Before we start

<table>
<thead>
<tr>
<th>Hands on Exercises Notes</th>
<th><a href="https://pegasus.isi.edu/tutorial/isil/">https://pegasus.isi.edu/tutorial/isil</a></th>
</tr>
</thead>
</table>

**System**

- workflow.isi.edu
- On terminal login in via ssh

**Training Accounts**

- Pick up from the instructor
EScience 2019: Pegasus Scientific Workflows with Containers

Karan Vahi
Mats Rynge

http://pegasus.isi.edu
| Introduction | Scientific Workflows
|             | Pegasus Overview
|             | Successful Stories |
| Pegasus Overview | Basic Concepts
|                 | Features
|                 | System Architecture |
| Hands-on Tutorial | Submitting a Workflow
|                 | Workflow Dashboard and Monitoring
|                 | Generating the Workflow |
| Understanding Pegasus Features | Information Catalogs
|                               | Containers |
| Hands-on Tutorial | Catalogs
|                 | Workflows with Containers
|                 | Clustering
|                 | Fault-Tolerance |
| Other Features   | Integrity Checking, Data Staging
|                 | Jupyter Notebooks
|                 | Metadata, Hierarchical Workflows, Data Reuse |
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.

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

[http://pegasus.isi.edu](http://pegasus.isi.edu)
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
XENONnT - Dark Matter Search

Two workflows: Monte Carlo simulations, and the main processing pipeline.

Workflows execute across Open Science Grid (OSG) and European Grid Infrastructure (EGI)

Rucio for data management

MongoDB instance to track science runs and data products.

<table>
<thead>
<tr>
<th>Type</th>
<th>Succeeded</th>
<th>Failed</th>
<th>Incomplete</th>
<th>Total</th>
<th>Retries</th>
<th>Total+Retries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks</td>
<td>4000</td>
<td>0</td>
<td>0</td>
<td>4000</td>
<td>267</td>
<td>4267</td>
</tr>
<tr>
<td>Jobs</td>
<td>4484</td>
<td>0</td>
<td>0</td>
<td>4484</td>
<td>267</td>
<td>4751</td>
</tr>
<tr>
<td>Sub-Workflows</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Workflow wall time : 5 hrs, 2 mins
Cumulative job wall time : 136 days, 9 hrs
Cumulative job wall time as seen from submit side : 141 days, 16 hrs
Cumulative job badput wall time : 1 day, 2 hrs
Cumulative job badput wall time as seen from submit side : 4 days, 20 hrs

Main processing pipeline is being developed for XENONnT - data taking will start at the end of 2019. Workflow in development:
**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)
OUTLINE

Introduction
Scientific Workflows
Pegasus Overview
Successful Stories

Pegasus Overview
Basic Concepts
Features
System Architecture

Hands-on Tutorial
Submitting a Workflow
Workflow Dashboard and Monitoring
Generating the Workflow

Understanding Pegasus Features
Information Catalogs
Containers

Hands-on Tutorial
Catalogs
Workflows with Containers
Clustering
Fault-Tolerance

Other Features
Integrity Checking, Data Staging
Jupyter Notebooks
Metadata, Hierarchal Workflows, Data Reuse
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
Portable Description

Users do not worry about low level execution details.

logical filename (LFN)
platform independent (abstraction)

transformation
executables (or programs)
platform independent

stage-in job
Transfers the workflow input data

cleanup job
Removes unused data

stage-out job
Transfers the workflow output data

registration job
Registers the workflow output data

DAG in XML

DAG in XML

directed-acyclic graphs

DAG

DAG

executable workflow

DAG

registered workflow

DAG

registered workflow
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(namespaces="hello_world",
            name="hello", version="1.0")
b = File("f.b")
hello.uses(b, link=Link.INPUT)
dax.addJob(hello)

# Add the world job (depends on the hello job)
world = Job(namespaces="hello_world",
            name="world", version="1.0")
c = File("f.c")
world.uses(c, link=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)
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<dag xmlns="http://pegasus.isi.edu/schema/DAX"
     version="1.4" name="hello_world">
    <!-- describe the jobs making up the hello world pipeline -->
    <job id="10000001" namespace="hello_world"
         name="hello" version="1.0">
        <uses name="f.b" link="output"/>
        <uses name="f.c" link="input"/>
    </job>
    <job id="10000002" namespace="hello_world"
         name="world" version="1.0">
        <uses name="f.b" link="input"/>
        <uses name="f.c" link="output"/>
    </job>
    <!-- describe the edges in the DAG -->
    <edge ref="10000001"/>
    <edge ref="10000002"/>
</dag>
```
An example
Split Workflow

Visualization Tools:
pegasus-graphviz
pegasus-plots

https://pegasus.isi.edu/documentation/tutorial_submitting_wf.php

```
#!/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.setStdout(count)
   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()
```
The Pegasus dashboard is a web interface for monitoring and debugging workflows. It provides 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
Reporting
Debugging
Troubleshooting
RESTful API
The Pegasus dashboard is a web interface for monitoring and debugging workflows. It provides 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.
Provenance data can be summarized with `pegasus-statistics` or used for debugging with `pegasus-analyzer`.

```
$ 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-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%)
```

```
$ 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
```

```
> command-line...

```
Hands-on Pegasus Tutorial...
Hands On Tutorial

• SSH to our training machine
  • Login with your user’s tutorial login and password
  • ssh pegtrainXX@workflow.isi.edu

• Open exercise notes in your browser
  • https://pegasus.isi.edu/tutorial/isi/tutorial.php
## OUTLINE

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

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

### Hands-on Tutorial
- Submitting a Workflow
- Workflow Dashboard and Monitoring
- Generating the Workflow

### Understanding Pegasus Features
- Information Catalogs
- Containers

### Hands-on Tutorial
- Catalogs
- Workflows with Containers
- Clustering
- Fault-Tolerance

### Other Features
- Integrity Checking, Data Staging
- Jupyter Notebooks
- Metadata, Hierarchical Workflows, Data Reuse
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

- **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 were 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

```
# 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"
  }
}
...```

http://pegasus.isi.edu
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
Replica catalog
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"
Pegasus Container Support

• 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)
Pegasus: Data Management

• Treat containers as input data dependency
  • Needs to be staged to compute node if not present

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

• If an image is specified to be residing in a hub
  ▪ The image is pulled down as a tar file as part of data stage-in jobs in the workflow
  ▪ The exported tar file is then shipped with the workflow and made available to the jobs
  ▪ Motivation: Avoid hitting Docker Hub/Singularity Library repeatedly for large workflows

• Symlink against a container image if available on shared filesystem
  ▪ For e.g. CVMFS hosted images on Open Science Grid
<table>
<thead>
<tr>
<th>Section</th>
<th>Overview/Features</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Introduction</strong></td>
<td>Scientific Workflows</td>
</tr>
<tr>
<td></td>
<td>Pegasus Overview</td>
</tr>
<tr>
<td></td>
<td>Successful Stories</td>
</tr>
<tr>
<td><strong>Pegasus Overview</strong></td>
<td>Basic Concepts</td>
</tr>
<tr>
<td></td>
<td>Features</td>
</tr>
<tr>
<td></td>
<td>System Architecture</td>
</tr>
<tr>
<td><strong>Hands-on Tutorial</strong></td>
<td>Submitting a Workflow</td>
</tr>
<tr>
<td></td>
<td>Workflow Dashboard and Monitoring</td>
</tr>
<tr>
<td></td>
<td>Generating the Workflow</td>
</tr>
<tr>
<td><strong>Understanding Pegasus Features</strong></td>
<td>Information Catalogs</td>
</tr>
<tr>
<td></td>
<td>Catalogs</td>
</tr>
<tr>
<td></td>
<td>Workflows with Containers</td>
</tr>
<tr>
<td></td>
<td>Clustering</td>
</tr>
<tr>
<td></td>
<td>Fault-Tolerance</td>
</tr>
<tr>
<td><strong>Hands-on Tutorial</strong></td>
<td>Catalogs</td>
</tr>
<tr>
<td></td>
<td>Workflows with Containers</td>
</tr>
<tr>
<td></td>
<td>Clustering</td>
</tr>
<tr>
<td></td>
<td>Fault-Tolerance</td>
</tr>
<tr>
<td><strong>Other Features</strong></td>
<td>Integrity Checking, Data Staging</td>
</tr>
<tr>
<td></td>
<td>Jupyter Notebooks</td>
</tr>
<tr>
<td></td>
<td>Metadata, Hierarchical Workflows, Data Reuse</td>
</tr>
</tbody>
</table>
Hands-on Pegasus Tutorial...
OUTLINE

Introduction  Scientific Workflows
           Pegasus Overview
           Successful Stories

Pegasus Overview  Basic Concepts
            Features
            System Architecture

Hands-on Tutorial  Submitting a Workflow
              Workflow Dashboard and Monitoring
              Generating the Workflow

Understanding Pegasus Features  Information Catalogs
                                   Containers

Hands-on Tutorial  Catalogs
             Workflows with Containers
             Clustering
             Fault-Tolerance

Other Features  Integrity Checking, Data Staging
                Jupyter Notebooks
                Metadata, Hierarchal Workflows, Data Reuse
A few more features...
Challenges to Scientific Data Integrity

Modern IT systems are not perfect - errors creep in.

At modern “Big Data” sizes we are starting to see checksums breaking down.

Plus there is the threat of intentional changes: malicious attackers, insider threats, etc.

User Perception: “Am I not already protected? I have heard about TCP checksums, encrypted transfers, checksum validation, RAID and erasure coding – is that not enough?”
Motivation:
Network Corruption

Network router software inadvertently corrupts TCP data and/or checksum!

XSEDE and Internet2 example from 2013.

Second similar case in 2017: University of Chicago network upgrade caused data corruption for the FreeSurfer/Fsurf project.

https://www.xsede.org/news/-/news/item/6390
Motivation:
Software failures

Bug in StashCache data transfer software would occasionally cause silent failure (failed but returned zero).

Failures in the final staging out of data were not detected.

The workflow management system, believing workflow was complete, cleaned up. With the final data being incomplete and all intermediary data lost, ten CPU-years of computing came to naught.

How is this an data integrity issue? The workflow system should have verified that the data at the storage system after the transfer, is the expected data.
Integrity validation is on by default since the Pegasus 4.9.0 release (Oct 31st, 2018). Users who upgrade will automatically get the protection, but can opt out.

Sharing of detailed monitoring data with the Pegasus team is off by default. Users can opt-in. (We will come back to this at the end of the talk)
Pegasus performs integrity checksums on input files right before a job starts on the remote node.

- For raw inputs, checksums specified in the input replica catalog along with file locations
- All intermediate and output files checksums are generated and tracked within the system.
- Support for sha256 checksums

Job failure is triggered if checksums fail
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...

submit host
(e.g., user’s laptop)

Compute Site

shared filesystem

Input data site
Data staging site
Output data site

typically most HPC sites
Using Shared File System for Data Access

XSEDE Wrangler

LOGIN NODE

Abstract Workflow

Pegasus Planner

Workflow Setup Job

Workflow Staging Job

Executable Workflow

Condor DAGMan

Condor Schedd

Data Cleanup Job

BLAHHP

HTCondor QUEUE

shared disk such as Lustre Filesystem

Local Disk

Local Disk

Legend

- Directory Setup Job
- Data Stageout Job
- Data Staging Job
- Directory Cleanup Job
Cloud Computing

submit host (e.g., user's laptop)

high-scalable object storages

Typical cloud computing deployment (Amazon S3, Google Storage)

http://pegasus.isi.edu
Grid Computing

local data management

Typical OSG sites
Open Science Grid

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

HPC System

Master (rank 0)

worker
(rank 1)

worker
(rank n-1)

workflow restructuring
workflow reduction
hierarchical workflows

pegasus-mpi-cluster

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

- **clustered job**: Groups small jobs together to improve performance.
- **task**: Small granularity.
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

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

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

**Rescue DAGs**
workflow can be restarted from checkpoint file
recover from failures with minimal loss
Running Pegasus workflows with Jupyter

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

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

creating an instance of the DAX

```python
instance = Instance(dax)
```

running a workflow

```python
instance.run(site='condorpool')
```

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)

instance.run(site='condorpool')
```

monitoring a workflow execution

```python
instance.status(loop=True, delay=5)
```

Progress: 100.0% (Success) (Completed: 17, Queued: 0, Running: 0, Failed: 0)

http://pegasus.isi.edu
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

Pegasus
http://pegasus.isi.edu

since Pegasus 4.6

<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>
    ...
    <uses name="eq.restart.coord" link="output" transfer="false">
      <metadata key="type">coordinates</metadata>
    </uses>
    ...
  </job>
</adag>

register data with metadata
select data based on metadata
What about **data reuse**?

Jobs which output data is already available are pruned from the DAG.
Pegasus also handles large-scale workflows.

Workflow restructuring
Workflow reduction
Hierarchical workflows
Pegasus-mpi-cluster

Sub-workflow

Recursion ends when DAX with only compute jobs is encountered.
Job Submissions

Local

Submit Machine
  Personal HTCondor

Local Campus Cluster accessible via Submit Machine *
  HTCondor via Glite

Remote

BOSCO + SSH**
  Each node in executable workflow submitted via SSH connection to remote cluster

BOSCO based Glideins**
  SSH based submission of Glideins

PyGlidein
  ICE Cube Glidein service

OSG using glideinWMS

CREAMCE
  Uses CondorG

Globus GRAM
  Uses CondorG

** Both Glite and BOSCO build on HTCondor BLAHP Support.

Supported schedulers

PBS  SGE  SLURM  MOAB
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