Workflow Tools

International HPC Summer School
June 26, 2015

Gideon Juve
gideon@isi.edu
Overview

- What are scientific workflows and why use them?
- Example workflow applications
- Overview of available workflow systems
- Introduction to Pegasus WMS
What are workflows?
Scientific Workflows

- Formal way to capture multi-step computations
- Relatively coarse grained
- Capture the steps and their parameters
- Define the input/output data of each step
- Describe dependencies between steps
Workflows can be simple!
Why Scientific Workflows?

- Automate complex, multi-stage processing pipelines
- Enable parallel, distributed computations
- Use existing code, no rewrites
- Simple to construct and modify
- Reusable, aid reproducibility
- Can be shared with others
- Record how data was produced (provenance)
- Handle failures with to provide reliability
- Keep track of data and files
Science-grade Mosaic of the Sky
Science-grade Mosaic of the Sky

Montage Workflow

Input → Reprojection → Background Rectification → Co-addition → Output

- Image1
  - Project → Diff → Fitplane
- Image2
  - Project → Diff → Fitplane
- Image3
  - Project → Diff → Fitplane

- Background
- BgModel
- Add

Montage Workflow: montage.ipac.caltech.edu

<table>
<thead>
<tr>
<th>Size of mosaic in degrees square</th>
<th>Number of input data files</th>
<th>Number of tasks</th>
<th>Number of intermediate files</th>
<th>Total data footprint</th>
<th>Cumulative wall time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>84</td>
<td>387</td>
<td>850</td>
<td>1.9 GB</td>
<td>21 mins</td>
</tr>
<tr>
<td>2</td>
<td>300</td>
<td>1442</td>
<td>3176</td>
<td>6.8 GB</td>
<td>54 mins</td>
</tr>
<tr>
<td>4</td>
<td>685</td>
<td>3738</td>
<td>8258</td>
<td>18 GB</td>
<td>3 hours, 18 mins</td>
</tr>
<tr>
<td>6</td>
<td>1461</td>
<td>7462</td>
<td>16458</td>
<td>37 GB</td>
<td>7 hours, 7 mins</td>
</tr>
<tr>
<td>8</td>
<td>2565</td>
<td>12757</td>
<td>28113</td>
<td>64 GB</td>
<td>11 hours, 44 mins</td>
</tr>
</tbody>
</table>
Bag of Tasks: Periodogram Workflow

- Kepler continuously monitors the brightness of over 175,000 stars
  - Search for periodic dips in signals as Earth-like planets transit in front of host star.
- For each star, Kepler data is used to create a “light curve”
- Need to perform a bulk analysis of all the data to search for these periodic signals

2012 Run at SDSC
- 1.1M tasks, 180 jobs
- 1.1M input, 12M output files
- ~101,000 CPU hours
- 16 TB output data

exoplanetarchive.ipac.caltech.edu
Workflows with MPI Codes: Neutron Scattering

- **Spallation Neutron Source at ORNL**
- **Parameter sweeps of MD and neutron scattering simulations**
  - Fit simulation to experimental data
  - e.g. temperature, charge, force
- **Nanodiamond Workflow**
  - Feb 2015 on Hopper using GRAM and GridFTP
  - 19 parameter values for nonbonded interactions between ND and H2O
  - 800 core NAMD jobs x 22 hrs
  - 400 core Sassena jobs x 3 hrs
  - ~380,000 CPU hours
  - ~1/2 TB output
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)

2014: 286 Sites, 4 models

- Each site = one workflow
- Each workflow has 420,000 tasks in 21 jobs using task clustering w/ PMC
- BlueWaters@NCSA, Stampede@TACC
Workflow Management Systems

- Automate execution of workflows

- Workflow language
  - Used to describe the workflow
  - Visual with GUI or text-based
  - Frequently based on DAGs, but some provide loops and branches or more exotic semantics

- Workflow engine
  - Manages the scheduling, submission, and monitoring of tasks
  - Orchestrates the movement of data
  - Interfaces with diverse cyberinfrastructure (grids, clusters, clouds)

- There are lots of workflow management systems
  - Some are abandoned research projects
Swift (swift-lang.org)

- Developed at the University of Chicago
- Workflow defined via parallel scripting language

```
//Create new type
type messagefile;
//Create app definition, returns messagefile
app (messagefile t) greeting() {
    //Print and pipe stdout to t
    echo "Hello, world!" stdout=filename(t);
}
//Create a new messagefile, linked to hello.txt
messagefile outfile <"hello.txt">
//Run greeting() and store results
outfile = greeting();
```

- Supports workflows with many tasks and large data
- Interfaces with many different cluster, grid and cloud infrastructures
Kepler (kepler-project.org)

- Developed by a diverse group of collaborators
- GUI-based
  - Composition and execution
  - View outputs
- Many different models of computation
  - Actor model with different execution semantics
- Interfaces with grids, clusters, and web services
- Component repository for sharing and lots of built-in components
Taverna (www.taverna.org.uk)

- Developed by a collaboration of UK universities
- GUI workflow composition
  - DAGs, loops, data parallel, merges
- Web services and local scripts/commands (mostly)
- Particularly good for bioinformatics
- Integrates with myExperiment for sharing workflows
- Leverages service catalogs for easy workflow composition
WS-PGRADE/gUSE (guse.hu)

- Developed at the Hungarian Academy of Sciences
- GUI interface for workflow composition
- Supports template DAGs for parameter sweep, WoW
- Integrated web portal/gateway
- Interfaces with many different infrastructures
- Extensive documentation
Other Workflow Systems

- **VisTrails** (vistrails.org)
  - Used for visualization pipelines with VTK

- **Galaxy** (galaxyproject.org)
  - Oriented toward biomedical research
  - Interfaces with many web services
  - Web-based GUI interface

- **UNICORE Workflow System** (unicore.eu)
  - GUI for workflow composition, or XML
  - Branches, loops, parallel loops

- **Makeflow** (ccl.cse.nd.edu/software)
  - Simple, make-like workflow language
  - Targets many different grid, cluster systems
Pegasus Workflow Management System

- Under development since 2001
- A collaboration between USC/ISI and the Condor Team at UW Madison
  - USC/ISI develops Pegasus
  - UW Madison develops DAGMan and Condor
- Actively used in a wide variety of domains
  - Earth science, physics, astronomy, bioinformatics, climate modeling, neutron science, and many others
  - About 600 workflows a day
Why Pegasus?

- Maps abstract workflows to diverse computing infrastructures
  - Desktop, Condor Pool, HPC Cluster, Grid, Cloud
- Supports large-scale, data-intensive workflows
  - $O(1M)$ tasks and $O(TB)$ of data
- Automatically plans and executes data transfers
- Manages failures to provide reliability
  - Retries and checkpointing
- Provides workflow monitoring and debugging tools to allow users to debug large workflows
- Technical support
  - full-time staff, mailing lists, public repository and bug tracker, regular releases, decent documentation
Pegasus Workflows

- Expressed as a DAG: nodes=tasks, edges=dependencies
- Tasks are command-line programs, executed as batch jobs
- Dependencies are usually data dependencies
- Data is exchanged via files

Setup

Split

Filter & Convert

Map

Merge

Analyze

Epigenomics Workflow
Pegasus WMS Environment

API Interfaces

- Python
- Java
- Perl

Portals

- hubzero

Other Workflow Composition Tools: Grayson, Triana, Wings

Users

Pegasus WMS

- Mapper
- Engine
- Scheduler

Monitoring

Workflow DB

Notifications

Logs

Clouds

- Cloudware
  - OpenStack, Eucalyptus, Nimbus

- Compute
  - Amazon EC2, RackSpace, FutureGrid

Storage

- S3

Distributed Resources

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

MIDDLEWARE

- GRAM
- PBS
- LSF
- SGE
- CONDOR

COMPUTE

- GridFTP
- HTTP
- FTP
- SRM
- IRDOS
- SCP

STORAGE
Pegasus WMS Data Flow

DAX

Site Catalog

Trans. Catalog

Replica Catalog

Config

Pegasus <Planner>

DAG

Submit Script

DAGMan <Engine>

Condor <Scheduler>
Workflow Planning (Mapping)

- Pegasus converts abstract workflow descriptions into executable workflows (similar to compiler)
  - Facilitates portability
  - Separates data management from workflow composition
  - Enables workflow-level optimizations
  - Others...

- Planning process:
  - Choose a site for each job (site selection)
  - Add resource-specific information
  - Choose input files (replica selection)
  - Plan data movements and add data management jobs
  - Perform optimizations
  - Add setup and cleanup jobs
  - Generate executable workflow artifacts
Abstract to Executable Workflow Mapping

Abstract Workflow

Executable Workflow

LEGEND
- Unmapped Job
- Compute Job mapped to a site
- Stage-in Job
- Stage-Out Job
- Registration Job
- Make Dir Job
- Cleanup Job
Data Management

- Input Data Site, Compute Site and Output Data Sites can be co-located
  - Example: Input data is already present on the compute site.

- Most of the tasks in scientific workflow applications require POSIX file semantics
  - Each task in the workflow opens one or more input files
  - Read or write a portion of it and then close the file.

- Data Staging Site can be the shared filesystem on the compute cluster!
Data Staging Configurations

Shared File System (typical of most HPC sites)
- Worker nodes and the head node have a shared filesystem, usually a parallel filesystem with high-performance I/O
- Can leverage symlinking against pre-staged datasets
- Staging site is the compute site

Non-shared File System (typical of OSG and EC2)
- Worker nodes don’t share a file system
- Uses a staging site separate from the compute site such as Amazon S3
- Data is pulled from / pushed to the staging site
- Also known as “PegasusLite”
Data Staging Configurations

Condor I/O (Typical of Condor Pools like OSG sites)
- Worker nodes don’t share a file system
- Data is pulled from / pushed to the submit host via Condor file transfers
- Staging site is the submit host

Using Pegasus allows you to move from one deployment to another without changing the workflow description

Many Data Protocols Supported:
- SCP
- HTTP
- FTP
- GridFTP
- Amazon S3
- iRODS
- SRM
- cp
- symlink
- FDT
- Google Storage
- StashCache
Workflow Monitoring and Reporting

- **Data collection**
  - Data extracted from log files and stored in a relational database
  - DB contains workflow structure, status information, runtimes, host info, task stdout/stderr

- **Reporting tools**
  - Status of the workflow
    - `pegasus-status path/to/submit/directory`
  - Detailed runtime statistics
    - `pegasus-statistics -s all path/to/submit/directory`

<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)
Pegasus Dashboard

- Web-based workflow monitoring GUI
  - Data comes from monitoring database
  - Supports monitoring, troubleshooting, and reporting
Failure Management

- Pegasus detects job failures
  - non-zero exit code
  - output does not contain a specified “success message”
  - output does contain a specified “failure message”
  - it exceeds a specified time limit
  - it fails to produce expected output files

- Job Retries
  - Helps with transient failures
  - Each job has a set number of retries per run

- Rescue DAGs
  - DAGMan writes a checkpoint file so workflow can be restarted
  - Can recover from almost any failure with minimal loss

- Checkpoint files
  - Job generates checkpoint files
  - Staging of checkpoint files is automatic on restarts
Workflow Debugging

- **Problem:** You have 1M tasks, and one of them fails
- **pegasus-analyzer:** Provides summary of workflow execution

- **Outputs**
  - 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
Task Clustering

- 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 jobs that take at least 10/30/60/? minutes
  - Clustered tasks can reuse common input data – less data transfers

Horizontal clustering

Label-based clustering

Also: time-based clustering
Pegasus-MPI-Cluster

- A master/worker task scheduler for running fine-grained workflows and ensembles on HPC systems
- Runs as an MPI job → Works on most HPC systems
- Allows sub-graphs of a Pegasus workflow to be submitted as monolithic jobs to remote resources
- Can be used on a sub-graph, or the entire workflow

```
Master (rank 0)

Worker (rank 1-N)

EDGE A B
EDGE A C

TASK A /bin/echo I am A
TASK B /bin/echo I am B
TASK C /bin/echo I am C
```

A
B
C
PMC Features

- **Fault Tolerance**
  - Retries at the task level (master resends task to another worker)
  - Retries at the workflow level (using a transaction log to record progress)

- **Resource-aware scheduling**
  - Many HPC machines have low memory/core
  - PMC can allocate memory and cores to a task, and force other slots on the same node to be idle

- **I/O Forwarding**
  - Small tasks == small I/O == poor performance
  - PMC reads data off of pipes from worker and forwards it using MPI messages to a central I/O process, which collects the data and writes it to disk
  - Writes are not interleaved, no locking required for synchronization
Resource Provisioning with Pilot Jobs

- Key idea: Use HPC scheduler to run application scheduler
- Parallel pilot jobs
- Amortize queue delays over many application jobs
- Apply application-specific policy
Data Cleanup

Problem: Workflow uses more disk space than quota
Solution: Add cleanup jobs to the workflow

Montage 1 degree workflow run with cleanup
Workflow Reduction (Data Reuse, Restarts)

Data reuse happens automatically when output files are found in the replica catalog.
Large-scale, Hierarchical Workflows

DAX A
A1
A2
A3
A4

DAX B
B1
B2
B3
B4

DAX C
C1
C2
C3
C4

DAX D
D1
D2
D3
D4

Compute Job
Pegasus Plan
And Execute Job

RECURSION ENDS WHEN DAX WITH ONLY COMPUTE JOBS IS ENCOUNTERED
Other Features

- **Job and Transfer Throttling**
  - Prevents too many jobs/transfers from overloading system

- **Notifications**
  - System calls a script when certain events occur: send email, text, etc.

- **Executable and Worker Package Staging**
  - Enables dynamic deployment of code on remote sites
  - Planner matches the executable in the TC to the site in the SC

- **Kickstart Job Wrapper**
  - Records detailed information about job execution (execution host, environment, memory usage, I/O, files accessed, CPU time, etc.)

- **Shell planner mode**
  - Generate a shell script of your workflow
Final Thoughts

- Probably using a workflow already
  - Replaces scripts, manual hand-offs and polling to monitor

- Automation is vital
  - Eliminate babysitting your jobs: your time is valuable!
  - Able to recover from failures without losing work

- Put ALL processing steps in the workflow
  - Include validation, visualization, data publishing, notifications

- Does add additional software layers and complexity
  - Some development time is required

- Choose workflow system carefully
  - Consider required features, target environment, maturity, support

- We want to help you!
Questions?
Some Computational Science Challenges

- Integrate several programs into one pipeline
- Run an ensemble of simulations
- Repeat processing steps on new data or parameters
- Reproduce previous results, or similar results
- Share analysis steps with other researchers
- Recreate the history of data products
- Run code on hundreds or thousands of inputs
- Execute analyses in parallel on distributed resources
- Reliably execute pipelines on unreliable infrastructure

**Scientific workflows can help with these problems**
Workflow Management System Functionality

- **Job execution**
  - Interfaces with middleware and batch systems to submit and monitor jobs

- **Data and control dependencies between jobs**
  - Tracks dependencies and makes sure jobs are executed in the right order

- **Scheduling**
  - Some jobs may be able to run in parallel with others
  - Ordering and placement can improve performance

- **Data management**
  - Transfers of input and output files to/from machine

- **Provenance**
  - Track when a job was run, where it was run, what data it produced, key parameters, metadata

- **Reliability**
  - Keeps track of what finished successfully, and what did not

- **Resource provisioning**
  - Allocating resources to run jobs
Example Workflow

- first_job
  - input.txt
    - simul_job
      - output.0.dat
    - simul_job
      - output.1.dat
    - simul_job
      - output.2.dat
    - simul_job
      - output.3.dat
    - simul_job
      - output.4.dat
  - tmp.txt
Example DAX Generator in Python

# Create DAX object
dax = ADAG("test_dax")
# Define first job
firstJob = Job(name="first_job")
# Input and output files to first job
firstInputFile = File("input.txt")
firstOutputFile = File("tmp.txt")
# Args to first_job (first_job input=input.txt output=tmp.txt)
firstJob.addArgument("input=input.txt", "output=tmp.txt")
# Role of the files for the job
firstJob.uses(firstInputFile, link=Link.INPUT)
firstJob.uses(firstOutputFile, link=Link.OUTPUT)
# Add the job to the workflow
dax.addJob(firstJob)
for i in range(0, 5):
    # Create simulation job
    simulJob = Job(id="%s" % (i+1), name="simul_job")
    # Define files
    simulInputFile = File("tmp.txt")
    simulOutputFile = File("output.%d.dat" % i)
    # Arguments to job
    # simulJob parameter=<i> input=tmp.txt output=output<i>.dat
    simulJob.addArgument("parameter=%d" % i, "input=tmp.txt",
                         "output=%s" % simulOutputFile.getName())
    # Role of files
    simulJob.uses(simulInputFile, link=Link.INPUT)
    simulJob.uses(simulOutputFile, line=Link.OUTPUT)
    # Add job to dax
    dax.addJob(simulJob)
    # Dependency on firstJob
    dax.depends(parent=firstJob, child=simulJob)
    # Write to file
    fp = open("test.dax", "w")
    dax.writeXML(fp)
    fp.close()
Site Catalog

- Stores details about each target execution/storage site
  - Job submission endpoints (GRAM URL, etc.)
  - Paths to storage/scratch directories
  - Data transfer services (GridFTP servers, etc.)
  - Paths to credentials (X509 proxy, ssh key, etc.)
  - Site-level configuration (environment variables, etc.)
  - “local” site is special—refers to submit host

<!-- Example site catalog -->
<sitecatalog>
  <site handle="example" arch="x86_64" os="LINUX">
    <grid type="gt5" contact="example.isi.edu/jobmanager-fork" jobtype="auxillary"/>
    <grid type="gt5" contact="example.isi.edu/jobmanager-pbs" jobtype="compute"/>
    <directory type="shared-scratch" path="/scratch">
      <file-server operation="all" url="gsiftp://example.isi.edu/scratch"/>
    </directory>
  </site>
</sitecatalog>
Transformation Catalog

- Maps transformations to executables on each site
  - Physical path or URL of executable and dependent data/configuration files
  - Executable characteristics (OS, architecture, glibc, etc.)
  - Job-level configuration (e.g. environment variables, profiles)

```plaintext
# Example transformation catalog
tr example::date {
  profile env "TZ" "America/Los_Angeles"

  site example {
    pfn "/bin/date"
    os "linux"
    arch "x86_64"
    type "INSTALLED"
  }
}
```
Replica Catalog

- Maps logical files to physical files
  - LFN (name) to PFN (path or URL)
  - Mappings annotated with metadata (e.g. site/pool, size, etc.)

- Enables Pegasus to choose “best” replica (replica selection phase of planner)

- Where Pegasus registers workflow output locations

- Support file-based or DB-based RC (also callout)

# Example replica catalog

```
f.1   gsiftp://example.isi.edu/inputs/f.1  pool="example"
f.1   file:///inputs/f.1                  pool="example"
f.2   file:///inputs/f.2                  pool="example"
f.2   file:///inputs/f.2                  pool="local"
```

“pool” == site
Configuration Properties and Profiles

- Specify all the tuning knobs for Pegasus
- Unification of properties and profiles several years ago
- Often in a “pegasus.properties” file (or command-line)
- Some are global and apply to all sites and jobs
- Some (profiles) can also be specified in the TC, SC and DAX with different scopes

Examples
- pegasus.data.configuration = sharedfs
- pegasus.style = condor
- dagman.retry = 2
- pegasus.exitcode.successmsg = “All data processed”
Data Management

- **Pegasus supports several different data configurations**
  - Many protocols
  - Complex data flows

- **Workflow file types**
  - Input
  - Intermediate
  - Output

- **Sites**
  - Local site: Pegasus WMS
  - Storage site: inputs and outputs
  - Staging site: intermediate
  - Compute site: compute jobs

Submit Host (Local Site)

Input Site

Staging Site

Compute Site

Output Site
Checkpoint Files

- A job can specify that it uses one or more checkpoint files
- Checkpoint files are both input files and output files
- Pegasus will stage-out these files in the case that job fails
  - Typically due to a timeout on long-running jobs
  - Jobs must periodically write checkpoint files (no signals are given)
- Pegasus will stage-in these files before retrying the job
  - They will appear in the working directory of the job
Workflow and Task Notifications

- Users want to be notified at certain points in the workflow or on certain events
- Support for adding notification to workflow and tasks
- Event based callouts
  - On Start, On End, On Failure, On Success
  - Examples contain email and jabber notification scripts
  - Can run any user provided scripts
  - Defined in the DAX
Pegasus clients for data management

- pegasus-transfer, pegasus-create-dir, pegasus-cleanup
- Support many different protocols
  - HTTP
  - SCP
  - GridFTP
  - IRODS
  - Amazon S3
  - SRM
  - cp
  - ln -s
- Remote directory creation and removal
- Support client discovery, parallel transfers, retries, and many other things to improve transfer performance and reliability
Different Directories used by Pegasus

1. **Submit Directory**
   - The directory where pegasus-plan generates the executable workflow i.e HTCondor DAGMan and job submit files.
   - Specified by `--dir` option to pegasus-plan

2. **Input Directory**
   - Mostly input file locations are catalogued in the Replica Catalog.
   - However, if inputs are on the submit host, then you can pass `--input-dir` option to pegasus-plan

3. **Scratch Directory**
   - Workflow specific directory created on the staging site by the `create-dir` job. This is where all the workflow inputs and outputs are gathered.
   - The base directory specified in the site catalog entry in `sites.xml` file.

4. **Output Directory**
   - The output directory where the outputs of the workflow appear.
   - Specified in the output site entry in the `sites.xml` file.
   - Can also be optionally specified by `--output-dir` option to pegasus-plan
Planning and Submitting workflows

- **pegasus-plan**
  - Interface to the Pegasus planner
  - Specify input dir
  - Specify compute site(s)
  - Specify staging site(s)
  - Specify output dir or output site

- **Pegasus-run**
  - Start and restart the workflow
Problems Workflows Solve

- **Task executions**
  - Workflow tools will retry and checkpoint if needed

- **Data management**
  - Stage-in and stage-out data
  - Ensure data is available for jobs automatically

- **Task scheduling**
  - Optimal execution on available resources

- **Metadata**
  - Automatically track runtime, environment, arguments, inputs

- **Getting cores**
  - Whether large parallel jobs or high throughput
Askalon (askalon.org)

- Developed at University of Innsbruck in Austria
- Create workflow description in AGWL (XML) or UML
  - if, for, parallelFor, DAGs
- Conversion: like planning, to bind to specific execution
- Submit jobs to Enactment Engine, which distributes jobs for execution at remote cluster, grid or cloud sites
- GUI for composition and monitoring
Example Hierarchical Workflow

- `<dax>` element behaves like `<job>`
  - Arguments are for pegasus-plan (most are inherited)
- Planner is invoked when DAX job is ready to run

```xml
<?xml version="1.0" encoding="UTF-8"?>
<adag version="3.4" name="multi-level">
  <job id="ID0000001" namespace="example" name="sleep">
    <argument>5</argument>
  </job>
  <dax id="ID0000002" file="sub.dax">
    <argument>--output-site local</argument>
  </dax>
  <job id="ID0000003" namespace="example" name="sleep">
    <argument>5</argument>
  </job>
  <child ref="ID0000002">
    <parent ref="ID0000001"/>
  </child>
  <child ref="ID0000003">
    <parent ref="ID0000002"/>
  </child>
</adag>
```
Integration with HUBzero

Credit: Frank McKenna UC Berkeley, NEES, HUBzero
Key Pegasus Concepts

- Workflows are DAGs (or hierarchical DAGs)
  - No loops, no conditional branches

- Pegasus WMS == Pegasus planner (mapper) + DAGMan workflow engine + Condor scheduler
  - The planner does not schedule jobs

- Planning occurs ahead of execution
  - (Except hierarchical workflows)

- Planning converts an abstract workflow into a concrete, executable workflow
  - Planner is like a compiler
Data-intensive Workflows

- Montage Galactic Plane Workflow
  - 18 million input images (~2.5 TB)
  - 900 output images (2.5 GB each, 2.4 TB total)
  - 10.5 million tasks (34,000 CPU hours)
  - Run on Amazon EC2 2013-2014

- Need to support hierarchical workflows and scale
Workflow Application: CyberShake

- What will peak ground motion be over the next 50 years?
  - Used in building codes, insurance, government, planning
  - Answered via Probabilistic Seismic Hazard Analysis (PSHA)
  - Communicated with hazard curves and maps

- Hazard curve for downtown LA: 2% in 50 years
- Probability of exceeding 0.1g in 50 yrs

Hazard curve for downtown LA

Probability of exceeding 0.1g in 50 yrs
Seismic Hazard Analysis Calculation

- Tensor simulation
  - Create 1.5 billion point mesh with material properties
  - Generate tensors across volume
  - Parallel, ~8,000 CPU-hrs
Post-Processing

- Individual earthquake contributions
  - Get list of earthquakes of interest (~415,000)
  - Simulate seismograms for each earthquake
  - Loosely-coupled, short-running serial jobs

- Combine the levels of shaking with probabilities to produce a hazard curve.
## Computational Requirements

<table>
<thead>
<tr>
<th>Component</th>
<th>Data</th>
<th>Executions</th>
<th>Cores/execute</th>
<th>Core hours</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Tensor Creation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mesh generation</td>
<td>15 GB</td>
<td>1</td>
<td>320</td>
<td>50</td>
</tr>
<tr>
<td>Tensor simulation</td>
<td>40 GB</td>
<td>2</td>
<td>10,000 CPU 100 GPU</td>
<td>16,000 CPU 1,200 GPU</td>
</tr>
<tr>
<td><strong>Post Processing</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tensor extraction</td>
<td>690 GB</td>
<td>6</td>
<td>256</td>
<td>275</td>
</tr>
<tr>
<td>Seismogram synthesis</td>
<td>12 GB</td>
<td>415,000</td>
<td>1</td>
<td>2,300</td>
</tr>
<tr>
<td>Curve generation</td>
<td>1 MB</td>
<td>1</td>
<td>1</td>
<td>&lt; 1</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>757 GB</td>
<td>415,000</td>
<td></td>
<td>18,625</td>
</tr>
</tbody>
</table>

This is for **one** location of interest; we wanted to run ~1000
Why Scientific Workflows?

- Large-scale, heterogeneous, high throughput
  - Parallel and many (~415,000) serial tasks
  - Task duration 100 ms – 2 hours
- Automation
- Data management
- Error recovery
- Resource provisioning
- Scalable
- System-independent description
CyberShake workflows

Tensor Workflow:
- Mesh generation
- Tensor simulation
- Tensor extraction

Post-Processing Workflow:
- Seismogram synthesis
- Hazard Curve

x1 -> x2
x6 -> x415,000

USC Viterbi
School of Engineering
Challenge: Resource Provisioning

- In tensor workflow, submit job to remote scheduler
  - GRAM puts jobs in remote queue
  - Runs like a normal batch job
  - Can specify either CPU or GPU nodes

- For post-processing workflow, need high throughput
  - Putting lots of jobs in the batch queue is ill-advised
    - Scheduler isn’t designed for heavy job load
    - Scheduler cycle is ~5 minutes
    - Policy limits too

- Solution: Pegasus-mpi-cluster (PMC)
Challenge: Data Management

- Millions of data files
  - Pegasus provides staging
    - symlinks files if possible, transfers files if needed
    - Supports running parts of workflows on separate machines
  - Transfers output back to local archival disk
  - Pegasus registers data products in catalog
  - Cleans up temporary files when no longer needed

- Directory hierarchy to reduce files per directory

- Added automated checks to check integrity
  - Correct number of files, NaN, zero-value checks
  - Included as new jobs in workflow
Challenge: File System Load

- **Seismogram tasks cause heavy I/O load**
  - Reads an earthquake description
  - Writes a seismogram file

- **Reduce reads**
  - Generate earthquake description on the fly, from geometry
  - Added memcached to cache rupture geometry
    - Local memory cache on compute node
    - Pegasus-mpi-cluster hook for custom startup script

- **Reduce writes**
  - Pegasus-mpi-cluster supports “pipe forwarding”
  - Workers write to pipes, master aggregates to fewer files
CyberShake Study 14.2

- Hazard curves for 1144 sites
- 46,720 CPUs + 225 GPUs for 14 days (Blue Waters)
  - Peak of 295,040 CPUs, 1100 GPUs
- 99.8 million tasks executed
  - 81 tasks/sec
  - Only 31,463 jobs in Blue Waters queue
- On average, 26.2 workflows running concurrently
- Managed 830 TB of data
  - 57 TB output files
  - 12.3 TB staged back to local disk (~16M files)
- Workflow tools scale!