Simplify Your Science with Workflow Tools

International HPC Summer School
June 26, 2013

Scott Callaghan
Southern California Earthquake Center
University of Southern California
scottcal@usc.edu
Southern California Earthquake Center

- Collaboration of 600+ scientists at 60+ institutions
- Focus on Southern California
- Earthquake system science
- Computational models of earthquake processes
  - Shaking from one earthquake (MPI codes)
  - Hazard at one location (high throughput)
For example
Overview

• What are scientific workflows?
• Available workflow tools
  – GUI tools
  – Scripting tools
• CyberShake (seismic hazard application)
  – Computational overview
  – Challenges and solutions
• Ways to simplify your work
• Goal: Help you figure out if this would be useful
Scientific Workflows

• Formal way to express a scientific calculation
• Multiple tasks with dependencies between them
• No limitations on tasks
  – Short or long
  – Loosely or tightly coupled
• Independence of workflow process and data
  – Often, run same workflow with different data
  – Workflow could be data-dependent
• You use workflows all the time…
#!/bin/bash

1) Stage-in input data to compute environment
   scp myself@datastore.com:/data/input.txt /scratch/input.txt

2) Run a serial job with an input and output
   bin/pre-processing in=input.txt out=tmp.txt

3) Run a parallel job with the resulting data
   mpiexec bin/parallel-job in=tmp.txt out_prefix=output

4) Run a set of independent serial jobs in parallel – scheduling by hand
   for i in `seq 0 $np`; do
      bin/integrity-check output.$i &
   done

5) While those are running, get metadata and run another serial job
   ts=`date +%s`
   bin/merge prefix=output out=output.$ts

6) Finally, stage results back to permanent storage
   scp /scratch/output.$ts myself@datastore.com:/data/output.$ts
Could think of shell script as a workflow

- input.txt
- pre-processing
- parallel-job
- merge
- date
- integrity-check
- output.*

stage-in

- input.txt
- tmp.txt
- output.*

stage-out

output.$ts
Workflow Components

- **Task executions**
  - Specify a series of tasks to run
- **Data and control dependencies between tasks**
  - Outputs from one task may be inputs for another
- **Task scheduling**
  - Some tasks may be able to run in parallel with other tasks
- **File and metadata management**
  - Track when a task was run, key parameters
- **Resource provisioning (getting cores)**
  - Computational resources are needed to run jobs on
What do we need help with?

• **Task executions**
  – What if something fails in the middle?
• **Data and control dependencies**
  – Make sure inputs are available for tasks
  – May have complicated dependencies
• **Task scheduling**
  – Minimize execution time while preserving dependencies
• **Metadata**
  – Automatically capture and track
• **Getting cores**
Types of Tools

• Workflow management systems take care of these concerns

• GUI-based (generally targeted at medium-scale)
  – Kepler
  – Taverna, Triana, VisTrails

• Scripting (generally more scalable, more complexity)
  – Pegasus (and Condor)
  – Swift

• Most tools are free and open source
• Not a complete list!
Kepler

- Developed by NSF-funded Kepler/CORE team (UCs)
- Actor and Director model
  - Actors = tasks
  - Director = controls execution of tasks
    - Serial, parallel, discrete time modeling
- Many built-in math and statistics modules
- Generally, execution on local machine
- Extensive documentation
Other Graphical Tools

• Taverna
  – Developed by myGrid (UK)
  – Runs workflow to minimize completion time
  – Commonly used in life science community

• Triana
  – Signal analysis, image manipulation

• VisTrails
  – Visualization
Scripting Tools

- Define workflow via programming
- Can support large numbers of tasks
- Provide many kinds of fancy features and capabilities
  - More flexibility
  - More complex
- Today, simple overview
- Will focus on Pegasus, but concepts are shared
Pegasus

- Developed at USC’s Information Sciences Institute
- Designed to address our earlier problems:
  - Task execution
  - Data and control dependencies
  - Data and metadata management
  - Error recovery
- Uses Condor DAGMan for
  - Task scheduling
  - Resource provisioning
Pegasus Concepts

• Separation of “submit host” and “execution site”
  – Create workflow using code on your local machine
  – Can run on local machine or on distributed resources

• Workflow represented with directed acyclic graphs

• You use API to write code describing workflow
  – Python, Java, Perl
  – Tasks with parent / child relationships
  – Files and their roles
  – Can have nested workflows

• Pegasus creates XML file of workflow called a DAX
Sample Workflow

```plaintext
input.txt

first_job

tmp.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
```
public static void main(String[] args) {
    //Create DAX object
    ADAG dax = new ADAG("test_dax");
    //Define first job
    Job firstJob = new Job("0", "my_namespace", "first_job", "v1.0");
    //Input and output files to first job
    File firstInputFile = new File("input.txt");
    File firstOutputFile = new File("tmp.txt");
    //Arguments to first_job (first_job input=input.txt output=tmp.txt)
    firstJob.addArgument("input=input.txt");
    firstJob.addArgument("output=tmp.txt");
    //Role of the files for the job
    firstJob.uses(firstInputFile, File.LINK.INPUT);
    firstJob.uses(firstOutputFile, File.LINK.OUTPUT);
    //Add the job to the workflow
    dax.addJob(firstJob);
for (int i=0; i<5; i++) {
    //Create simulation job
    Job simulJob = new Job(i+1, "my_namespace", "simul_job", "v2.1");
    //Define files
    File simulInputFile = new File("tmp.txt");
    File simulOutputFile = new File("output." + i + ".dat");
    //Arguments to job
    simulJob.addArgument("parameter=" + i);
    simulJob.addArgument("input=tmp.txt");
    simulJob.addArgument("output=" + simulOutputFile.getName());
    //Role of files
    simulJob.uses(simulInputFile, File.LINK.INPUT);
    simulJob.uses(simulOutputFile, File.LINK.OUTPUT);
    //Add job to dax
    dax.addJob(simulJob);
    //Dependency on firstJob
    dax.addDependency(firstJob, simulJob);
}
//Write to file
dax.writeToFile("test.dax");
Planning

• DAX is “abstract workflow”
  – Logical filenames and executables
  – Algorithm description

• Prepare workflow to execute on a certain system

• Use Pegasus to “plan” workflow
  – Uses catalogs to resolve logical names, compute info
  – Pegasus automatically augments workflow
    • Stages jobs (if needed) with GridFTP or Globus Online (soon)
    • Registers output files in a catalog to find later
    • Wraps jobs in pegasus-kickstart for detailed statistics
  – Generates a DAG
    • Top-level workflow description (tasks and dependencies)
    • Submission file for each job (Condor format)
Pegasus Workflow Path

Create workflow description (you write this)

Abstract workflow (DAX)
Logical names, algorithm

Concrete workflow (DAG)
Physical paths, job scripts

Planning

Running

Jobs Execute

Scheduler (Condor)
Running with HTCondor

• Developed by HTCondor group at U of Wisconsin

• Pegasus “submits” workflow to Condor DAGMan
  – Contains local queue of jobs
  – Monitors dependencies
  – Schedules jobs to resources
  – Automatically retries failed jobs
    • Writes rescue DAG to restart if job keeps failing
  – Updates status (jobs ready, complete, failed, etc.)

• Can run jobs locally or on remote system
  – Condor-G uses GRAM to submit jobs to remote scheduler
GRAM

• Part of the Globus Toolkit
• Uses certificate-based authentication
  – Like gsissh, GridFTP, Globus Online
  – Requires X509 certificate and account on remote machine
• Enables submission of jobs into a remote queue
• Supported by many university and XSEDE resources
  – Lonestar, Kraken, Stampede (for example)
Pegasus/Condor/GRAM stack

What you do:
Create workflow description

What the tools do:
Pegasus -> DAG

Local machine
Condor
remote queue
GRAM
Remote scheduler
Remote machine
Other Tools

• Swift
  – Similar, but workflow defined via scripting language

```swift
//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();
```

• Catalogs used to resolve executables and resources
• Workflow compiled internally and executed
• Which tool is better depends on the app and you
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

Probability of exceeding 0.1g in 50 yrs
How do you do PSHA?

1. Pick a location of interest.
2. Determine what future earthquakes might happen which could affect that location.
3. Estimate the magnitude and probability for each earthquake.
The computationally intensive part...

4. Determine the shaking caused by each earthquake at the site of interest.

Can calculate with ‘attenuation relationships’ – extrapolate from historical data. Very quick, but simple.

Alternatively, can use 3D physics-based simulation. Captures more complex phenomena.
Physics-based approach

• Wave propagation simulation
  – Create 1.5 billion point mesh with material properties
  – Generate Strain Green Tensors across volume
  – Parallel, ~12,000 CPU-hrs
Post-Processing

- Individual earthquake contributions
  - Use “seismic reciprocity” to simulate seismograms for each of ~415,000 earthquakes
  - Loosely-coupled, short-running serial jobs

5) 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/exec</th>
<th>CPU 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>160</td>
<td>50</td>
</tr>
<tr>
<td>Tensor simulation</td>
<td>40 GB</td>
<td>2</td>
<td>4,000</td>
<td>12,000</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>8</td>
<td>192</td>
<td>200</td>
</tr>
<tr>
<td>Seismogram synthesis</td>
<td>10 GB</td>
<td><strong>415,000</strong></td>
<td>1</td>
<td>800</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>755 GB</td>
<td><strong>415,000</strong></td>
<td></td>
<td>13,000</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
  - $x_1$
  - $x_2$

**Post-Processing Workflow**
- Tensor extraction
  - $x_8$
- Seismogram synthesis
  - $x_415,000$
- Hazard Curve
  - $x_1$
Challenge: Resource Provisioning

- In tensor workflow, submit job to remote scheduler
  - GRAM puts jobs in remote queue
  - Runs like a normal batch job
- 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
Pegasus-mpi-cluster

• MPI wrapper around serial or thread-parallel jobs
  – Master-worker paradigm
  – Specify jobs as usual, Pegasus does wrapping
  – Preserves dependencies

• Uses intelligent scheduling
  – Core counts, memory requirements, priorities
  – Locality preferences under development
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 SCEC disk
  – Pegasus registers data products in catalog

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

- **Seismogram jobs causing 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
Science Run

- Hazard curves for 1144 sites
- 21100 processors for 23 days (Blue Waters, Stampede)
  - Limited by queuing policies
  - Tensor workflow outpaced post-processing
- 470 million tasks executed
  - 165 tasks/sec
  - Only 9152 jobs in Stampede queue
  - Task failure rate: 1 in a million
- Managed 830 TB of data
  - 57 TB output files
  - 12.3 TB staged back to local disk (~16M files)
- Workflow tools scale!
Results (ratio)

CyberShake map compared to attenuation map – red is higher risk, green lower
Should you use workflow tools?

• Probably using a workflow already
  – Replaces manual hand-offs and polling to monitor
• Provides framework to assemble community codes
• Scales from local computer to large clusters
• Provide portable algorithm description independent of data
• Does add additional software layers and complexity
  – Some development time is required
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
Final Thoughts

- Automation is vital
  - Eliminate human polling
  - Get everything to run automatically if successful
  - Be able to recover from common errors

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

- Avoid premature optimization

- Consider new compute environments (dream big!)
  - Larger clusters, XSEDE / PRACE, Amazon EC2

- Tool developers want to help you!
Links

- SCEC:  http://www.scec.org
- Kepler:  http://kepler-project.org/
- Taverna: http://www.taverna.org.uk/
- Triana:  http://www.trianacode.org/
- VisTrails: http://www.vistrails.org
- Pegasus:  http://pegasus.isi.edu
- Condor: http://www.cs.wisc.edu/condor/
- Globus:  http://www.globus.org/
- CyberShake: http://scec.usc.edu/scecpedia/CyberShake
Questions?