### Before we start

#### Hands on Exercises Notes

[https://pegasus.isi.edu/tutorial/CC-IN2P3/](https://pegasus.isi.edu/tutorial/CC-IN2P3/)

<table>
<thead>
<tr>
<th>System</th>
<th>cctbcondor09.in2p3.fr</th>
</tr>
</thead>
</table>
CC-IN2P3 Pegasus Tutorial

Pegasus Workflow Management System

Mats Rynge
Rafael Ferreira da Silva

https://pegasus.isi.edu
<table>
<thead>
<tr>
<th>OUTLINE</th>
</tr>
</thead>
</table>

| Introduction | Scientific Workflows  
Pegasus Overview  
Success 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 | Workflows with Containers  
Clustering  
Fault-Tolerance |
| Other Features | Data Staging  
Jupyter Notebooks  
Metadata, Hierarchal Workflows, Data Reuse |

http://pegasus.isi.edu
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

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
Files, site services, endpoints, policies, …

Debug and Monitor Computations
Correlate data across lots of tasks / metadata / log files
Need to know what host a job ran on, how it was invoked, and in what environment

Restructure Workflows for Improved Performance
Short running tasks / Data placement and management / …
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 success stories…
60,000 compute tasks
Input Data: 5000 files (10GB total)
Output Data: 60,000 files (60GB total)
executed on LIGO Data Grid, EGI, 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 100,000s 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, 23,000 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

Each workflow has **420,000 tasks**
Soykb Workflow
TACC Wrangler as Execution Environment

HTCondor 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 great on Wrangler due to the flash filesystem, and the memory per core (48 cores, 128 GB RAM)
**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:
# OUTLINE

| **Introduction** | Scientific Workflows  
Pegasus Overview  
Success 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** | Workflows with Containers  
Clustering  
Fault-Tolerance |
| **Other Features** | Data Staging  
Jupyter Notebooks  
Metadata, Hierarchal Workflows, Data Reuse |

http://pegasus.isi.edu
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)

DAG in XML

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

Portable Description

Users do not worry about low level execution details.

logical filename (LFN)

platform independent (abstraction)

DAG in XML

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

Portable Description

Users do not worry about low level execution details.

logical filename (LFN)

platform independent (abstraction)

DAG in XML

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
Pegasus also provides tools to generate the abstract workflow.

```python
#!/usr/bin/env python
from Pegasus.DAX3 import *
import sys

# Create a abstract dag
dag = DAG("hello_world")

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

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

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

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

DAG in XML

http://pegasus.isi.edu
An example

Split Workflow

```python
#!/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)
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 = 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)

    # job dependency
    dax.depends(wc, split)

f = open("split.dax", "w")
dax.writeXML(f)
f.close()
```
Pegasus dashboard

web interface for monitoring and debugging workflows

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
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

```bash
$ 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
```

or used for debugging

```bash
$ 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%)
```

```bash
$ pegasus-statistics -s all pegasus/examples/split/run0001
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Succeeded</th>
<th>Failed</th>
<th>Incomplete</th>
<th>Total Retries</th>
<th>Total+Retries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tasks</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Jobs</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>Sub-Workflows</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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 :
# 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>Success Stories</td>
</tr>
</tbody>
</table>

### Pegasus Overview

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

### Hands-on Tutorial

<table>
<thead>
<tr>
<th>Submitting a Workflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow Dashboard and Monitoring</td>
</tr>
<tr>
<td>Generating the Workflow</td>
</tr>
</tbody>
</table>

### Understanding Pegasus Features

<table>
<thead>
<tr>
<th>Information Catalogs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Containers</td>
</tr>
</tbody>
</table>

### Hands-on Tutorial

<table>
<thead>
<tr>
<th>Workflows with Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clustering</td>
</tr>
<tr>
<td>Fault-Tolerance</td>
</tr>
</tbody>
</table>

### Other Features

<table>
<thead>
<tr>
<th>Data Staging</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jupyter Notebooks</td>
</tr>
<tr>
<td>Metadata, Hierarchal Workflows, Data Reuse</td>
</tr>
</tbody>
</table>

---

[23]
Hands-on Pegasus Tutorial…
OUTLINE

Introduction
Scientific Workflows
Pegasus Overview
Success 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
Workflows with Containers
Clustering
Fault-Tolerance

Other Features
Data Staging
Jupyter Notebooks
Metadata, Hierarchal 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
<site handle="local" arch="x86_64" os="LINUX">

<!-- These are the paths on the submit host were Pegasus stores data -->
<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.

```bash
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"
Replica catalog

multiple sources

```
# 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=.*
```

```
# 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
  - Shifter (coming soon)

- 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)
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.
<table>
<thead>
<tr>
<th>OUTLINE</th>
</tr>
</thead>
</table>
| **Introduction** | Scientific Workflows  
| | Pegasus Overview  
| | Success 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** | Workflows with Containers  
| | Clustering  
| | Fault-Tolerance  |
| **Other Features** | Data Staging  
| | Jupyter Notebooks  
| | Metadata, Hierarchal Workflows, Data Reuse  |
Hands-on Pegasus Tutorial…
OUTLINE

Introduction
Scientific Workflows
Pegasus Overview
Success 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
Workflows with Containers
Clustering
Fault-Tolerance

Other Features
Data Staging
Jupyter Notebooks
Metadata, Hierarchal Workflows, Data Reuse
A few more features…
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...
Typical cloud computing deployment (Amazon S3, Google Storage)
Grid Computing

local data management

Typical OSG sites
Open Science Grid
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 can support it, also 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
Performance, why not improve it?

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

[Diagram showing workflow restructuring and hierarchical workflows: pegasus-mpi-cluster]
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

http://pegasus.isi.edu
Running Pegasus workflows with Jupyter

http://pegasus.isi.edu
from Pegasus.jupyter.instance import *

# 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","pl"))
dax.addJob(split)

instance = Instance(dax)

# creating an instance of the DAX
instance.run(site='condorpool')

# running a workflow

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

# monitoring a workflow execution

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

Example of Pegasus workflow with metadata:

```
<adag ...>
    <job id="ID0000001" name="namd">
        <argument><file name="equilibrate.conf"/></argument>
        <metadata key="timesteps" value="500000"/>
        <metadata key="temperature" value="200"/>
        <metadata key="pressure" value="1.01325"/>
        <uses name="Q42.psf" link="input">
            <metadata key="type" value="psf"/>
            <metadata key="charge" value="42"/>
        </uses>
        ...
        <uses name="eq.restart.coord" link="output" transfer="false">
            <metadata key="type" value="coordinates"/>
        </uses>
        ...
    </job>
    ...
</adag>
```

Workflow, job, file

Pegasus

https://pegasus.isi.edu
What about **data reuse**?

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

** Local Campus Cluster accessible via Submit Machine **
HTCondor via BLAHP

** Both Glite and BOSCO build on HTCondor BLAHP

Currently supported schedulers:

SLURM  SGE  PBS  MOAB

---

** Remote **

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

BOSCO based Glideins**
SSH based submission of glideins

PyGlidein
IceCube glidein service

OSG using glideinWMS
Infrastructure provisioned glideins

CREAMCE
Uses CondorG

Globus GRAM
Uses CondorG
Using Shared FileSystem for Data Access

XSEDE Wrangler

LOGIN NODE

Abstract Workflow

Workflow Setup Job

Workflow Staging Job

Execution Workflow

Workflow Staging Job

Data Cleanup Job

Pegasus Planner

SLURM QUEUE

HTCondor QUEUE

J

W

Shared Disk such as Lustre Filesystem

C1

Local Disk

Cn

Local Disk

J

W

LEGEND

- Orange: Directory Setup Job
- Green: Data Stageout Job
- Light Green: Data Staging Job
- Red: Directory Cleanup Job

Concord DAGMan

Condor Schedd

BLAHP
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*