ACCESS Pegasus
A hosted workflow system for ACCESS users, bringing their own capacity

Mats Rynge
University of Southern California
Information Sciences Institute

Supported by NSF #2138286
Pegasus Workflow Management System
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*
Laser Interferometer Gravitational Wave Observatory (LIGO) develops large scale analysis pipelines used for gravitational wave detection.

Southern California Earthquake Center (SCEC) CyberShake project generates hazard maps using hierarchical workflows.

Oak Ridge National Lab (ORNL) conducted studies on tRNA and nanodiamonds to improve drug delivery design principles.

- Develop portable scientific workflows in Python, Java, and R
- Compile workflows to be run on heterogeneous resources
- Monitor and debug workflow execution via CLI and web-based tools
- Recover from failures with built-in fault tolerance mechanisms
- Regular release schedule incorporating latest research and development

Visualization of water on nanodiamond spheres from ORNL

LIGO observation of colliding black holes

Hazard map indicating maximum amount of shaking at a particular geographic location generated from SCEC's CyberShake Pegasus 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
Pegasus provides APIs to generate the Abstract Workflow

```python
#!/usr/bin/env python3
import os
import logging
from pathlib import Path
from argparse import ArgumentParser

logging.basicConfig(level=logging.DEBUG)

# --- Import Pegasus API ---
from Pegasus.api import *

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

# create node
node = Node('curl]
.add_arg('-o', 'webpage', 'http://pegasus.isi.edu')
.add_output('webpage', stage_out=True, register_replica=True)

# create job
job = Job('curl]
.add_arg('-o', 'webpage')
.add_output('webpage', stage_out=True, register_replica=True)

# create dependent job
mc_job = (Job('curl]
.add_arg('-o', 'nonwebpage')
.add_task('webpage', stage_out=True, register_replica=True)

# add jobs to the abstract workflow
wf.add_jobs(curl_job, mc_job)

# add control flow dependency
wf.add_dependency(mc_job, parent=curl_job)

# write out the abstract workflow
wf.write()
```

YAML Formatted

```
x-pegasus:
  apilang: python
  createdBy: valhi
  pegasus: '5.0'
  name: pipeline
  jobs:
    - type: job
      name: curl
      id: 100000001
      arguments:
        - type: pegasus.html
        - http://pegasus.isi.edu
      uses:
        - fn: pegasus.html
          type: output
          stageOut: false
          registerReplica: false
        - type: job
          name: wc
          id: 100000002
      stdout: count.txt
      arguments:
        - type: pegasus.html
      uses:
        - fn: count.txt
          type: output
          stageOut: true
          registerReplica: true
        - fn: pegasus.html
          type: input
      jobDependencies:
        - id: 100000001
        - 100000002
```

Abstract Workflow
So, what other 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.
Performance. Why not improve it?

Clustered Job
Groups small jobs together to improve performance

Task
Small granularity
Data Reuse prune jobs if output data already exists

Jobs which output data is already available are pruned from the DAG.
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., SCP to S3 = SCP 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)
Automatic Integrity Checking

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.
And if a job fails?

- **Postscript**: 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.
Success Stories
Southern California Earthquake Center’s CyberShake

**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

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)

---

**SCEC/USC**

**Pre-processing**
Configuration Files (2 GB)
Mesh Generation
Velocity Mesh (120 GB)
SGT X Simulation
SGT Y Simulation

**Simulation**

**Post-processing**
Seismograms & intensity measures (8TB)
Strain Green Tensors (1.5 TB)

**Titan, Blue Waters**

**Serial, 1 core x 0.1 hr**
Configuration Files

**Parallel, 3840 cores x 0.4 hr**
Mesh Generation
Velocity Mesh

**Parallel, 800 GPUs x 1 hr**
SGT X Simulation
SGT Y Simulation

**Blue Waters**

**Parallel, 3712 cores x 11 hr**
Seismograms & intensity measures

---

286 Sites
4 Models
Each Workflow Has 420,000 Tasks
Monte Carlo simulations and the main processing pipeline.

- Workflows execute across Open Science Grid (OSG) & 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
Event Horizon Telescope
Bringing Black Holes into Focus

8 telescopes: 5 PB of data
60 simulations: 35 TB data

First images of black hole at the center of the M87 galaxy

Improve constraints on Einstein’s theory of general relativity by 500x

Pegasus-SYMBA Pipeline

Physically accurate synthetic observation data from simulations are keys to develop calibration and imaging algorithms, as well as comparing the observation with theory and interpreting the results.

1. Pull simulated image from CyVerse Data Store
2. Model interstellar scattering
3. Synthetic interferometry data
4. Model observation systematics
5. Calibrate synthetic observation
6. Push synthetic observation data back to CyVerse

480,000 jobs - 2,600,000 core hours
#15 in all OSG projects in last 6 months
#2 in all OSG astronomy projects in the last 6 months

480,000 jobs - 2,600,000 core hours
#15 in all OSG projects in last 6 months
#2 in all OSG astronomy projects in the last 6 months
ACCESS Pegasus
ACCESS Researcher Support Services

- Enable innovative research through equitable and scalable support
- Four tiers of support
- Tools, growing knowledge base
- Match-making with experts
- Student engagement
- Engagement from community
- CSSN incentives

https://support.access-ci.org
ACCESS Support Strategy

Reduce the need for support by simplifying access to resources

**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.

Powerful Tools & Workflows
ACCESS Pegasus
Leveraging Open OnDemand and Jupyter Notebooks

[Image with text and website link]

https://pegasus.access-ci.org/
ACCESS Pegasus
Leveraging Open OnDemand and Jupyter Notebooks
Example Workflows

In addition to tutorial workflows, a set of example workflows are automatically installed into each user account - easy to explore, execute and modify!

- Artificial Intelligence
  - Lung Segmentation
  - Mask Detection
  - Orca Sound
  - LLM + RAG
- Astronomy
  - Montage
- Bioinformatics
  - AlphaFold
  - Rosetta
  - VariantCalling
Bring Your Own Capacity

- ACCESS CI consists of allocatable resources
- Motivations
  - Increased / known capacity
  - GPUs
  - Parallel filesystems, I/O
  - Clouds / configurable
- ACCESS Pegasus connects to both shared and BYOC resources
The TestPool is a small resource which is always available. This can be used for development, debugging and running small workflows.

For cloud resources, currently IU Jetstream2, a custom image is provided. The image is started with a provided security token, which makes it available to your jobs.

HTCondor Annex enables you to provision compute resources on HPC clusters such as SDSC Expanse, PSC Bridges2, and Purdue Anvil.

The OSG Open Science Pool (OSPool) will run jobs in a distributed infrastructure. You do not need to provision these resources, but there are special rules and capabilities.
Shared Capacity - TestPool

*Helps users with an ACCESS account but no allocation try the capability*

Small amount of compute resources attached to pegasus.access-ci.org

- CPU: 32 cores, 128 GB RAM, 256 GB disk
- GPU: 32 cores, 2 GPUs, 128 GB RAM, 256 GB disk
- hosted on IU Jetstream2, provisioned when needed

Always available, no allocation needed

Can be used for quick turnaround jobs

- workflow development and debugging
- tutorials (not all users might have an allocation at the time of the tutorial)
BYOC - Cloud

- IU JetStream2
- Provided VM image
- Users have to add pegasus.access-ci.org username and token in the cloud-init yaml
- Instances self-terminates when there are no more jobs
BYOC - HTCondor Annex

- Bring your own HPC allocation
- Semi-managed, submits glideins via SSH.
  - A glidein 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 glidein 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 glidein will only run jobs for the user who started it.
- Documentation: [https://htcondor.org/experimental/ospool/byoc/](https://htcondor.org/experimental/ospool/byoc/)

$ htcondor annex create --nodes 1 --lifetime 7200 --project sta230005p \  
  --gpu-type v100-16 $USER GPU@bridges2
Shared or BYOC - OSPool

- pegasus.access-ci.org is an OSPool access point, via flocking
- Jobs have to set +ProjectName, which can be an ACCESS allocation (if the user have one), or any registered OSG project name
- No OSDF origin
  - Can use the ACCESS allocated Open Storage Network (OSN), which is a S3 compatible object store
Example AI Workflow - LLM + RAG

LLM RAG (Large Language Model Retrieval-Augmented Generation) is a technique that enhances large language models by incorporating information retrieval mechanisms. It involves retrieving relevant information from a database or document corpus and combining it with the original query to provide additional context. This augmented input is then processed by the large language model to generate more accurate and contextually relevant responses. LLM RAG offers benefits such as improved accuracy, access to up-to-date information, and better contextual understanding, making it useful for applications like question answering, summarization, and conversational AI.

- Container includes LLM
- Inputs: IU Jetstream2 documentation PDF
- Same query twice, once without RAG and one with RAG
- “Please provide a summary of the GPU capabilities of the IU Jetstream2 system. Include the instance flavors, and any details about the GPUs.”
- Can execute on GPUS from any of the capacity providers: TestPool, Cloud, HTCondor Annex, OSPool

Summary without RAG:

The IU Jetstream2 system is a high-performance computing resource provided by Indiana University for research purposes. Regarding its GPU capabilities, Jetstream2 offers several instance flavors that include GPUs:

1. `gpu96`: This flavor provides 4 NVIDIA V100 GPUs (with 32GB of memory each) and 128GB of RAM on the host CPU node.
2. `gpu60`: This flavor offers 2 NVIDIA V100 GPUs (each with 32GB of memory) and 64GB of RAM on the host CPU node.

Summary with RAG:

The IU Jetstream2 system has 90 GPU-enabled nodes with four NVIDIA A100 GPUs each. These are divided using NVIDIA virtual GPU (vGPU) to allow allocations to utilize from 1/5th of a GPU to an entire GPU in instances, facilitating use for educational and research workloads requiring varying amounts of GPU processing power.

The Jetstream2 system offers several GPU instance flavors: g3.small, g3.medium, g3.large, and g3.xl. Here’s a summary of their specifications:

- g3.small: 4 vCPUs, 15 GB RAM, 60 GB local storage, 20% GPU compute, 20 GB GPU RAM
- g3.medium: 8 vCPUs, 30 GB RAM, 60 GB local storage, 25% GPU compute, 10 GB GPU RAM
- g3.large: 16 vCPUs, 60 GB RAM, 60 GB local storage, 50% GPU compute, 20 GB GPU RAM
- g3.xl: 32 vCPUs, 125 GB RAM, 60 GB local storage, 100% GPU compute, 40 GB GPU RAM
Thank You!

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

https://pegasus.isi.edu
## Deployment Scenarios

<table>
<thead>
<tr>
<th>HTCondor Pool</th>
<th>Slurm (+HTCondor)</th>
<th>Glideins / Pilot Jobs</th>
<th>Hosted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Workflow jobs are submitted to an existing HTCondor pool, and handled directly by HTCondor.</td>
<td>Common setup on campus clusters. Pegasus and HTCondor are installed on a head node. Workflow jobs are submitted to HTCondor, which translates the jobs to Slurm jobs. Good solution for parallel jobs.</td>
<td>Variation of HTCondor Pool, but the capacity is dynamically added to the pool by submitting pilot jobs. The pilot jobs will become part of the pool, and will start executing workflow jobs.</td>
<td>Larger infrastructure projects might have hosted instances, commonly built on top of either a static pool or glideins. ACCESS Pegasus described later is an example.</td>
</tr>
</tbody>
</table>
The Ocean Observatories Initiative (OOI), through a network of sensors, supports critical research in ocean science and marine life. Orcasound is a community driven project that leverages hydrophone sensors deployed in three locations in the state of Washington (San Juan Island, Point Bush, and Port Townsend as shown in the figure below) in order to study Orca whales in the Pacific Northwest region.

The workflow processes the hydrophone data of one or more sensors in batches for each timestamp, and converts them to a WAV format. Using the WAV output it creates spectrogram images that are stored in the final output location. Furthermore, using the pretrained Orcasound model, the workflow scans the WAV files to identify potential sounds produced by the orcas.
Precise detection of the borders of organs and lesions in medical images such as X-rays, CT, or MRI scans is an essential step towards correct diagnosis and treatment planning. We implement a workflow that employs supervised learning techniques to locate lungs on X-ray images. Lung instance segmentation workflow uses Chest X-ray for predicting lung masks from the images using U-Net model.
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
Pegasus Container Support

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

Users can refer to a container they want to use – Pegasus stages their executables and containers 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 as a tar file and ship them around.

Containers Execution Model

- Directory Setup
- Pull image
- Start container
- $PWD bind-mounted as /srv
- Pull worker package (if needed)
- Set job environment
- Stage in inputs
- Execute user application
- Stage out outputs
- Stop container
- Cleanup
Data Management for Containers

Containers are data too!

Pegasus treats containers as input data dependency
- Staged to compute node if not present
- Docker or Singularity Hub URL's
- Docker Image exported as a TAR file and available at a server, just like any other input dataset

Scaling up for larger workflows
- 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
- Pricing considerations. You are now charged if you exceed a certain rate of pulls from Hubs

Other Optimizations
- Symlink against existing images on shared file system such as CVMFS
- The exported tar file is then shipped with the workflow and made available to the jobs