ACCESS Pegasus
A hosted scientific workflow system
part of ACCESS Support

Mats Rynge, USC/ISI
ACCESS Support Strategy

Powerful Tools & Workflows

Dynamic Knowledge Base

Community Expertise and Experience
Powerful Tools & Workflows

**OnDemand**

INTEGRATED WEB-BASED INTERFACES
Schedule jobs, manage files, create remote visualizations and use a host of other valuable services.

**Pegasus**

AUTOMATED WORKFLOWS
Simplify complex data workflows on distributed computing resources, such as clusters, grids, and clouds.
Knowledge Base
Self-service resources reduce the learning curve

DOCS
RP guides, code and best practices

LINKS
Provided and vetted by the community

TICKETS
Answers build the Knowledge Base

ASK.CI
Community Q&A forum
Computational Science & Support Network

CSSN

Utilizing community expertise
MATCH Engagements
Assistance is available from CSSN experts matched to Researcher needs.

\[ M^+ = R + SM \]
MATCHPlus
Researchers assisted by a Student and their Mentor

\[ M^{++} = R + C \]
MATCHPremier
Researchers assisted by an expert Consultant

- \( R \) < 6 MONTHS
- \( C \) 6+ MONTHS
Pegasus
Scientific Workflows

- An abstraction to express ensemble of complex computational operations
  - Eg: retrieving data from remote storage services, executing applications, and transferring data products to designated storage sites

- A workflow is represented as a directed acyclic graph (DAG)
  - Nodes: tasks or jobs to be executed
  - Edges: depend between the tasks

- Have a monolithic application/experiment?
  - Find the inherent DAG structure in your application to convert into a workflow
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
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
Stage-out generated output data

Registration Job
Registers the workflow output data

directed-acyclic graphs
Pegasus provides tools to generate the Abstract Workflow

```
#!/usr/bin/env python3

import os
import logging
from pegasus import Workflow
from pegasus import Job
from pegasus import Pipeline

logging.basicConfig(level=logging.INFO)

# --- Create Pegasus API -----------------
from pegasus.api import *

# --- Create Abstract Workflow -----------
wf = Workflow('pipeline')

webpage = File('pegasus.html')

# --- Create Parent Job -----------------
json = Job()
   .add_arg('-a', webpage, 'http://pegasus.isi.edu')
   .add_outputs(webpage, stage_out=True, register_replica=True)
   count = File('count.txt')

# --- Create Dependent Job ---------------
mc_job = Job()
   .add_arg('-i', webpage)
   .set_stderr(count, stage_out=True, register_replica=True)

# --- Add Jobs to the Abstract Workflow ---
wf.add_jobs(json, mc_job)

# --- Add control flow dependency ------
wf.add_dependency(wf, json, parents=[mc_job])

# --- Write out the Abstract Workflow ----
wf.write()
```

**YAML Formatted**

```yaml
x-pegasus:
  appname: pegasus
  createdby: vali
  createdon: 11-19-2014:57:56Z
  pegasus: '5.0'
  name: pipeline
  jobs:
    type: job
      name: curl
      id: ID0000001
      arguments:
        -p
        -pegasus.html
        -http://pegasus.isi.edu
      uses:
        -f:' f: pegasus.html
          type: output
          stageOut: false
          registerReplica: false
        -type: job
          name: mc
          id: ID0000002
          stdout: count.txt
          arguments:
            -l
            -pegasus.html
            -f: f: count.txt
            type: output
            stageOut: true
            registerReplica: true
            -f: f: pegasus.html
            type: input
            jobDependencies:
              -id: ID0000001
              children:
              -ID0000002
```
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.
command-line...

$ 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
------------------------------------------------------------------------------
<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>5</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Jobs</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>17</td>
<td>0</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>
<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

$ 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
Success Stories
Data Flow for LIGO Pegasus Workflows in OSG

60,000 Compute Tasks
Input Data: 5000 files (10GB total)
Output Data: 60,000 files (60GB total)
Processed Data: 725 GB

Executed on LIGO Data Grid, EGI, Open Science Grid and XSEDE
SoyKB

PI: Dong Xu, Trupti Joshi, Saad Kahn, Yang Liu, Juexin Wang, Badu Valliyodan, Jiaojiao Wang
Processing instrument data in real time

Cryo-EM Imaging Facility

Raw data

Thermofisher Data PC

CARC Data Server

Automatic Data Transfer

Real-time Feedback

Semi-Automatic Pre-processing: Motion Correction, CTF Estimation

Cryo-EM GPU Cluster

Discovery/Endeavour HPC Cluster

Processing: CryoSPARC RELION EMAN2

10PB Project FS

CARC HPC Facility

User Interaction with the Cryo-EM Web Portal

Session Start Detection
User Input Task Initiation
Relay Notifications Start
Relay Notifications End
Session End Detection
Stage Out the Data

Computational Processing on GPU Cluster

Pegasus WMS

Apply Motion Correction
Compute CTF Estimate
Prepare the Preview Image
Prepare the Notice
Send the Notice via Slack

Progress of Time

User Reacts to the Notice

Data Transfer End

Data Session End
ACCESS Pegasus

https://support.access-ci.org/pegasus
Prepare

Logging In

CILogin with your ACCESS ID and institutional login

- https://access.pegasus.isi.edu

All registered ACCESS users with an active allocation automatically have access
Prepare

Setting Up Resources

One time setup

Use **Open OnDemand instances** at resource providers to install ssh keys, and determine local allocation id.
Step 1

Designing Workflow

Pegasus API in Jupyter Notebook

Fully hosted environment, based on Open Ondemand
Step 2
Provision Resources

Use the HTCondor Annex tool to dynamically bring in compute nodes from one or more resource providers.

[Code example]

```
[rynge@access ~]$ htcondor annex annex
--nodes 1 --lifetime 86400 --project
ddm160003 $USER standard@anvil
```
HTCondor Annex / Pilot Jobs

• A pilot can run multiple user jobs - it stays active until no more user jobs are available or until end of life has been reached, whichever comes first.

• A pilot is partitionable - job slots will dynamically be created based on the resource requirements in the user jobs. This means you can fit multiple user jobs on a compute node at the same time.

• A pilot will only run jobs for the user who started it.
HTCondor with BLAHP translation layer
HTCondor Pilot Jobs
Step 3
Monitoring Workflow and Resources

Workflows can be monitored from within the Jupyter notebook, or via command line.

HTCondor Annex can be monitored on the command line.
Try it out!

Documentation:

https://support.access-ci.org/pegasus

Open a ticket:

https://support.access-ci.org/open-a-ticket

Questions?
Tutorial
Tutorial

This is **not** using ACCESS resources - jobs are staying local in the container.

If we are not finishing here today, feel free to keep exploring on your own

In-person: handout
Remote:  [https://tinyurl.com/pegasus-ern](https://tinyurl.com/pegasus-ern)