
- Add nodes to create an execution directory on a remote site
- Write out the workflow in a form understandable by a workflow engine
  - Include provenance capture steps
Pegasus Workflow Mapping

Original workflow: 15 compute nodes devoid of resource assignment

Resulting workflow mapped onto 3 Grid sites:

- 11 compute nodes (4 reduced based on available intermediate data)
- 13 data stage-in nodes
- 8 inter-site data transfers
- 14 data stage-out nodes to long-term storage
- 14 data registration nodes (data cataloging)
Exercise: 2.4

- Plan using Pegasus and submit the workflow to Condor DAGMan/CondorG for local job submissions
  
  ```
  $ pegasus-plan -Dpegasus.user.properties=<properties file> 
  --dax <dax file> --dir <dags directory> -s local -o local 
  --nocleanup 
  
  $ pegasus-run --Dpegasus.user.properties=<properties file> 
  --nodatabase <dag directory>
  ```
Run (pegasus-run)
Exercise: 2.4 (cont.)

- Submits the workflow to Condor DAGMAN/CondorG for local job submissions
Exercise: 2.5 - (Monitor)

Pegasus-status

- A perl wrapper around condor_q
- Allows you to see only the jobs of a particular workflow
- Also can see what different type of jobs that are executing
- Pegasus-status <dag directory>
- Pegasus-status –w <workflow> -t <time>
Condor_submit_dag

- Creates a Condor submit file for DAGMan
- Also submits it (unless –no_submit option is given)
- -f option forces overwriting of existing files
Condor Monitoring

- Monitoring your DAG
  - `Condor_q [–dag] [name]`
  - Dagman.out file

```bash
% condor_q –dag train15

-- Submitter: train15@isi.edu : <128.9.72.178:43684> : viz-login.isi.edu
ID      OWNER/NODENAME   SUBMITTED     RUN_TIME ST PRI SIZE CMD
1860.0   train15         5/31 10:59   0+00:00:26 R  0   9.8  condor_dagman -f -
1861.0    |-Setup        5/31 10:59   0+00:00:12 R  0   9.8  nodejob Setup node

2 jobs; 0 idle, 2 running, 0 held
```

`pegasus-status calls internally to condor_q with –dag option`
Exercise 2.6 - A simple DAG
DAG file

- Defines the DAG shown previously
- Node names are case-sensitive
- Keywords are not case-sensitive

JOB generate_ID000001 generate_ID000001.sub
JOB findrange_ID000002 findrange_ID000002.sub
JOB findrange_ID000003 findrange_ID000003.sub
JOB analyze_ID000004 analyze_ID000004.sub
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 /bin/exitpost
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
DAG node

- Treated as a unit
- Job or POST script determines node success or failure
PRE/POST in DAGMan scripts

- SCRIPT PRE|POST node script [arguments]
- All scripts run on submit machine
- If PRE script fails, node fails w/o running job or POST script (for now…)
- If job fails, POST script is run
- If POST script fails, node fails
- Special macros:
  - $JOB
  - $RETURN (POST only)

In PegasusWMS the kickstart xml output is parsed by invoking a postscript. The postscript parses and determines the exit code with which the job failed.
Exercise 2.7 - pegasus-remove

- Remove your workflow and associated jobs
- In future, would cleanup the remote directories that are created during workflow execution.
- Pegasus-remove <dag directory>
Outline of Tutorial

- Pegasus-WMS and Composition of a Simple Workflow In terms of DAX.
- Pegasus Internals
- Mapping and Executing Simple Workflow Locally
- Mapping and Executing Simple Workflow On the Grid
- Optimization techniques for mapping and executing Large Scale workflows

Exercise Notes and Slides online at
http://pegasus.isi.edu/tutorial/tg08/index.php
Execution of jobs on storage systems shared between the worker nodes and headnode / storage element.

1) Creates a unique work directory (RDy) per workflow on the Shared Storage System of Site Y

2) Stages in input data for job from remote storage system at Site Y to the work directory (RDy)

3) Job is executed on one of the worker nodes in the directory (RDy) on the Shared Storage System

4) Stages out Data from work directory (RDy) on the shared storage system to storage system on Grid Site Z

5) Removes the directory (RDy) on the shared storage system of Site Y

Legend
- Visibility / Accessibility
- Compute Job
- Stage-in Job
- Stage-Out Job
- Cleanup Job
- Make Dir Job
Map and Execute Montage Workflow on Grid

- Take a montage abstract workflow (DAX) and map it to an executable workflow that runs on the Grid
- The available sites are tg_ncsa, tg_sdsc, viz
- You can either use a single site or a combination of these by specifying comma separated sites on the command line
Exercise: 2.8

- Plan using Pegasus and submit the workflow to Condor DAGMan/CondorG for remote job submissions
- Pegasus-run starts the monitoring daemon (tailstatd) in the directory containing the condor submit files
- Tailstatd parses the condor output and updates the status of the workflow to a database
- Tailstatd updates job status to a text file jobstate.log in the directory containing the condor submit files
Exercise: 2.8 - Debugging

- The status of the workflow can be determined by
  - Looking at the jobstate.log
  - Or looking at the dagman out file (with suffix .dag.dagman.out)

- All jobs in Pegasus are launched by a wrapper executable kickstart. Kickstart generates provenance information including the exit code, and part of the remote application’s stdout.

- In case of job failure look at kickstart output of the failed job.
Outline of Tutorial

- Pegasus-WMS and Composition of a Simple Workflow In terms of DAX.
- Pegasus Internals
- Mapping and Executing Simple Workflow Locally
- Mapping and Executing Simple Workflow On the Grid
- Optimization techniques for mapping and executing Large Scale workflows

Exercise Notes and Slides online at
http://pegasus.isi.edu/tutorial/tg08/index.php
Workflow Restructuring to improve Application Performance

- Cluster small running jobs together to achieve better performance

Why?

- Each job has scheduling overhead
- Need to make this overhead worthwhile
- Ideally users should run a job on the grid that takes at least 10 minutes to execute
Job clustering

Level-based clustering

Vertical clustering

Arbitrary clustering

Useful for small granularity jobs

Ewa Deelman, deelman@isi.edu

www.isi.edu/~deelman

pegasus.isi.edu
Exercise 3.1
Optional clustering exercise

- To trigger specify --cluster horizontal option to pegasus-plan

- The granularity of clustering configured via pegasus profile key bundle
  - Can be specified with a transformation in the transformation catalog, or with sites in the site catalog
  - Pegasus profile *bundle* specified in the site catalog
  - Bundle means how many clustered jobs for that transformation you need on a particular site
How to: Files need to be cataloged in replica catalog at runtime. The registration flags for these files need to be set in the DAX.
**File cleanup**

- Problem: Running out of space on shared scratch
  - In OSG scratch space is limited to 30Gb for all users
- Why does it occur
  - Workflows bring in huge amounts of data
  - Data is generated during workflow execution
  - Users don’t worry about cleaning up after they are done
- Solution
  - Do cleanup after workflows finish
    - Does not work as the scratch may get filled much before during execution
  - Interleave cleanup automatically during workflow execution.
    - Requires an analysis of the workflow to determine, when a file is no longer required

*How to:* remove the `--nocleanup` option to the `pegasus-plan` invocation in exercises 2.5 and 2.8.
Montage 1 degree workflow run with cleanup on OSG-PSU
Running using different styles

- Need to specify pegasus namespace profile keys with the sites in the site catalog

- Submitting directly to condor pool
  - The submit host is a part of a local condor pool
  - Bypasses CondorG submissions avoiding Condor/GRAM delays

- Using Condor GlideIn
  - User glides in nodes from a remote grid site to his local pool
  - Condor is deployed dynamically on glided in nodes for e.g. you glide in nodes from the teragrid site running PBS
  - Only have to wait in the remote queue once when gliding in nodes
Transfer of Executables

- Allows the user to dynamically deploy scientific code on remote sites
- Makes for easier debugging of scientific code
- The executables are transferred as part of the workflow
- Currently, only statically compiled executables can be transferred
- Also we transfer any dependant executables that maybe required. In your workflow, the mDiffFit job is dependant on mDiff and mFitplane executables
Staging of executable exercise

- All the workflows that you ran had staging of executables

- In your transformation catalog, the entries were marked as STATIC_BINARY on site “local”

- Selection of what executable to transfer
  - pegasus.transformation.mapper property
  - pegasus.transformation.selector property
Nested DAGs
Managing execution environment changes through partitioning
**Resulting Meta-Workflow/Nested DAG**

- **Pegasus(X):** Pegasus generates the concrete workflow and the submit files for Partition X -- Su(X)

- **DAGMan(Su(X)):** DAGMan executes the concrete workflow for X

1. **Pegasus(A) = Su(A)**
   - **DAGMan(Su(A))**
   - Retry X times

2. **Pegasus(B) = Su(B)**
   - **DAGMan(Su(B))**
   - Retry X times

3. **Pegasus(C) = Su(C)**
   - **DAGMan(Su(C))**
   - Retry X times
Workflow-level checkpointing

Pegasus(A) = Su(A)

DAGMan(Su(A))

Retry Y times

Original abstract workflow partition

Pegasus mapping , f2 and f3 were found in a replica catalog

Workflow submitted to DAGMan

Pegasus is called again with original partition

New mapping, here assuming R1 was picked again
Exercise 3.3 – Nested DAG and Deferred Planning

- Partition the workflow using partitiondax
  - `partitiondax -Dpegasus.user.properties=./config/properties --dax dax/montage.dax --dir ./pdags/ --type horizontal`

- Submit the outer level workflow by submitting the pdax file created. Use the --pdax option
  - `pegasus-plan -Dpegasus.user.properties=`\`pwd`/config/properties --pdax `\`pwd`/pdags/montage.pdax --dir `\`pwd`/dags -s tg_ncsa -o local --nocleanup --force`
VARS (per-node variables)

- VARS JobName
  macroname="string" [macroname="string"... ]
- Macroname can only contain alphanumeric characters and underscore
- Value can’t contain single quotes; double quotes must be escaped
- Macronames cannot begin with “queue”
- Macronames are not case-sensitive
Exercise 3.4 – VARS and CONFIG
Exercise 3.4, continued

```bash
% cat dagman/vars/vars.dag
# DAG to illustrate VARS and CONFIG.

CONFIG vars.config

JOB Setup setup.submit
SCRIPT PRE Setup setup.pre

JOB Proc1 pijob.submit
VARS Proc1 ARGS = "-sleep 60 -trials 10000 -seed 1234567"
PARENT Setup CHILD Proc1

JOB Proc2 pijob.submit
VARS Proc2 ARGS = "-sleep 80 -trials 20000 -seed 7654321"
PARENT Setup CHILD Proc2

JOB Proc3 pijob.submit
PARENT Setup CHILD Proc3
VARS Proc3 ARGS = "YOUR ARGS HERE"
[.....]
```
Exercise 3.4, continued

% cat dagman/vars/vars.config
# DAGMan configuration file for vars.dag.

DAGMAN_MAX_JOBS_SUBMITTED = 3
DAGMAN_STARTUP_CYCLE_DETECT = true
DAGMAN_SUBMIT_DEPTH_FIRST = true

% cat dagman/vars/pijob.submit
# Simple Condor submit file.

Executable = ../pi/pi
 Universe = scheduler
 #Error = pi.err.$(cluster)
 Output = output/pi.out.$(cluster)
 Getenv = true
 Log = pi.log

Arguments = $(ARGS)
 Notification = never
 Queue
Exercise 3.5- Running your Jobs on non shared filesystem

Set the property pegasus.execute.*.filesystem.local true
Transfer Throttling

- Large Sized Workflows result in large number of transfer jobs being executed at once. Results in:
  - Grid FTP server overload (connection refused errors etc)
  - May result in a high load on the head node if transfers are not configured for being executed as third party transfers

- Need to throttle transfers:
  - Set pegasus.transfer.refiner property
  - Allows you to create chained transfer jobs or bundles of transfer jobs
  - Looks in your site catalog for pegasus profile "bundle.stagein"
Throttling in DAGMan

- Maxjobs (limits jobs in queue/running)
- Maxidle (limits idle jobs)
- Maxpre (limits PRE scripts)
- Maxpost (limits POST scripts)
- All limits are *per DAGMan*, not global for the pool

The above parameters can be configured in Pegasus Properties
Throttling by category

- CATEGORY JobName CategoryName
- MAXJOBS CategoryName MaxJobsValue
- Applies the maxjobs setting to only jobs assigned to the given category
- Global throttles still apply
- Useful with different types of jobs that cause different loads
Pegasus throttling properties

- Specifying for the whole workflow
  - pegasus.dagman.maxidle
  - pegasus.dagman.maxjobs
  - pegasus.dagman.maxpre
  - pegasus.dagman.maxpost

- Specifying per category
  - pegasus.dagman.[category-name].maxjobs
Node categories

Setup

Big job

Small job

Cleanup

Big job

Small job

Big job

Small job
Node retries

- RETRY JobName NumberOfRetries [UNLESS-EXIT value]
- Node is retried as a whole

Diagram:
- Node
  - PRE
  - Job
  - POST
- Success
- One node failure – retry
- Unless-exit value – node fails
Node Categories and Retries

% more montage.dag
#
# DAG to illustrate node categories/category throttles.

MAXJOBS projection 2

CATEGORY mProjectPP_ID000002 projection
JOB mProjectPP_ID000002 mProjectPP_ID000002.sub
SCRIPT POST mProjectPP_ID000002 /nfs/software/pegasus/
    default/bin/exitpost ......
RETRY mProjectPP_ID000002 2
...
...
Rescue DAG

- Generated when a node fails or DAGMan is condor_rm’ed
- Saves state of DAG
- Run the rescue DAG to restart from where you left off
- DAGMan 7.1.0 has improvements in how rescue DAGs work
Recovery/bootstrap mode

- Most commonly, after condor_hold/condor_release of DAGMan
- Also after DAGMan crash/restart
- Restores DAG state by reading node job logs
Depth-first DAG traversal

- Get results more quickly
- Possibly clean up intermediate files more quickly
- `DAGMAN_SUBMITDEPTH_FIRST=True`
DAG node priorities

- **PRIORITY** JobName PriorityValue
- Determines order of submission of ready nodes
- Does *not* violate/change DAG semantics
- Mostly useful when DAG is throttled
- Higher Numerical value equals higher priority
- Version 6.9.5+

Node priorities can be configured in Pegasus Properties
Pegasus node priority properties

- `pegasus.job.priority=<N>`
- `pegasus.transfer.stagein.priority=N`
- `pegasus.transfer.stageout.priority=N`
- `pegasus.transfer.inter.priority=N`
- `pegasus.transfer.*.priority=N`
- For each job in TC or DAX define profile
  `CONDOR::priority=N`
What does Pegasus do for an application?

- Provides an Teragrid-aware workflow management tool
  - Queries Web MDS to get information about sites in the Teragrid
  - Deploys user executables as part of the workflow.
  - Reduced Storage footprint. Data is also cleaned as the workflow progresses.

- Data Management within the workflow
  - Interfaces with the variety of Replica Catalog’s (including RLS) to discover data
  - Does replica selection to select replicas.
  - Manages data transfer by interfacing to various transfer services like RFT, Stork and clients like globus-url-copy.
  - No need to stage-in data before hand. We do it within the workflow as and when it is required.

- Improves application performance and execution
  - Job clustering
  - Support for condor glidein’s
  - Techniques exist to minimize load on remote Grid resources during large scale execution of workflows.
  - Data Reuse
    - Avoids duplicate computations
    - Can reuse data that has been generated earlier.
Current and Future Research

- Resource selection
- Resource provisioning
- Workflow restructuring
- Adaptive computing
  - Workflow refinement adapts to changing execution environment
- Workflow provenance (including provenance of the mapping process)
- Management and optimization across multiple workflows
- Workflow debugging
- Streaming data workflows
- Automated guidance for workflow restructuring
- Support for long-lived and recurrent workflows
Relevant Links

- **Pegasus**: [pegasus.isi.edu](http://pegasus.isi.edu)
- **DAGMan**: [www.cs.wisc.edu/condor/dagman](http://www.cs.wisc.edu/condor/dagman)


- For more questions: [pegasus@isi.edu](mailto:pegasus@isi.edu), [condor-admin@cs.wisc.edu](mailto:condor-admin@cs.wisc.edu)
Relevant Links

- NSF Workshop on Challenges of Scientific Workflows: [www.isi.edu/nsf-workflows06](http://www.isi.edu/nsf-workflows06), E. Deelman and Y. Gil (chairs)

- Open Science Grid: [www.opensciencegrid.org](http://www.opensciencegrid.org)
- LIGO: [www.ligo.caltech.edu/](http://www.ligo.caltech.edu/)
- SCEC: [www.scec.org](http://www.scec.org)
- Montage: [montage.ipac.caltech.edu/](http://montage.ipac.caltech.edu/)
- Condor: [www.cs.wisc.edu/condor/](http://www.cs.wisc.edu/condor/)
- Globus: [www.globus.org](http://www.globus.org)
- TeraGrid: [www.teragrid.org](http://www.teragrid.org)