Pegasus

Automate, recover, and debug scientific computations.

Rafael Ferreira da Silva

http://pegasus.isi.edu
**Scientific Problem**
Earth Science, Astronomy, Neuroinformatics, Bioinformatics, etc.

**Computational Scripts**
Shell scripts, Python, Matlab, etc.

**Analytical Solution**

**Distributed Computing**
Clusters, HPC, Cloud, Grid, etc.

**Automation**
Workflows, MapReduce, etc.

**Scientific Result**
Models, Quality Control, Image Analysis, etc.

**Monitoring and Debug**
Fault-tolerance, Provenance, etc.
What is involved in an experiment execution?
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

http://pegasus.isi.edu
Taking a closer look into a workflow...

- **job**
- **dependency**
  - Usually data dependencies

- **split**
- **merge**
- **pipeline**

**directed-acyclic graphs**

**abstract workflow**
**executable workflow**
**optimizations**
**storage constraints**

**DAG in XML**

**Pegasus**

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

abstract workflow
executable workflow
registrations
storage constraints

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

cleanup job
Removes unused data

http://pegasus.isi.edu
In a nutshell...

...and all automatically!
Pegasus also provides tools to generate the abstract workflow

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="%d" % (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, line=Link.OUTPUT)
dax.addJob(simulJob)
dax.depends(parent=firstJob, child=simulJob)
fp = open("test_dax", "w")
dax.writeXML(fp)
fp.close()
While you wait...

...or the execution is finished.

Does everything executed successfully?

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

How my workflow behaves?

Debug
Set of debugging tools to unveil issues

RESTful API
Monitoring and reporting information on your own application interface

Past executions?

Statistics
Workflow execution and job performance metrics

Command-line tools
Tools for monitor and debug workflows

http://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.
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 l=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

http://pegasus.isi.edu
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)

Compute site A

Compute site B

Data transfers

1. Input data site
2. Data staging site
3. Compute site A
4. Output data site
However, there are several possible configurations for data sites...

submit host
(e.g., user’s laptop)

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...

Typical OSG sites
Open Science Grid

submit host
(e.g., user’s laptop)
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
<!-- The local site contains information about the submit host -->
<!-- The arch and os keywords are used to match binaries in the transformation catalog -->
<site handle="local" arch="x86_64" os="LINUX">
  <!-- These are the paths on the submit host were Pegasus stores data -->
  <!-- Scratch is where temporary files go -->
  <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"
    }
}
...```
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.

```plaintext
# 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

Workflow restructuring
Workflow reduction
Hierarchical workflows
Pegasus-MPI-Cluster
What about data reuse?

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

Data also available

Data already available

Workflow reduction

Data reuse

Workflow restructuring

Hierarchical workflows

Pegasus-mpi-cluster

http://pegasus.isi.edu
Pegasus also handles large-scale workflows

sub-workflow

sub-workflow

recursion ends when DAX with only compute jobs is encountered
Running fine-grained workflows on HPC systems...

Workflow wrapped as an MPI job

Allows sub-graphs of a Pegasus workflow to be submitted as monolithic jobs to remote resources
Real-time collection of time-series of workflow performance metrics

**Averaged CPU Time for NAMD**

- **Metric**: Simulation time [s]
- **Values**: 0, 10, 20, 30, 40, 50, 60, 70

**Write I/O for NAMD**

- **Metric**: Bytes written (MB)
- **Values**: Run 1, Run 2, Run 3, Run 4, Run 5

**Time-series data in real-time** integrated with Pegasus dashboard

**Time-series data**

I/O (read, write), memory, CPU
Pegasus’ flow at a glance

Data Reuse
- Replica Catalog

Task Clustering
- Transformation Catalog

Directory Creation and File Cleanup
- Site Catalog

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

Transfer Refiner
- Replica Selector
- Replica Catalog

Remoter Workflow Engine
- Site Catalog
- Transformation Catalog

abstract workflow

Code Generation

executable workflow
Science-grade Mosaic of the Sky
(Galactic Plane - Montage)

18 million input images (~2.5TB)
900 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

ORNL Spallation Neutron Source (SNS)
5 jobs that consumes about
900 cores for more than 12 hours

executed on Hopper (NERSC)

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)
Pegasus est. 2001
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 est. 2001
Automate, recover, and debug scientific computations.

Thank You
Questions?

Rafael Ferreira da Silva

Meet our team

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