Why workflows?

*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*
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 (or hierarchical DAGs)

- Directed acyclic graph
- Nodes: jobs, edges: dependencies
- No while loops, no conditional branches

Planning occurs ahead of execution

- (Except hierarchical workflows)

Planning converts an abstract workflow into a concrete, executable workflow

- Planner is like a compiler
Taking a closer look into a workflow…

- **DAG** (Directed Acyclic Graphs)
- **job**
- **dependency**
- **split**
- **merge**
- **pipeline**

Command-line programs

Usually data dependencies

**DAG in XML**

- **abstract workflow**
- **executable workflow**
- **optimizations**
- **storage constraints**
From the abstraction to execution!

- **stage-in job**: Transfers the workflow input data
- **stage-out job**: Transfers the workflow output data
- **registration job**: Registers the workflow output data

abstract workflow
executable workflow
optimizations
storage constraints
Optimizing storage usage...

cleanup job
Removes unused data
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(namespace="hello_world",
            name="hello", version="1.0")
b = File("f.b")
hello.uses(a, link=Link.INPUT)
hello.uses(b, link=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=Link.INPUT)
world.uses(c, link=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)
```
While you wait…

Web-based interface
Real-time monitoring, graphs, provenance, etc.

Statistics
Workflow execution and job performance metrics

Command-line tools
Tools for monitor and debug workflows

RESTful API
Monitoring and reporting information on your own application interface

Debug
Set of debugging tools to unveil issues

Does everything executed successfully?

How my workflow behaves?

Past executions?

…or the execution is finished.

https://pegasus.isi.edu
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 (pegasus-statistics) or used for debugging (pegasus-analyzer)

$ pegasus-statistics

<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>100000</td>
<td>0</td>
<td>0</td>
<td>100000</td>
<td>543</td>
<td>100543</td>
</tr>
<tr>
<td>Jobs</td>
<td>20206</td>
<td>0</td>
<td>0</td>
<td>20206</td>
<td>604</td>
<td>20810</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: 19 hrs, 37 mins
Cumulative job wall time: 1 year, 5 days
Cumulative job wall time as seen from submit side: 1 year, 27 days
Cumulative job badput wall time: 2 hrs, 42 mins
Cumulative job badput wall time as seen from submit side: 2 days, 2 hrs

$ 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%)
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

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

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

**Rescue DAGs**
workflow can be restarted from checkpoint
file recover from failures with minimal loss
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…

Typically most HPC sites
Cloud Computing

High-scalable object storages

Workflow Engine

Submit host

Compute Site

Object storage

Input data site
Data staging site
Output data site

Staging Site

Typical cloud computing deployment
(Amazon S3, Google Storage)
And yes… you can mix everything!
• Pegasus’ internal data transfer tool
• Supports many 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
• Checkpoint and restart transfers
• Credential management
  • Uses the appropriate credential for each site and each protocol (even 3rd party transfers)
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 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">/wf/key.priv</profile>
</site>
```

- **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
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

```plaintext
... 
# 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"
    }
}
... 
```
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 http://storage.mysite.edu/examples/diamond/input/f.a site="storage"

**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**

```bash
# 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**

```bash
# 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"
```
A few more features...
Performance, why not improve it?

*clustered job*
Groups small jobs together to improve performance

*task*
small granularity

workflow restructuring
workflow reduction
hierarchical workflows
pegasus-mpi-cluster
What about data reuse?

Data already available

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

Workflow reduction

Workflow restructuring

Hierarchical workflows

Pegasus-mpi-cluster
Pegasus also handles large-scale workflows

workflow restructuring
workflow reduction
hierarchical workflows
pegasus-mpi-cluster

recursion ends when DAX with only compute jobs is encountered
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
Metadata

- Can associate arbitrary key-value pairs with workflows, jobs, and files

- Replica selection
  - Input files are selected based on metadata attributes

- Data registration
  - Output files get tagged with metadata on registration

- Static and runtime metadata
  - Static: application parameters
  - Runtime: performance metrics

New in Pegasus 4.6, added to support users who want to select data based on attributes rather than names (e.g. LIGO)
DAX Metadata Example

1. `<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>
   </job>
   ...<uses name="eq.restart.coord" link="output" transfer="false">
     <metadata key="type">coordinates</metadata>
   </uses>
   ...
</adag>
Pegasus’ flow at a glance

- **Data Reuse**
  - Replica Catalog

- **Task Clustering**
  - Transformation Catalog

- **Directory Creation and File Cleanup**
  - Site Catalog

- **Code Generation**
  - Executable workflow

- **abstract workflow**

- **Site Selection**
  - Site Selector
  - Site Catalog
  - Transformation Catalog
  - Replica Catalog

- **Transfer Refiner**
  - Replica Selector
  - Replica Catalog

- **Remote Workflow Engine**
  - Site Catalog
  - Transformation Catalog

http://pegasus.isi.edu
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.

PyCBC Papers: An improved pipeline to search for gravitational waves from compact binary coalescence. Samantha Usman, Duncan Brown et al.
The PyCBC search for gravitational waves from compact binary coalescence, Samantha Usman et al (https://arxiv.org/abs/1508.02357)
PyCBC Detection GW150914: First results from the search for binary black hole coalescence with Advanced LIGO. B. P. Abbott et al.
http://soykb.org

XSEDE Allocation
PI: Dong Xu
Trupti Joshi, Saad Kahn, Yang Liu, Juexin Wang, Badu Valliyodan, Jiaojiao Wang

https://github.com/pegasus-isi/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 in the submit host - Workflow runs at a finer granularity

Works well on TACC Wrangler due to more cores and memory per node (48 cores, 128 GB RAM)
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).

Southern California Earthquake Center’s CyberShake

286 sites, 4 models each workflow has 420,000 tasks
Upcoming Features

4.8 – Late summer 2017
  • Containers
  • Jupyter

4.9 – Early 2018?
  • Data integrity
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

HipChat