Pegasus and DAGMan
Generating and running workflows on the Grid

Gaurang Mehta, Karan Vahi
USC Information Sciences Institute, Marina Del Rey, CA

Kent Wenger
University of Wisconsin Madison, Madison, WI
Outline of Tutorial

- Introduction (15 mins)
- DAGMan (1 hr)
  - Overview
  - Hands On
- Break (20 mins)
- Pegasus (1 hr 45 mins)
  - Overview
  - Hands On
- Summary (10 mins)
**Generating mosaics of the sky**

<table>
<thead>
<tr>
<th>Size of the mosaic is degrees square*</th>
<th>Number of input data files</th>
<th>Number of Intermediate files</th>
<th>Total data footprint</th>
<th>Number of jobs</th>
<th>Approx. execution time (20 procs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>53</td>
<td>588</td>
<td>1.2GB</td>
<td>232</td>
<td>40 mins</td>
</tr>
<tr>
<td>2</td>
<td>212</td>
<td>3,906</td>
<td>5.5GB</td>
<td>1,444</td>
<td>49 mins</td>
</tr>
<tr>
<td>4</td>
<td>747</td>
<td>13,061</td>
<td>20GB</td>
<td>4,856</td>
<td>1hr 46 mins</td>
</tr>
<tr>
<td>6</td>
<td>1,444</td>
<td>22,850</td>
<td>38GB</td>
<td>8,586</td>
<td>2 hrs. 14 mins</td>
</tr>
<tr>
<td>10</td>
<td>3,722</td>
<td>54,434</td>
<td>97GB</td>
<td>20,652</td>
<td>6 hours</td>
</tr>
</tbody>
</table>

*The full moon is 0.5 deg. sq. when viewed from Earth, Full Sky is ~ 400,000 deg. sq.*
Computational Workflows

- Montage example: Composing a large image based on many individual images (Bruce Berriman, Caltech)
- Involves several interdependent computational steps
  - applied to each image
  - applied to sets of images
Workflow Building Blocks

- Standalone computations
- Data transfers
- Result (final and intermediate) registration in catalogs (optional)

- In distributed environments there are many choices of compute and data resources
- In many cases data movement depends on the scheduling of the computation
Pegasus

- Based on the programming language principles
  - Leverage abstraction for workflow description to obtain ease of use, scalability, and portability
  - Provide a compiler to map from high-level descriptions to executable workflows
    - Correct mapping
    - Performance enhanced mapping
  - Rely on a runtime engine to carry out the instructions
    - Scalable manner
    - Reliable manner
Pegasus

- Enables the construction of complex workflows based using computational blocks
- Infers data transfers
- Infers data registrations
- Provides a portable workflow description

- May improve application performance
- May improve the reliability of the execution
- Relies on Condor DAGMan for correct, scalable, and reliable execution
DAGMan (Directed Acyclic Graph MANager)

- Runs workflows that can be specified as Directed Acyclic Graphs
- Enforces DAG dependencies
- Progresses as far as possible in the face of failures
- Provides retries, throttling, etc.
- Runs on top of Condor (and is itself a Condor job)
Benefits of Scientific Workflows (from the point of view of an application scientist)

- Conducts a series of computational tasks.
  - Resources distributed across Internet.
- Chaining (outputs become inputs) replaces manual hand-offs.
  - Accelerated creation of products.
- Ease of use - gives non-developers access to sophisticated codes.
  - Avoids need to download-install-learn how to use someone else's code.
- Provides framework to host or assemble community set of applications.
  - Honors original codes. Allows for heterogeneous coding styles.
- Framework to define common formats or standards when useful.
  - Promotes exchange of data, products, codes. Community metadata.
- Multi-disciplinary workflows can promote even broader collaborations.
  - E.g., ground motions fed into simulation of building shaking.
- Certain rules or guidelines make it easier to add a code into a workflow.

Slide courtesy of David Okaya, SCEC, USC
DAGMan Hands-On

Kent Wenger
(wenger@cs.wisc.edu)
University of Wisconsin
Madison, Madison, WI
Outline of Tutorial

- Introduction (15 mins)
- DAGMan ( 1 hr )
  - Overview
  - Hands On
- Break ( 20 mins)
- Pegasus ( 1 hr 45 mins )
  - Overview
  - Hands On
- Summary ( 10 mins)
General info

- Already set up in /scratch/trainxx/tg07_dagman_tutorial
- These slides at: http://www.cs.wisc.edu/condor/tutorials/tg07_dagman.ppt
- Tar file of exercises available: http://www.cs.wisc.edu/condor/tutorials/tg07_dagman_tutorial.tgz
- DAGMan exercises can run on any Condor pool
Exercise 1 (run a Condor job)

% cd tg07_tutorial/nodejob
% make
  cc nodejob.c -o nodejob

% cd ../ex1

% condor_submit ex1.submit
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 1859.
Exercise 1, continued

- Monitoring your Condor job
  - `Condor_q [-sub name]`
  - `Condor_history [name]`
Exercise 1, continued

```
% condor_q -sub train15

-- Submitter: train15@isi.edu : <128.9.72.178:43684> : viz-login.isi.edu
  ID    OWNER                  SUBMITTED     RUN_TIME ST PRI SIZE CMD
1859.0 train15               5/31 10:53   0+00:00:07 R  0   9.8  nodejob Miguel Ind
1 jobs; 0 idle, 1 running, 0 held

...  

% condor_q -sub train15

-- Submitter: train15@isi.edu : <128.9.72.178:43684> : viz-login.isi.edu
  ID    OWNER                  SUBMITTED     RUN_TIME ST PRI SIZE CMD
 0 jobs; 0 idle, 0 running, 0 held
```
## Exercise 1, continued

```
% condor_history train15

<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>ST</th>
<th>COMPLETED</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1015.0</td>
<td>train15</td>
<td>5/28 11:34</td>
<td>0+00:01:00</td>
<td>C</td>
<td>5/28 11:35</td>
<td>/nfs/home/train</td>
</tr>
<tr>
<td>1017.0</td>
<td>train15</td>
<td>5/28 11:45</td>
<td>0+00:01:00</td>
<td>C</td>
<td>5/28 11:46</td>
<td>/nfs/home/train</td>
</tr>
<tr>
<td>1018.0</td>
<td>train15</td>
<td>5/28 11:46</td>
<td>0+00:01:00</td>
<td>C</td>
<td>5/28 11:47</td>
<td>/nfs/home/train</td>
</tr>
</tbody>
</table>

...
Exercise 1, continued

- The Condor submit file

```
% more ex1.submit
# Simple Condor submit file.

Executable      = ../nodejob/nodejob
Universe        = scheduler
Error           = job.err
Output          = job.out
Getenv          = true
Log             = job.log

Arguments       = Miguel Indurain
Notification    = never
Queue
```
A simple DAG

- We will use this in exercise 2
DAG file

● Defines the DAG shown previously
● Node names are case-sensitive
● Keywords are not case-sensitive

```bash
# Simple DAG for exercise 2.

JOB Setup setup.submit
JOB Proc1 proc1.submit
JOB Proc2 proc2.submit
JOB Cleanup cleanup.submit
PARENT Setup CHILD Proc1 Proc2
PARENT Proc1 Proc2 CHILD Cleanup
```
- Treated as a unit
- Job or POST script determines node success or failure
Staging data on the TeraGrid

- DAGMan does not automatically handle this
- To be discussed in the Pegasus portion of the tutorial
Condor_submit_dag

- Creates a Condor submit file for DAGMan
- Also submits it (unless \texttt{--no_submit})
- \texttt{-f} option forces overwriting of existing files
User logs (for node jobs)

- This is how DAGMan monitors state
- Not on NFS!
- Truncated at the start of the DAG
Exercise 2 (run a basic DAG)

- Node jobs must have log files

```
% cd ../ex2

% condor_submit_dag -f ex2.dag

Checking all your submit files for log file names.
This might take a while...
5/31 10:58:58 MultiLogFiles: No 'log =' value found in submit file cleanup
submit for node Cleanup
ERROR: Failed to locate Condor job log files: No 'log =' value found in submit
file cleanup.submit for node Cleanup
Aborting -- try again with the -AllowLogError flag if you *really* think this
shouldn't be a fatal error
```
Exercise 2, continued

- Edit cleanup.submit
- Re-submit the DAG

```
% condor_submit_dag -f ex2.dag

Checking all your submit files for log file names.
This might take a while...
checking /scratch/train15/tg07_tutorial/ex2 instead...
Done.

File for submitting this DAG to Condor : ex2.dag.condor.sub
Log of DAGMan debugging messages : ex2.dag.dagman.out
Log of Condor library debug messages : ex2.dag.lib.out
Log of the life of condor_dagman itself : ex2.dag.dagman.log

Condor Log file for all jobs of this DAG : /scratch/train15/tg07_tutorial/ex
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 1860.
```
Exercise 2, continued

- Monitoring your DAG
  - `Condor_q -dag [-sub name]`
  - Dagman.out file

% condor_q -sub train15 -dag

-- Submitter: train15@isi.edu : <128.9.72.178:43684> : viz-login.isi.edu

<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER/NODENAME</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>ST</th>
<th>PRI</th>
<th>SIZE</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1860.0</td>
<td>train15</td>
<td>5/31 10:59</td>
<td>0+00:00:26</td>
<td>R</td>
<td>0</td>
<td>9.8</td>
<td>condor_dagman -f -</td>
</tr>
<tr>
<td>1861.0</td>
<td></td>
<td>-Setup</td>
<td>5/31 10:59</td>
<td>0+00:00:12</td>
<td>R</td>
<td>0</td>
<td>9.8</td>
</tr>
</tbody>
</table>

2 jobs; 0 idle, 2 running, 0 held
Exercise 2, continued

% tail -f ex2.dag.dagman.out
5/31 11:01:09 Event: ULOG_SUBMIT for Condor Node Proc1 (1862.0)
5/31 11:01:09 Number of idle job procs: 1
5/31 11:01:09 Event: ULOG_EXECUTE for Condor Node Proc1 (1862.0)
5/31 11:01:09 Number of idle job procs: 0
5/31 11:01:09 Event: ULOG_SUBMIT for Condor Node Proc2 (1863.0)
5/31 11:01:09 Number of idle job procs: 1
5/31 11:01:09 Of 4 nodes total:
5/31 11:01:09 Done Pre Queued Post Ready Un-Ready Failed
5/31 11:01:09 === === === === === === ===
5/31 11:01:09 1 0 2 0 0 1 0
PRE/POST scripts

- SCRIPT PRE|POST node script [arguments]
- All scripts run on submit machine
- If PRE script fails, node fails w/o running job or POST script (for now…)
- If job fails, POST script is run
- If POST script fails, node fails
- Special macros:
  - $JOB
  - $RETURN (POST only)
Exercise 3 (PRE/POST scripts)

- Proc2 job will fail, but POST script will not

% cd ../ex3
% condor_submit_dag -f ex3.dag
Checking all your submit files for log file names. This might take a while...
Done.

File for submitting this DAG to Condor: ex3.dag.condor.sub
Log of DAGMan debugging messages: ex3.dag.dagman.out
Log of Condor library debug messages: ex3.dag.lib.out
Log of the life of condor_dagman itself: ex3.dag.dagman.log

Condor Log file for all jobs of this DAG: /scratch/train15/tg07
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 1905.
Exercise 3, continued

% more ex3.dag
# DAG with PRE and POST scripts.

JOB Setup setup.submit
SCRIPT PRE Setup pre_script $JOB
SCRIPT POST Setup post_script $JOB $RETURN

JOB Proc1 proc1.submit
SCRIPT PRE Proc1 pre_script $JOB
SCRIPT POST Proc1 post_script $JOB $RETURN

JOB Proc2 proc2.submit
SCRIPT PRE Proc2 pre_script $JOB
SCRIPT POST Proc2 post_script $JOB $RETURN

JOB Cleanup cleanup.submit
SCRIPT PRE Cleanup pre_script $JOB
SCRIPT POST Cleanup post_script $JOB $RETURN

PARENT Setup CHILD Proc1 Proc2
PARENT Proc1 Proc2 CHILD Cleanup
Exercise 3, continued

- From dagman.out:

  5/31 11:12:55 Event: ULOG_JOB_TERMINATED for Condor Node Proc2 (1868.0)
  5/31 11:12:55 Node Proc2 job completed
  5/31 11:12:55 Running POST script of Node Proc2...

  ...

  5/31 11:13:00 Event: ULOG_POST_SCRIPT_TERMINATED for Condor Node Proc2 (1868.0)
  5/31 11:13:00 POST Script of Node Proc2 completed successfully.

  ...
VARS (per-node variables)

- VARS JobName macroname="string"
  
  [macroname="string"... ]

- Macroname can only contain alphanumeric characters and underscore

- Value can’t contain single quotes; double quotes must be escaped

- Macronames are not case-sensitive
Rescue DAG

- Generated when a node fails or DAGMan is condor_rm’ed
- Saves state of DAG
- Run the rescue DAG to restart from where you left off
Exercise 4 (VARS/rescue DAG)
Exercise 4, continued

% cd ../ex4

% more ex4.dag
# DAG to show VARS and rescue DAG.

JOB Setup setup.submit

JOB Proc1.1 proc.submit
VARS Proc1.1 Args = "Eddy Merckx"

JOB Proc1.2 proc.submit
VARS Proc1.2 ARGS = "Bjarne Riis -fail"

JOB Proc1.3 proc.submit
VARS Proc1.3 ARGS = "Sean Yates"

JOB Proc2.1 proc.submit
VARS Proc2.1 ARGS = "Axel Merckx"
...

Exercise 4, continued

% condor_submit_dag -f ex4.dag

...

% tail ex4.dag.dagman.out
5/31 11:19:57 Aborting DAG...
5/31 11:19:57 Writing Rescue DAG to ex4.dag.rescue...
5/31 11:19:57 Note: 0 total job deferrals because of -MaxJobs limit (0)
5/31 11:19:57 Note: 0 total job deferrals because of -MaxIdle limit (0)
5/31 11:19:57 Note: 0 total PRE script deferrals because of -MaxPre limit (0)
5/31 11:19:57 Note: 0 total POST script deferrals because of -MaxPost limit (0)
5/31 11:19:57 **** condor_scheduniv_exec.1870.0 (condor_DAGMAN) EXITING WITH STATUS 1
Exercise 4, continued

- Edit ex4.dag.rescue (remove "-fail" in ARGS for Proc1.2)
- Submit rescue DAG

```
% condor_submit_dag -f ex4.dag.rescue
...

% tail -f ex4.dag.rescue.dagman.out
5/31 11:46:16 All jobs Completed!
5/31 11:46:16 Note: 0 total job deferrals because of -MaxJobs limit (0)
5/31 11:46:16 Note: 0 total job deferrals because of -MaxIdle limit (0)
5/31 11:46:16 Note: 0 total PRE script deferrals because of -MaxPre limit (0)
5/31 11:46:16 Note: 0 total POST script deferrals because of -MaxPost limit (0)
5/31 11:46:16 **** condor_scheduniv_exec.1877.0 (condor_DAGMAN) EXITING WITH STATUS 0
```
Throttling

- Maxjobs (limits jobs in queue/running)
- Maxidle (limits idle jobs)
- Maxpre (limits PRE scripts)
- Maxpost (limits POST scripts)
- All limits are *per DAGMan*, not global for the pool
Configuration

- Condor configuration files
- Environment variables (`_CONDOR_<macroname>`) (6.9.2+)
- DAGMan configuration file (6.9.2+)
- Condor_submit_dag command line
Exercise 5 (config/throttling)
Exercise 5, continued

```
% cd ../ex5

% more ex5.dag
# DAG with lots of siblings to illustrate throttling.
# This only works with version 6.9.2 or later.
# CONFIG ex5.config

JOB Setup setup.submit

JOB Proc1 proc.submit
VARS Proc1 ARGS = "Alpe d'Huez"
PARENT Setup CHILD Proc1
...

% more ex5.config
DAGMAN_MAX_JOBS_SUBMITTED = 4
```
Exercise 5, continued

% condor_submit_dag -f -maxjobs 4 ex5.dag

...

$ condor_q -dag -sub train15

-- Submitter: train15@isi.edu : <128.9.72.178:43684> : viz-login.isi.edu

<table>
<thead>
<tr>
<th>ID</th>
<th>OWNER/NODENAME</th>
<th>SUBMITTED</th>
<th>RUN_TIME</th>
<th>ST</th>
<th>PRI</th>
<th>SIZE</th>
<th>CMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1910.0</td>
<td>train15</td>
<td>6/1 08:17</td>
<td>0+00:00:46</td>
<td>R</td>
<td>0</td>
<td>9.8</td>
<td>condor_dagman -f -</td>
</tr>
<tr>
<td>1912.0</td>
<td>-Proc1</td>
<td>6/1 08:17</td>
<td>0+00:00:03</td>
<td>R</td>
<td>0</td>
<td>9.8</td>
<td>nodejob Processing</td>
</tr>
<tr>
<td>1913.0</td>
<td>-Proc2</td>
<td>6/1 08:17</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>9.8</td>
<td>nodejob Processing</td>
</tr>
<tr>
<td>1914.0</td>
<td>-Proc3</td>
<td>6/1 08:17</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>9.8</td>
<td>nodejob Processing</td>
</tr>
<tr>
<td>1915.0</td>
<td>-Proc4</td>
<td>6/1 08:17</td>
<td>0+00:00:00</td>
<td>I</td>
<td>0</td>
<td>9.8</td>
<td>nodejob Processing</td>
</tr>
</tbody>
</table>

5 jobs; 3 idle, 2 running, 0 held
Exercise 5, continued

% tail ex5.dag.dagman.out
6/1 08:19:51 Of 12 nodes total:
6/1 08:19:51 Done  Pre  Queued  Post  Ready  Un-Ready  Failed
6/1 08:19:51 ===  ===  ===  ===  ===  ===  ===
6/1 08:19:51 12  0  0  0  0  0  0
6/1 08:19:51 Note: 50 total job deferrals because of -MaxJobs limit (4)
6/1 08:19:51 All jobs Completed!
6/1 08:19:51 Note: 50 total job deferrals because of -MaxJobs limit (4)
6/1 08:19:51 Note: 0 total job deferrals because of -MaxIdle limit (0)
6/1 08:19:51 Note: 0 total PRE script deferrals because of -MaxPre limit (0)
6/1 08:19:51 Note: 0 total POST script deferrals because of -MaxPost limit (0)
6/1 08:19:51 **** condor_scheduniv_exec.1910.0 (condor_DAGMAN) EXITING WITH STATUS 0
Recovery/bootstrap mode

- Most commonly, after `condor_hold/condor_release` of DAGMan
- Also after DAGMan crash/restart
- Restores DAG state by reading node job logs
Node retries

- RETRY JobName NumberOfRetries [UNLESS-EXIT value]
- Node is retried as a whole
Exercise 6 (recovery/node retries)

% cd ../ex6

% more ex6.dag
# DAG illustrating node retries.

JOB Setup setup.submit
SCRIPT PRE Setup pre_script $JOB
SCRIPT POST Setup post_script $JOB $RETURN

JOB Proc proc.submit
SCRIPT PRE Proc pre_script $JOB
SCRIPT POST Proc post_script $JOB $RETURN
RETRY Proc 2 UNLESS-EXIT 2

PARENT Setup CHILD Proc
Exercise 6, continued

% condor_submit_dag -f ex6.dag

...

% condor_q -sub train15
-- Submitter: viz-login.isi.edu : <128.9.72.178:43684> : viz-login.isi.edu
   ID      OWNER                  SUBMITTED    RUN_TIME ST PRI SIZE CMD
1895.0   train15              5/31 11:58    0+00:00:21 R  0  9.8  condor_dagman -f -
1896.0   train15              5/31 11:58    0+00:00:08 R  0  9.8  nodejob Setup node

2 jobs; 0 idle, 2 running, 0 held

% condor_hold 1895
Cluster 1895 held.

% condor_q -sub train15 -dag
-- Submitter: train15@isi.edu : <128.9.72.178:43684> : viz-login.isi.edu
   ID      OWNER/NODENAME    SUBMITTED    RUN_TIME ST PRI SIZE CMD
1895.0   train15           5/31 11:58    0+00:00:33 H  0  9.8  condor_dagman -f -

1 jobs; 0 idle, 0 running, 1 held
Exercise 6, continued

% condor_release 1895
Cluster 1895 released.

% condor_q --sub train15
-- Submitter: viz-login.isi.edu : <128.9.72.178:43684> : viz-login.isi.edu
  ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
1895.0   train15         5/31 11:58   0+00:00:45 R  0   9.8  condor_dagman -f -

% more ex6.dag.dagman.out
5/31 11:59:38 Number of pre-completed nodes: 0
5/31 11:59:38 Running in RECOVERY mode...
5/31 11:59:38 Event: ULOG_SUBMIT for Condor Node Setup (1896.0)
5/31 11:59:38 Number of idle job procs: 1
5/31 11:59:38 Event: ULOG_EXECUTE for Condor Node Setup (1896.0)
5/31 11:59:38 Number of idle job procs: 0
5/31 11:59:38 Event: ULOG_JOB_TERMINATED for Condor Node Setup (1896.0)
5/31 11:59:38 Node Setup job proc (1896.0) completed successfully.
5/31 11:59:38 Node Setup job completed
5/31 11:59:38 Number of idle job procs: 0
5/31 11:59:38 ------------------------------
5/31 11:59:38      Condor Recovery Complete
5/31 11:59:38 ------------------------------...
Exercise 6, continued

```
% tail ex6.dag.dagman.out

5/31 12:01:25 ERROR: the following job(s) failed:
5/31 12:01:25 ---------------------- Job ----------------------
5/31 12:01:25       Node Name: Proc
5/31 12:01:25          NodeID: 1
5/31 12:01:25     Node Status: STATUS_ERROR
5/31 12:01:25 Node return val: 1
5/31 12:01:25       Error: Job exited with status 1 and POST Script failed with status 1 (after 2 node retries)
5/31 12:01:25 Job Submit File: proc.submit
5/31 12:01:25     PRE Script: pre_script $JOB
5/31 12:01:25     POST Script: post_script $JOB $RETURN
5/31 12:01:25       Retry: 2
5/31 12:01:25     Condor Job ID: (1899)
5/31 12:01:25     Q_PARENTS: 0, <END>
5/31 12:01:25     Q_WAITING: <END>
5/31 12:01:25     Q_CHILDREN: <END>
5/31 12:01:25 --------------------------------------- <END>

...
What we’re skipping

- Nested DAGs
- Multiple DAGs per DAGMan instance
- Stork
- DAG abort
- Visualizing DAGs with *dot*
- See the DAGMan manual section online!
Pegasus-general
Mapping workflows onto the Grid
Outline of Tutorial

- Introduction (15 mins)
- DAGMan (1 hr)
  - Overview
  - Hands On
- Break (20 mins)
- Pegasus (1 hr 45 mins)
  - Overview
  - Hands On
- Summary (10 mins)
DAGMan Functionality

- DAGMan workflow description--DAG
  - “workflow assembly language”
- DAGMan
  - “workflow executor”
- Need a higher level language to provide:
  - Portability
  - Ease of programming
  - Expose opportunities for optimization and improved reliability
Describing Abstract Workflows

- Pegasus workflow description—DAX
  - workflow “high-level language”
  - devoid of resource descriptions
  - devoid of data locations

- Pegasus
  - a workflow “compiler”
  - target language---DAGMan’s DAG and Condor submit files
  - transforms the workflow for performance and reliability
Bringing a workflow to execution

Abstract Workflow

Workflow Mapping
Pegasus

Executable Workflow

Workflow Execution
Condor DAGMan

Workflow Tasks

Task Execution at
Remote Site
Condor -G/Globus
Supported Abstract Workflow Generation tools:

Custom Scripts Portal Interfaces Virtual Data Language Wings Triana GUI (prototype)

Abstract Workflow (Resource-independent)

Executable Workflow (Resources Identified)

Ready Tasks

LOCAL SUBMIT HOST (Community resource)

Resource Information and Data Location Information

NMI: Globus MDS, RLS, SRB

Job monitoring Information

NMI: Condor

NMI: GridFTP

HTTP

Storage

National CyberInfrastructure
Execution Environment

Globus and Condor Services for job scheduling
Globus Services for data transfer and Cataloging

Information Services:
--- information about data location
--- information about the execution sites
Pegasus: Planning for Execution in Grids

- Maps from a workflow instance to an executable workflow
- Automatically locates physical locations for both workflow components and data
- Finds appropriate resources to execute the components
- Reuses existing data products where applicable
- Publishes newly derived data products
  - Provides provenance information
Basic Workflow Mapping

● Select where to run the computations
  ● Change task nodes into nodes with executable descriptions
    ● Execution location
    ● Environment variables initializes
    ● Appropriate command-line parameters set

● Select which data to access
  ● Add stage-in nodes to move data to computations
  ● Add stage-out nodes to transfer data out of remote sites to storage
  ● Add data transfer nodes between computation nodes that execute on different resources
Basic Workflow Mapping

- Add nodes that register the newly-created data products
- Add nodes to create an execution directory on a remote site
- Write out the workflow in a form understandable by a workflow engine
  - Include provenance capture steps
Information Components used by Pegasus

- Pegasus maintains interfaces to support a variety of information sources

**Information about resources**
- Globus Monitoring and Discovery Service (MDS)
  - Finds resource properties
  - Dynamic: load, queue length
  - Static: location of GridFTP server, RLS, etc
- VORS - to discover information about Open Science Grid

**Information about data location**
- Globus Replica Location Service
  - Locates data that may be replicated
  - Registers new data products

**Information about executables**
- Transformation Catalog
Original workflow: 15 compute nodes devoid of resource assignment

Resulting workflow mapped onto 3 Grid sites:

- 13 data stage-in nodes
- 11 compute nodes (4 reduced based on available intermediate data)
- 8 inter-site data transfers
- 14 data stage-out nodes to long-term storage
- 14 data registration nodes (data cataloging)

60 tasks
Outline of Tutorial

- Introduction (15 mins)
- DAGMan (1 hr)
  - Overview
  - Hands On
- Break (20 mins)
- Pegasus (1 hr 45 mins)
  - Overview
  - Hands On
- Summary (10 mins)
Pegasus-hands on

- Describing workflows at a higher-level of abstraction
- Catalogs needed to support the mapping
  - Transformation Catalog
  - Replica Catalog
  - Site Catalog
- Running end-to-end workflows
- Debugging Workflows
- Pegasus Advanced Features and Optimizations
Outline

- By now, you know how to run a workflow on the Grid using Condor DAGMan
- What are we going to do?
  - Step a level higher
  - Describe your computation in abstract terms (DAX)
  - Setup your catalogs so that we can discover data, codes, and resource during planning
  - plan, run, monitor and debug workflows
High-level system view

<table>
<thead>
<tr>
<th>Portals</th>
<th>Scripts</th>
<th>VDL</th>
<th>Wings</th>
<th>Kepler</th>
<th>Triana</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract Workflow Description</td>
<td>Devoid of resource specifications</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Pegasus

Executable Workflow Includes computations on specific resources and data management steps

DAGMan/Condor-G

Tasks are being dispatched to remote resources

NMI software (Condor, Globus)

TeraGrid, Open Science Grid, Condor Pools

National CyberInfrastructure
Pegasus workflow

- DAX
  - What it describes
  - How to read a DAX
  - How to generate a DAX
    - Describe the various methods
      - Direct XML
      - Wings
      - DAX API
      - Behind portals
    - Migrating from a DAG to DAX
Abstract Workflow (DAX)

Exercise: 2.1

- Pegasus workflow description—DAX
  - workflow “high-level language”
  - devoid of resource descriptions
  - devoid of data locations
  - refers to codes as logical transformations
  - refers to data as logical files
Understanding DAX (1)

<!-- part 1: list of all files used (may be empty) -->
<filename file="f.input" link="input"/>
<filename file="f.intermediate" link="input"/>
<filename file="f.output" link="output"/>

<!-- part 2: definition of all jobs (at least one) -->
<job id="ID000001" namespace="pegasus" name="preprocess" version="1.0">
  <argument>-a top -T 6 -i <filename file="f.input"/> -o <filename file="f.intermediate"/>
</argument>
<uses file="f.input" link="input" dontRegister="false" dontTransfer="false"/>
<uses file="f.intermediate" link="output" dontRegister="true" dontTransfer="true"/>
</job>

<job id="ID000002" namespace="pegasus" name="analyze" version="1.0">
  <argument>-a top -T 6 -i <filename file="f.intermediate"/> -o <filename file="f.output"/>
</argument>
<uses file="f.input" link="input" dontRegister="false" dontTransfer="false"/>
<uses file="f.intermediate" link="output" dontRegister="false" dontTransfer="false"/>
</job>

<!-- part 3: list of control-flow dependencies (empty for single jobs) -->
<child ref="ID000002">
  <parent ref="ID000001"/>
</child>
(excerpted for display)
Creating Workflow Template with Wings GUI
### Comparison of a DAG and DAX

<table>
<thead>
<tr>
<th><strong>DAX</strong></th>
<th><strong>DAG</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Single XML File</td>
<td>One dag file and multiple submit files</td>
</tr>
<tr>
<td>Describes your workflow at a logical level</td>
<td>Describes your workflow in terms of physical files and paths</td>
</tr>
<tr>
<td>Site Independent</td>
<td>Site Specific</td>
</tr>
<tr>
<td>Captures just the computation that the user (you) want to do</td>
<td>Has auxiliary jobs like data movement etc.</td>
</tr>
</tbody>
</table>
Outline

- Describing workflows at a higher-level of abstraction
- Catalogs needed to support the mapping
  - Replica Catalog
  - Transformation Catalog
  - Site Catalog
- Running end-to-end workflows
- Debugging Workflows
- Pegasus Advanced Features and Optimizations
Replica Catalog
Overview—finding data

- Replica Catalog stores mappings between logical files and their target locations.
- Used to
  - discover input files for the workflow
  - track data products created
  - data reuse
- Data is replicated for scalability, reliability and availability
Replica Catalog

- Pegasus interfaces with a variety of replica catalogs
  - File based Replica Catalog
    - useful for small datasets (like this tutorial)
    - cannot be shared across users.
  - Database based Replica Catalog
    - useful for medium sized datasets.
    - can be used across users.
  - Globus Replica Location Service
    - useful for large scale data sets across multiple users.
    - LIGO’s LDR deployment.
The rc-client is a command line tool to interact with Replica Catalog.

- One client talks to all types of Replica Catalog

Practical exercise (Refer Exercise 2.2):

- Use the rc-client to
  - Populate the Replica Catalog
  - Query the Replica Catalog
  - Remove entries (Offline exercise)
Site Catalog—finding resources

- Contains information about various sites on which workflows may execute.
- For each site following information is stored
  - Installed job-managers for different types of schedulers
  - Installed GridFTP servers
  - Local Replica Catalogs where data residing in that site has to be catalogued
  - Site Wide Profiles like environment variables
  - Work and storage directories
Site Catalog Exercise (Ex 2.3 10 minutes)

- Two clients for generating a site catalog
  - pegasus-get-sites
    - Allows you to generate a site catalog
      - for OSG grid sites by querying VORS
      - for ISI skynet, Teragrid, UC SofaGrid by querying a SQLite database
  - sc-client
    - Allows you to generate a site catalog
      - By specifying information about a site in a textual format in a file.
      - One file per site
Site Catalog Entry

<site handle="isi_skynet" sysinfo="INTEL32::LINUX"
      gridlaunch="/nfs/software/vds/vds/bin/kickstart">

  <profile namespace="env" key="PEGASUS_HOME">/nfs/software/pegasus</profile>
  <lrc url="rlsn://smarty.isi.edu"/>
  <gridftp url="gsiftp://skynet-data.isi.edu" storage="/nfs/storage01" major="2" minor="4" patch="3" />
  <gridftp url="gsiftp://skynet-2.isi.edu" storage="/nfs/storage01" major="2" minor="4" patch="3" />
  <jobmanager universe="vanilla" url="skynet-login.isi.edu/jobmanager-pbs" major="2" minor="4" patch="3" total-nodes="93" />
  <jobmanager universe="transfer" url="skynet-login.isi.edu/jobmanager-fork" major="2" minor="4" patch="3" total-nodes="93" />
  <workdirectory>/nfs/scratch01</workdirectory>

</site>
Transformation Catalog ---- finding codes

- Transformation Catalog maps logical transformations to their physical locations

- Used to
  - discover application codes installed on the grid sites
  - discover statically compiled codes, that can be deployed at grid sites on demand
Transformation Catalog Overview

- For each transformation following are stored
  - logical name of the transformation
  - Type of transformation (INSTALLED or STATIC_BINARY)
  - Architecture, OS, Glibc version
  - the resource on which the transformation is available
  - the URL for the physical transformation
  - Profiles that associate runtime parameters like environment variables, scheduler related information
Transformation Catalog Exercise (Offline)

- tc-client is a command line client that is primarily used to configure the database TC.
- Works even for file based transformation catalog.
Outline

- Describing workflows at a higher-level of abstraction
- Catalogs needed to support the mapping
  - Transformation Catalog
  - Replica Catalog
  - Site Catalog
- Running end-to-end workflows
- Debugging Workflows
- Pegasus Advanced Features and Optimizations
Component Configuration

- Component Configuration using Properties File
- Most of the configuration of Pegasus is done by properties.

- Properties can be specified
  - On the command line
  - In $HOME/.pegasusrc file
  - In $PEGASUS_HOME/etc/properties

- All properties are described in
  $PEGASUS_HOME/doc/properties.pdf

- For the tutorial the properties are configured in the
  $HOME/tutorial/config/properties file
Plan First (pegasus-plan)
Exercise: 2.4

- Takes in the abstract workflow (DAX) and generates the executable workflow in terms condor submit files.
Executable Workflow

1) Creates a unique work directory (RDy) per workflow on the Shared Storage System of Site Y

2) Stages in input data for job from remote storage system at Site Y to the work directory (RDy)

3) Job is executed on one of the worker nodes in the directory (RDy) on the Shared Storage System

4) Stages out Data from work directory (RDy) on the shared storage system to storage system on Grid Site Z

5) Removes the directory (RDy) on the shared storage system of Site Y

- **Legend**: Visibility / Accessibility
  - Compute Job
  - Stage-in Job
  - Stage-Out Job
  - Cleanup Job
  - Make Dir Job

Execution of jobs on storage systems shared between the worker nodes and headnode / storage element.
Run (pegasus-run)  
Exercise: 2.4 (cont.)

- Submits the workflow to Condor DAGMAN/ CondorG for remote job submissions

- Starts the monitoring daemon (tailstatd) in the directory containing the condor submit files

- Tailstatd parses the condor output and updates the status of the workflow to a database

- Tailstatd updates job status to a text file jobstate.log in the directory containing the condor submit files.
(Monitor) Pegasus-status
Exercise: 2.5

- A perl wrapper around condor_q
- Allows you to see only the jobs of a particular workflow
- Also can see what different type of jobs that are executing
pegasus-remove

- Remove your workflow and associated jobs
- In future, would cleanup the remote directories that are created during workflow execution.
Data Reuse

Sometimes it is cheaper to access the data than to regenerate it.

Keeping track of data as it is generated supports workflow-level checkpointing.

Files C and B 2 are available.

Reduced Workflow
File cleanup

- **Problem:** Running out of space on shared scratch
  - In OSG scratch space is limited to 30Gb for all users
- **Why does it occur**
  - Workflows bring in huge amounts of data
  - Data is generated during workflow execution
  - Users don’t worry about cleaning up after they are done
- **Solution**
  - Do cleanup after workflows finish
    - Does not work as the scratch may get filled much before during execution.
  - Interleave cleanup automatically during workflow execution.
    - Requires an analysis of the workflow to determine, when a file is no longer required.
Storage Improvement for Montage Workflows

Montage 1 degree workflow run with cleanup on OSG-PSU
Running using different styles

- Need to specify pegasus namespace profile keys with the sites in the site catalog.

- Submitting directly to condor pool
  - The submit host is a part of a local condor pool
  - Bypasses CondorG submissions avoiding Condor/GRAM delays.

- Using Condor GlideIn
  - User glides in nodes from a remote grid site to his local pool
  - Condor is deployed dynamically on glided in nodes for e.g. you glide in nodes from the teragrid site running PBS.
  - Only have to wait in the remote queue once when gliding in nodes.
Outline

- Describing workflows at a higher-level of abstraction
- Catalogs needed to support the mapping
  - Transformation Catalog
  - Replica Catalog
  - Site Catalog
- Running end-to-end workflows
- Debugging Workflows
- Pegasus Advanced Features and Optimizations
Debugging (Exercise: 2.5)

- The status of the workflow can be determined by
  - Looking at the jobstate.log
  - Or looking at the dagman out file (with suffix .dag.dagman.out)

- All jobs in Pegasus are launched by a wrapper executable kickstart. Kickstart generates provenance information including the exit code, and part of the remote application's stdout.

- In case of job failure look at kickstart output of the failed job.
Outline

- Describing workflows at a higher-level of abstraction
- Catalogs needed to support the mapping
  - Transformation Catalog
  - Replica Catalog
  - Site Catalog
- Running end-to-end workflows
- Debugging Workflows
- Pegasus Advanced Features and Optimizations
Workflow Restructuring to improve Application Performance

- Cluster small running jobs together to achieve better performance.

- Why?
  - Each job has scheduling overhead
  - Need to make this overhead worthwhile.
  - Ideally users should run a job on the grid that takes at least 10 minutes to execute
Job clustering

Vertical clustering

Level-based clustering

Arbitrary clustering

Useful for small granularity jobs

Ewa Deelman, deelman@isi.edu

www.isi.edu/~deelman

pegasus.isi.edu
Optional clustering exercise (Exercise 2.7)

- To trigger specify `--cluster horizontal` option to `pegasus-plan`

- The granularity of clustering configured via `pegasus profile key bundle`
  - Can be specified with a transformation in the transformation catalog, or with sites in the site catalog
  - Pegasus profile `bundle` specified in the site catalog.
  - Bundle means how many clustered jobs for that transformation you need on a particular site.
Transfer of Executables

- Allows the user to dynamically deploy scientific code on remote sites
- Makes for easier debugging of scientific code.
- The executables are transferred as part of the workflow
- Currently, only statically compiled executables can be transferred
- Also we transfer any dependant executables that maybe required. In your workflow, the mDiffFit job is dependant on mDiff and mFitplane executables
All the workflows that you ran had staging of executables

In your transformation catalog, the entries were marked as STATIC_BINARY on site “local”

Selection of what executable to transfer
- pegasus.transformation.mapper property
- pegasus.transformation.selector property
Transfer Throttling

- Large Sized Workflows result in large number of transfer jobs being executed at once. Results in:
  - Grid FTP server overload (connection refused errors etc)
  - May result in a high load on the head node if transfers are not configured for being executed as third party transfers

- Need to throttle transfers
  - Set pegasus.transfer.refiner property.
  - Allows you to create chained transfer jobs or bundles of transfer jobs
  - Looks in your site catalog for pegasus profile "bundle.stagein"
Managing execution environment changes through partitioning

Original Abstract Workflow

A Particular Partitioning

New Abstract Workflow
**Resulting Meta-Workflow**

**Pegasus**: Pegasus generates the concrete workflow and the submit files for Park in Man X = Sub(X)

**DAGMan(Sub(X))**: DAGMan executes the concrete workflow for X.
Workflow-level checkpointing
Outline of Tutorial

- Introduction (15 mins)
- DAGMan (1 hr)
  - Overview
  - Hands On
- Break (20 mins)
- Pegasus (1 hr 45 mins)
  - Overview
  - Hands On
- Summary (10 mins)
What do Pegasus & DAGMan do for an application?

- Provide a level of abstraction above gridftp, condor-submit, globus-job-run, etc commands
- Provide automated mapping and execution of workflow applications onto distributed resources
- Manage data files, can store and catalog intermediate and final data products
- Improve successful application execution
- Improve application performance
- Provide provenance tracking capabilities
- Provides a Grid-aware workflow management tool
Pegasus Summary (1)

- Data Management within the workflow
  - Interfaces with the Replica Location Service to discover data
  - Does replica selection to select replica.
  - Manages data transfer by interfacing to various transfer services like RFT, Stork and clients like globus-url-copy.
  - No need to stage-in data before hand. We do it within the workflow as and when it is required.
  - Reduced Storage footprint. Data is also cleaned as the workflow progresses.

- Data Reuse
  - Avoids duplicate computations
  - Can reuse data that has been generated earlier.
Pegasus Summary (2)

- Dynamic Deployment of Executable
  - Allows the user to dynamically deploy scientific code on remote sites
  - Makes for easier debugging of scientific code.
- Allows for Workflow Level Restructuring to improve application performance
- Reliability
  - Leverages DAGMan retry facilities to provide reliability
  - Throttles transfers for large workflows
DAGMan Summary

-Runs large, complex workflows
-DAG files are simple text files easily generated by automated tools (e.g., Pegasus)
-Provides retries, error handling, and throttling
-Highly configurable
-Used on DAGs up to ~500k nodes
-Reliable and mature software
-Users’ needs drive development
Applications using Pegasus and DAGMan
Laser Gravitational Wave Observatory

LIGO’s Binary Inspiral Analysis

- Based on the precise comparison of known waveforms from the final moments of orbital evolution in a system of two neutron stars or black holes

- Pegasus and DAGMan runs large-scale LIGO workflows on the Open Science Grid
- A month of LIGO data requires many thousands of jobs, running for days on hundreds of CPUs

LIGO OSG effort is led by Kent Blackburn and David Meyers (Caltech)
Eleven major projects and surveys worldwide, such as the Spitzer Space Telescope Legacy teams, have integrated Montage into their pipelines and processing environments to generate science and browse products for dissemination to the astronomy community.

Montage is a collaboration between IPAC, JPL, and CACR.


Pegasus maps workflows with thousands of tasks onto NSF’s TeraGrid.

Pegasus improved overall runtime by 90% through automatic workflow restructuring and minimizing execution overhead.

Eleven major projects and surveys worldwide, such as the Spitzer Space Telescope Legacy teams, have integrated Montage into their pipelines and processing environments to generate science and browse products for dissemination to the astronomy community.
Southern California Earthquake Center (SCEC)

- SCEC’s Cybershake is used to create Hazard Maps that specify the maximum shaking expected over a long period of time
- Used by civil engineers to determine building design tolerances

Pegasus mapped SCEC CyberShake workflows onto the TeraGrid in Fall 2005. The workflows ran over a period of 23 days and processed 20TB of data using 1.8 CPU Years. Total tasks in all workflows: 261,823.

**CyberShake Science result:** CyberShake delivers new insights into how rupture directivity and sedimentary basin effects contribute to the shaking experienced at different geographic locations. As a result more accurate hazard maps can be created.

*SCEC is led by Tom Jordan, USC*
BLAST runs at BioMagResBank using DAGMan

- BLAST (Basic Local Alignment Search Tool): rapid searching of protein & nucleotide databases
- Compares novel sequences with previously-characterized genes
- DAG consists of:
  - Main BLAST runs (~200 jobs of 20 BMRB sequences vs Non-Redundant database)
  - Post-processing (parsing) jobs in Java (again ~200)
- Completes within 2 days on ~20 nodes
- Runs weekly (databases are updated)
Current and Future Research

- Resource selection
- Resource provisioning
- Workflow restructuring
- Adaptive computing
  - Workflow refinement adapts to changing execution environment
- Workflow provenance (including provenance of the mapping process)
- Management and optimization across multiple workflows
- Workflow debugging
- Streaming data workflows
- Automated guidance for workflow restructuring
- Support for long-lived and recurrent workflows
Relevant Links

- Pegasus: pegasus.isi.edu
- DAGMan: www.cs.wisc.edu/condor/dagman

- Tutorial materials available at: http://pegasus.isi.edu/tutorial/tg07

- For more questions: pegasus@isi.edu
Relevant Links

- NSF Workshop on Challenges of Scientific Workflows: [www.isi.edu/nsf-workflows06](http://www.isi.edu/nsf-workflows06), E. Deelman and Y. Gil (chairs)

- Open Science Grid: [www.opensciencegrid.org](http://www.opensciencegrid.org)
- LIGO: [www.ligo.caltech.edu/](http://www.ligo.caltech.edu/)
- SCEC: [www.scec.org](http://www.scec.org)
- Montage: [montage.ipac.caltech.edu/](http://montage.ipac.caltech.edu/)
- Condor: [www.cs.wisc.edu/condor/](http://www.cs.wisc.edu/condor/)
- Globus: [www.globus.org](http://www.globus.org)
- TeraGrid: [www.teragrid.org](http://www.teragrid.org)