Introduction to the Pegasus Workflow Management System
Before we begin

- If you plan to participate in the hands-on portion
- Copy the .vmdk.zip file and unzip it
  - It is about 900 MB zipped, 2.5 GB unzipped
- Install VirtualBox
  - [http://virtualbox.org](http://virtualbox.org)
  - The VM is 64-bit, and won’t work on 32-bit machines
- We will do the hands-on portion after the break
Agenda

- 9:00 – 10:00 Introduction to workflows and Pegasus
- 10:00 – 10:15 Break
- 10:15 – 11:15 Hands-on Pegasus Tutorial
- 11:15 – 11:30 Infrastructure Examples
- 11:30 – 12:00 Advanced Topics
- 12:00 – 1:00 Lunch
- 1:00 – 3:00 Bring your workflow
- 3:00 Adjourn
Computational Science Challenges

- Computational scientists need to:
  - Integrate diverse models and data
  - Automate parallel data processing steps
  - Repeat processing steps on new data
  - Reproduce previous results
  - Share their analysis steps with other researchers
  - Track the provenance of data products
  - Execute analyses in parallel on distributed resources
  - Reliably execute analyses on unreliable infrastructure

Scientific workflow management systems provide solutions to these problems
Scientific Workflows

- Orchestrate complex, multi-stage scientific computations
- Often expressed as directed acyclic graphs (DAGs)
- Capture analysis pipelines for sharing and reuse
- Can execute in parallel on distributed resources
Workflows can be simple!

J1  J2  J3  J4  J5  J6  J7  J8  J9  ...  Jn
Hunting Exoplanets with Kepler

• 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”

• Can take 1 hour to perform periodogram analysis of Kepler light curve

• Need to perform a bulk analysis of all the data to search for these periodic signals

  • 210K input, 630K output files
  • 210K tasks total
Some workflows are structurally complex

Gravitational-wave physics workflow

Genomics Workflow
Some workflows are large-scale and data-intensive

- **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)

- Need to support hierarchical workflows and scale
Some workflows couple large-scale simulations with data analysis

CyberShake PSHA Workflow

- 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
Why Scientific Workflows?

- Automate complex processing pipelines
- Support parallel, distributed computations
- Use existing codes, no rewrites
- Relatively simple to construct
- Reusable, aid reproducibility
- Can be shared with others
- Capture provenance of data
Workflow challenges

- **Portability**
  - How can you run a pipeline on Amazon EC2 one day, and a PBS cluster the next?

- **Data Management**
  - How do you ship in the small/large amounts data required by your pipeline?
  - Different protocols for different sites: Can I use SRM? How about GridFTP? HTTP and Squid proxies?
  - Can I use Cloud based storage like S3 on EC2?

- **Debug and Monitor Computations**
  - Users need automated tools to go through the log files
  - Need to correlate data across lots of log files
  - Need to know what host a job ran on and how it was invoked

- **Restructure Workflows for Improved Performance**
  - Short running tasks?
  - Data placement?
Pegasus Workflow Management System (WMS)

- 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 by many applications in a variety of domains
  - Earth science, physics, astronomy, bioinformatics
Why use Pegasus?

- Maps abstract workflows to diverse computing infrastructure
  - 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

- 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
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
Pegasus Workflows are Directed Acyclic Graphs

- **Nodes are tasks**
  - Typically, executables with arguments
  - Nodes can also be other workflows

- **Edges are dependencies**
  - Represent data flow
  - Can also be control dependencies
  - Pegasus can infer edges from data use

- **No loops, no branches**
  - Recursion is possible
  - Can generate workflows in a workflow
  - Can conditionally skip tasks with wrapper
Pegasus WMS Environment

API Interfaces
- Python
- Java
- Perl

Portals
- HubZero

Other Workflow Composition Tools: Grayson, Triana, Wings

Users

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

COMPUTE

STORAGE
- GridFTP
- HTTP
- FTP
- SRM
- IRDSS
- SCP
General Workflow Execution Model

- 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!
Pegasus WMS Data Flow

DAX

Site Catalog

Xform Catalog

Replica Catalog

Config

Pegasus <Planner>

DAG

Submit Script

DAGMan <Engine>

Condor <Scheduler>
Workflow Mapping/Planning

- Pegasus converts abstract workflow descriptions into executable workflows
  - 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
    - Executable locations
    - Scratch directories
    - Job submission endpoints
    - Data transfer endpoints
  - Choose input files (replica selection)
  - Plan data movements
    - Add data transfer jobs (stage-in, stage-out, inter-site)
    - Add data registration jobs
  - Perform optimizations
    - Workflow reduction / data reuse
    - Task clustering
  - Add setup and cleanup jobs
  - Generate executable workflow artifacts
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)
Abstract Workflow (DAX)

- **Abstract description of workflow**
  - Contains jobs, data, and the relationships between them
  - Excludes resource-specific information (hosts, endpoints, etc.)
  - Excludes physical file names (paths, URLs, etc.)
  - Executables represented by logical transformations
  - Data represented by logical files
  - Dependencies are job-job, but can be inferred based on data

- **Simple XML format**
  - `<job>`, `<file>`, `<uses>`, `<parent>`, `<child>`

- **DAX Generator**
  - Use Java, Perl, Python APIs to generate DAXes
  - The “real” workflow
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>
    <profile namespace="env" key="GLOBUS_LOCATION">/usr/local/globus</profile>
    <profile namespace="pegasus" key="style">globus</profile>
    <profile namespace="pegasus" key="X509_USER_PROXY">/tmp/x509_u40001</profile>
  </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
Data Staging Configurations

Shared File System (typical of XSEDE and 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
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
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
Failures

- Pegasus knows a job has failed if:
  - it returns a non-zero exit code
  - its output does not contain a specified “success message”
  - its output does contain a specified “failure message”
  - it exceeds a specified time limit (useful for checkpointing)
  - (in some cases) it fails to produce expected output files

- Pegasus automatically retries failed jobs
  - Helps with transient failures
  - Each job has a set number of retries per run (profile: dagman.retry)
  - When a job has exhausted all retries, it is marked as permanently failed
  - Workflow runs with permanently failed jobs are checkpointed via a “rescue DAG” so that they can be restarted
Workflow Debugging

- **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
Steps to Run Pegasus

1. Specify your computation in terms of DAX
   - Write a DAX generator using API

2. Set up your catalogs and config file
   - Site, Transformation, Replica, pegasus.properties

3. Plan and Submit your workflow
   - pegasus-plan & pegasus-run (or pegasus-plan –submit)

4. Monitor and Debug your workflow
   - pegasus-status
   - pegasus-analyzer
   - Pegasus Dashboard

5. Gather performance statistics
   - pegasus-statistics
Set Up VirtualBox VM

- Launch VirtualBox
- Click “New”
- Enter a name, the OS is Linux, **Red Hat 64-bit** (continue)
- For memory use 1024 MB (continue)
- For hard disk, choose “Existing” and pick .vmdk file (create)
Boot the new VM

- Username: tutorial
- Password: pegasus

- Getting your cursor back: Left alt or command key
Diamond Workflow

Abstract Workflow

Executable Workflow
Plan

- DAX Generator
- DAX
- Transformation Catalog
- Replica Catalog
- Site Catalog
- Config File

- pegasus-plan
- pegasus-run
- pegasus-status
- submit directory
- Pegasus Dashboard
- pegasus-analyzer
- pegasus-statistics
Infrastructure Examples

- This section describes some relevant infrastructures that Pegasus supports and how to configure a workflow to access those infrastructures.
- Most of these configurations are documented in the User Guide Chapter 6.
Condor Pool

- The easiest mode, no special configuration required

- Profiles:
  - pegasus.style = condor
  - condor.universe = vanilla
  - Other condor.* profiles can be used in SC, TC, and DAX to set priority, rank, requirements and other Condor job attributes

```xml
<site handle="condor" arch="x86_64" os="LINUX">
  <profile namespace="pegasus" key="style">condor</profile>
  <profile namespace="condor" key="universe">vanilla</profile>
</site>
```
Local PBS Cluster

- Pegasus can submit to local PBS clusters using Condor
  - Condor must be running on a machine that can call qsub, qstat, etc.
  - Condor uses glite to interface with PBS
  - Requires modifications to Condor configuration in /usr/libexec/condor/glite to add a script from /usr/share/pegasus/htcondor/glite
  - Similar approach for LSF and SGE

- Profiles
  - pegasus.style = glite
  - condor.grid_resource = pbs
  - Other profiles can be used in TC and DAX to specify queue, wall time, nodes, cores, etc.

```xml
<site handle="hpcc" arch="x86_64" os="LINUX">
  <directory type="shared-scratch" path="/staging/gmj/juve">
    <file-server operation="all" url="scp://juve@hpc-login2.usc.edu/staging/gmj/juve"/>
  </directory>
  <profile namespace="env" key="PEGASUS_HOME">/home/rcf-40/juve/pegasus-4.4.1</profile>
  <profile namespace="pegasus" key="style">glite</profile>
  <profile namespace="condor" key="grid_resource">pbs</profile>
  <profile namespace="pegasus" key="change.dir">true</profile>
  <profile namespace="pegasus" key="SSH_PRIVATE_KEY">/home/gideon/.ssh/id_rsa</profile>
</site>
```
Remote Cluster / Grid Site

- Pegasus can submit jobs to remote sites via Globus GRAM
  - For example, GRAM is supported at NERSC
  - Authentication is via X.509, which can be very challenging
  - GRAM often requires debugging

- Profiles
  - `pegasus.style = globus` (also the default for historical reasons)
  - `globus.*` profiles can be used to set queue, account, max wall time, count, hostcount, etc

```xml
<site handle="hopper" arch="x86_64" os="LINUX">
  <grid type="gt5" contact="hoppergrid.nersc.gov/jobmanager" scheduler="Fork" jobtype="auxillary"/>
  <grid type="gt5" contact="hoppergrid.nersc.gov/jobmanager-pbs" scheduler="PBS" jobtype="compute"/>
  <directory type="shared-scratch" path="/scratch/scratchdirs/juve">
    <file-server operation="all" url="gsiftp://hoppergrid.nersc.gov/scratch/scratchdirs/juve"/>
  </directory>
  <profile namespace="env" key="PEGASUS_HOME">/project/projectdirs/m1503/pegasus/pegasus-4.4.0</profile>
  <profile namespace="globus" key="project">m1503</profile>
  <profile namespace="globus" key="queue">regular</profile>
</site>
```
Pilot Jobs / Condor Glideins

- Condor workers can be submitted as batch jobs to HPC system
  - HPC system resources become part of user’s Condor pool
  - Workflow jobs bypass site’s batch scheduler (higher throughput)
  - Pegasus is configured to target a Condor pool with shared file system
Example: SCEC CyberShake on Titan

- **Titan is ideal for SCEC because of GPUs**
  - New GPU code much faster than previous CPU code
  - Titan has many GPUs

- **Titan has no remote job submission interface**
  - Other systems like Kraken and Blue Waters have GRAM
  - Security policy prohibits GRAM and similar on Titan
  - Incoming connections require 2-factor authentication

- **SCEC prefers to manage their workload from their own submit host**
  - Easier to monitor and manage workload
  - Only need to maintain one submit infrastructure to target many sites (also use: Kraken, Stampede, Blue Waters)
Pilot Jobs on Titan: Challenges

- **How to enable network connections from USC to Titan?**
  - Network policy prevents incoming connections to Titan w/o 2-factor auth
  - Solution: Condor connection brokering
  - Condor worker makes persistent *outgoing* connections to Condor master at USC, which arranges connections (like passive FTP)

- **Where to run Condor worker?**
  - Compute nodes can’t talk to outside network – Condor can’t run there
  - Solution: Use service nodes, aprun to launch jobs on compute nodes

- **How to run MPI jobs?**
  - Condor has poor support for MPI
  - Use wrapper scripts to call aprun directly

- **Submitting pilot jobs is done manually**
Pilot Jobs on Titan

User logs in

User submits pilot job for n compute nodes

Titan
Head Node

Titan
PBS Queue

Pilot job starts on service node and runs C. worker

Submit Host

Condor Master

Worker sets up reverse connection to master
Advertises N << n virtual slots

Service Node

Condor Worker

Job Wrapper

Job gets matched with worker and Sent via reverse connection

Job wrapper uses aprun to launch tasks on compute nodes

Notes:
1. Data staging jobs run on service node using GridFTP 3rd party transfer
2. Job wrappers launch both serial and MPI tasks
3. Most serial tasks in workflow handled by Pegasus-MPI-Cluster
4. Size of pilot job (n) and num slots (N) are carefully chosen based on workload
Advanced Topics

- All these topics are documented in the user guide
  - [https://pegasus.isi.edu](https://pegasus.isi.edu) → Documentation → User Guide

- There are many examples in the User Guide and in /usr/share/pegasus/examples
Fine-Grained Workflows

- Problem: Many scientific workflows are fine-grained
  - Thousands of tasks
  - Short duration
  - Serial

- Collectively, these tasks require distributed resources to finish in a reasonable time, but individually they are relatively small
  - Touch many GB or TB of data
  - Consume thousands of CPU hours

- Many large-scale compute resources are optimized for a few, large, parallel jobs, not many small, serial jobs
  - Serial tasks face long queue times due to low priority
  - Batch schedulers have low throughput

Results in poor workflow performance
Solutions

- **Advance Reservations**
  - No waiting in the queue
  - Require admin support – not supported at most sites
  - Negatively affect system utilization

- **Pilot jobs (e.g. Condor Glideins)**
  - Can have utilization problems
  - Introduce security holes
  - Very complex
  - Have problems with firewalls, NAT
  - Some view it as “cheating”

- **Task Clustering**
  - Group small tasks into a smaller number of batch jobs
  - Difficult to get the right clustering parameters – end up guessing
  - Clustered jobs are still serial
  - Clustering reduces parallelism, and introduces false dependencies
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 on batch systems
- Runs as an MPI job
  - Uses MPI to implement master/worker protocol
- Works on most HPC systems
  - Requires: MPI, a shared file system, and fork()
- Allows sub-graphs of a Pegasus workflow to be submitted as monolithic jobs to remote resources
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
PMC Modes

- **Approach 1: Stand alone**
  - Run PMC jobs without Pegasus at all

- **Approach 2: Task Clustering Implementation in Pegasus**
  - Pegasus partitions a large workflow into sub-graphs
  - Each sub-graph is rendered as a PMC job
  - The workflow becomes a small number of PMC jobs plus auxiliary jobs (setup, stage-in, stage-out, cleanup, data registration)

- **Approach 3: PMC Only Mode in Pegasus**
  - Pegasus converts the entire DAX into a single PMC job
Data Cleanup

- **Problem: Running out of space on shared scratch**
- **Why does it occur**
  - Workflows bring in huge amounts of data
  - Data is generated during workflow execution
  - Application jobs can’t clean up because other jobs may need the data
- **Solutions**
  - Perform cleanup at the end of the workflow
    - Does not work if scratch fills up during workflow execution
  - Interleave cleanup with compute jobs
    - Requires an analysis of the workflow to determine when a file is no longer required
Data Cleanup Example

Montage 1 degree workflow run with cleanup

Use the --cleanup option for pegasus-plan
- “none” for no cleanup
- “inplace” to clean up as soon as possible
- “leaf” to clean up at the end of the workflow
Workflow Reduction (Data Reuse)

Abstract Workflow

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

RECURSIVE DAX

INCREASING LEVEL OF RECURSION

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

RECURATION ENDS WHEN DAX WITH ONLY COMPUTE JOBS IS ENCOUNTERED
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>
```
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
Integration with HUBzero

Credit: Frank McKenna UC Berkeley, NEES, HUBzero
Other Advanced Features

- **Job Throttling**
  - Prevents too many jobs from being queued at once

- **Transfer Throttling**
  - Prevents workflow from starting too many transfers that overwhelm file servers

- **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
Summary – What Does Pegasus provide an Application

- **Portability / Reuse**
  - User created workflows can be run in different environments without alteration.

- **Performance**
  - The Pegasus mapper can reorder, group, and prioritize tasks in order to increase the overall workflow performance.

- **Scalability**
  - Pegasus can easily scale both the size of the workflow, and the resources that the workflow is distributed over. Pegasus runs workflows ranging from just a few computational tasks up to 1 million.
Summary – What Does Pegasus provide an Application

- **Provenance**
  - Provenance data is collected in a database, and the data can be summaries with tools such as pegasus-statistics, Pegasus Dashboard, or directly with SQL queries.

- **Data Management**
  - Pegasus handles replica selection, data transfers and output registrations in data catalogs. These tasks are added to a workflow as auxiliary jobs by the Pegasus planner.

- **Reliability**
  - Jobs and data transfers are automatically retried in case of failures. Debugging tools such as pegasus-analyzer help the user to debug the workflow in case of non-recoverable failures.
If you get stuck...

And you can draw....

We can help you!
More Information

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

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

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

- **Email addresses:**
  - Pegasus users list (public): pegasus-users@isi.edu
  - Pegasus support (private): pegasus-support@isi.edu