

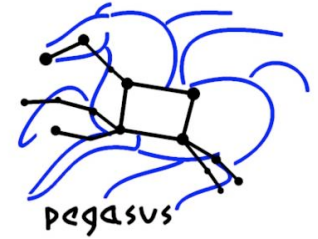
Pegasus WMS Tutorial

Gaurang Mehta¹, Kent Wenger²

(gmehta@isi.edu, wenger@cs.wisc.edu)

¹ Center for Grid Technologies, USC Information Sciences
Institute

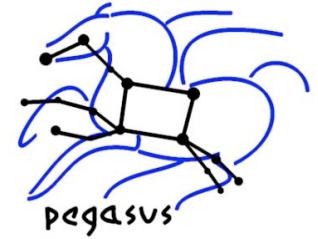
² University of Wisconsin Madison, Madison, WI



Outline of Tutorial

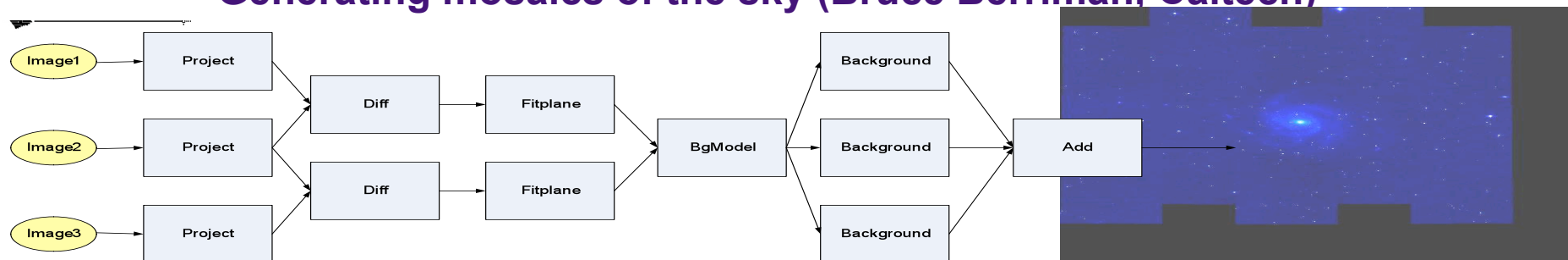
- Introduction to Pegasus WMS
- Composing a Simple Workflow In terms of DAX.
- Pegasus Internals
- Mapping and Running Workflows Locally
- Mapping and Running Workflows on the Grid
- Optimization techniques for mapping and executing Large Scale workflows

Scientific Workflows



- Capture individual data transformation and analysis steps
- Large monolithic applications broken down to smaller jobs.
 - Smaller jobs can be independent or connected by some control flow/ data flow dependencies.
 - Usually expressed as a Directed Acyclic Graph of tasks
- Allows the scientists to modularize their application
- Scaled up execution over several computational resources

Generating mosaics of the sky (Bruce Berriman, Caltech)

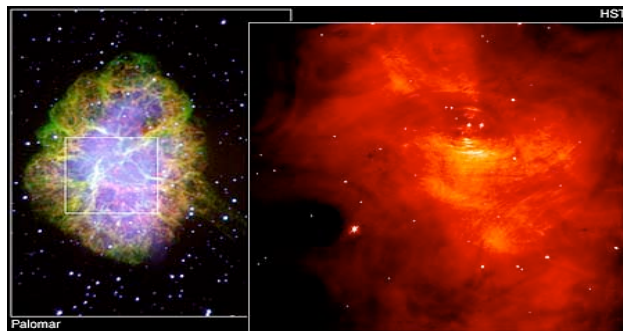


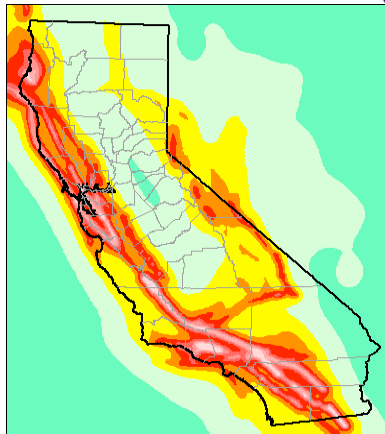
Size of the mosaic is degrees square*	Number of jobs	Number of input data files	Number of Intermediate files	Total data footprint	Approx. execution time (20 procs)
1	232	53	588	1.2GB	40 mins
2	1,444	212	3,906	5.5GB	49 mins
4	4,856	747	13,061	20GB	1hr 46 mins
6	8,586	1,444	22,850	38GB	2 hrs. 14 mins
10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

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

LIGO Scientific Collaboration

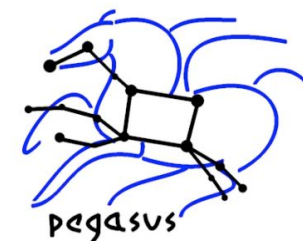
- Continuous gravitational waves are expected to be produced by a variety of celestial objects
- Only a small fraction of potential sources are known
- Need to perform blind searches, scanning the regions of the sky where we have no a priori information of the presence of a source
 - Wide area, wide frequency searches
- Search is performed for potential sources of continuous periodic waves near the Galactic Center and the galactic core
- Search for binary inspirals collapsing into black holes.
- The search is very compute and data intensive



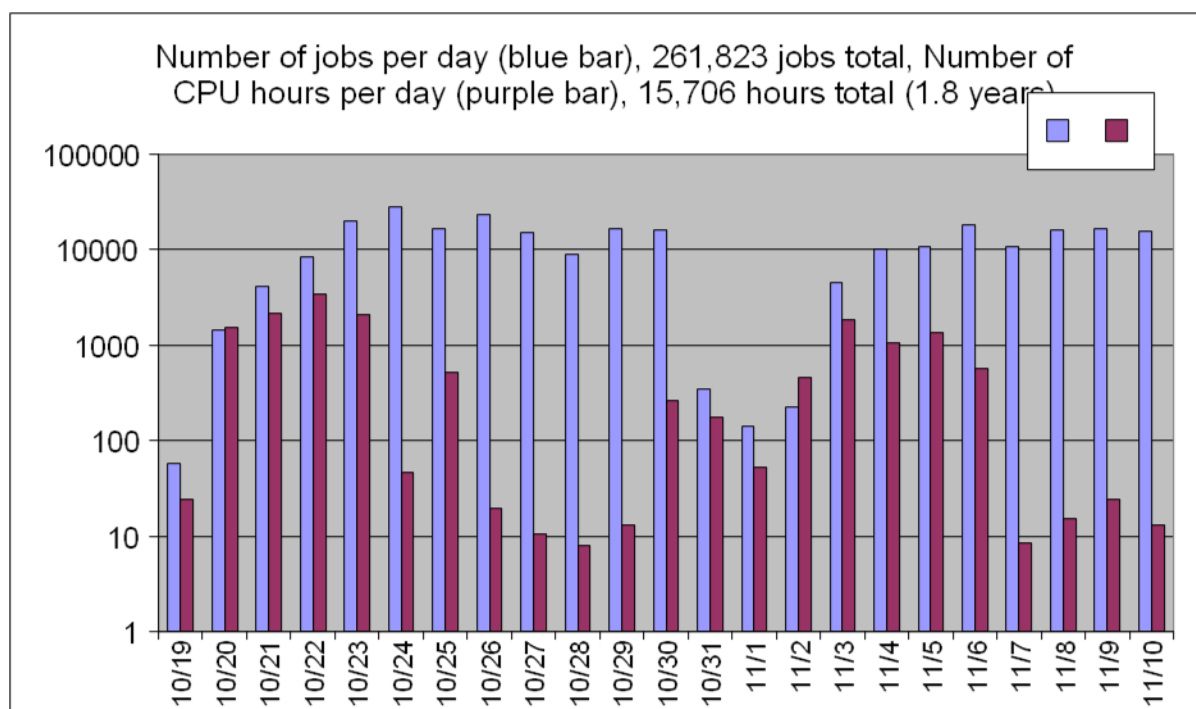


Southern California Earthquake Center (SCEC)

- 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



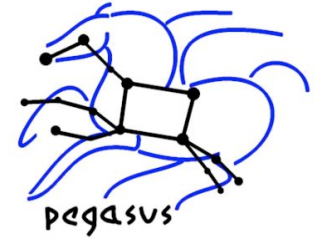
Pegasus mapped SCEC CyberShake workflows onto the TeraGrid in Fall 2005. The workflows ran over a period of 23 days and processed 20TB of data using 1.8 CPU Years. **Total tasks in all workflows: 261,823.**



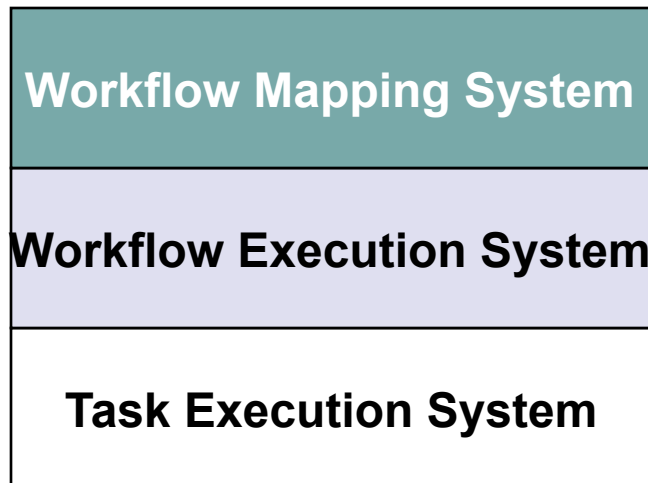
CyberShake Science result: *CyberShake delivers new insights into how rupture directivity and sedimentary basin effects contribute to the shaking experienced at different geographic locations. As a result more accurate hazard maps can be created.*

SCEC is led by Tom Jordan, USC

Pegasus-Workflow Management System a layered approach



A reliable, scalable workflow management system that an application or workflow composition service can depend on to get the job done



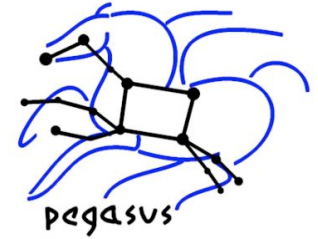
A decision system that develops strategies for reliable and efficient execution in a variety of environments

Reliable and scalable execution of dependent tasks

Reliable, scalable execution of independent tasks (locally, across the network), priorities, scheduling

Cyberinfrastructure: Local machine, cluster, Condor pool, Grid

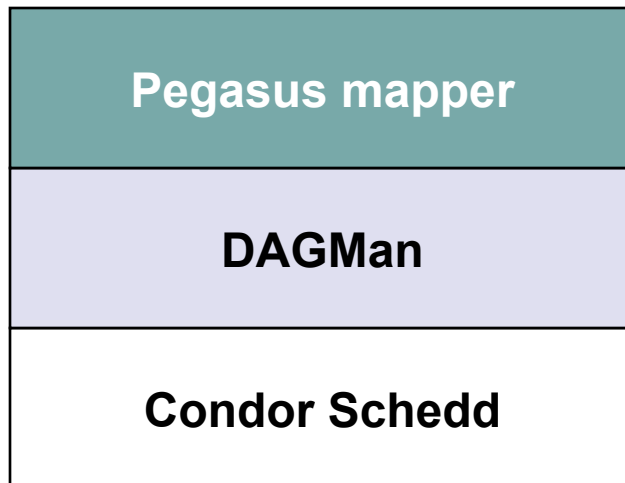
Pegasus Workflow Management System



Abstract Workflow



A reliable, scalable workflow management system that an application or workflow composition service can depend on to get the job done

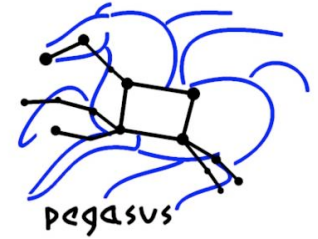


A decision system that develops strategies for reliable and efficient execution in a variety of environments

Reliable and scalable execution of dependent tasks

Reliable, scalable execution of independent tasks (locally, across the network), priorities, scheduling

Cyberinfrastructure: Local machine, cluster, Condor pool, OSG, TeraGrid

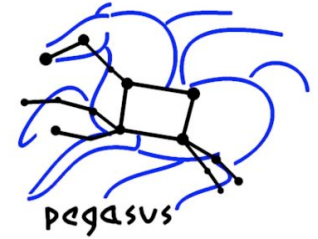


Mapping Correctly

- Select where to run the computations
 - Apply a scheduling algorithm
 - HEFT, min-min, round-robin, random
 - Schedule in a data-aware fashion (data transfers, amount of storage)
 - The quality of the scheduling depends on the quality of information
 - Transform 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
- Add nodes to create an execution directory on a remote site

Additional Mapping Elements

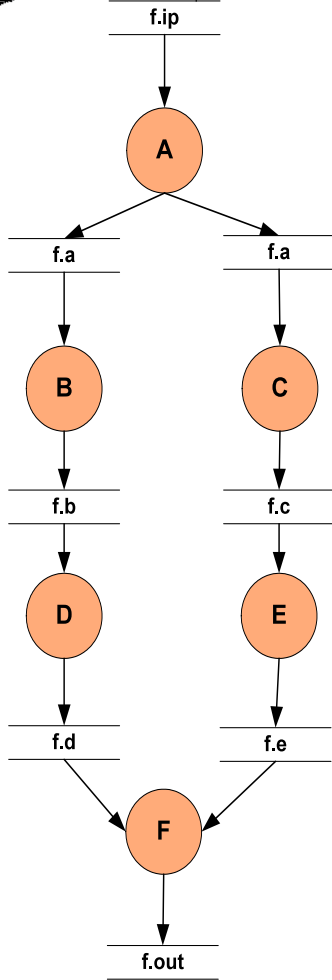
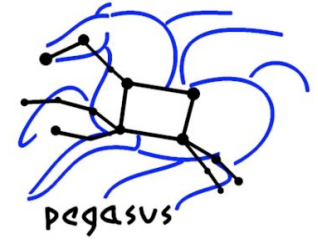
- Cluster compute nodes in small granularity applications
- Add data cleanup nodes to remove data from remote sites when no longer needed
 - reduces workflow data footprint
- Add nodes that register the newly-created data products
- Provide provenance capture steps
 - Information about source of data, executables invoked, environment variables, parameters, machines used, performance
- **Scale matters--today we can handle:**
 - 1 million tasks in the workflow instance (SCEC)
 - 10TB input data (LIGO)



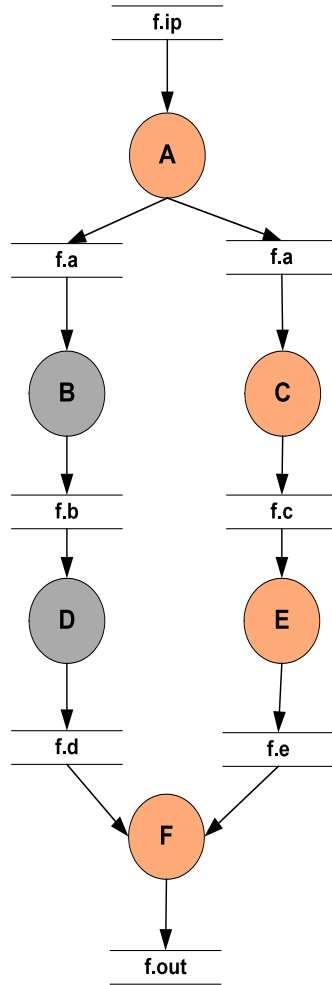
Optimizations during Mapping

- Node clustering for fine-grained computations
 - Can obtain significant performance benefits for some applications (in Montage ~80%, SCEC ~50%)
- Data reuse in case intermediate data products are available
 - Performance and reliability advantages—workflow-level checkpointing
- Data cleanup nodes can reduce workflow data footprint
 - by ~50% for Montage, applications such as LIGO need restructuring
- Workflow partitioning to adapt to changes in the environment
 - Map and execute small portions of the workflow at a time

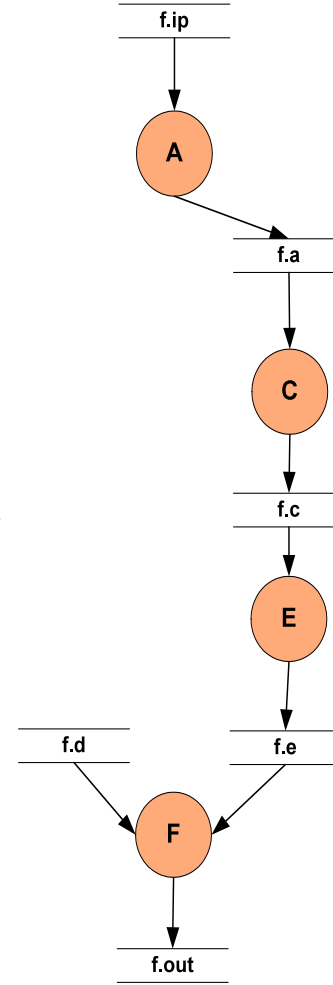
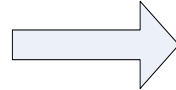
Workflow Reduction (Data Reuse)



Abstract Workflow

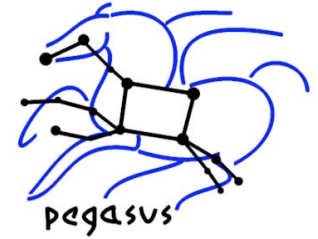


File f.d exists somewhere.
Reuse it.
Mark Jobs D and B to delete



Delete Job D and Job B

How to: Files need to be cataloged in replica catalog at runtime. The registration flags for these files need to be set in the DAX

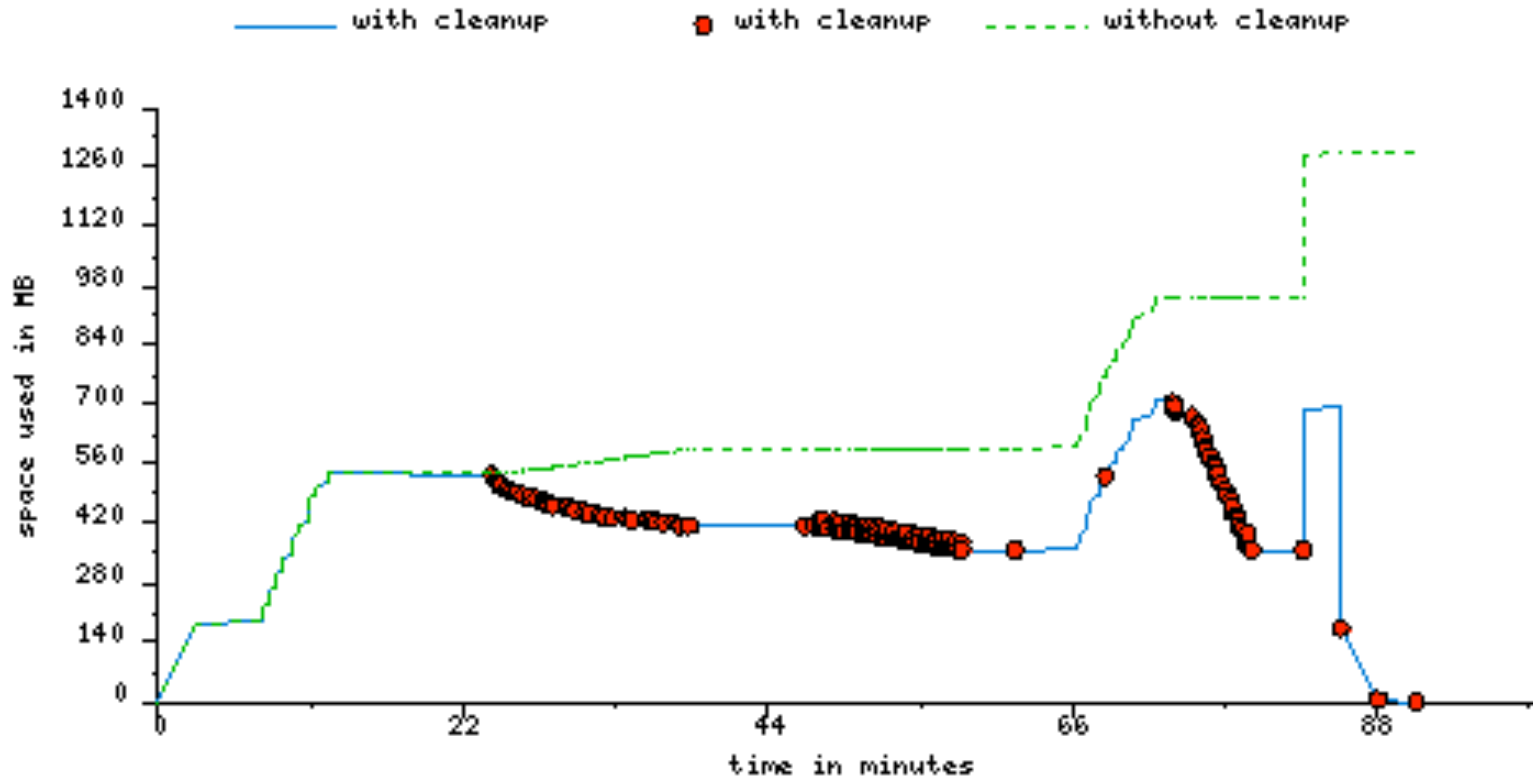
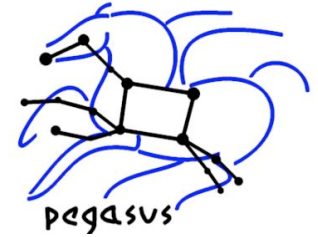


File cleanup

- 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.

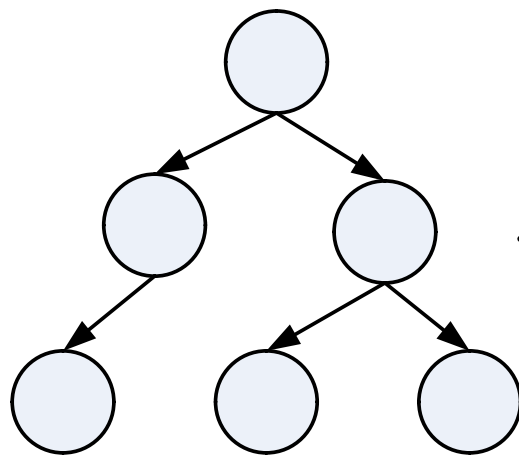
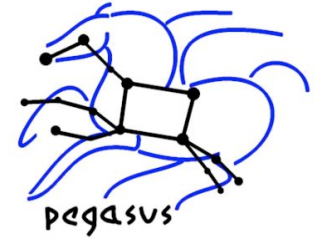


Storage Improvement for Montage Workflows

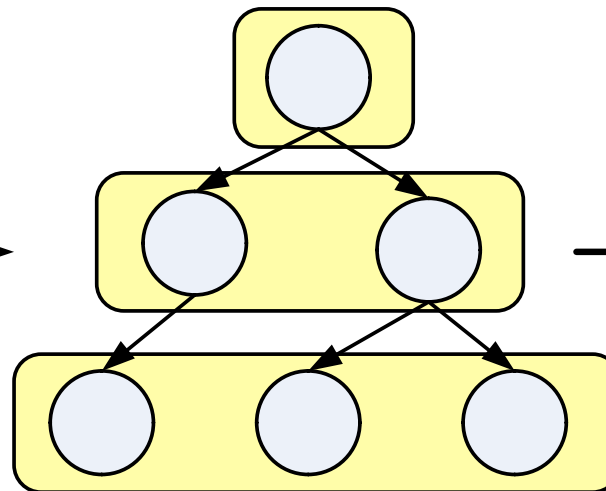


Montage 1 degree workflow run with cleanup on OSG-PSU

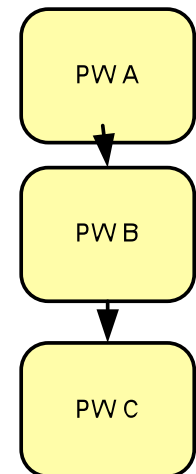
Managing execution environment changes through partitioning



Original Abstract Workflow



A Particular Partitioning

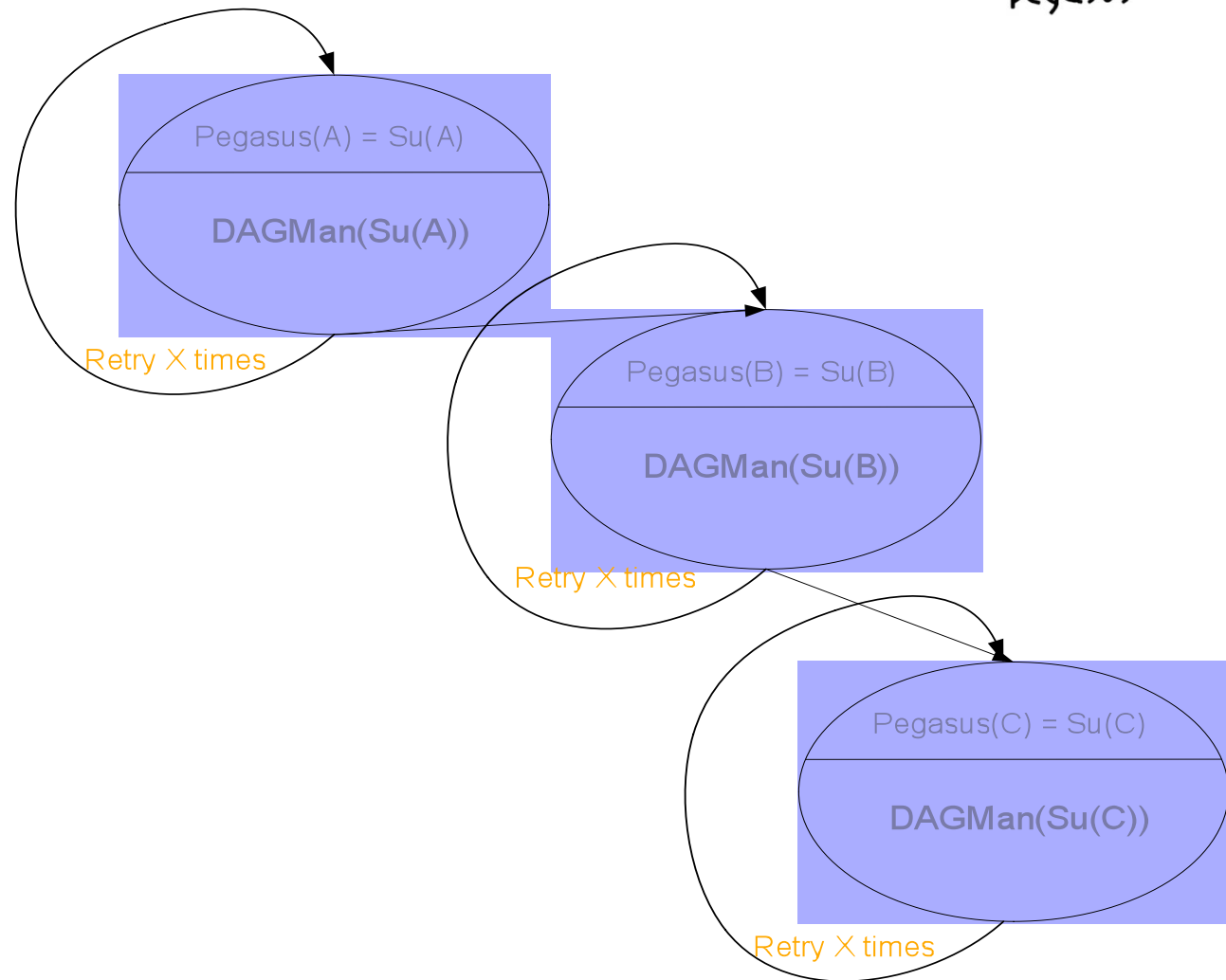


New Abstract Workflow

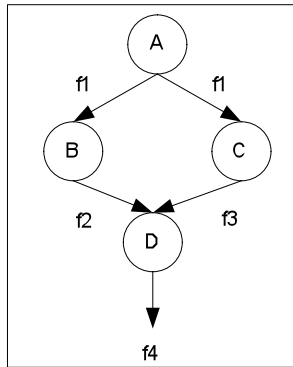
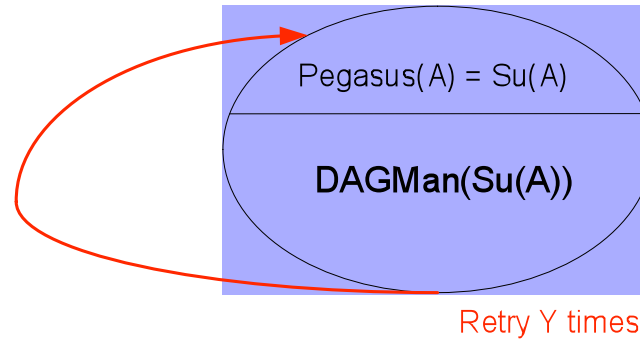
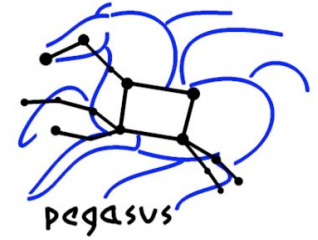
Resulting Meta-Workflow

Pegasus(X): Pegasus generates the concrete workflow and the submit files for Partition X -- Su(X)

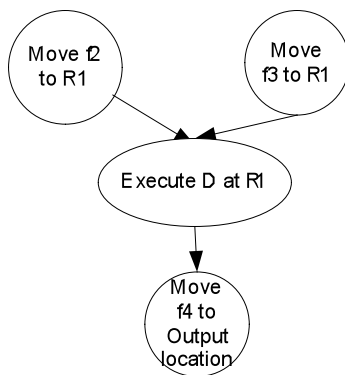
DAGMan(Su(X)): DAGMan executes the concrete workflow for X



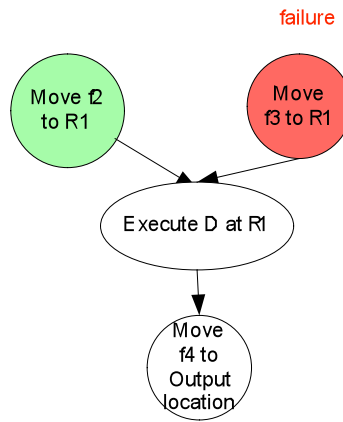
Workflow-level checkpointing



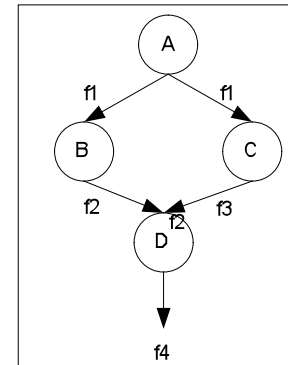
Original abstract workflow partition



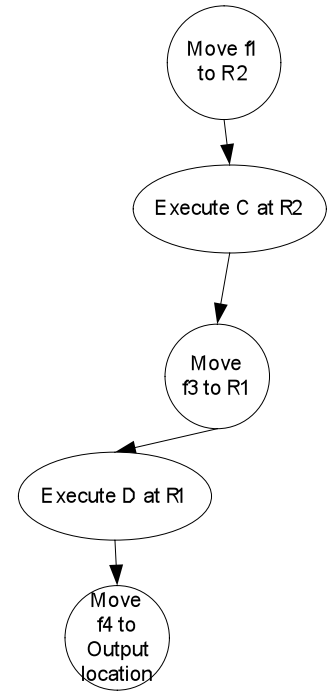
Pegasus mapping, f2 and f3 were found in a replica catalog



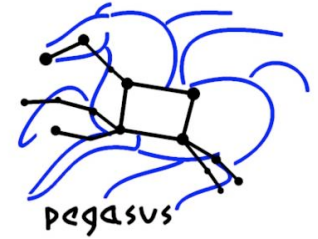
Workflow submitted to DAGMan



Pegasus is called again with original partition



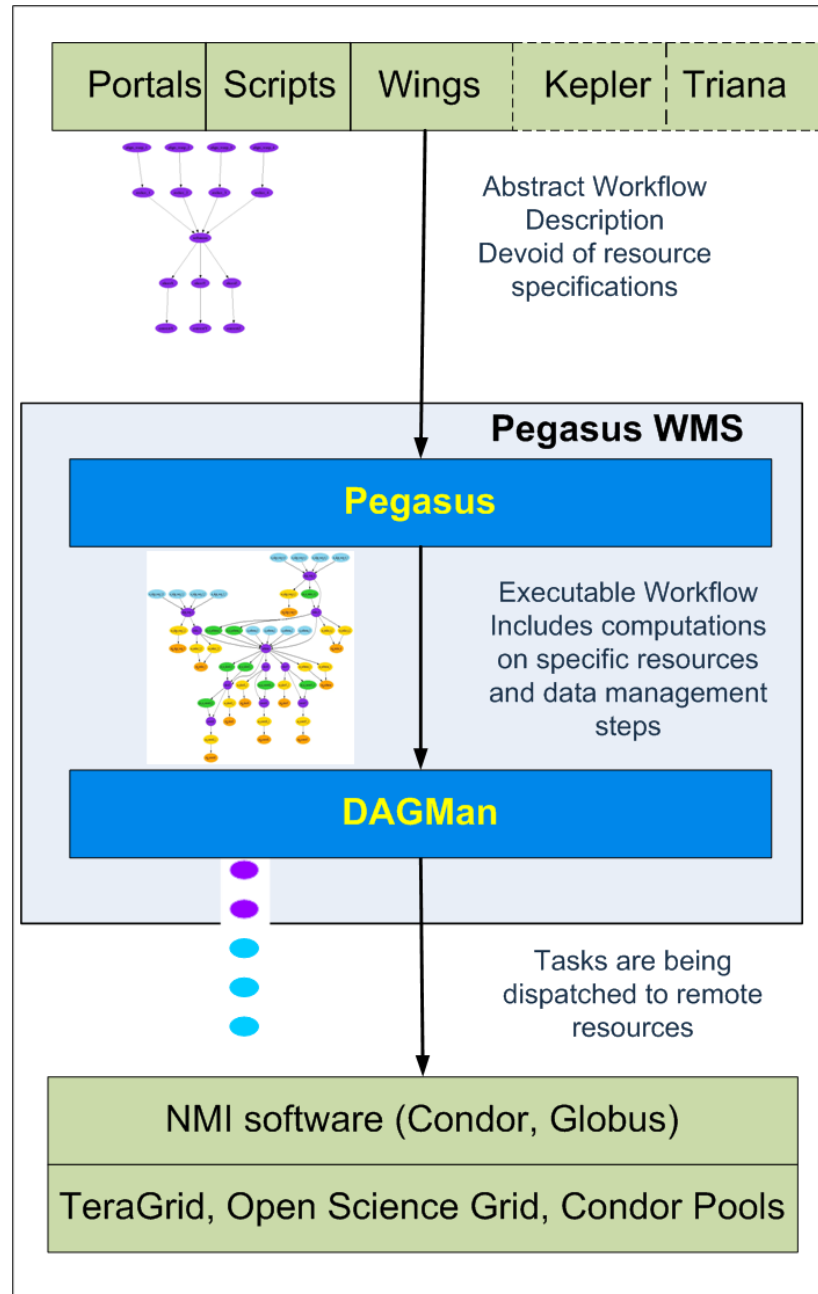
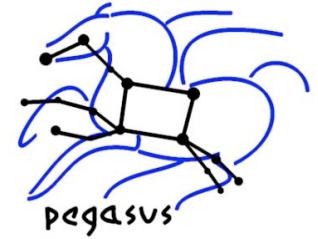
New mapping, here assuming R1 was picked again

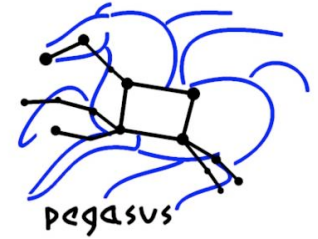


Outline of Tutorial

- Introduction to Pegasus WMS
- Composing a Simple Workflow In terms of DAX.
- Pegasus Internals
- Mapping and Running Workflows Locally
- Mapping and Running Workflows on the Grid
- Optimization techniques for mapping and executing Large Scale workflows

High-level system view





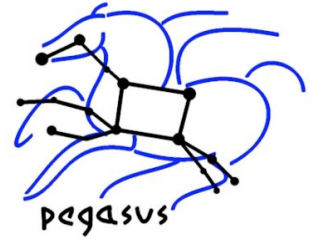
Pegasus workflow

- 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

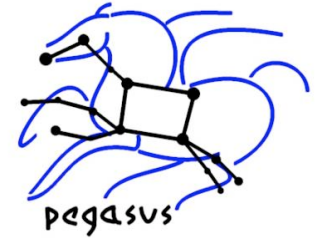


Abstract Workflow (DAX)

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



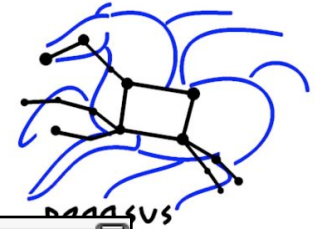
Understanding DAX (1)

```
<!-- 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"/>

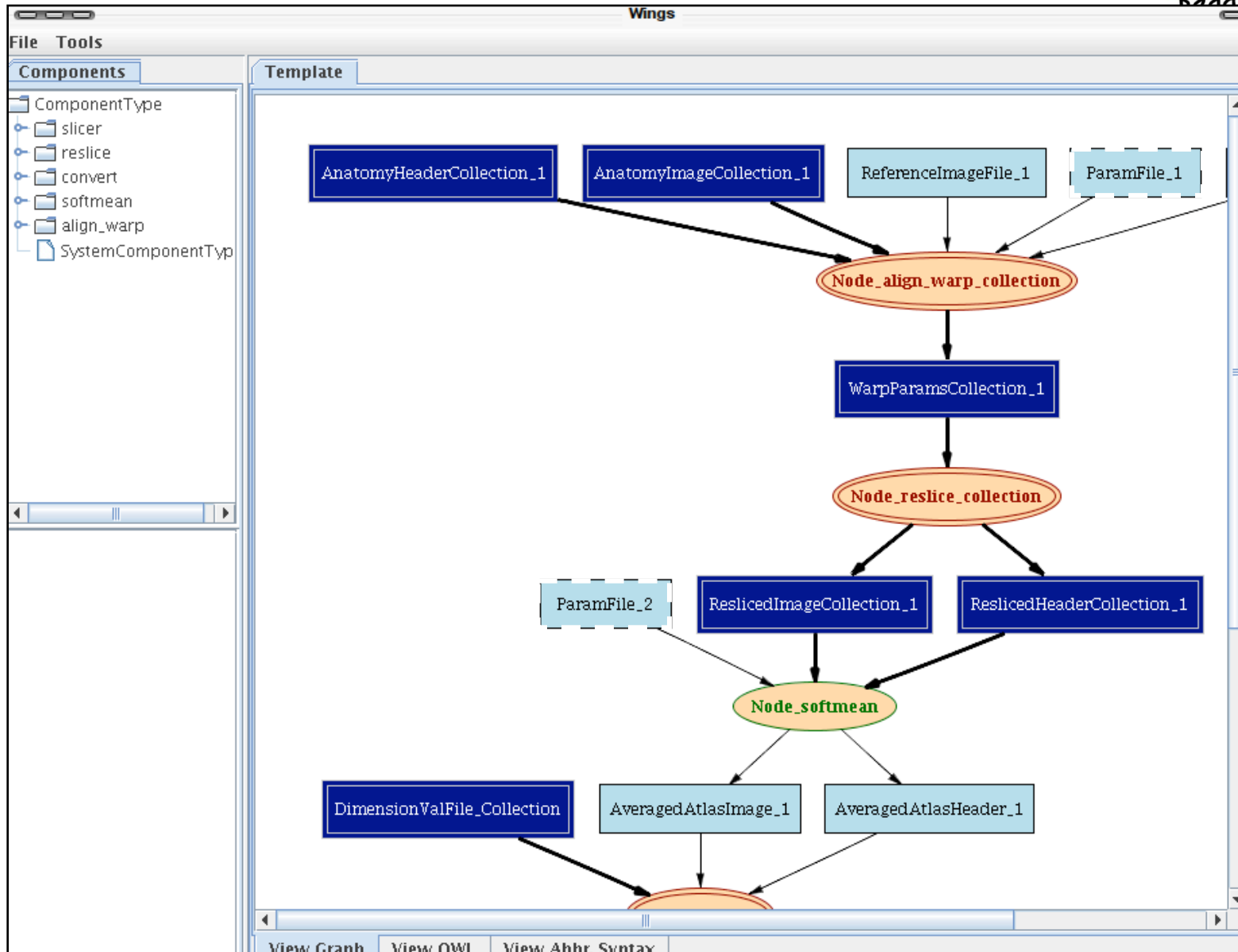
<!-- part 2: definition of all jobs (at least one) -->
  <job id="ID000001" namespace="pegasus" name="preprocess" version="1.0" >
    <argument>-a top -T 6 -i <filename file="f.input"/> -o <filename file="f.intermediate"/>
    </argument>
    <uses file="f.input" link="input" dontRegister="false" dontTransfer="false"/>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

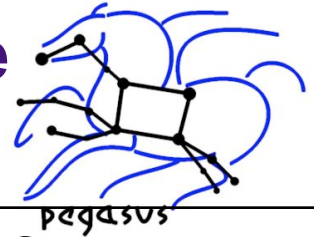


Creating Workflow Template with Wings GUI

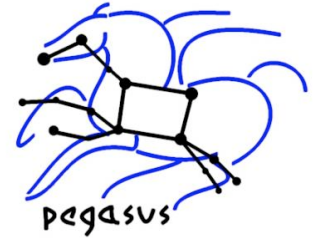




Comparison of abstract and executable workflows

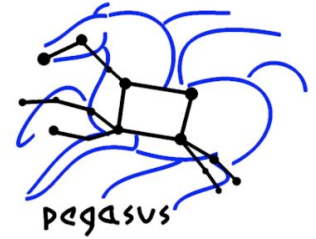


Abstract Workflow	Executable Workflow
Describes your workflow at a logical level	Describes your workflow in terms of physical files and paths
Site Independent	Site Specific
Captures just the computation that the user (you) want to do	Has additional jobs for data movement etc.



Outline of Tutorial

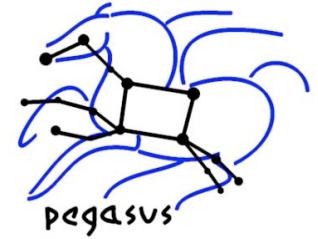
- Introduction to Pegasus WMS
- Composing a Simple Workflow In terms of DAX.
- Pegasus Internals
- Mapping and Running Workflows Locally
- Mapping and Running Workflows on the Grid
- Optimization techniques for mapping and executing Large Scale workflows



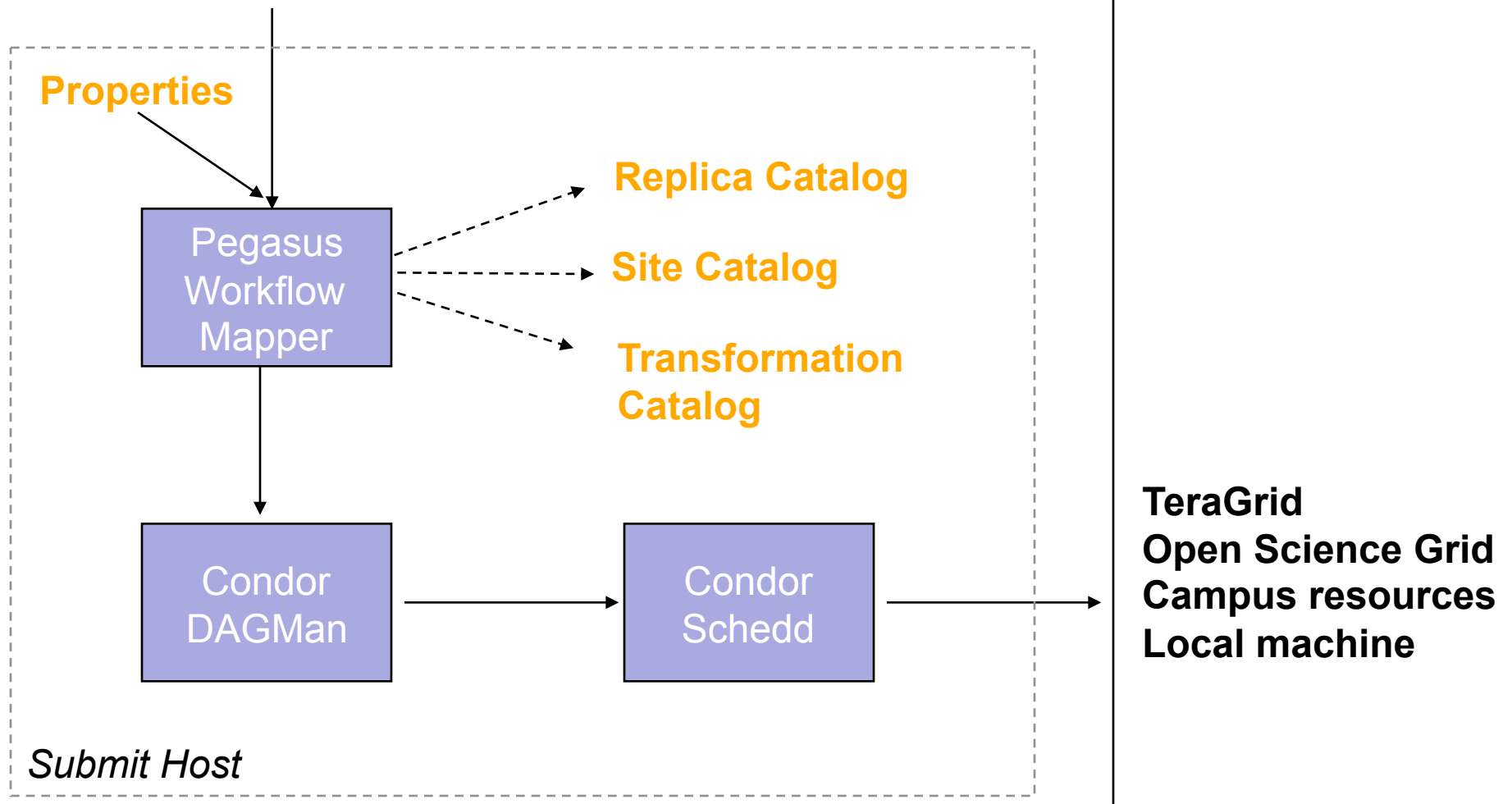
Discovery

- Data
 - Where do the input datasets reside?
- Executables
 - Where are the executables installed ?
 - Do binaries exist somewhere that can be staged to remote grid sites?
- Site Layout
 - What does a grid site look like?

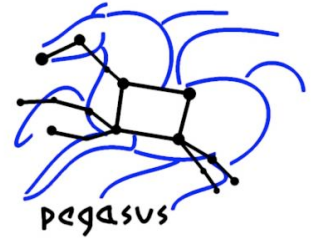
Pegasus WMS



Workflow Description in XML

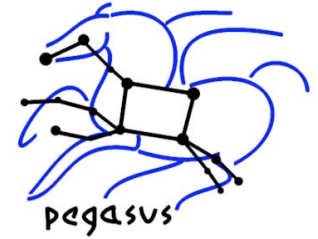


Pegasus WMS restructures and optimizes the workflow, provides reliability



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

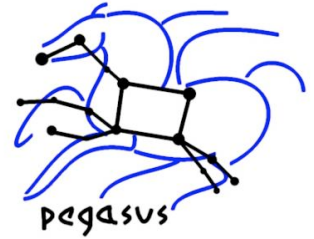


Replica Catalog

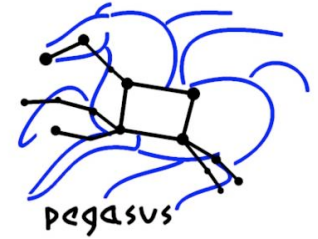
- Pegasus interfaces with a variety of replica catalogs
 - File based Replica Catalog
 - useful for small datasets (like this tutorial)
 - cannot be shared across users.
 - Database based Replica Catalog
 - useful for medium sized datasets.
 - can be used across users.
 - Globus Replica Location Service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

Replica Catalog

Exercise: 2.2



- 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 to Exercise 2.2):
 - Use the rc-client to
 - Populate the Replica Catalog
 - Query the Replica Catalog
 - Remove entries (offline exercise)



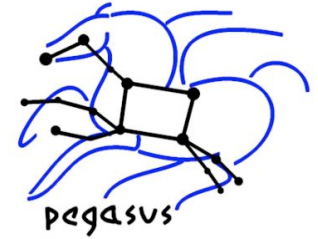
Site Catalog—finding resources

- 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

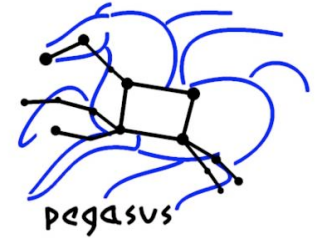
Exercise: 2.3



- 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 SQLite2 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



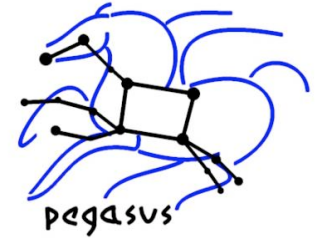
```
<site handle="isi_skynet" sysinfo="INTEL32::LINUX" gridlaunch="/nfs/software/vds/vds/bin/
kickstart">

  <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>

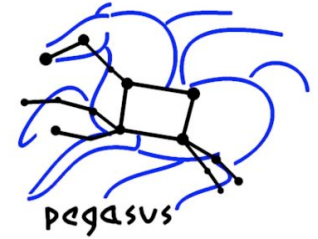
</site>
```



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

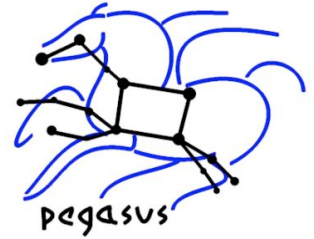


Transformation Catalog Overview

- 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 the which the transformation is available
 - the URL for the physical transformation
 - Profiles that associate runtime parameters like environment variables, scheduler related information



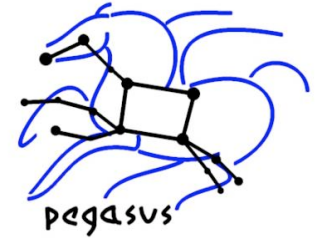
Transformation Catalog Exercise (Offline)



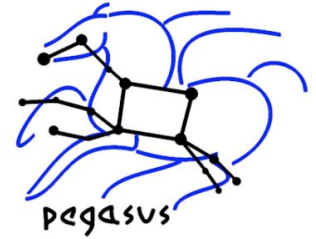
- tc-client is a command line client that is primarily used to configure the database TC
- Works even for file based transformation catalog.



Pegasus-WMS Configuration



- 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/pegasus-wms/config/properties` file

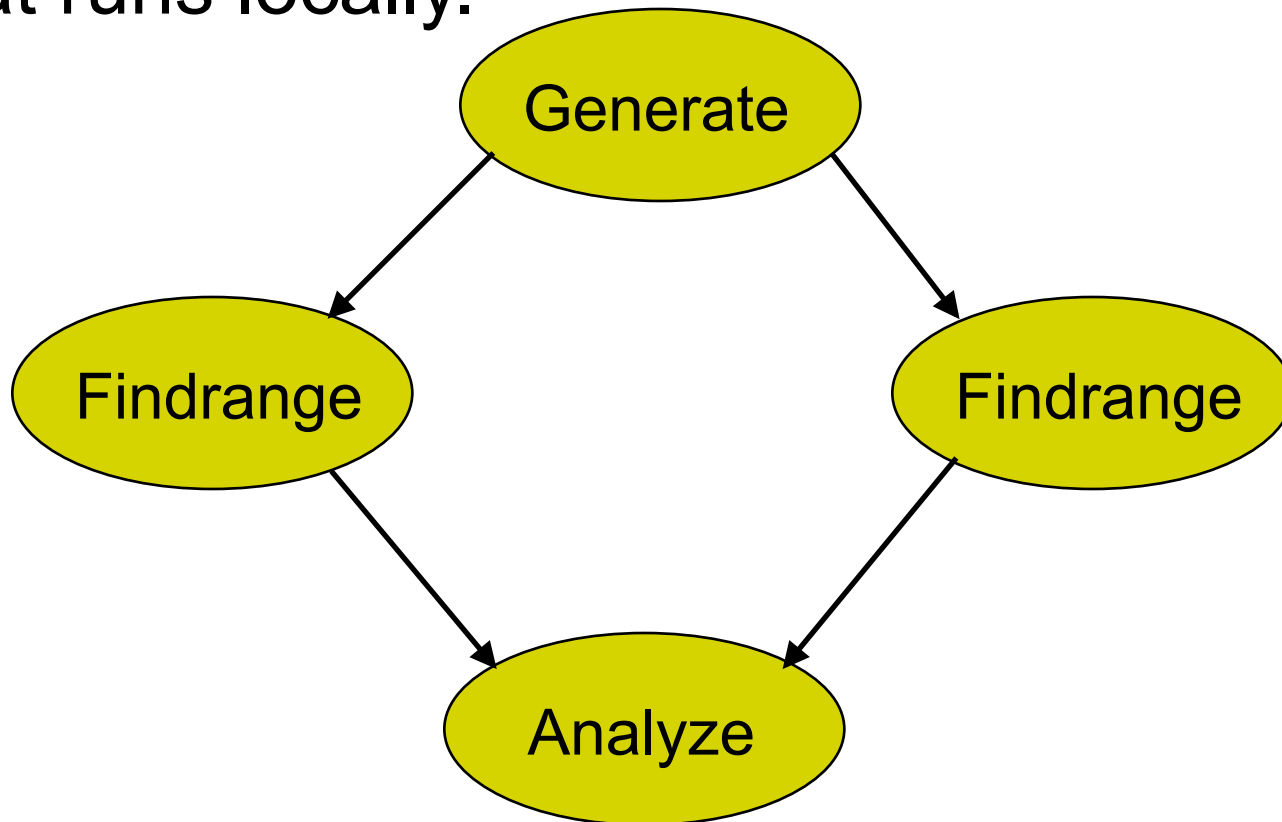


Outline of Tutorial

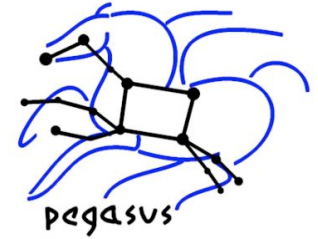
- Introduction to Pegasus WMS
- Composing a Simple Workflow In terms of DAX.
- Pegasus Internals
- Mapping and Running Workflows Locally
- Mapping and Running Workflows on the Grid
- Optimization techniques for mapping and executing Large Scale workflows

Map and Execute Workflow Locally

- Take a 4 node diamond abstract workflow (DAX) and map it to an executable workflow that runs locally.

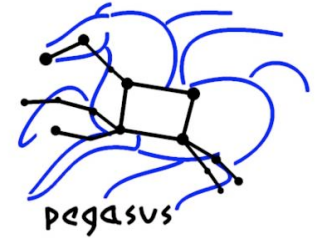


Basic Workflow Mapping



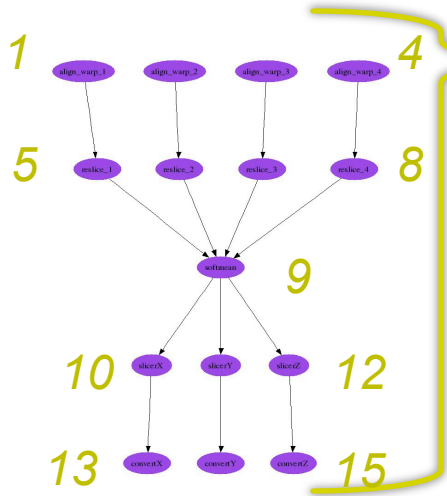
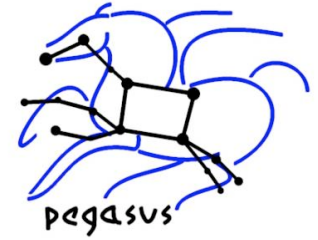
- Select where to run the computations
 - Change task nodes into nodes with executable descriptions
 - Execution location
 - Environment variables initializes
 - Appropriate command-line parameters set
- Select which data to access
 - Add stage-in nodes to move data to computations
 - Add stage-out nodes 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

Pegasus Workflow Mapping



Original workflow: 15 compute nodes devoid of resource assignment

Resulting workflow mapped onto 3 Grid sites:

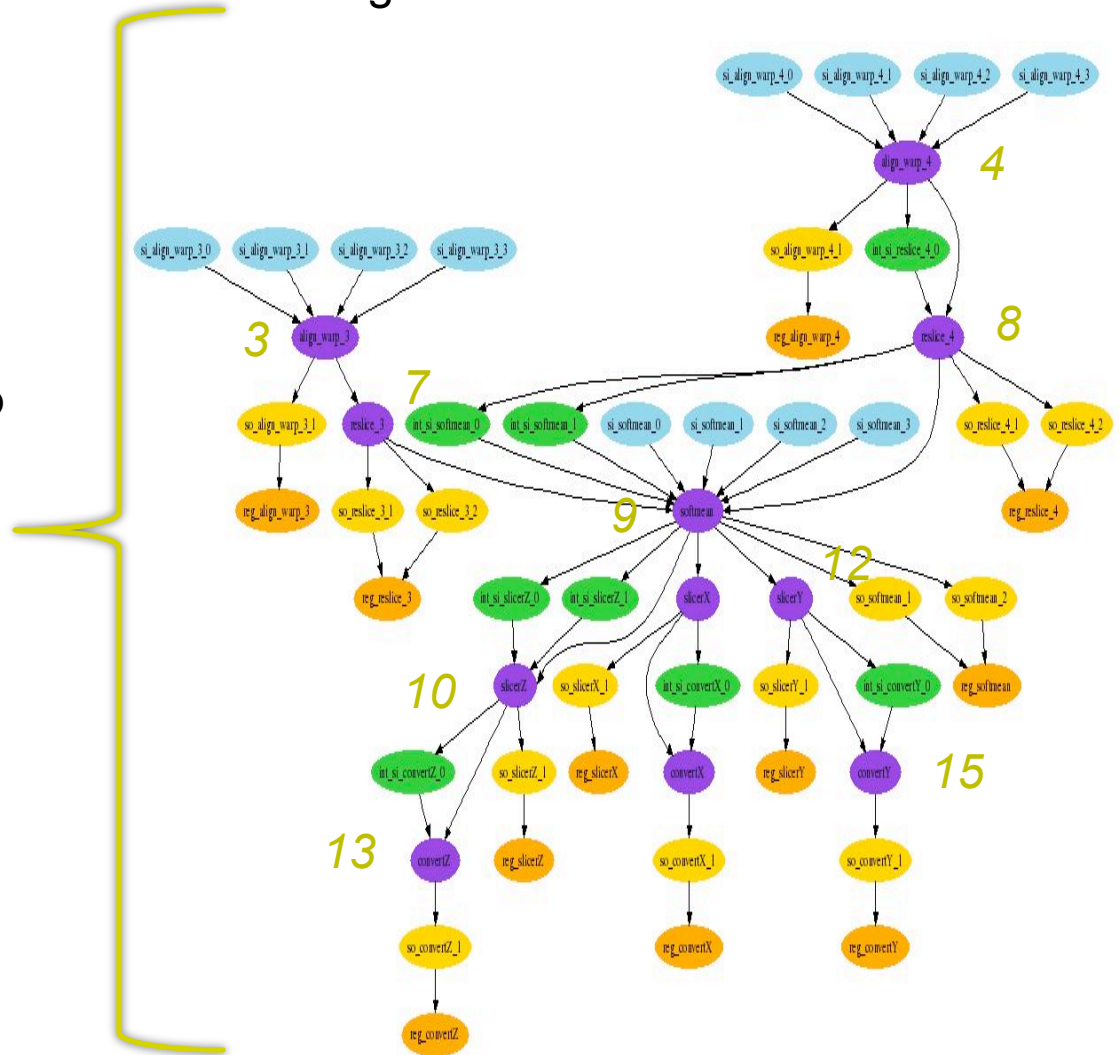
11 compute nodes (4 reduced based on available intermediate data)

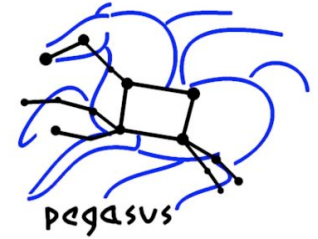
13 data stage-in nodes

8 inter-site data transfers

14 data stage-out nodes to long-term storage

14 data registration nodes (data cataloging)



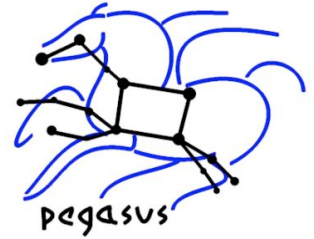


Exercise: 2.4

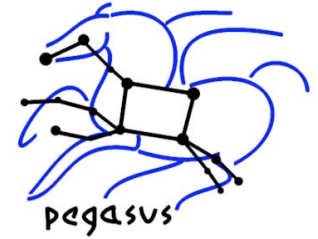
- Plan using Pegasus and submit the workflow to Condor DAGMan/CondorG for local job submissions
- **\$ pegasus-plan -Dpegasus.user.properties=<properties file> --dax <dax file> --dir <dags directory> -s local -o local --nocleanup**
- The output of pegasus-plan tells you the next command to run.
- **\$ pegasus-run -Dpegasus.user.properties=<properties file> --nodatabase <dag directory>**



Exercise: 2.5 – Monitor using Pegasus-status

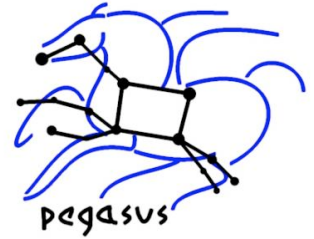


- A perl wrapper around condor_q
- Allows you to see only the jobs of a particular workflow
- Also can see what different type of jobs that are executing
- Pegasus-status <dag directory>
- Pegasus-status -w <workflow> -t <time>



Exercise: 2.5 - Debugging

- 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.
 - Jobname.out.XXX where XXX=000 to NNNN

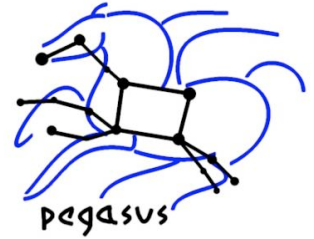


DAGMan (“under the hood” of Pegasus)

- Pegasus uses DAGMan to run the executable workflow
- Users may not have to interact with DAGMan directly...
- ...but they may (for debugging, optimization)
- Pegasus doesn't expose all DAGMan features

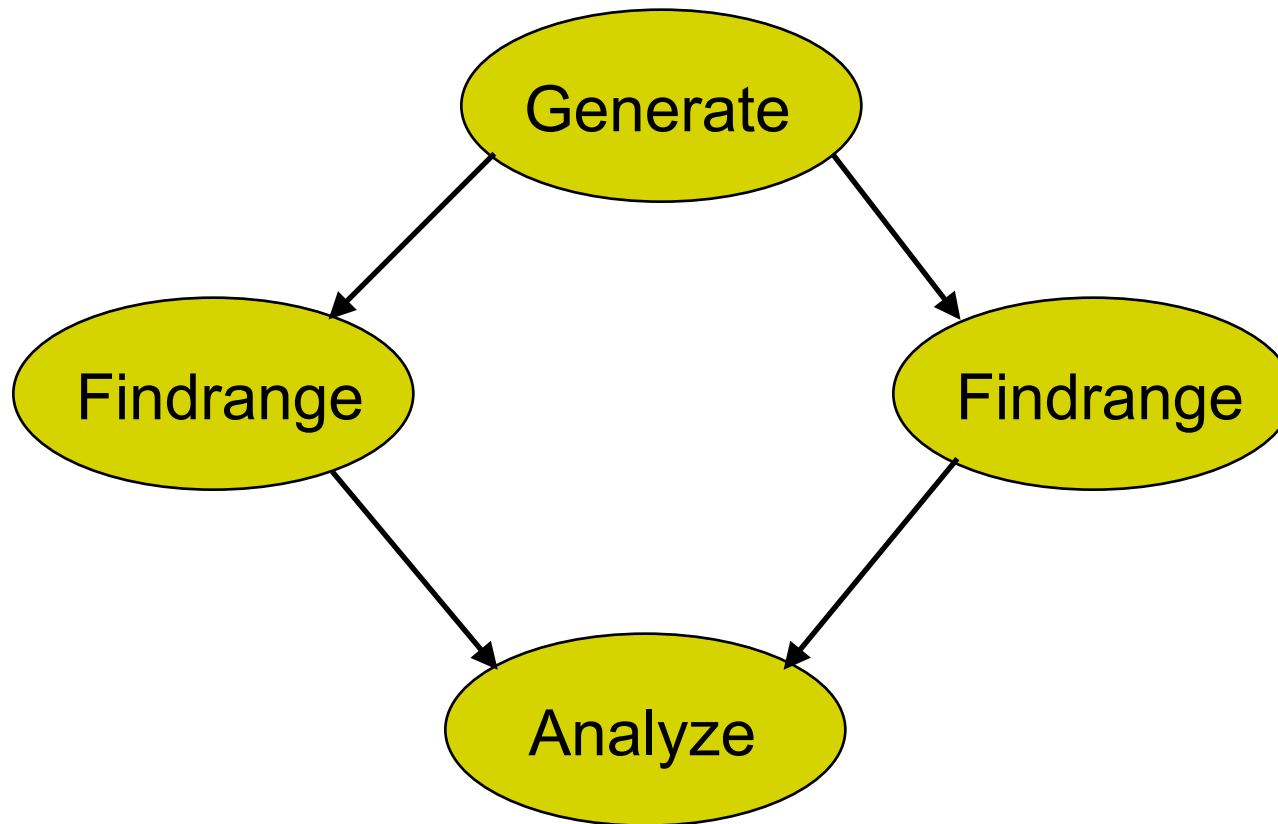


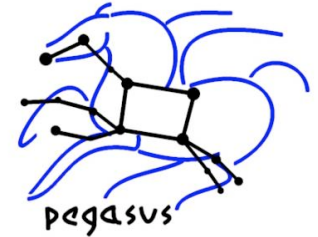
DAGMan (Directed Acyclic Graph MANager)



- 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)
- Doesn't "care" whether node jobs are local or Grid jobs

A simple DAG - Exercise 2.6





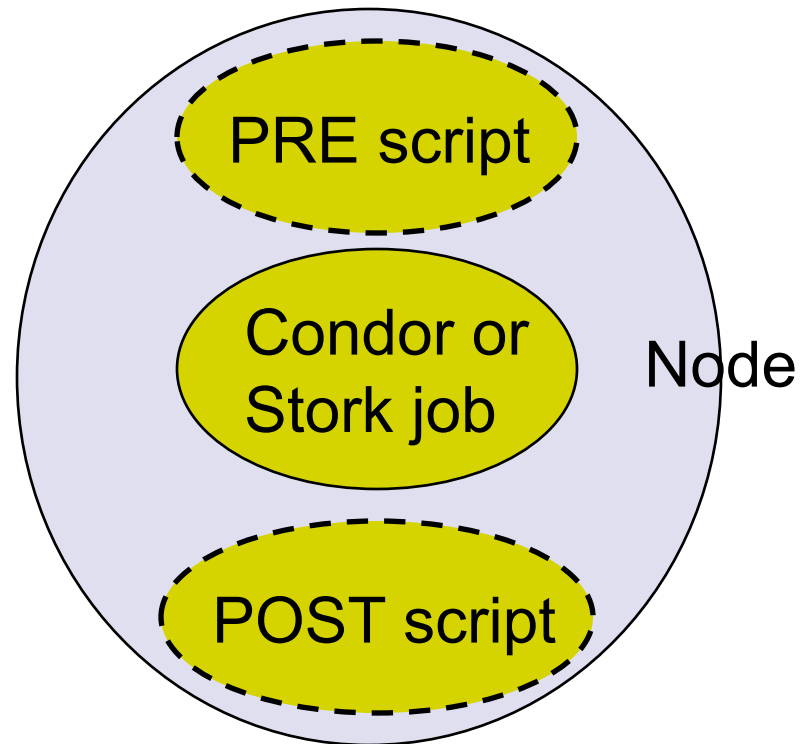
DAG file

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

```
JOB generate_ID000001 generate_ID000001.sub
JOB findrange_ID000002 findrange_ID000002.sub
JOB findrange_ID000003 findrange_ID000003.sub
JOB analyze_ID000004 analyze_ID000004.sub
JOB diamond_0_pegasus_concat diamond_0_pegasus_concat.sub
JOB diamond_0_local_cdir diamond_0_local_cdir.sub
```

```
SCRIPT POST diamond_0_local_cdir /bin/exitpost
PARENT generate_ID000001 CHILD findrange_ID000002
PARENT generate_ID000001 CHILD findrange_ID000003
PARENT findrange_ID000002 CHILD analyze_ID000004
PARENT findrange_ID000003 CHILD analyze_ID000004
PARENT diamond_0_pegasus_concat CHILD generate_ID000001
PARENT diamond_0_local_cdir CHILD diamond_0_pegasus_concat
```

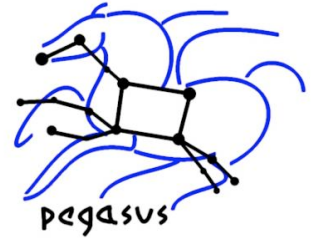
DAG node



- Treated as a unit
- Job or POST script determines node success or failure



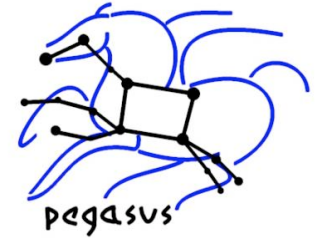
Condor_submit_dag



- Creates a Condor submit file for DAGMan
- Also submits it (unless `-no_submit` option is given)
- `-f` option forces overwriting of existing files



Condor Monitoring



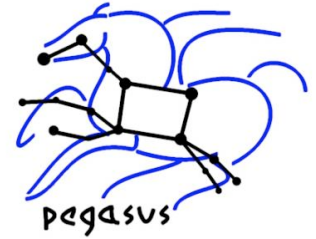
- Monitoring your DAG
 - `Condor_q -dag [name]`
 - Dagman.out file

```
% condor_q -dag train15
```

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

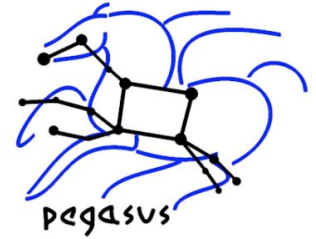
ID	OWNER/NODENAME	SUBMITTED	RUN_TIME	ST	PRI	SIZE	CMD
1860.0	train15	5/31 10:59	0+00:00:26	R	0	9.8	condor_dagman -f -
1861.0	-Setup	5/31 10:59	0+00:00:12	R	0	9.8	nodejob Setup node

```
2 jobs; 0 idle, 2 running, 0 held
```



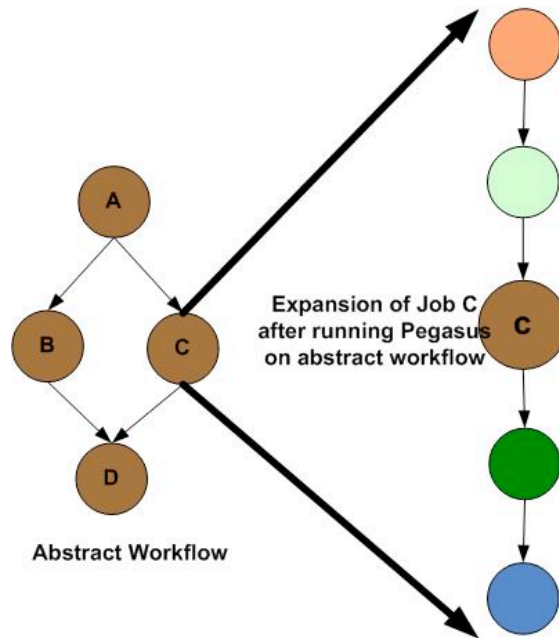
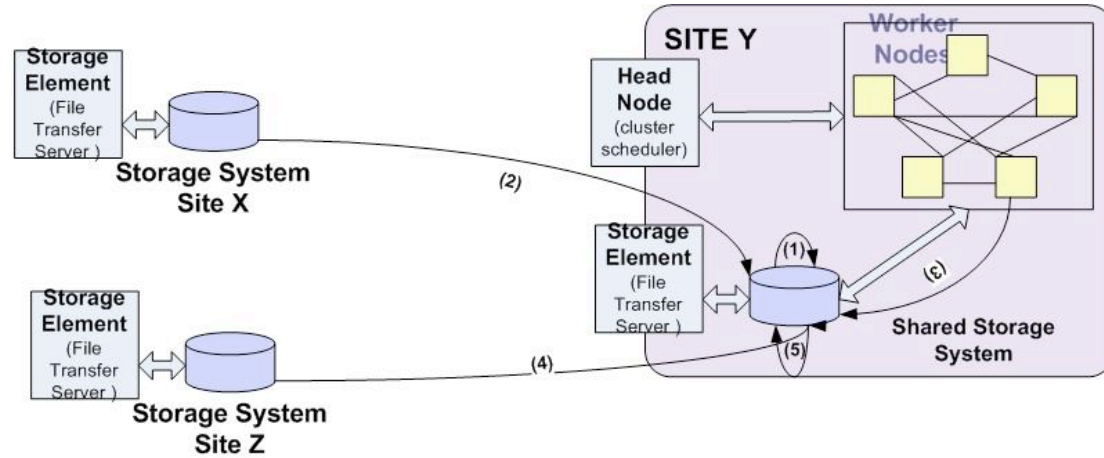
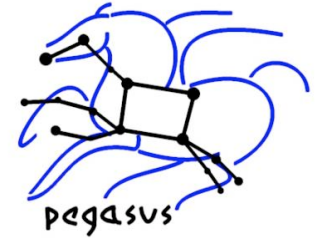
Exercise 2.7 - pegasus-remove

- Remove your workflow and associated jobs
- In future, would cleanup the remote directories that are created during workflow execution.
- Pegasus-remove <dag directory>



Outline of Tutorial

- Introduction to Pegasus WMS
- Composing a Simple Workflow In terms of DAX.
- Pegasus Internals
- Mapping and Running Workflows Locally
- Mapping and Running Workflows on the Grid
- Optimization techniques for mapping and executing Large Scale workflows



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

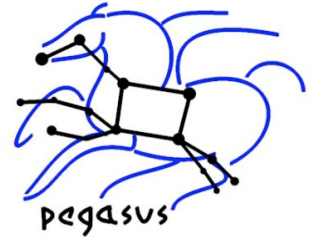
5) Removes the directory (RDy) on the shared storage system of Site Y

Execution of jobs on storage systems shared between the worker nodes and headnode / storage element.

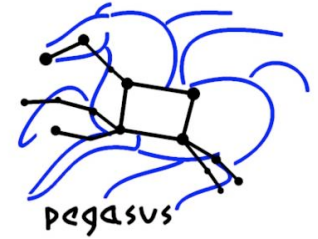
Legend	
	Visibility / Accessibility
	Compute Job
	Stage-in Job
	Stage-Out Job
	Cleanup Job
	Make Dir Job



Map and Execute Montage Workflow on Grid

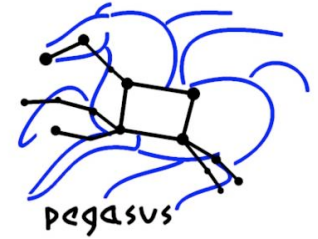


- Take a montage abstract workflow (DAX) and map it to an executable workflow that runs on the Grid.
- The available sites are viz and wind.
- You can either use a single site or a combination of these by specifying comma separated sites on the command line.



Exercise: 2.8

- Plan using Pegasus and submit the workflow to Condor DAGMan/CondorG for remote job submissions
- Pegasus-run 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.

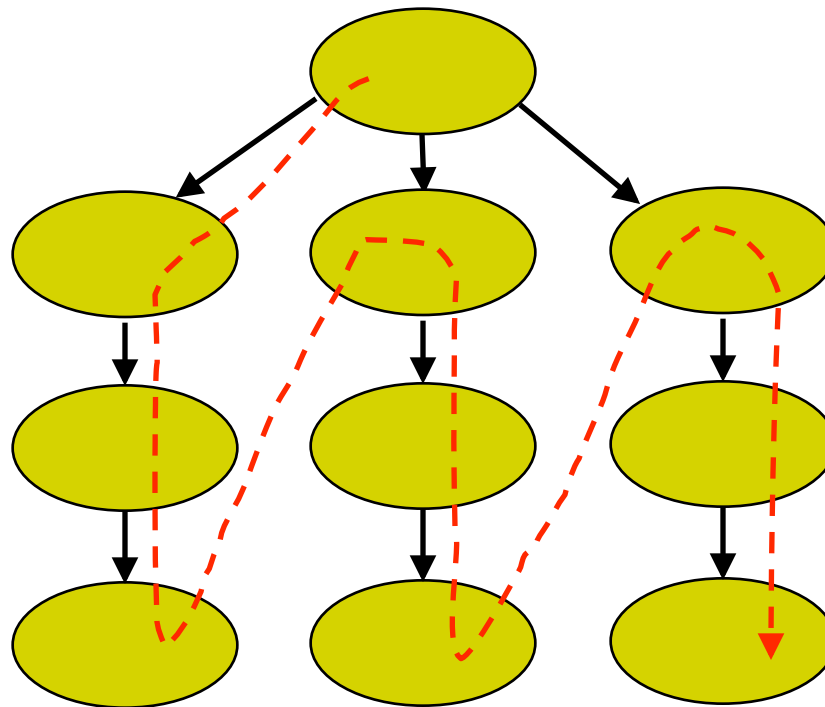


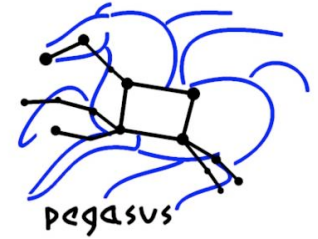
Outline of Tutorial

- Introduction to Pegasus WMS
- Composing a Simple Workflow In terms of DAX.
- Pegasus Internals
- Mapping and Running Workflows Locally
- Mapping and Running Workflows on the Grid
- Optimization techniques for mapping and executing Large Scale workflows

Depth-first DAG traversal

- Get results more quickly
- Possibly clean up intermediate files more quickly
- `DAGMAN_SUBMIT_DEPTH_FIRST=True`

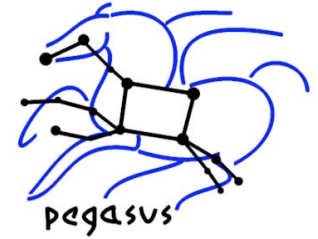




DAG node priorities

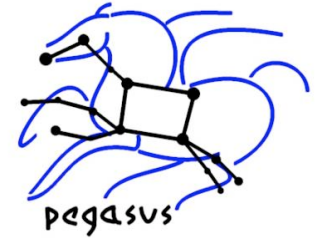
- **PRIORITY** *JobName PriorityValue*
- Determines order of submission of ready nodes
- Does *not* violate/change DAG semantics
- Mostly useful when DAG is throttled
- Higher Numerical value equals higher priority
- Version 6.9.5+

Node priorities can be configured in Pegasus Properties



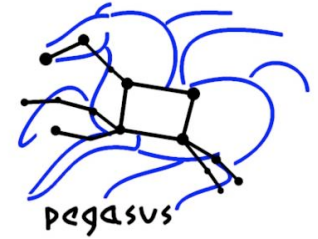
Pegasus node priority properties

- `pegasus.job.priority=<N>`
- `pegasus.transfer.stagein.priority=N`
- `pegasus.transfer.stageout.priority=N`
- `pegasus.transfer.inter.priority=N`
- `pegasus.transfer.*.priority=N`
- For each job in TC or DAX define profile
`CONDOR::priority=N`



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 to execute 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"*

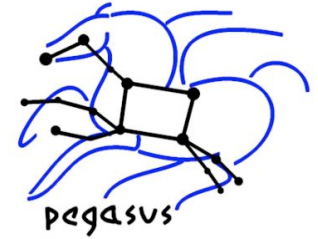


Throttling in DAGMan

- 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

The above parameters can be configured in Pegasus Properties

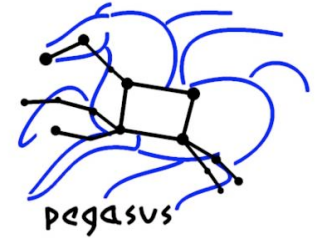
Pegasus throttling properties



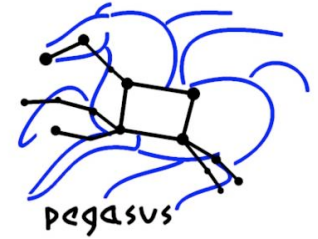
- `pegasus.dagman.maxidle`
- `pegasus.dagman.maxjobs`
- `pegasus.dagman.maxpre`
- `pegasus.dagman.maxpost`



Condor/DAGMan Throttling

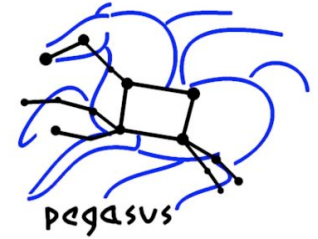


- Condor configuration files
- Environment variables
(`_CONDOR_<macroname>`)
- DAGMan configuration file (6.9.2+)
- `Condor_submit_dag` command line



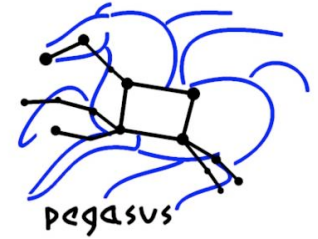
Throttling by category

- `CATEGORY JobName CategoryName`
- `MAXJOBS CategoryName MaxJobsValue`
- Applies the maxjobs setting to only jobs assigned to the given category
- Global throttles still apply
- Useful with different types of jobs that cause different loads
- Available in version 6.9.5+



PRE/POST scripts

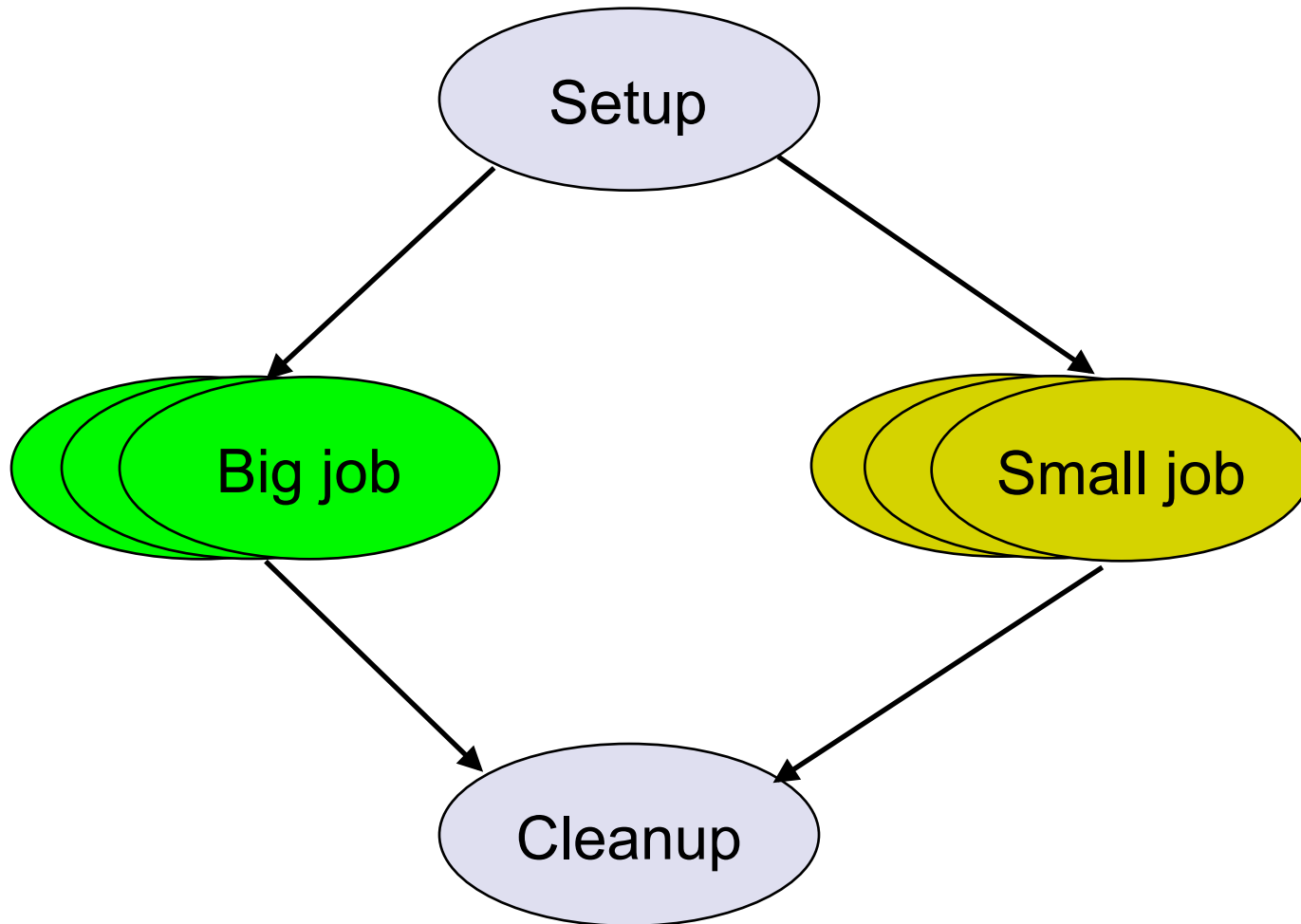
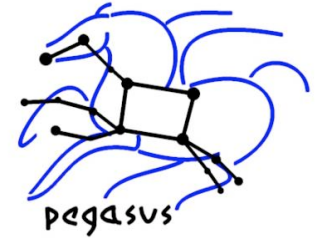
- `SCRIPT PRE|POST node script [arguments]`
- All scripts run on submit machine
- If PRE script fails, node fails w/o running job or POST script (for now...)
- If job fails, POST script is run
- If POST script fails, node fails
- Special macros:
 - `$JOB`
 - `$RETURN` (POST only)

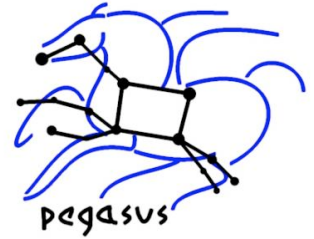


VARs (per-node variables)

- 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 cannot begin with "queue"
- Macronames are not case-sensitive

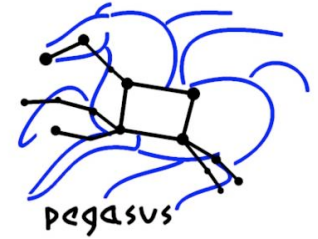
Exercise 3.1 – node categories/throttles, VARS, and scripts





Nested DAGs

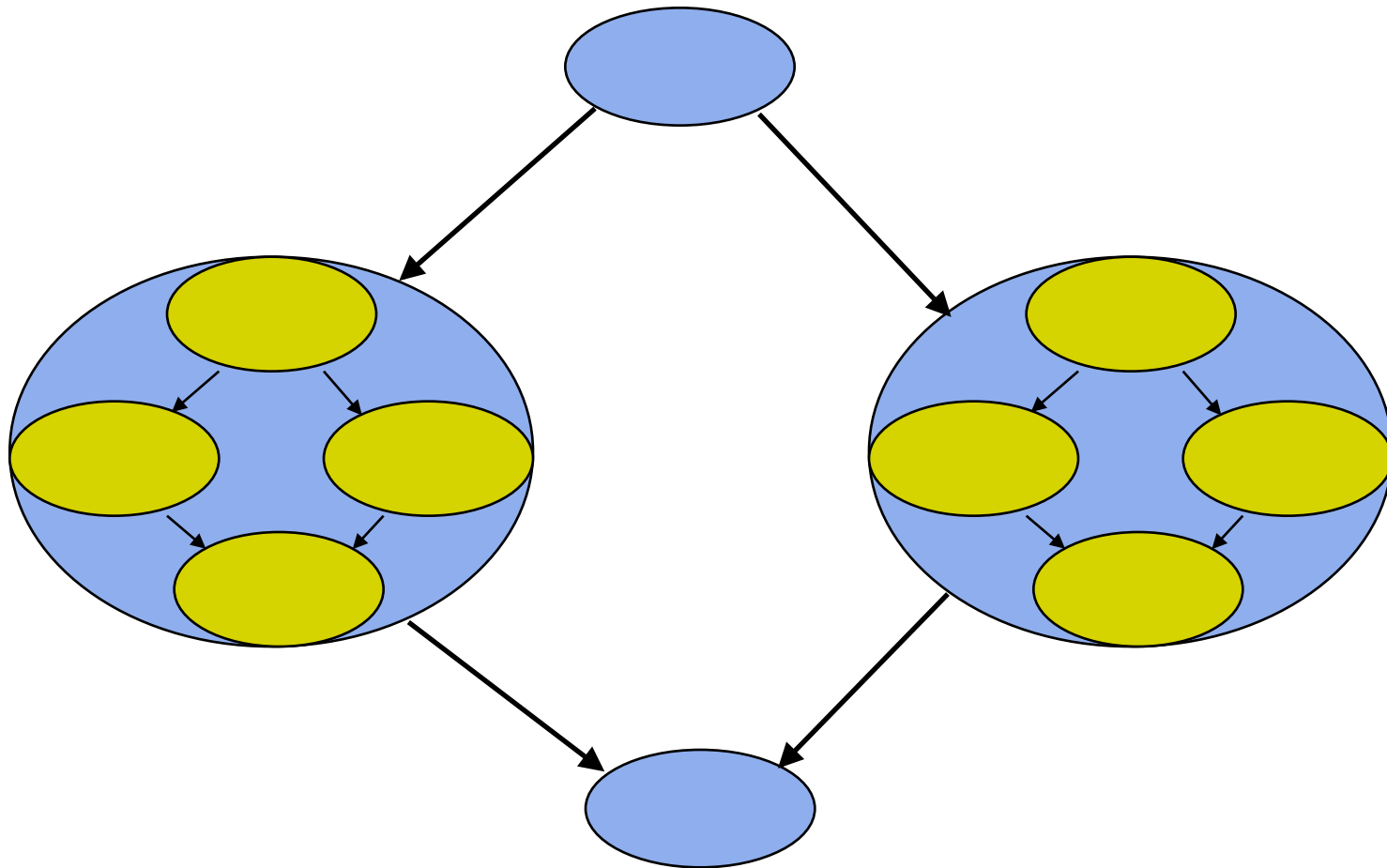
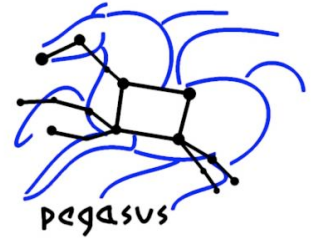
- One DAG is a node within a higher-level DAG
- `Condor_submit_dag -no_submit`
- Can be nested to arbitrary depth
- New rescue DAG semantics make this work better

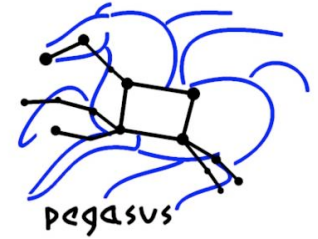


DAG config files

- In DAG file:
 - CONFIG <filename>
- In config file:
 - <macroname> = <value>
 - Any DAGMan-related config macro
- Overrides global config
- Condor_submit_dag command-line flags override this

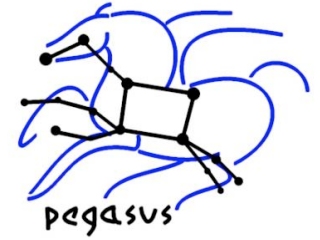
Exercise 3.2 – Nested DAGs, DAG config files, and overall throttling



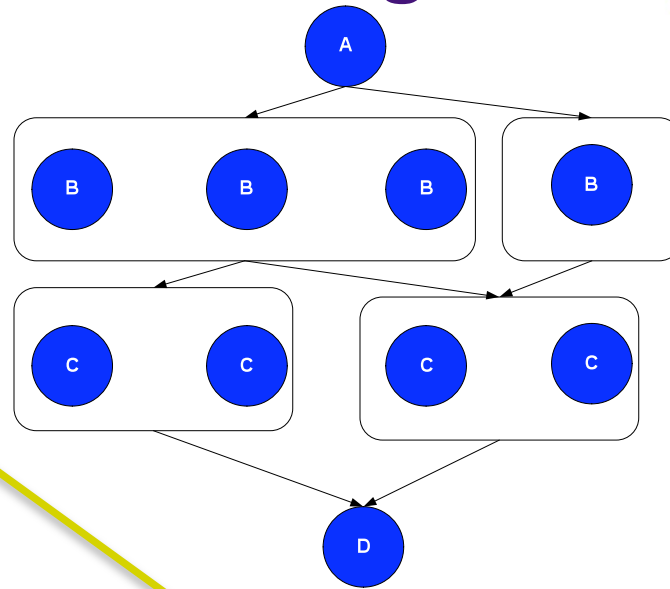
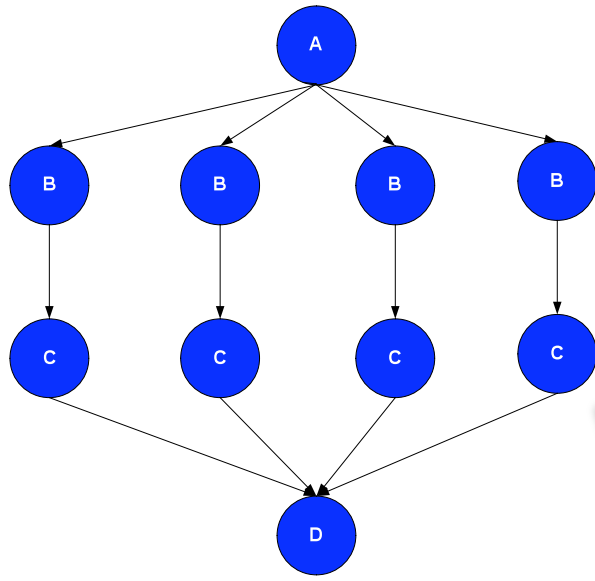


Workflow Restructuring to improve Application Performance

- 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

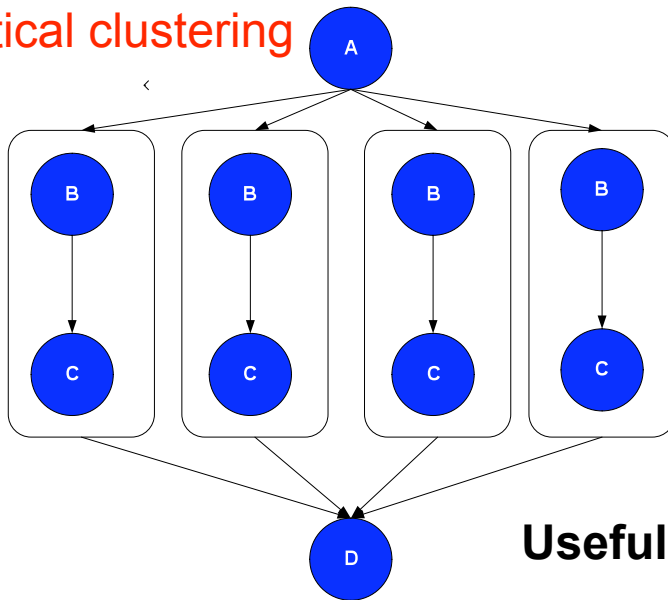


Job clustering

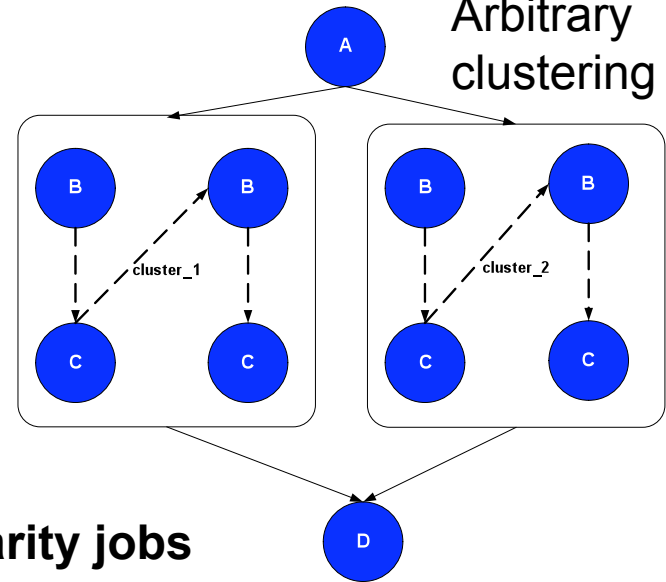


Level-based
clustering

Vertical clustering



Arbitrary
clustering

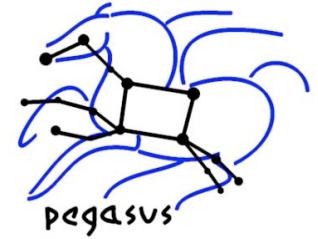


Useful for small granularity jobs

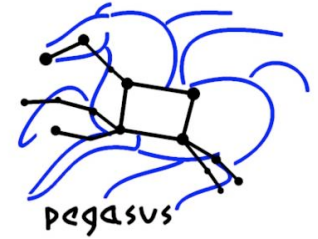


Exercise 3.3

Optional clustering exercise

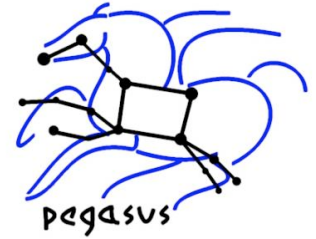


- To trigger specify `--cluster horizontal` option to `pegasus-plan`
- The granularity of clustering configured via Pegasus profile key bundle
 - Can be specified with a transformation in the transformation catalog, or with sites in the site catalog
 - Pegasus profile *bundle* specified in the site catalog.
 - Bundle means how many clustered jobs for that transformation you need on a particular site.



Transfer of Executables

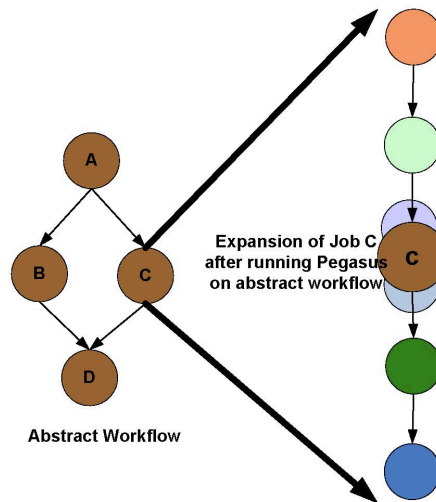
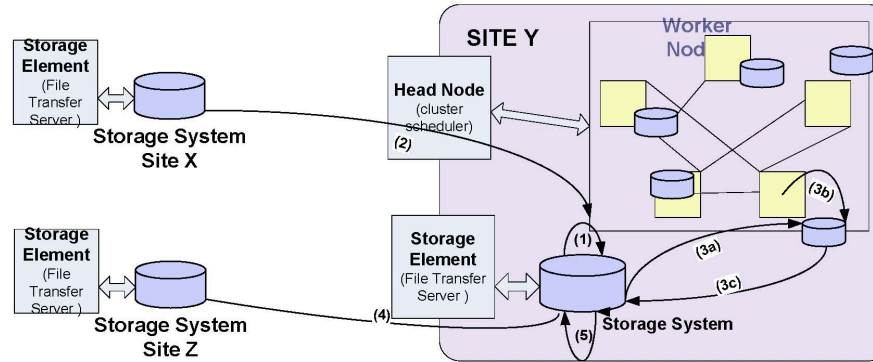
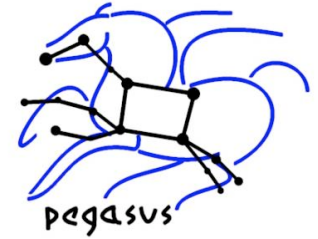
- Allows the user to dynamically deploy scientific code on remote sites
- Makes for easier debugging of scientific code.
- The executables are transferred as part of the workflow
- Currently, only statically compiled executables can be transferred
- Also we transfer any dependant executables that maybe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



Staging of executable exercise

- 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

Exercise 3.4- Running your Jobs on non shared filesystem

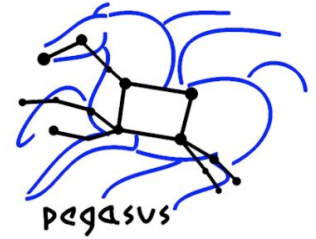


Execution of jobs on host local non shared file systems .

- 1) Creates a unique work directory (RDy) per workflow on the Storage Element at Site Y
- 2) Stages in input data for job from remote storage system at Site Y to the work directory (RDy)
- 3 a) Kickstart PRE Job that transfers staged input data from (RDy) to a host local directory on the worker node.
- 3 b) Job is executed in the host local directory of the worker node
- 3 c) Kickstart POST Job that transfers of materialized data from the host local directory of the worker node to Storage Element Directory (RDy)
- 4) Stages out Data from work directory (RDy) on the storage element at Site Y to storage system on Grid Site Z
- 5) Removes the directory (RDy) on the storage element of Site Y

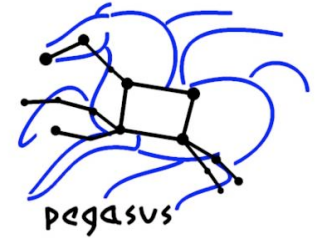
Legend	
	Visibility / Accessibility
	Compute Job
	Stage-in Job
	Stage-Out Job
	Cleanup Job
	Make Dir Job

Set the property `pegasus.execute.*.filesystem.local true`



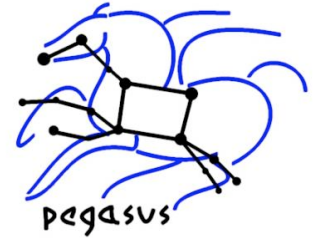
Rescue DAG

- 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
- DAGMan 7.1.0 has improvements in how rescue DAGs work



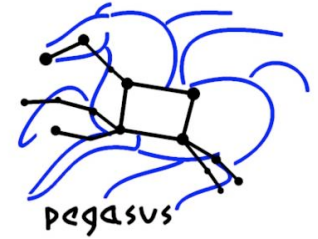
Recovery/bootstrap mode

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



Node retries

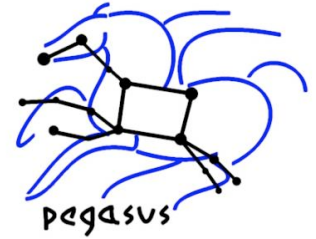
- `RETRY JobName NumberOfRetries`
`[UNLESS-EXIT value]`
- Node is retried as a whole
- `pegasus.dagman.retry=<N>`



What we're skipping

- Recursive Workflows
- Partitioning Workflows
- Multiple DAGs per DAGMan instance
- Stork
- DAG abort
- Visualizing DAGs with *dot*

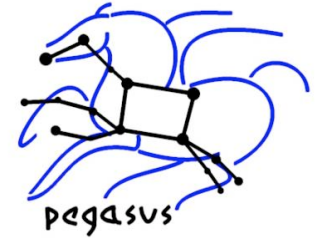
- See the Pegasus and DAGMan manuals online!



Relevant Links

- Pegasus: pegasus.isi.edu
- DAGMan: www.cs.wisc.edu/condor/dagman
- Tutorial materials available at:
<http://pegasus.isi.edu/tutorial/condor08/index.php>
- For more questions: pegasus@isi.edu

Relevant Links



- NSF Workshop on Challenges of Scientific Workflows : www.isi.edu/nsf-workflows06, E. Deelman and Y. Gil (chairs)
- Workflows for e-Science, Taylor, I.J.; Deelman, E.; Gannon, D.B.; Shields, M. (Eds.), Dec. 2006

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

