Pegasus
Automate, recover, and debug scientific computations.

Mats Rynge
rynge@isi.edu

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
Taking a closer look into a workflow…

usually data dependencies

Command-line programs

job

dependency

merge

pipeline

split

DAG in XML

http://pegasus.isi.edu
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

http://pegasus.isi.edu
Optimizing storage usage...

`cleanup job`
Removes unused data

http://pegasus.isi.edu
Pegasus also provides tools to generate the abstract workflow

```python
import dax

dax = ADAG("test_dax")
firstJob = Job(name="first_job")
firstInputFile = File("input.txt")
firstOutputFile = File("tmp.txt")
firstJob.addArgument("input=input.txt", "output=tmp.txt")
firstJob.uses(firstInputFile, link=Link.INPUT)
firstJob.uses(firstOutputFile, link=Link.OUTPUT)
dax.addJob(firstJob)

for i in range(0, 5):
    simulJob = Job(id="%s" % (i+1), name="simul_job")
simulInputFile = File("tmp.txt")
simulOutputFile = File("output.%d.dat" % i)
simulJob.addArgument("parameter=%d" % i, "input=tmp.txt", "output=%s" % simulOutputFile.getName())
simulJob.uses(simulInputFile, link=Link.INPUT)
simulJob.uses(simulOutputFile, link=Link.OUTPUT)
dax.addJob(simulJob)
dax.depends(p=firstJob, c=simulJob)

fp = open("test_dax", "w")
dax.writeXML(fp)
fp.close()
```
While you wait...

...or the execution is finished.

Does everything executed successfully?

Statistics
Workflow execution and job performance metrics

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

Past executions?

Command-line tools
Tools for monitor and debug workflows

How my workflow behaves?

Debug
Set of debugging tools to unveil issues

RESTful API
Monitoring and reporting information on your own application interface
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.
But, if you prefer the 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-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%)

$ pegasus-statistics -s all pegasus/examples/split/run0001
Type                  Succeeded Failed Incomplete Total Retries Total+Retries
---------------------------------------------------------------
Tasks                  5         0         0         5         0         5
Jobs                  17         0         0       17         0      17
Sub-Workflows         0         0         0         0         0      0
---------------------------------------------------------------
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 :

...Pegasus provides a set of concise and powerful tools
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
Worried about data? Let Pegasus manage it for you
How we handle it:

submit host
(e.g., user’s laptop)

Input data site
Data staging site
Output data site

Compute site A

Compute site B

data transfers
However, there are several possible configurations for data sites...

submit host (e.g., user’s laptop)

Compute Site

shared filesystem

Input data site
Data staging site
Output data site

typically most HPC sites
Pegasus also handles high-scalable object storages

submit host
(e.g., user’s laptop)

Typical cloud computing deployment
(Amazon S3, Google Storage)
Pegasus can also manage data over the submit host...

submit host
(e.g., user’s laptop)

Typical OSG sites
Open Science Grid
And yes... you can mix everything!
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">
  <!-- The local site contains information about the submit host -->
  <!-- The arch and os keywords are used to match binaries in the transformation catalog -->
  <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

```bash
... # 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"
    }
}
...```

http://pegasus.isi.edu
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"

**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
A few more features...
Performance, why not improve it?

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

Related terms:
- workflow restructuring
- workflow reduction
- hierarchical workflows
- pegasus-mpi-cluster

http://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

workflow restructuring
workflow reduction
hierarchical workflows
pegasus-mpi-cluster

sub-workflow
recursion ends when DAX with only compute jobs is encountered

http://pegasus.isi.edu
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

**HPC System**

**Master** (rank 0)

**worker**

**rank n-1**

**rank 1**
Pegasus’ flow at a glance

Data Reuse
- Replica Catalog

Task Clustering
- Transformation Catalog

Directory Creation and File Cleanup
- Site Catalog

Code Generation
- Remote Workflow Engine
  - Site Catalog
  - Transformation Catalog

abstract workflow

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

Transfer Refiner
- Replica Selector
- Replica Catalog

Code Generation
- Remote Workflow Engine
  - Site Catalog
  - Transformation Catalog

executable workflow

http://pegasus.isi.edu
Science-grade Mosaic of the Sky (Galactic Plane - Montage)

18 million input images (~2.5TB)
1,100 output images (2.5GB each, 2.4TB total)
17 workflows, each of which contains
900 sub-workflows (hierarchical workflows)
10.5 million tasks (34,000 CPU hours)

executed on the cloud (Amazon EC2)

How Pegasus has been used?

Periodogram

1.1M tasks grouped into 180 jobs
1.1M input, 12M output files
~101,000 CPU hours
16 TB output data

executed at SDSC

Pegasus

Advanced LIGO – Laser Interferometer Gravitational Wave Observatory

60,000 compute tasks
Input Data: 5000 files (10GB total)
Output Data: 60,000 files (60GB total)

executed on LIGO Data Grid, Open Science Grid and XSEDE

SCEC CyberShake

286 sites, 4 models
each workflow has 420,000 tasks
described as 21 jobs using PMC

executed on BlueWaters (NCSA) and Stampede (TACC)
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 Wrangler due to more cores and memory per node (48 cores, 128 GB RAM)
Workflow supports 3 different execution environments

1. SDSC Comet (glideins)
2. TACC Stampede (pegasus-mpi-cluster)
3. Distributed (local HTCondor pool)
submit host (workflow.isi.edu)
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

HipChat
Pegasus

Automate, recover, and debug scientific computations.

Thank You

Questions?

Mats Rynge
rynge@isi.edu

USC Viterbi
School of Engineering
Information & Computing

Meet our team

Ewa Deelman
Karan Vahi
Gideon Juve
Mats Rynge
Rajiv Mayani
Rafael Ferreira da Silva

NSF
DEPARTMENT OF
ENERGY
NIH