Enhancing Scientific Computations with Scientific Workflows

Pegasus Workflow Management System

George Papadimitriou
Rafael Ferreira da Silva
Ewa Deelman
<table>
<thead>
<tr>
<th>Section</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td>Scientific Workflows</td>
</tr>
<tr>
<td></td>
<td>Pegasus Overview</td>
</tr>
<tr>
<td></td>
<td>Successful Stories</td>
</tr>
<tr>
<td>Pegasus Overview</td>
<td>Basic Concepts</td>
</tr>
<tr>
<td></td>
<td>Features</td>
</tr>
<tr>
<td></td>
<td>System Architecture</td>
</tr>
<tr>
<td>Features</td>
<td>Data Staging</td>
</tr>
<tr>
<td></td>
<td>Information Catalogs</td>
</tr>
<tr>
<td></td>
<td>Fault-Tolerance</td>
</tr>
<tr>
<td>Break</td>
<td>10min Break</td>
</tr>
<tr>
<td>Hands On Tutorial</td>
<td></td>
</tr>
<tr>
<td>Introduction</td>
<td>Scientific Workflows</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td></td>
<td>Pegasus Overview</td>
</tr>
<tr>
<td></td>
<td>Successful Stories</td>
</tr>
<tr>
<td>Pegasus Overview</td>
<td>Basic Concepts</td>
</tr>
<tr>
<td></td>
<td>Features</td>
</tr>
<tr>
<td></td>
<td>System Architecture</td>
</tr>
<tr>
<td>Features</td>
<td>Data Staging</td>
</tr>
<tr>
<td></td>
<td>Information Catalogs</td>
</tr>
<tr>
<td></td>
<td>Fault-Tolerance</td>
</tr>
<tr>
<td>Break</td>
<td>10min Break</td>
</tr>
<tr>
<td>Hands On Tutorial</td>
<td></td>
</tr>
</tbody>
</table>
## Compute Pipelines

**Building Blocks**

- **Compute Pipelines**
  - Allows scientists to connect different codes together and execute their analysis
  - Pipelines can be very simple (independent or parallel) jobs or complex represented as DAG’s
  - Helps users to automate scale up

**However, it is still up-to user to figure out**

### Data Management

- How do you ship in the small/large amounts data required by your pipeline and protocols to use?

**How best to leverage different infrastructure setups**

- OSG has no shared filesystem while XSEDE and your local campus cluster has one!

### Debug and Monitor Computations

- Correlate data across lots of log files
- Need to know what host a job ran on and how it was invoked

### Restructure Workflows for Improved Performance

- Short running tasks? Data placement

---

**Pegasus**

http://pegasus.isi.edu
Why Pegasus?

**Automates** complex, multi-stage processing pipelines

Enables parallel, **distributed computations**

Automatically executes data transfers

Reusable, aids **reproducibility**

Records how data was produced (**provenance**)  

Handles **failures** with to provide reliability

Keeps track of data and **files**

NSF funded project since 2001, with close collaboration with HTCondor team
Some of the successful stories...
60,000 compute tasks
Input Data: 5000 files (10GB total)
Output Data: 60,000 files (60GB total)

executed on LIGO Data Grid, Open Science Grid and XSEDE
Advanced LIGO

PyCBC Workflow

One of the main pipelines to measure the statistical significance of data needed for discovery

Contains 100's of thousands of jobs and accesses on order of terabytes of data

Uses data from multiple detectors

For the detection, the pipeline was executed on Syracuse and Albert Einstein Institute Hannover

A single run of the binary black hole + binary neutron star search through the O1 data (about 3 calendar months of data with 50% duty cycle) requires a workflow with 194,364 jobs

Generating the final O1 results with all the review required for the first discovery took about 20 million core hours
Southern California Earthquake Center’s CyberShake

Builders ask seismologists: What will the peak ground motion be at my new building in the next 50 years?

Seismologists answer this question using Probabilistic Seismic Hazard Analysis (PSHA)

CPU jobs (Mesh generation, seismogram synthesis):
1,094,000 node-hours

GPU jobs: 439,000 node-hours
AWP-ODC finite-difference code
5 billion points per volume, 23000 timesteps
200 GPUs for 1 hour

Titan:
421,000 CPU node-hours, 110,000 GPU node-hours

Blue Waters:
673,000 CPU node-hours, 329,000 GPU node-hours

286 sites, 4 models
each workflow has 420,000 tasks
Impact on DOE Science

Enabled cutting-edge domain science (e.g., drug delivery) through collaboration with scientists at the DoE Spallation Neutron Source (SNS) facility

A Pegasus workflow was developed that confirmed that nanodiamonds can enhance the dynamics of tRNA

It compared SNS neutron scattering data with MD simulations by calculating the epsilon that best matches experimental data

Ran on a Cray XE6 at NERSC using 400,000 CPU hours, and generated 3TB of data.

Water is seen as small red and white molecules on large nanodiamond spheres. The colored tRNA can be seen on the nanodiamond surface.

(Image Credit: Michael Mattheson, OLCF, ORNL)

Soybean Workflow

TACC Wrangler as Execution Environment

Flash Based Shared Storage

Switched to glideins (pilot jobs) - Brings in remote compute nodes and joins them to the HTCondor pool on the submit host - Workflow runs at a finer granularity

Works well on Wrangler due to more cores and memory per node (48 cores, 128 GB RAM)
<table>
<thead>
<tr>
<th>Introduction</th>
<th>Scientific Workflows</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pegasus Overview</td>
</tr>
<tr>
<td></td>
<td>Successful Stories</td>
</tr>
<tr>
<td>Pegasus Overview</td>
<td>Basic Concepts</td>
</tr>
<tr>
<td></td>
<td>Features</td>
</tr>
<tr>
<td></td>
<td>System Architecture</td>
</tr>
<tr>
<td>Features</td>
<td>Data Staging</td>
</tr>
<tr>
<td></td>
<td>Information Catalogs</td>
</tr>
<tr>
<td></td>
<td>Fault-Tolerance</td>
</tr>
<tr>
<td>Break</td>
<td>10min Break</td>
</tr>
<tr>
<td>Hands On Tutorial</td>
<td></td>
</tr>
</tbody>
</table>
Basic concepts...
Key Pegasus Concepts

Pegasus WMS == Pegasus planner (mapper) + DAGMan workflow engine + HTCondor scheduler/broker

- Pegasus maps workflows to infrastructure
- DAGMan manages dependencies and reliability
- HTCondor is used as a broker to interface with different schedulers

Workflows are DAGs

- Nodes: jobs, edges: dependencies
- No while loops, no conditional branches
- Jobs are standalone executables

Planning occurs ahead of execution

Planning converts an abstract workflow into a concrete, executable workflow

Planner is like a compiler

https://pegasus.isi.edu
**DAX**

**DAG in XML**

**Portable Description**

Users do not worry about low level execution details

- **logical filename (LFN)**
  platform independent (abstraction)

- **transformation**
  executables (or programs)
  platform independent

---

**DAG**

**directed-acyclic graphs**

**stage-in job**

Transfers the workflow input data

**stage-out job**

Transfers the workflow output data

**cleanup job**

Removes unused data

**registration job**

Registers the workflow output data

**executable workflow**

Users do not worry about low level execution details

Portable Description

- abstract workflow

- transformation
  executables (or programs)
  platform independent
Pegasus also provides tools to generate the abstract workflow

```python
#!/usr/bin/env python

from Pegasus.DAX3 import *
import sys
import os

# Create a abstract dag
dax = ADAG("hello_world")

# Add the hello job
hello = Job(namespace="hello_world",
            name="hello", version="1.0")
b = File("f.b")
hello.uses(a, Link.INPUT)
hello.uses(b, Link.INPUT)
dax.addJob(hello)

# Add the world job (depends on the hello job)
world = Job(namespace="world",
            name="world", version="1.0")
c = File("f.c")
world.uses(b, Link.INPUT)
world.uses(c, Link.INPUT)
dax.addJob(world)

# Add control-flow dependencies
dax.addDependency(Dependency(parent=hello, child=world))

# Write the DAX to stdout
dax.writeXML(sys.stdout)
```

DAG in XML

http://pegasus.isi.edu
#!/usr/bin/env python
import os, pwd, sys, time
from Pegasus.DAX3 import *

# Create an abstract dag
dax = ADAG("split")

webpage = File("pegasus.html")

# the split job that splits the webpage into smaller chunks
split = Job("split")
split.addArguments("-l","100","-a","1",webpage,"part.")
split.uses(webpage, link=Link.INPUT)

# associate the label with the job. all jobs with same label
# are run with PMC when doing job clustering
split.addProfile( Profile("pegasus","label","p1"))
dax.addJob(split)

# we do a parameter sweep on the first 4 chunks created
for c in "abcd":
    part = File("part.%s" % c)
    split.uses(part, link=Link.OUTPUT, transfer=False, register=False)
    count = File("count.txt.%s" % c)
    wc = Job("wc")
    wc.addProfile( Profile("pegasus","label","p1"))
    wc.addArguments("-l",part)
    wc.uses(part, link=Link.INPUT)
    wc.uses(count, link=Link.OUTPUT, transfer=True, register=True)
dax.addJob(wc)

# adding dependency
dax.depends(wc, split)

f = open("split.dax", "w")
dax.writeXML(f)
f.close()
Running Pegasus workflows with Jupyter

http://pegasus.isi.edu
importing the API

```python
from Pegasus.jupyter.instance import *
```

using the Pegasus DAX3 API to write a workflow

```python
# Create an abstract dag
dax = ADAG("split")

# the split job that splits the webpage into smaller chunks
split = Job("split")
split.addArguments("-l","100","-a","1",webpage,"part.")
split.uses(webpage, link=Link.INPUT)
# associate the label with the job. All jobs with same label
# are run with PMC when doing job clustering
split.addProfile(Profile("pegasus","label","p1"))
dax.addJob(split)

# creating an instance
instance = Instance(dax)

# running a workflow
instance.run(site='condorpool')

# monitoring a workflow execution
instance.status(loop=True, delay=5)
```

Progress: 100.0% (Success)   (Completed: 17, Queued: 0, Running: 0, Failed: 0)
Support for
  Docker
  Singularity – Widely supported on OSG

Users can refer to containers in the Transformation Catalog with their executable preinstalled.

Users can refer to a container they want to use. However, they let Pegasus stage their executable to the node.
  Useful if you want to use a site recommended/standard container image.
  Users are using generic image with executable staging.

Future Plans
  Users can specify an image buildfile for their jobs.
  *Pegasus will build the Docker image as separate jobs in the executable workflow, export them at tar file and ship them around* (planned for 4.8.X)

http://pegasus.isi.edu
Data Management for Containers

• Users can refer to container images as
  • Docker or Singularity Hub URL’s
  • Docker Image exported as a TAR file and available at a server, just like any other input dataset.

• We want to avoid hitting Docker/Singularity Hub repeatedly for large workflows
  • Extend pegasus-transfer to pull image from Docker Hub and then export it as tar file, that can be shipped around in the workflow.

• Ensure pegasus worker package gets installed at runtime inside the user container.
Real-time monitoring of workflow executions. It shows the status of the workflows and jobs, job characteristics, statistics and performance metrics. Provenance data is stored into a relational database.
Real-time monitoring of workflow executions. It shows the status of the workflows and jobs, job characteristics, statistics and performance metrics. Provenance data is stored into a relational database.
$ pegasus-status pegasus/examples/split/run0001
STAT IN_STATE JOB
Run 00:39 split-0 (/home/pegasus/examples/split/run0001)
Idle 00:03 ─split_ID0000001
Summary: 2 Condor jobs total (I:1 R:1)
UNRDY READY PRE IN_Q POST DONE FAIL %DONE STATE DAGNAME
14 0 0 1 0 2 0 11.8 Running *split-0.dag

$ pegasus-statistics –s all pegasus/examples/split/run0001
------------------------------------------------------------------------------
| Type               | Succeeded | Failed | Incomplete | Total | Retries | Total+Retries |
------------------------------------------------------------------------------
| Tasks              | 5         | 0      | 0          | 5     | 0       | 5             |
| Jobs               | 17        | 0      | 0          | 17    | 0       | 17            |
| Sub-Workflows      | 0         | 0      | 0          | 0     | 0       | 0             |
------------------------------------------------------------------------------
Workflow wall time : 2 mins, 6 secs
Workflow cumulative job wall time : 38 secs
Cumulative job wall time as seen from submit side : 42 secs
Workflow cumulative job badput wall time :
Cumulative job badput wall time as seen from submit side :

$ pegasus-analyzer pegasus/examples/split/run0001
pegasus-analyzer: initializing...
****************************Summary***************************
Total jobs : 7 (100.00%)
# jobs succeeded : 7 (100.00%)
# jobs failed : 0 (0.00%)
# jobs unsubmitted : 0 (0.00%)

Provenance data can be summarized

pegasus-statistics

or used for debugging

pegasus-analyzer
Automate, recover, and debug scientific computations.

Get Started

Pegasus Website
http://pegasus.isi.edu

Users Mailing List
pegasus-users@isi.edu

Support
pegasus-support@isi.edu

Pegasus Online Office Hours
https://pegasus.isi.edu/blog/online-pegasus-office-hours/

Bi-monthly basis on second Friday of the month, where we address user questions and also apprise the community of new developments

HipChat
# OUTLINE

## Introduction
- Scientific Workflows
- Pegasus Overview
- Successful Stories

## Pegasus Overview
- Basic Concepts
- Features
- System Architecture

## Features
- Data Staging
- Information Catalogs
- Fault-Tolerance

## Break
- 10min Break

## Hands On Tutorial
Understanding Pegasus features...
So, what information does Pegasus need?

- **Site Catalog**: describes the sites where the workflow jobs are to be executed.
- **Transformation Catalog**: describes all of the executables (called “transformations”) used by the workflow.
- **Replica Catalog**: describes all of the input data stored on external servers.
How does Pegasus decide where to execute?

- **site description**
  - describes the compute resources
  - tells where output data is stored

- **scratch**
  - tells where temporary data is stored

- **storage**
  - tells where output data is stored

- **profiles**
  - key-pair values associated per job level

```xml
|-- The local site contains information about the submit host -->
|-- The arch and os keywords are used to match binaries in the -->
|-- transformation catalog -->
<site handle="local" arch="x86_64" os="LINUX">

|-- These are the paths on the submit host where Pegasus stores data -->
|-- Scratch is where temporary files go -->
<directory type="shared-scratch" path="/home/tutorial/run">
  <file-server operation="all" url="file:///home/tutorial/run"/>
</directory>

|-- Storage is where pegasus stores output files -->
<directory type="local-storage" path="/home/tutorial/outputs">
  <file-server operation="all" url="file:///home/tutorial/outputs"/>
</directory>

|-- This profile tells Pegasus where to find the user's private key -->
|-- for SCP transfers -->
<profile namespace="env" key="SSH_PRIVATE_KEY">
  /home/tutorial/.ssh/id_rsa
</profile>

</site>
```
How does it know where the executables are or which ones to use?

executables description
list of executables locations per site

physical executables
mapped from logical transformations

transformation type
whether it is installed or available to stage

```bash
# This is the transformation catalog. It lists information about
# each of the executables that are used by the workflow.

tr ls {
  site PegasusVM {
    pfn "/bin/ls"
    arch "x86_64"
    os "linux"
    type "INSTALLED"
  }
}
...
What if data is not local to the submit host?

# This is the replica catalog. It lists information about each of the input files used by the workflow. You can use this to specify locations to input files present on external servers.

# The format is:
# LFN PFN site="SITE"

f.a file://home/tutorial/examples/diamond/input/f.a site="local"

**logical filename**
abstract data name

**physical filename**
data physical location on site different transfer protocols can be used (e.g., scp, http, ftp, gridFTP, etc.)

**site name**
in which site the file is available
Multiple sources

pegasus.conf

# Add Replica selection options so that it will try URLs first, then
# XrootD for OSG, then gridftp, then anything else
pegasus.selector.replica=Regex
pegasus.selector.replica.regex.rank.1=file:///cvmfs/.*
pegasus.selector.replica.regex.rank.2=file://.*
pegasus.selector.replica.regex.rank.3=root://.*
pegasus.selector.replica.regex.rank.4=gridftp://.*
pegasus.selector.replica.regex.rank.5=.*

rc.data

# This is the replica catalog. It lists information about each of the
# input files used by the workflow. You can use this to specify locations
# to input files present on external servers.

# The format is:
# LFN PFN site="SITE"

f.a  file:///cvmfs/oasis.opensciencegrid.org/diamond/input/f.a  site="cvmfs"
f.a  file:///local-storage/diamond/input/f.a  site="prestaged"
f.a  gridftp://storage.mysite.edu/examples/diamond/input/f.a  site="storage"
Data Staging Configurations

**HTCondor I/O** (HTCondor pools, OSG, ...)
- Worker nodes do not share a file system
- Data is pulled from / pushed to the submit host via HTCondor file transfers
- Staging site is the submit host

**Non-shared File System** (clouds, OSG, ...)
- Worker nodes do not share a file system
- Data is pulled / pushed from a staging site, possibly not co-located with the computation

**Shared File System** (HPC sites, XSEDE, Campus clusters, ...)
- I/O is directly against the shared file system
There are several possible configurations...

Typically most HPC sites
Cloud Computing

submit host (e.g., user’s laptop)

Typical cloud computing deployment (Amazon S3, Google Storage)

high-scalable object storages

Compute Site

Input data site
Data staging site
Output data site

Staging Site

object storage
Grid Computing

local data management

Typical OSG sites
Open Science Grid

submit host
(e.g., user’s laptop)
And yes... you can mix everything!
Running workflows on AWS

There are many different ways to set up an execution environment in Amazon EC2

The simplest way is to use a submit machine outside the cloud, and to provision several worker nodes and a file server node in the cloud

1. Launch the VM (Condor Worker) – requires configuration
2. The VM will appear as a new compute resource
3. Spawn job to the cloud VM
4. VMs shutdown itself in the absence of work

Guidelines for Tutorial VM:
https://pegasus.isi.edu/documentation/vm_amazon.php
**pegasus-transfer**

*Pegasus’ internal data transfer tool with support for a number of different protocols*

**Directory creation, file removal**
- If protocol supports, used for cleanup

**Two stage transfers**
- e.g., GridFTP to S3 = GridFTP to local file, local file to S3

**Parallel transfers**

**Automatic retries**

**Credential management**
- Uses the appropriate credential for each site and each protocol (even 3rd party transfers)

---

**Protocols Supported**
- HTTP
- SCP
- GridFTP
- Globus Online
- iRods
- Amazon S3
- Google Storage
- SRM
- FDT
- stashcp
- cp
- `ln -s`
And if a job fails?

**Job Failure Detection**
detects non-zero exit code
output parsing for success or failure message
exceeded timeout
do not produced expected output files

**Checkpoint Files**
job generates checkpoint files
staging of checkpoint files is
automatic on restarts

**Job Retry**
helps with transient failures
set number of retries per job and run

**Rescue DAGs**
workflow can be restarted from checkpoint file
recover from failures with minimal loss
A few more features...
Metadata

Can associate arbitrary key-value pairs with workflows, jobs, and files

Data registration

Output files get tagged with metadata on registration in the workflow database

Static and runtime metadata

Static: application parameters
Runtime: performance metrics

<adag ...>

  <metadata key="experiment">par_all27_prot_lipid</metadata>
  <job id="ID0000001" name="namd">
    <argument><file name="equilibrate.conf"/></argument>
    <metadata key="timesteps">500000</metadata>
    <metadata key="temperature">200</metadata>
    <metadata key="pressure">1.01325</metadata>
    <uses name="Q42.psf" link="input">
      <metadata key="type">psf</metadata>
      <metadata key="charge">42</metadata>
    </uses>
    ...
  </job>
  ...
</adag>

register data with metadata

select data based on metadata

Workflow Database

Static metadata from DAX and catalogs
Runtime metadata

Nailogger Events
Collected from kickstart records

Python Metadata API

S3
IRODS
pegasus-dashboard
pegasus-metadata (command line tool)

since Pegasus 4.6

Pegasus
http://pegasus.isi.edu
Performance, why not improve it?

Clustered job
Groups small jobs together to improve performance

Task
Small granularity

Workflow restructuring
Workflow reduction
Hierarchical workflows
Pegasus-mpi-cluster
What about **data reuse**?

Jobs which output data is already available are pruned from the DAG.

**Data reuse**

**Workflow reduction**

**Data already available**

**Data also available**
Pegasus also handles **large-scale workflows**

- **sub-workflow**
- **recursion ends when DAX with only compute jobs is encountered**

- **workflow restructuring**
- **workflow reduction**
- **hierarchical workflows**
- **pegasus-mpi-cluster**
Running **fine-grained** workflows on HPC systems...

**submit host** (e.g., user’s laptop)

**workflow wrapped as an MPI job**

Allows sub-graphs of a Pegasus workflow to be submitted as monolithic jobs to remote resources
Pegasus est. 2001
Automate, recover, and debug scientific computations.

Get Started

Pegasus Website
http://pegasus.isi.edu

Users Mailing List
pegasus-users@isi.edu

Support
pegasus-support@isi.edu

Pegasus Online Office Hours
https://pegasus.isi.edu/blog/online-pegasus-office-hours/

Bi-monthly basis on second Friday of the month, where we address user questions and also apprise the community of new developments

HipChat
# OUTLINE

<table>
<thead>
<tr>
<th>Introduction</th>
<th>Scientific Workflows</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pegasus Overview</td>
</tr>
<tr>
<td></td>
<td>Successful Stories</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pegasus Overview</th>
<th>Basic Concepts</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Features</td>
</tr>
<tr>
<td></td>
<td>System Architecture</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Features</th>
<th>Data Staging</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Information Catalogs</td>
</tr>
<tr>
<td></td>
<td>Fault-Tolerance</td>
</tr>
</tbody>
</table>

| Break              | 10min Break          |

| Hands On Tutorial  |                      |

[http://pegasus.isi.edu](http://pegasus.isi.edu)
Hands-On Pegasus Tutorial...
Pegasus Tutorial

- SSH to our training machine
  - You need to be connected to the ‘utk-open’ network
  - Login with your user’s tutorial login and password

```
ssh pegtrain42@workflow.isi.edu
```
Split Workflow Execution Steps

• Step 1: Change directory to split workflow dir
  cd ~/tutorial/split_example

• Step 2: Generate the Pegasus DAX file
  ./daxgen.py split.dax

• Step 3: Plan and submit the split workflow
  ./plan_dax.sh split.dax

• Step 4: Observe the progress of the workflow
  watch pegasus-status -l submit/USERNAME/pegasus/split_wf/run0001
Pegasus Dashboard

• Step 1: Change workflow db permissions
  
  \$ chmod -R 755 ~/.pegasus

• Step 2: Go to your browser and open
  
  https://workflow.isi.edu:8443

• Step 3: Login with your user’s tutorial login and password
NAMD Workflow Execution Steps

• Step 1: Change directory to namd workflow dir
cd  ~/tutorial/namd_example

• Step 2: Generate the Pegasus DAX file
./daxgen.py namd.dax

• Step 3: Plan and submit the namd workflow
./plan_dax.sh namd.dax

• Step 4: Observe the progress of the workflow
watch pegasus-status -l submit/USERNAME/pegasus/namd_wf/run0001
NAMD Workflow With Docker

• Step 1: Create a docker image with NAMD pre-installed
  There is already on here: https://hub.docker.com/r/papajim/namd_image

• Step 2: Edit the transformation catalog to use docker

```plaintext
cont namd_image {
  type "docker"
  image "docker:///papajim/namd_image:latest"
}
tr namd {
  site condorpool {
    container "namd_image"
    pfn "file:///opt/NAMD_2.12_Linux-x86_64-multicore/namd2"
    arch "x86_64"
    os "LINUX"
    type "INSTALLED"
  }
}```
NAMD Workflow With Docker

• Step 3: Change directory to namd workflow dir
cd ~/tutorial/namd_docker

• Step 4: Generate the Pegasus DAX file
  ./daxgen.py namd.dax

• Step 5: Plan and submit the namd workflow
  ./plan_dax.sh namd.dax

• Step 6: Observe the progress of the workflow
  watch pegasus-status –l submit/USERNAME/pegasus/namd_wf/run0001
Jupyter Notebook

• Go to https://workflow.isi.edu:8000
  • Login with your user’s tutorial login and password
  • Click the button to Launch the Jupyter server
  • Open the folder ‘jupyter’
  • Launch the ‘Pegasus-DAX3-Tutorial.ipynb’ notebook

• Instructions on how to execute the notebook
  • Update the ‘workflow_dir’ variable with your training account name

```
workflow_dir = '/scitech/home/pegtrain42/jupyter/wf-split-tutorial'
```

• Update the ‘replica catalog’ entry with your training account name

```
rc = ReplicaCatalog(workflow_dir)
rc.add('pegasus.html', 'file:///scitech/home/pegtrain42/jupyter/pegasus.html', site='local')
```
NAMD Workflow Execution Steps (NERSC)

• Step 1: Retrieve myproxy credential from NERSC
  myproxy-logon -s nerscca.nersc.gov:7512 -t 24 -T -l NERSC_USER

• Step 2: Change directory to namd workflow dir
  cd ~/tutorial/namd_example

• Step 3: Edit “plan_dax.sh” and update the execution site

```bash
  pegasus-plan \
    --conf pegasus.properties \
    --dax $DAXFILE \
    --dir $DIR/submit \
    --input-dir $DIR/input \
    --output-dir $DIR/output \
    --sites nersc \
    --cleanup leaf \
    --force \
    --submit
```
NAMD Workflow Execution Steps (NERSC)

• Step 4: Update sites catalog with your NERSC scratch folder

```xml
<directory type="shared-scratch" path="YOUR_SHARED_SCRATCH_DIR">
  <file-server operation="all" url="gsiftp://corigrid.nersc.gov/YOUR_SHARED_SCRATCH_DIR"/>
</directory>
```

• Step 5: Generate the Pegasus DAX file

```
./daxgen_nersc.py namd_nersc.dax
```

• Step 6: Plan and submit the namd workflow

```
./plan_dax.sh namd_nersc.dax
```

• Step 7: Observe the progress of the workflow

```
watch pegasus-status -l submit/USERNAME/pegasus/namd_wf/run0002
```