Pegasus
Workflow Management System

Rafael Ferreira da Silva

https://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 success stories...
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

286 sites, 4 models each workflow has 420,000 tasks
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
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:
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
Portable Description

Users do not worry about low level execution details

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

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

**DAG in XML**

**directed-acyclic graphs**

**DAG**

**executable workflow**

**stage-in job**
Transfers the workflow input data

**stage-out job**
Transfers the workflow output data

**registration job**
Registers the workflow output data

**cleanup job**
Removes unused data

**Pegasus**

Portable Description

Users do not worry about low level execution details

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

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

**DAG in XML**

**directed-acyclic graphs**

**DAG**

**executable workflow**

**stage-in job**
Transfers the workflow input data

**stage-out job**
Transfers the workflow output data

**registration job**
Registers the workflow output data

**cleanup job**
Removes unused data

**Pegasus**
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 = DAG(namespace="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=OUTPUT)
dax.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(b, Link=INPUT)
world.uses(c, Link=OUTPUT)
dax.addJob(world)

# Add control-flow dependencies

dax.addDependency(Dependency(parent="hello",
                            child="world")

# Write the DAX to stdout

dax.writeXML(sys.stdout)
```

<xml version="1.0" encoding="UTF-8">
<!-- generator: python -->
<doc xmlns="http://pegasus.isi.edu/schemas/DAX" version="3.4" name="hello_world">
  <!-- describe the jobs making up the hello world pipeline -->
  <job id="ID000001" namespace="hello_world" name="hello" version="1.0">
    <uses name="f.b" link="output"/>
    <uses name="f.a" link="input"/>
  </job>

  <job id="ID000002" 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="ID000001"/>
  <edge ref="ID000000"/>
</doc>
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...

```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_ID00000001
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
```

```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</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
Workflow cumulative job badput wall time :
Cumulative job badput wall time as seen from submit side :
```

```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%)
```

Provenance data can be summarized with `pegasus-statistics` or used for debugging with `pegasus-analyzer`. 
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.
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...

High Performance Computing

submit host

Compute Site

shared filesystem

Input data site
Data staging site
Output data site

typically most HPC sites
Cloud Computing

Typical cloud computing deployment (Amazon S3, Google Storage)

submit host

Compute Site

object storage

Input data site
Data staging site
Output data site

Staging Site

high-scalable object storages
Grid Computing

Typical OSG sites
Open Science Grid

submit host

local data management
**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
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
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"></file></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>
  ...
  <uses name="eq.restart.coord" link="output" transfer="false">
    <metadata key="type">coordinates</metadata>
  </uses>
  ...
</adag>
Pegasus also handles large-scale workflows

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 BLAHP

** 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

** Both Glite and BOSCO build on HTCondor BLAHP

Currently supported schedulers:
SLURM SGE PBS MOAB
Amazon AWS Batch

• AWS Batch
  Container based, dynamically scaled and efficient batch computing service

  Automatically launches compute nodes in Amazon based on demand in the associated job queue

  Users can specify compute environment that dictates what type of VM’s are launched

• Pegasus will allow clusters of jobs to be run on Amazon EC2 using AWS Batch Service

  New command line tool: pegasus-aws-batch

  Automates most of the batch setup programmatically
  • Sets up and Deprovisions
    • Compute Environment
    • Job Queues
  • Follows AWS Batch HTTP specification
Pegasus est. 2001
Automate, recover, and debug scientific computations.

Get Started

Pegasus Website
https://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
Backup slides...
And yes… you can mix everything!
What about **data reuse**?

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

**Data also available**

**Workflow reduction**

**Data reuse**

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