Managing HTC workflows with Pegasus

OSG Campus Infrastructures Community Webinar 4/26/13

Mats Rynge
USC Information Sciences Institute
Outline

- **Overview**
  - What is Pegasus?
  - Components of a Pegasus workflow
    - Abstract workflow
    - Replica, transformation and site catalogs
  - Common workflow transformations
  - Debugging and statistics

- **Demo**
  - Our first workflow
  - Failure / debugging
  - OSG-XSEDE example
  - Task clustering
  - Data management
Pegasus Workflow Management System

- NSF funded project and developed since 2001 as a collaboration between USC Information Sciences Institute and the Condor Team at UW Madison

- Builds on top of Condor DAGMan.

- **Abstract Workflows - Pegasus input workflow description**
  - Workflow “high-level language”
  - Only identifies the computation, devoid of resource descriptions, devoid of data locations

- **Pegasus is a workflow planner/mapper (“compiler”)**
  - Target is DAGMan DAGs and Condor submit files
  - Transforms the workflow for performance and reliability
  - Automatically locates physical locations for both workflow components and data
  - Collects runtime provenance
Workflows can be simple
Pegasus WMS

API Interfaces
- Python, Java, Perl

Portals
- hubzero

Other Workflow Composition Tools: Grayson, Triana, Wings

Users

Pegasus WMS
- Mapper
- Engine
- Scheduler

Notifications
- Monitoring
- Logs

Workflow DB

Clouds
- Cloudware
  - OpenStack
  - Eucalyptus, Nimbus
- Compute
  - Amazon EC2, RackSpace, FutureGrid
- Storage
  - S3

Distributed Resources
- Campus Clusters, Local Clusters, Open Science Grid, XSEDE

Middleware
- GRAM
- CONDOR
- PBS
- LSF
- SGE

Compute

Storage
- GridFTP
- HTTP
- FTP
- SRM
- IRODS
- SCP
Abstract to Executable Workflow Mapping

- Abstraction provides
  - Ease of Use (do not need to worry about low-level execution details)
  - Portability (can use the same workflow description to run on a number of resources and/or across them)
  - Gives opportunities for optimization and fault tolerance
    - automatically restructure the workflow
    - automatically provide fault recovery (retry, choose different resource)
Catalogs

- **Site catalog**
  - Defines the execution environment and potential data staging resources
  - Simple in the case of Condor pool, but can be more complex when running on grid resources

- **Transformation catalog**
  - Defines executables used by the workflow
  - Executables can be installed in different locations at different sites

- **Replica catalog**
  - Locations of existing data products – input files and intermediate files from previous runs
Supported Data Staging Approaches

- **NonShared filesystem setup using an existing storage element for staging (typical of OSG and campus Condor pools)**
  - Worker nodes don’t share a filesystem.
  - Data is pulled from / pushed to the existing storage element.
  - (Pictured on the next slide)

- **Condor IO**
  - Worker nodes don’t share a filesystem
  - Data is pulled from / pushed to the submit host via Condor file transfers

- **Shared Filesystem setup (typical of XSEDE and HPC sites)**
  - Worker nodes and the head node have a shared filesystem, usually a parallel filesystem with great I/O characteristics
  - Can leverage symlinking against existing datasets
Data Flow for Pegasus Workflows on OSG with GlideinWMS and Staging Storage Element

**Abstract Workflow**

1. **Workflow Setup Job**
   - Pegasus Planner
   - Head Node

2. **Workflow Stagein Job**
   - Executable Workflow
   - Data Stagein Job

3. **Workflow Stageout Job**
   - Data Stageout Job
   - Workflow Cleanup Job

4. **Directory Setup Job**
   - Condor DAGMan
   - Condor Queue

5. **Condor DAGMan**
   - Condor Queue

---

**STAGING STORAGE ELEMENT**

- Supports independent protocols for the get and put interfaces
- Protocols Supported:
  - SRM
  - GridFTP
  - HTTP
  - IRODS
  - S3
  - SCP

---

**EXECUTABLES**

- Workflow Setup
- Workflow Stagein
- Workflow Stageout
- Data Cleanup

---

**LEGEND**

- Directory Setup Job
- Data Stagein Job
- Data Stageout Job
- Directory Cleanup Job
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 10/30/60/? minutes to execute
- Clustered tasks can reuse common input data – less data transfers

Level-based clustering

Level-based clustering
Workflow Reduction (Data Reuse)

Abstract Workflow

File f.d exists somewhere. Use it. Mark Jobs D and B to delete

Delete Job D and Job B
Workflow Monitoring - Stampede

- **Leverage Stampede Monitoring framework with DB backend**
  - Populates data at runtime. A background daemon monitors the logs files and populates information about the workflow to a database
  - Stores workflow structure, and runtime stats for each task.

- **Tools for querying the monitoring framework**
  - `pegasus-status`
    - Status of the workflow
  - `pegasus-statistics`
    - Detailed statistics about your finished workflow
  - `pegasus-plots`
    - Visualization of your workflow execution

<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>135002</td>
<td>0</td>
<td>0</td>
<td>135002</td>
<td>0</td>
<td>135002</td>
</tr>
<tr>
<td>Jobs</td>
<td>4529</td>
<td>0</td>
<td>0</td>
<td>4529</td>
<td>0</td>
<td>4529</td>
</tr>
<tr>
<td>Sub-Workflows</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Workflow wall time : 13 hrs, 2 mins, (46973 secs)
Workflow cumulative job wall time : 384 days, 5 hrs, (33195705 secs)
Cumulative job walltime as seen from submit side : 384 days, 18 hrs, (33243709 secs)
Workflow Monitoring - Stampede

Hosts Over Time – Distribution of Different Job Types on Hosts

Workflow Gantt Chart

Jobs and Runtime over Time
Workflow Debugging Through Pegasus

- After a workflow has completed, we can run pegasus-analyzer to analyze the workflow and provide a summary of the run

- pegasus-analyzer's output contains
  - a brief summary section
    - showing how many jobs have succeeded
    - and how many have failed.
  - For each failed job
    - showing its last known state
    - exitcode
    - working directory
    - the location of its submit, output, and error files.
    - any stdout and stderr from the job.
Relevant Links

- **Pegasus:** [http://pegasus.isi.edu](http://pegasus.isi.edu)

- **Tutorial and documentation:** [http://pegasus.isi.edu/wms/docs/latest/](http://pegasus.isi.edu/wms/docs/latest/)