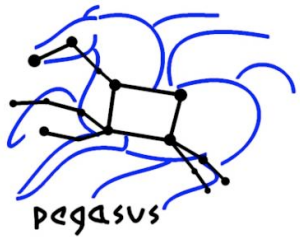
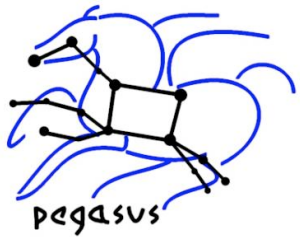


Pegasus WMS Tutorial

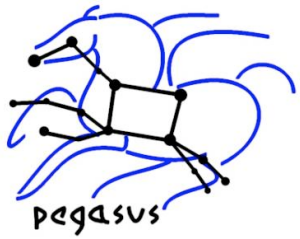
Gaurang Mehta[1], Kent Wenger [2]
(gmehta@isi.edu, wenger@cs.wisc.edu)
[1] Center for Grid Technologies, USC Information Sciences Institute
[2] University of Wisconsin Madison, Madison, WI

Outline of Tutorial



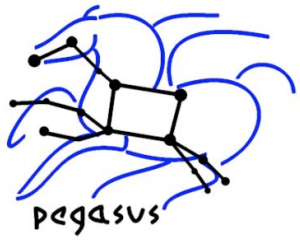
- Composing a Simple Workflow In terms of DAX.
- Pegasus Internals
- Mapping and Running Workflows Locally
- Mapping and Running Workflows on the Grid
- Optimization techniques for mapping and executing Large Scale workflows

Pegasus workflow



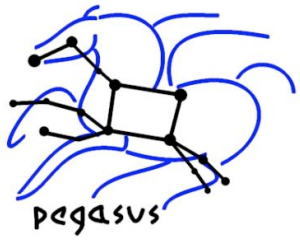
- DAX
  - What it describes
  - How to read a DAX
  - How to generate a DAX
    - Describe the various methods
      - Direct XML
      - Wings
      - DAX API
      - Behind portals
    - Migrating from a DAG to DAX

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

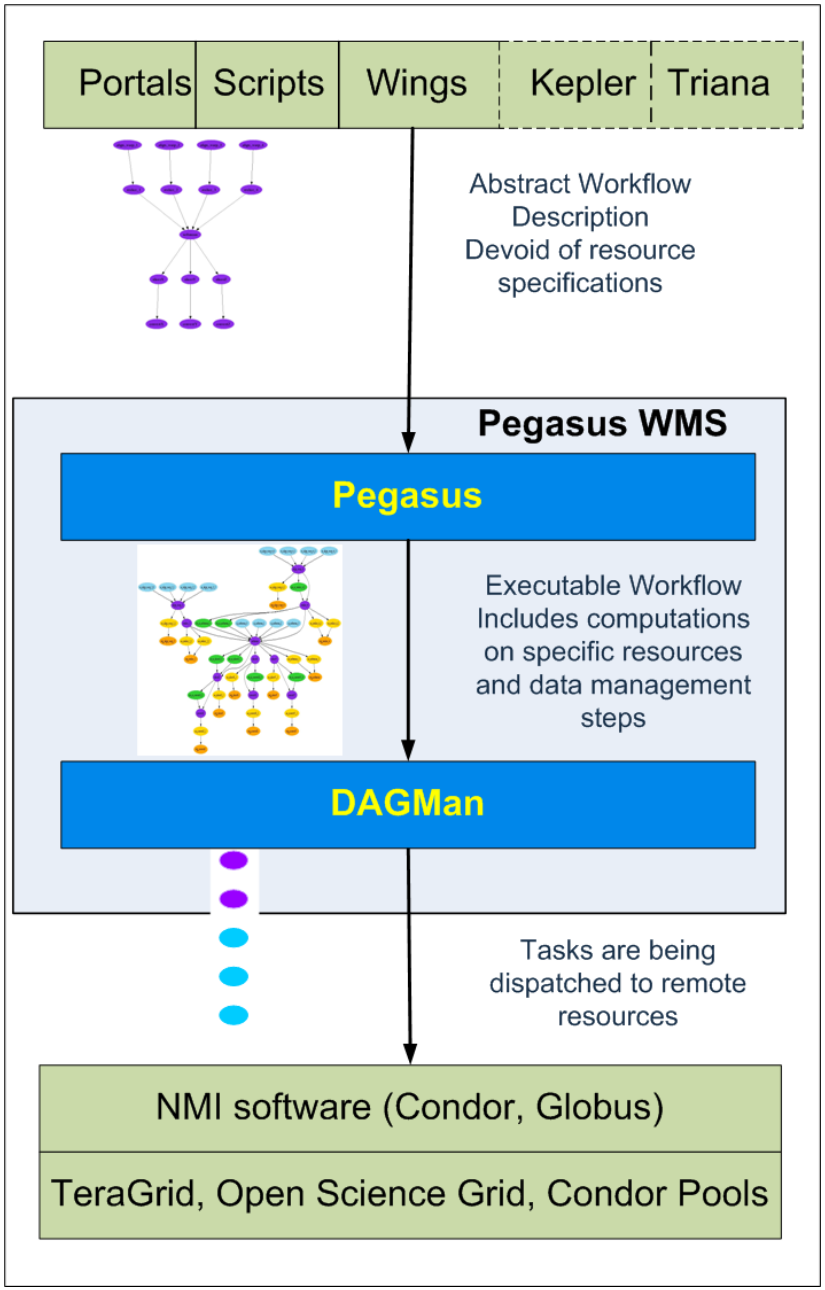
Understanding DAX (1)



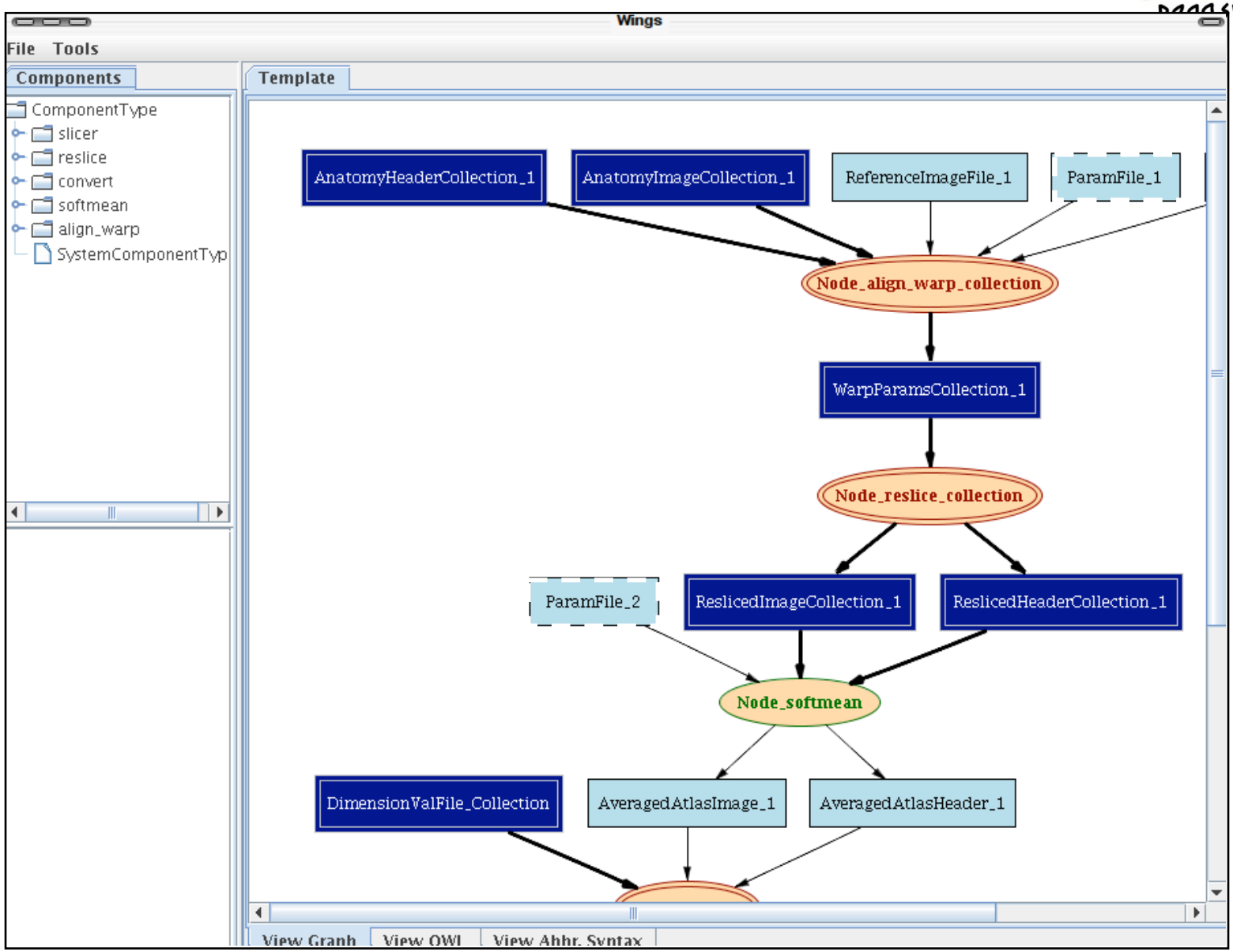

```
<!-- part 1: list of all files used (may be empty) -->
  <filename file="f.input" link="input"/>
  <filename file="f.intermediate" link="input"/>
  <filename file="f.output" link=”output"/>

<!-- 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)

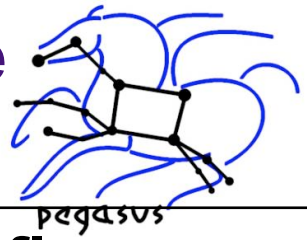
High-level system view



Creating Workflow Template with Wings GUI

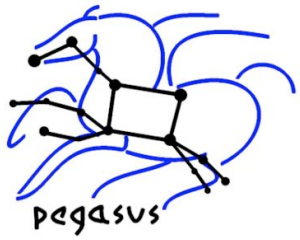


Comparison of abstract and executable workflows



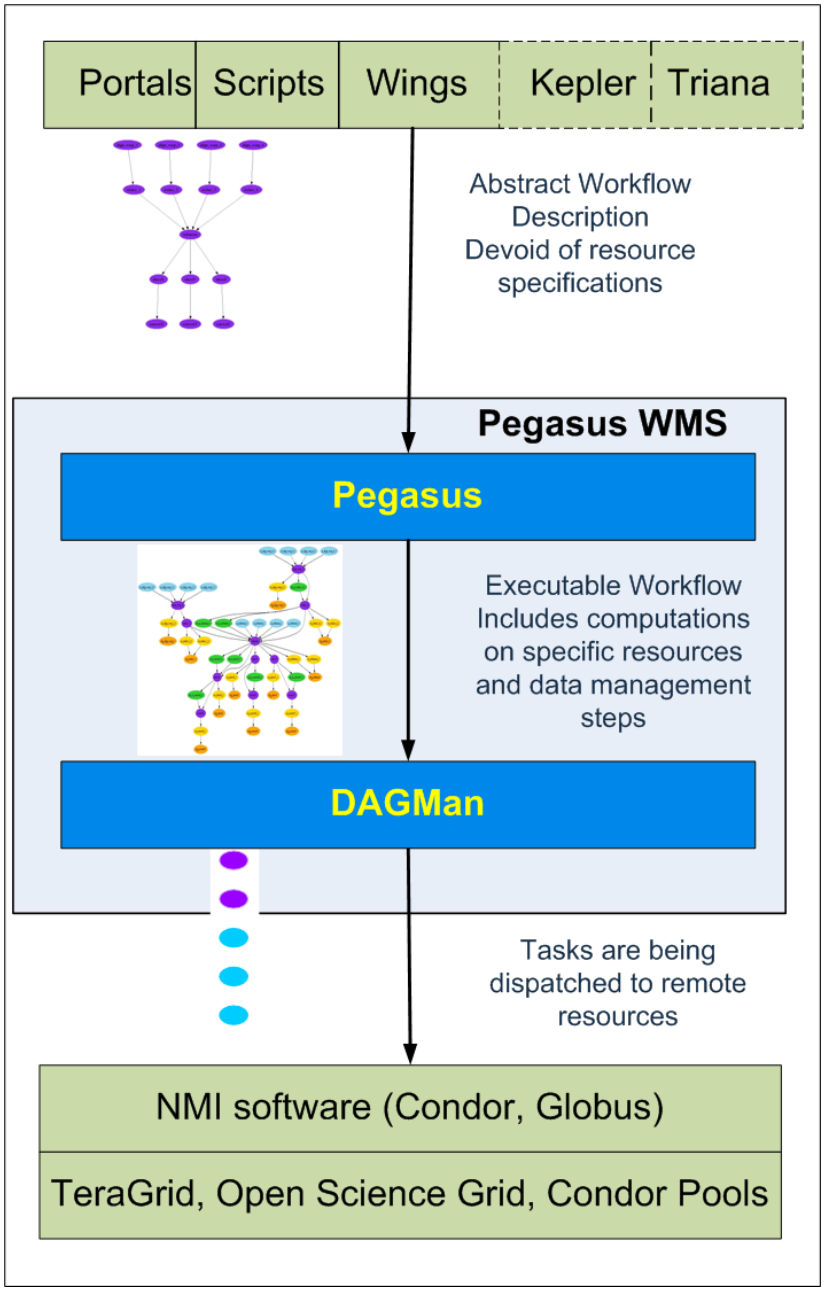
Abstract Workflow	Executable Workflow
Describes your workflow at a logical level	Describes your workflow in terms of physical files and paths
Site Independent	Site Specific
Captures just the computation that the user (you) want to do	Has additional jobs for data movement etc.

Outline of Tutorial

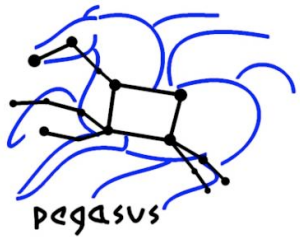


- Composing a Simple Workflow In terms of DAX.
- Pegasus Internals
- Mapping and Running Workflows Locally
- Mapping and Running Workflows on the Grid
- Optimization techniques for mapping and executing Large Scale workflows

High-level system view

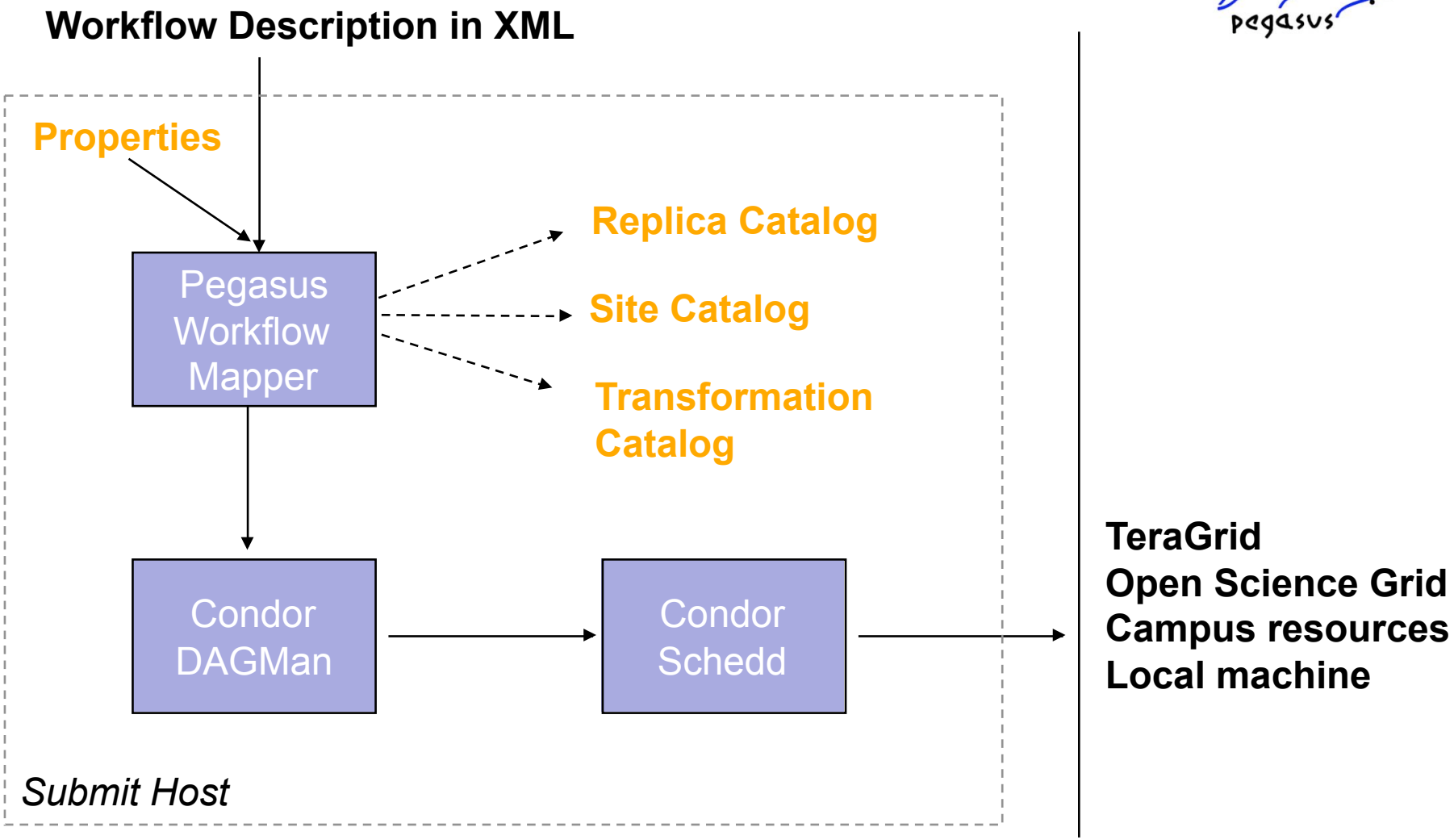


Discovery



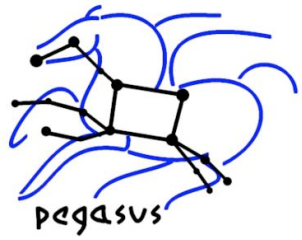
- Data
  - Where do the input datasets reside?
- Executables
  - Where are the executables installed ?
  - Do binaries exist somewhere that can be staged to remote grid sites?
- Site Layout
  - What does a grid site look like?

Pegasus WMS



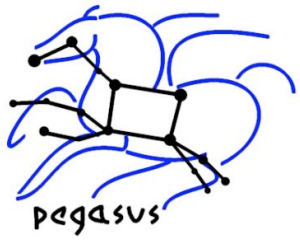
Pegasus WMS restructures and optimizes the workflow, provides reliability

Replica Catalog Overview—finding data



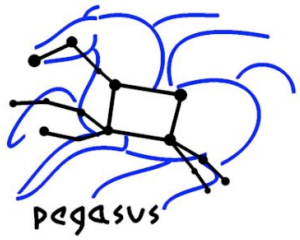
- Replica Catalog stores mappings between logical files and their target locations.
- Used to
  - discover input files for the workflow
  - track data products created
  - data reuse
- Data is replicated for scalability, reliability and availability

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.

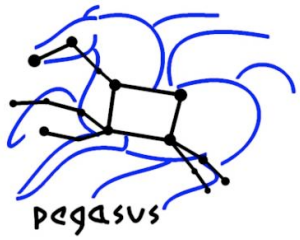
Replica Catalog Exercise: 2.2



- The rc-client is a command line tool to interact with Replica Catalog.
  - One client talks to all types of Replica Catalog

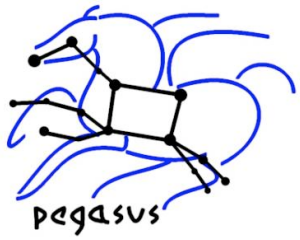
- Practical exercise (refer to Exercise 2.2):
  - Use the rc-client to
    - Populate the Replica Catalog
    - Query the Replica Catalog
    - Remove entries (offline exercise)

Site Catalog—finding resources



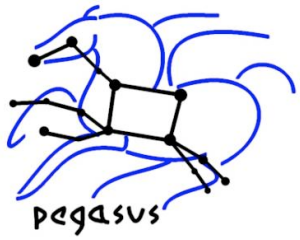
- Contains information about various sites on which workflows may execute.
- For each site following information is stored
  - Installed job-managers for different types of schedulers
  - Installed GridFTP servers
  - Local Replica Catalogs where data residing in that site has to be catalogued
  - Site Wide Profiles like environment variables
  - Work and storage directories

Site Catalog Exercise (Ex 2.3 10 minutes)



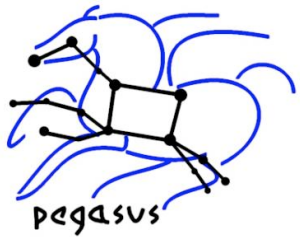
- Two clients for generating a site catalog
- pegasus-get-sites
  - Allows you to generate a site catalog
    - for OSG grid sites by querying VORS
    - for ISI skynet, TeraGrid, UC SofaGrid by querying a SQLLite2 database
- sc-client
  - Allows you to generate a site catalog
    - By specifying information about a site in a textual format in a file.
    - One file per site

Site Catalog Entry



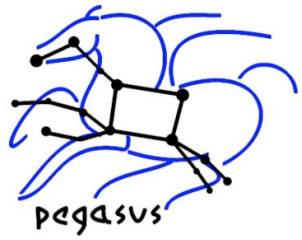

```
<site handle="isi_skynet" sysinfo="INTEL32::LINUX" gridlaunch="/nfs/software/vds/vds/bin/
    kickstart">

    <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

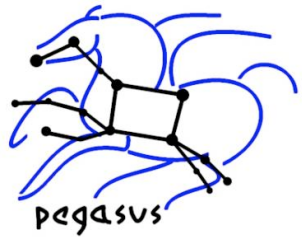
- Used to
  - discover application codes installed on the grid sites
  - discover statically compiled codes, that can be deployed at grid sites on demand



Transformation Catalog Overview

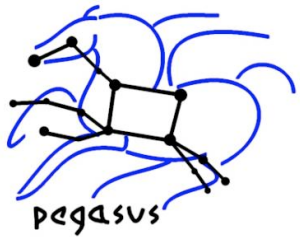
- For each transformation following are stored
  - logical name of the transformation
  - Type of transformation (INSTALLED or STATIC_BINARY)
  - Architecture, OS, Glibc version
  - the resource on the which the transformation is available
  - the URL for the physical transformation
  - Profiles that associate runtime parameters like environment variables, scheduler related information

Transformation Catalog Exercise (Offline)



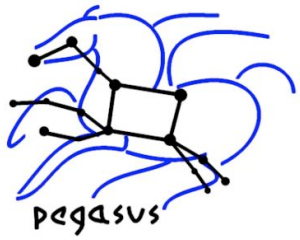
- tc-client is a command line client that is primarily used to configure the database TC
- Works even for file based transformation catalog.

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

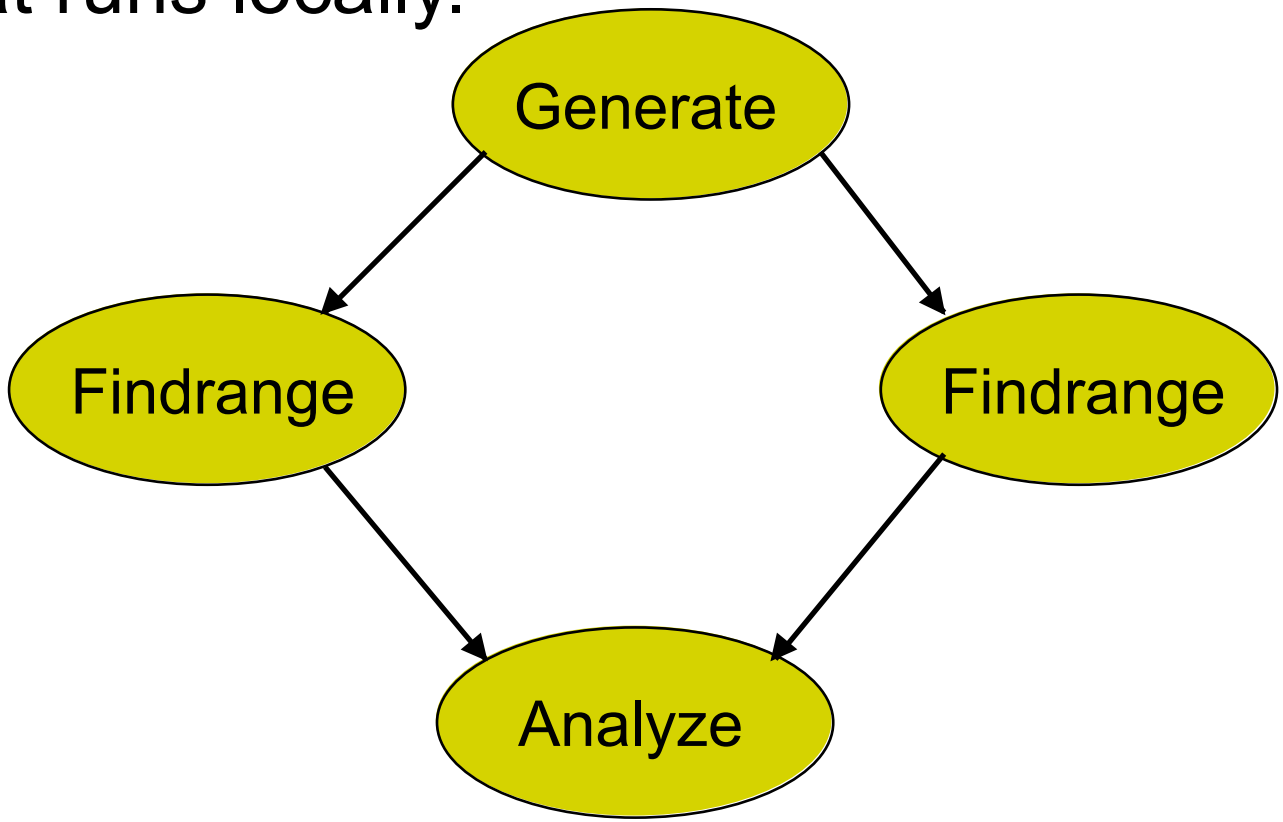
Outline of Tutorial



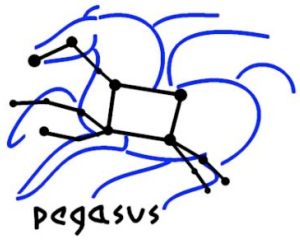
- Composing a Simple Workflow In terms of DAX.
- Pegasus Internals
- Mapping and Running Workflows Locally
- Mapping and Running Workflows on the Grid
- Optimization techniques for mapping and executing Large Scale workflows

Map and Execute Workflow Locally

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

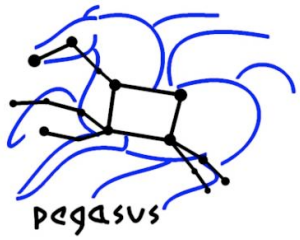


Basic Workflow Mapping



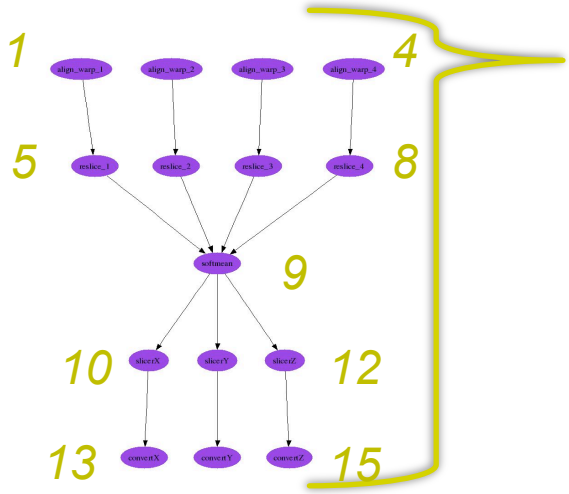
- Select where to run the computations
  - Change task nodes into nodes with executable descriptions
    - Execution location
    - Environment variables initializes
    - Appropriate command-line parameters set
- Select which data to access
  - Add stage-in nodes to move data to computations
  - Add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

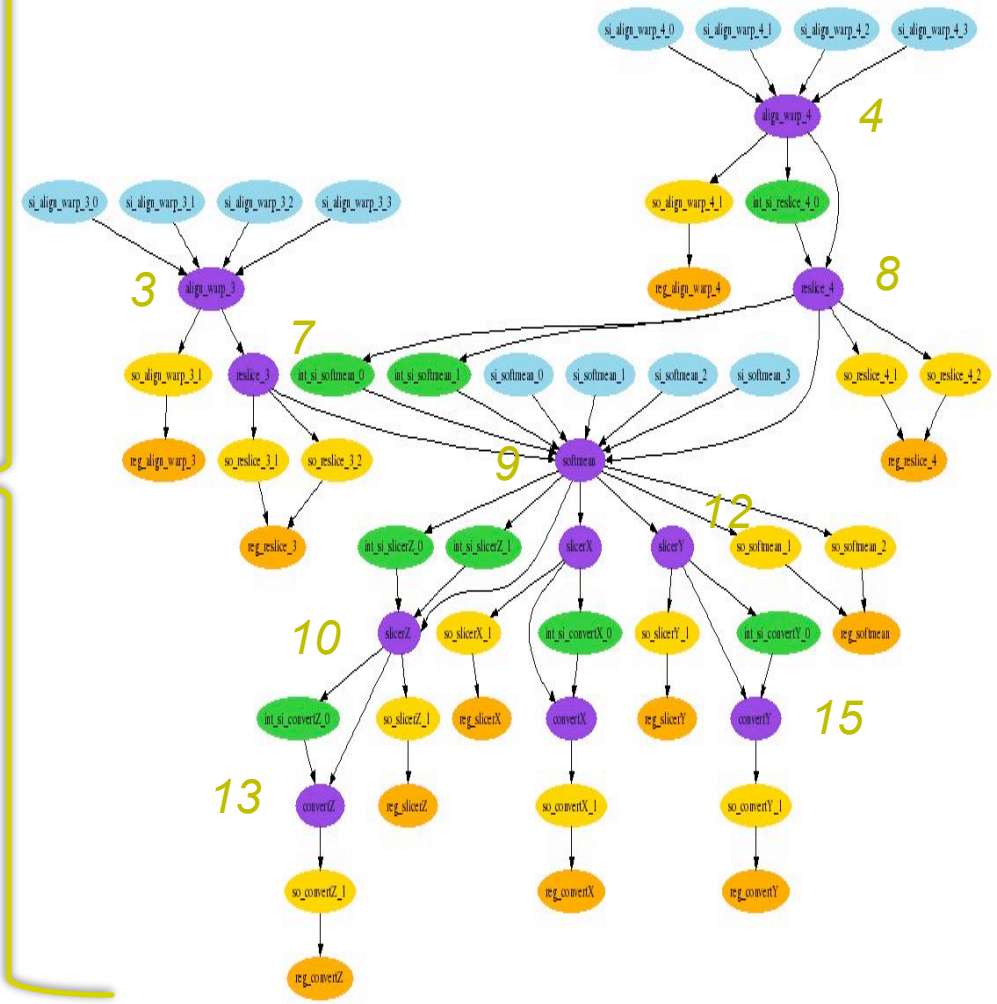
Pegasus Workflow Mapping



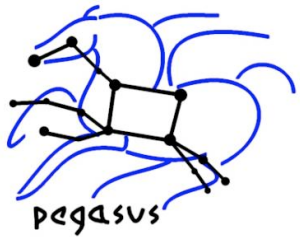
Original workflow: 15 compute nodes devoid of resource assignment

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

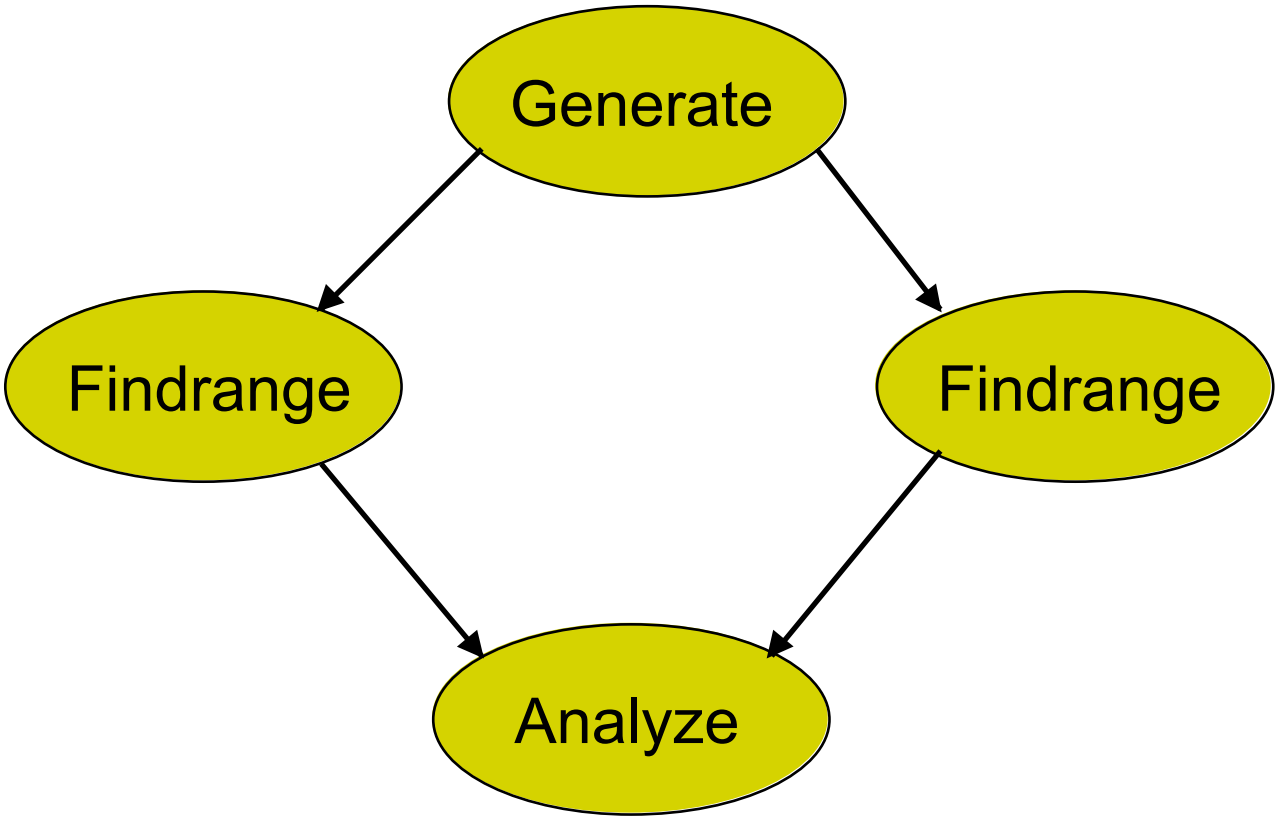


- Plan using Pegasus and submit the workflow to Condor DAGMan/CondorG for local job submissions

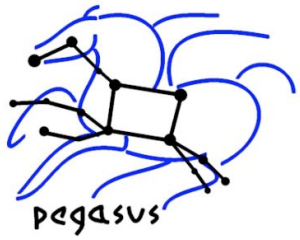
- **$ pegasus-plan -Dpegasus.user.properties=<properties file> --dax <dax file> --dir <dags directory> -s local –o local --nocleanup**

- **$ pegasus-run –Dpegasus.user.properties=<properties file> --nodatabase <dag directory>**

A simple DAG (Exercise 2.5)



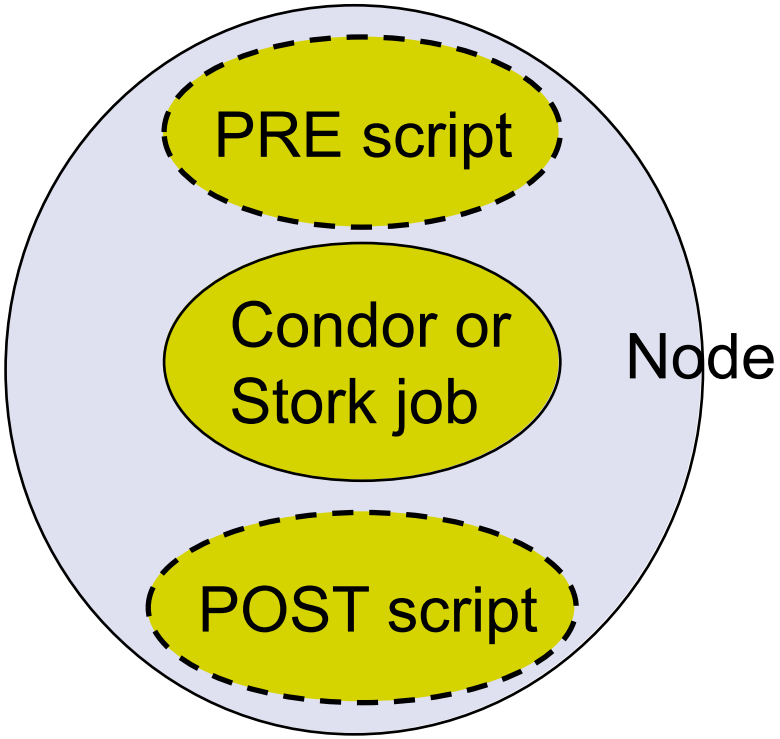
DAG file



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

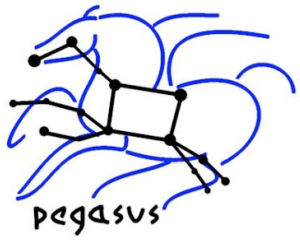
```
JOB generate_ID000001 generate_ID000001.sub
JOB findrange_ID000002 findrange_ID000002.sub
JOB findrange_ID000003 findrange_ID000003.sub
JOB analyze_ID000004 analyze_ID000004.sub
JOB diamond_0_pegasus_concat diamond_0_pegasus_concat.sub
JOB diamond_0_local_cdir diamond_0_local_cdir.sub

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



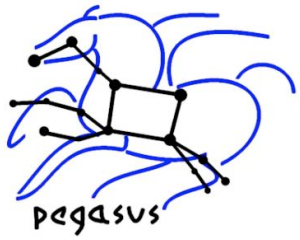
- Treated as a unit
- Job or POST script determines node success or failure

Condor_submit_dag



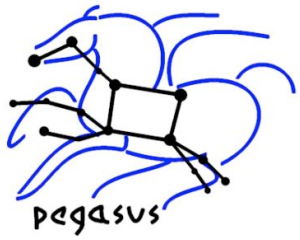
- Creates a Condor submit file for DAGMan
- Also submits it (unless –no_submit option is given)
- -f option forces overwriting of existing files

Condor Monitoring - 1



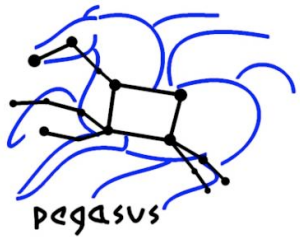
- Monitoring your Workflow jobs
  - Condor_q [*name*]
  - Condor_history [*name*]

Condor Monitoring - 2




```
% condor_q train15


-- Submitter: train15@isi.edu : <128.9.72.178:43684> : viz-login.isi.edu
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD
1859.0   train15         5/31 10:53   0+00:00:07 R  0   9.8  nodejob Miguel Ind

1 jobs; 0 idle, 1 running, 0 held

...

% condor_q train15


-- Submitter: train15@isi.edu : <128.9.72.178:43684> : viz-login.isi.edu
 ID      OWNER            SUBMITTED     RUN_TIME ST PRI SIZE CMD

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

Condor Monitoring - 3

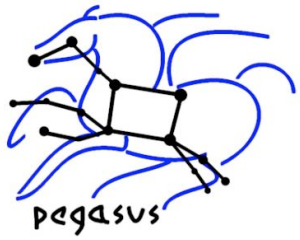



```
% condor_history train15
 ID      OWNER            SUBMITTED     RUN_TIME ST   COMPLETED CMD
1015.0   train15         5/28 11:34   0+00:01:00 C   5/28 11:35 /nfs/home/train
1017.0   train15         5/28 11:45   0+00:01:00 C   5/28 11:46 /nfs/home/train
1018.0   train15         5/28 11:46   0+00:01:00 C   5/28 11:47 /nfs/home/train
...
```

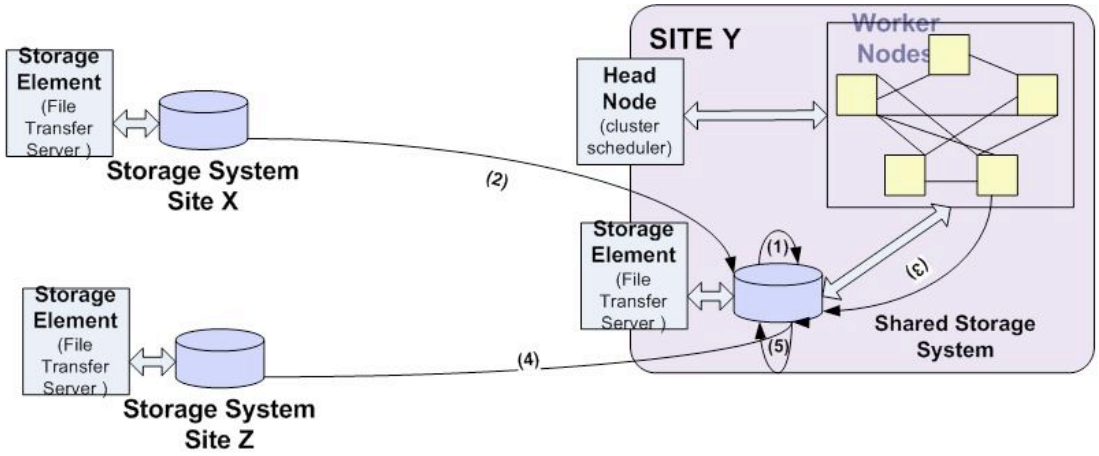
- Monitoring your DAG
  - Condor_q –dag [*name*]
  - Dagman.out file

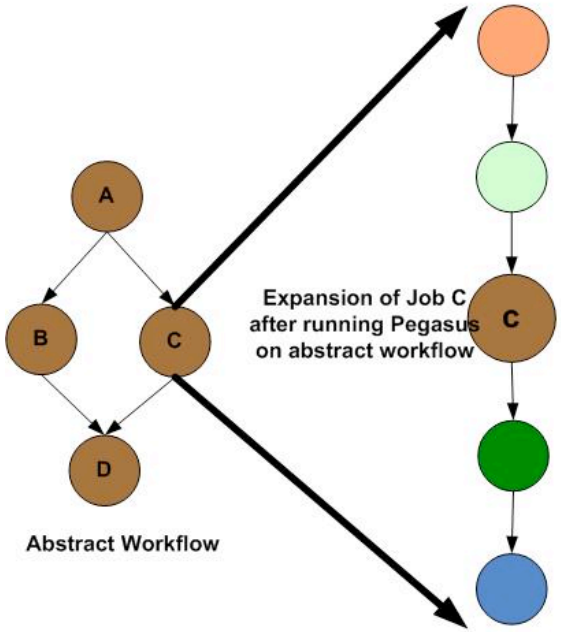
```
% condor_q -dag train15


-- 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
```

Outline of Tutorial



- 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





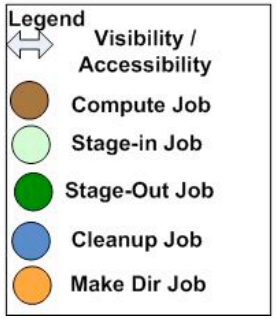
1) Creates a unique work directory (RDy) per workflow on the Shared Storage System of Site Y

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

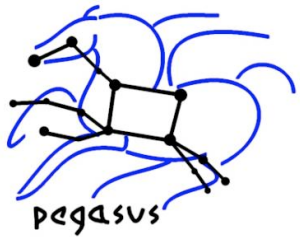
4) Stages out Data from work directory (RDy) on the shared storage system to storage system on Grid Site Z

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



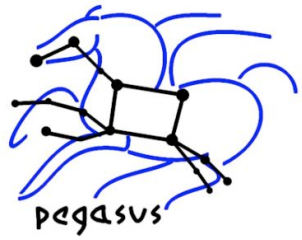
Execution of jobs on storage systems shared between the worker nodes and headnode / storage element.

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, OSG_LIGO_MIT and Uflorida_HPC.
- You can either use a single site or a cobmination of these by specifying comma seperates sites on the command line.

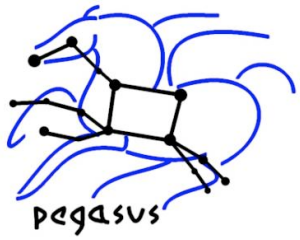
Condor Grid Universe



- Simplest grid universe submit file:

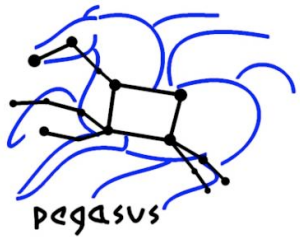
```
universe = grid
grid_resource = gt2 <gatekeeper-machine>/jobmanager-fork
executable = /bin/date
output = test.out
queue
```

Exercise: 2.6



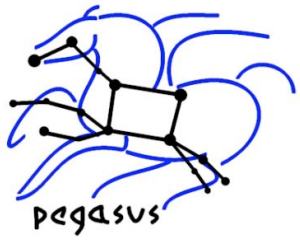
- Plan using Pegasus and submit the workflow to Condor DAGMan/CondorG for remote job submissions
- Pegasus-run starts the monitoring daemon (tailstatd) in the directory containing the condor submit files
- Tailstatd parses the condor output and updates the status of the workflow to a database
- Tailstatd updates job status to a text file jobstate.log in the directory containing the condor submit files.

Exercise: 2.7 - (Monitor) 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: 2.7- Debugging

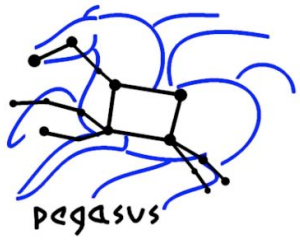


- The status of the workflow can be determined by
  - Looking at the jobstate.log
  - Or looking at the dagman out file (with suffix .dag.dagman.out)

- 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.

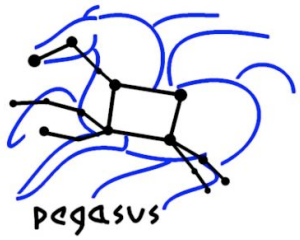
- In case of job failure look at kickstart output of the failed job.

Exercise 2.8 - pegasus-remove



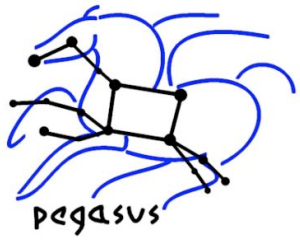
- Remove your workflow and associated jobs
- In future, would cleanup the remote directories that are created during workflow execution.
- Pegasus-remove <dag directory>

Outline of Tutorial



- 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

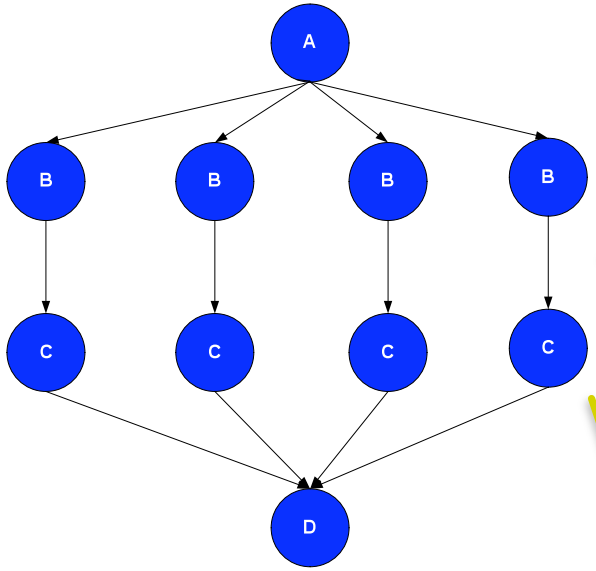
Workflow Restructuring to improve Application Performance

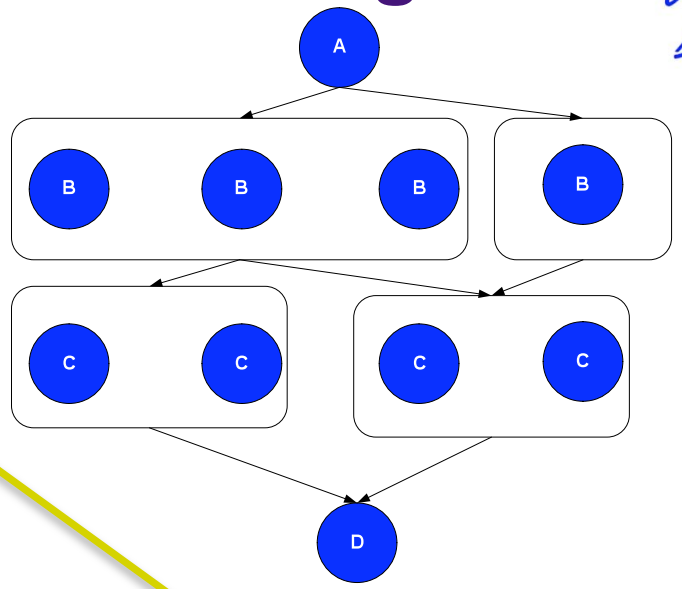


- Cluster small running jobs together to achieve better performance.

- 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

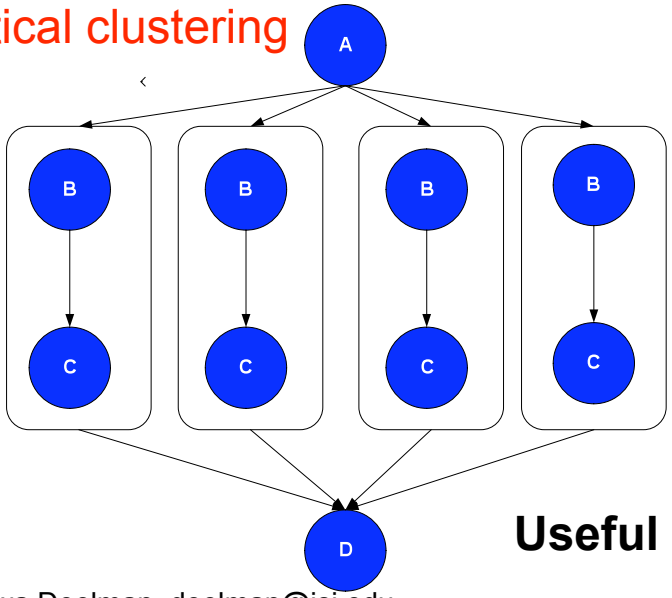
Job clustering



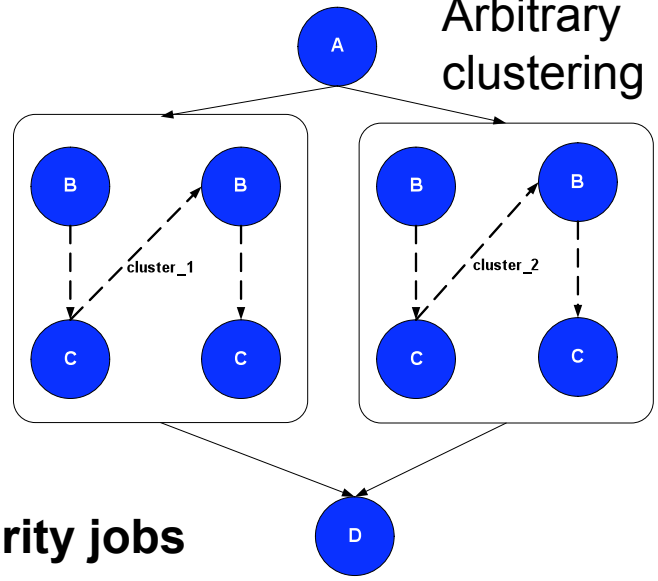


Level-based clustering

Vertical clustering



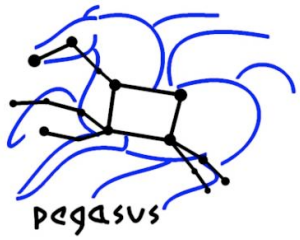
Arbitrary clustering



Useful for small granularity jobs

Ewa Deelman, deelman@isi.edu

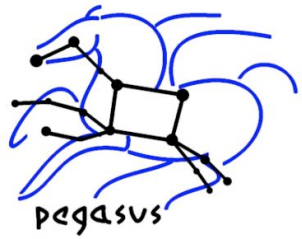
Exercise 3.1
Optional clustering exercise



- To trigger specify --cluster horizontal option to pegasus-plan

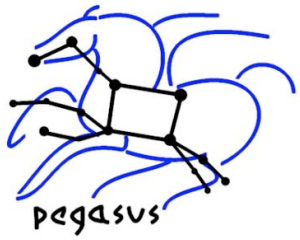
- The granularity of clustering configured via Pegasus profile key bundle
  - Can be specified with a transformation in the transformation catalog, or with sites in the site catalog
  - Pegasus profile *bundle* specified in the site catalog.
  - Bundle means how many clustered jobs for that transformation you need on a particular site.

Transfer of Executables



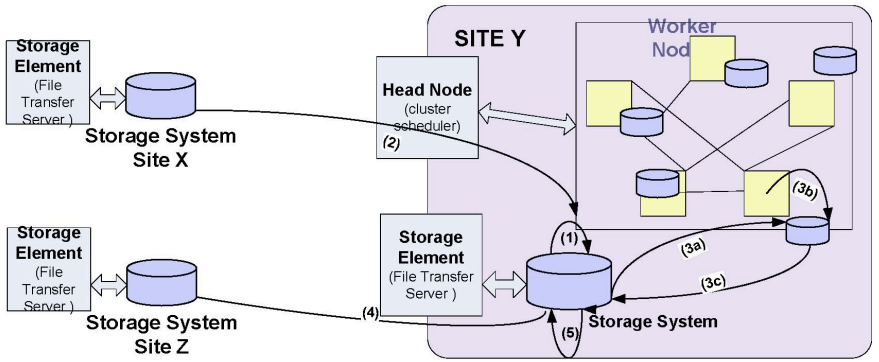
- Allows the user to dynamically deploy scientific code on remote sites
- Makes for easier debugging of scientific code.
- The executables are transferred as part of the workflow
- Currently, only statically compiled executables can be transferred
- Also we transfer any dependant executables that maybe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

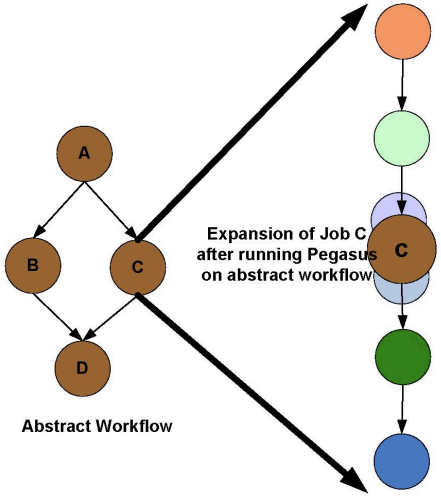
Staging of executable exercise



- 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 3.2- Running your Jobs on non shared filesystem





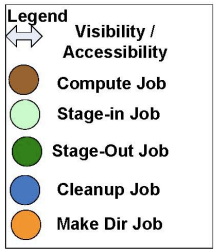
1) Creates a unique work directory (RDy) per workflow on the Storage Element at Site Y

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



Execution of jobs on host local non shared file systems .

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

Transfer Throttling

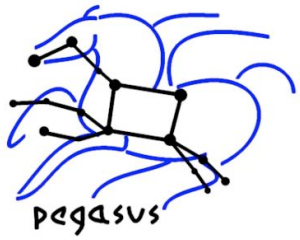
- Large-sized workflows result in large number of transfer jobs being executed at once. Results in:
  - Grid FTP server overload (connection refused errors etc)
  - May result in a high load on the head node if transfers are not configured to execute as third party transfers
- Need to throttle transfers
  - 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

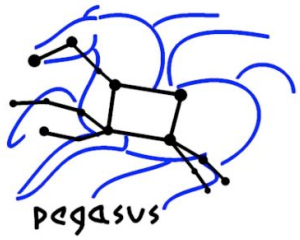
The above parameters can be configured in Pegasus Properties

Pegasus throttling properties



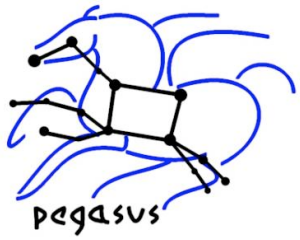
- pegasus.dagman.maxidle
- pegasus.dagman.maxjobs
- pegasus.dagman.maxpre
- pegasus.dagman.maxpost

Condor/DAGMan Throttling



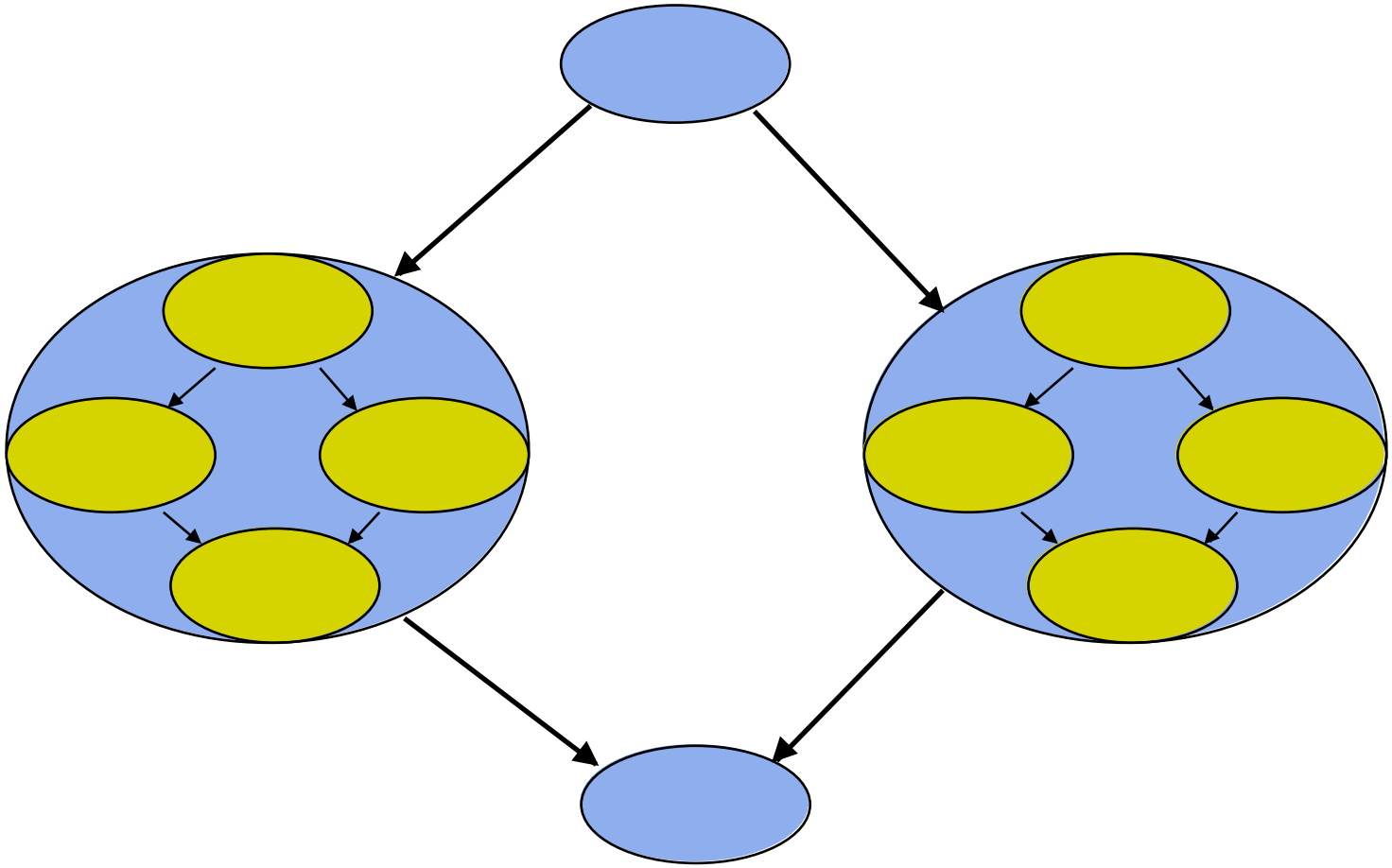
- Condor configuration files
- Environment variables (_CONDOR_<macroname>)
- DAGMan configuration file (6.9.2+)
- Condor_submit_dag command line

Node retries

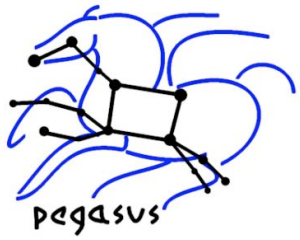


- RETRY *JobName NumberOfRetries* [UNLESS-EXIT *value*]
- Node is retried as a whole
- pegasus.dagman.retry=<N>

Nested DAGs

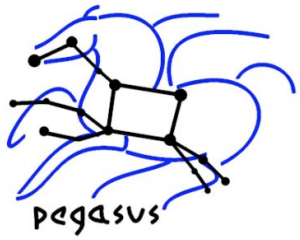


PRE/POST scripts



- 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)

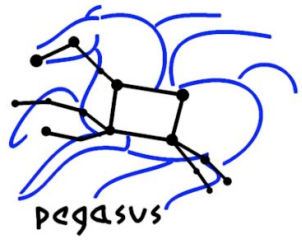
Exercise 3.3 - PRE/POST scripts



- Proc2 job will fail, but POST script will not

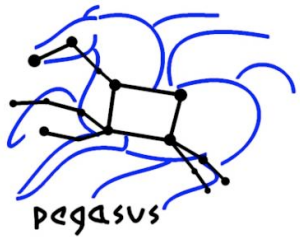
```
% condor_submit_dag -f scripts.dag
Checking all your submit files for log file names.
This might take a while...
Done.
-----------------------------------------------------------------------
File for submitting this DAG to Condor           : exD2.dag.condor.sub
Log of DAGMan debugging messages                 : exD2.dag.dagman.out
Log of Condor library debug messages             : exD2.dag.lib.out
Log of the life of condor_dagman itself          : exD2.dag.dagman.log

Condor Log file for all jobs of this DAG         :
                          /scratch/train15/tg07_tutorial/ex3/job.log
Submitting job(s).
Logging submit event(s).
1 job(s) submitted to cluster 1905.
-----------------------------------------------------------------------
```

Exercise 3.3, continued




```
% more scripts.dag
# DAG with PRE and POST scripts.

JOB Setup setup.submit
SCRIPT PRE Setup pre_script $JOB
SCRIPT POST Setup post_script $JOB $RETURN

JOB Proc1 proc1.submit
SCRIPT PRE Proc1 pre_script $JOB
SCRIPT POST Proc1 post_script $JOB $RETURN

JOB Proc2 proc2.submit
SCRIPT PRE Proc2 pre_script $JOB
SCRIPT POST Proc2 post_script $JOB $RETURN

JOB Cleanup cleanup.submit
SCRIPT PRE Cleanup pre_script $JOB
SCRIPT POST Cleanup post_script $JOB $RETURN

PARENT Setup CHILD Proc1 Proc2
PARENT Proc1 Proc2 CHILD Cleanup
```

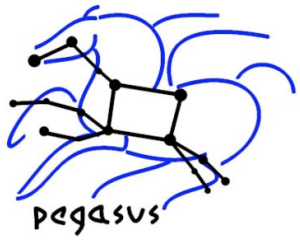
Exercise 3.3, continued



- From dagman.out:

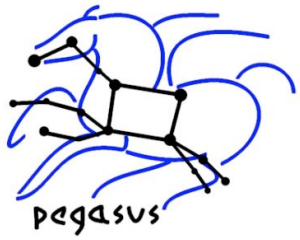
```
5/31 11:12:55 Event: ULOG_JOB_TERMINATED for Condor Node Proc2 (1868.0)
5/31 11:12:55 Node Proc2 job proc (1868.0) failed with status 1.
5/31 11:12:55 Node Proc2 job completed
5/31 11:12:55 Running POST script of Node Proc2...
...
5/31 11:13:00 Event: ULOG_POST_SCRIPT_TERMINATED for Condor Node Proc2 (1868.0)
5/31 11:13:00 POST Script of Node Proc2 completed successfully.
...
```

Rescue DAG



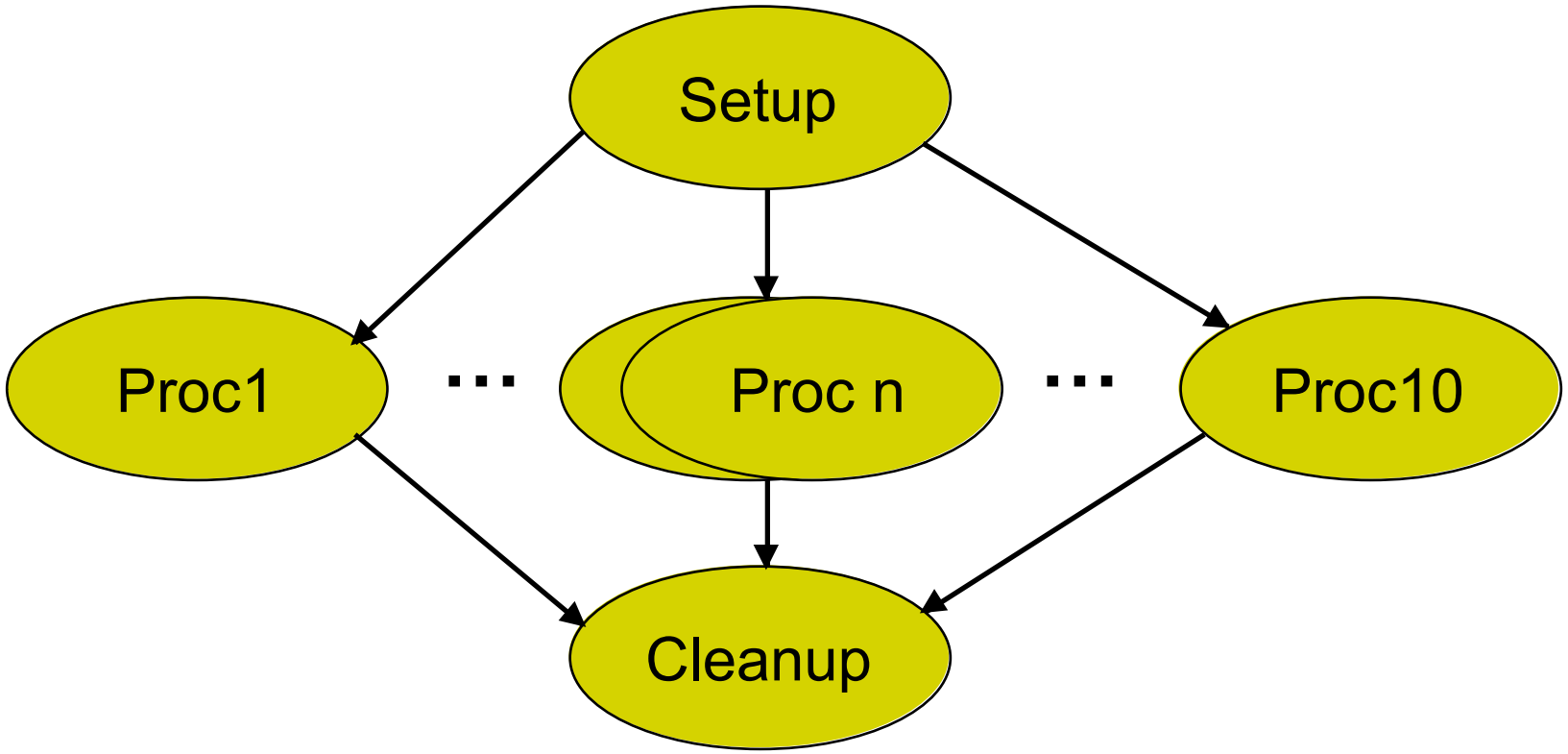
- Generated when a node fails or DAGMan is condor_rm'ed
- Saves state of DAG
- Run the rescue DAG to restart from where you left off
- DAGMan 7.1.0 has improvements in how rescue DAGs work

VARS (per-node variables)

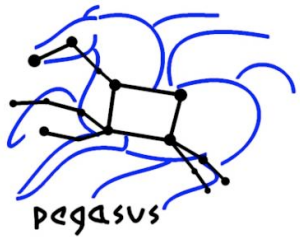


- VARS *JobName* *macroname*="*string*" [*macroname*="*string*"...]
- Macroname can only contain alphanumeric characters and underscore
- Value can’t contain single quotes; double quotes must be escaped
- Macronames cannot begin with “queue”
- Macronames are not case-sensitive

Exercise 3.4 - VARS

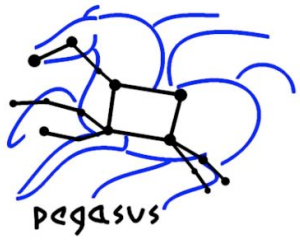


Exercise 3.4, continued




```
% more vars.dag
# DAG with lots of similar nodes to illustrate VARS.

JOB Setup setup.submit

JOB Proc1 proc.submit
VARS Proc1 ARGS = "Proc1 Alpe_dHuez"
PARENT Setup CHILD Proc1

JOB Proc2 proc.submit
VARS Proc2 ARGS = "Proc2 Col_du_Glandon"
PARENT Setup CHILD Proc2

JOB Proc3 proc.submit
VARS Proc3 ARGS = "Proc3 Col_de_la_Madeleine"
PARENT Setup CHILD Proc3

JOB Proc4 proc.submit
VARS Proc4 ARGS = "Proc4 Col_de_la_Forclaz"
PARENT Setup CHILD Proc4
[....]
```

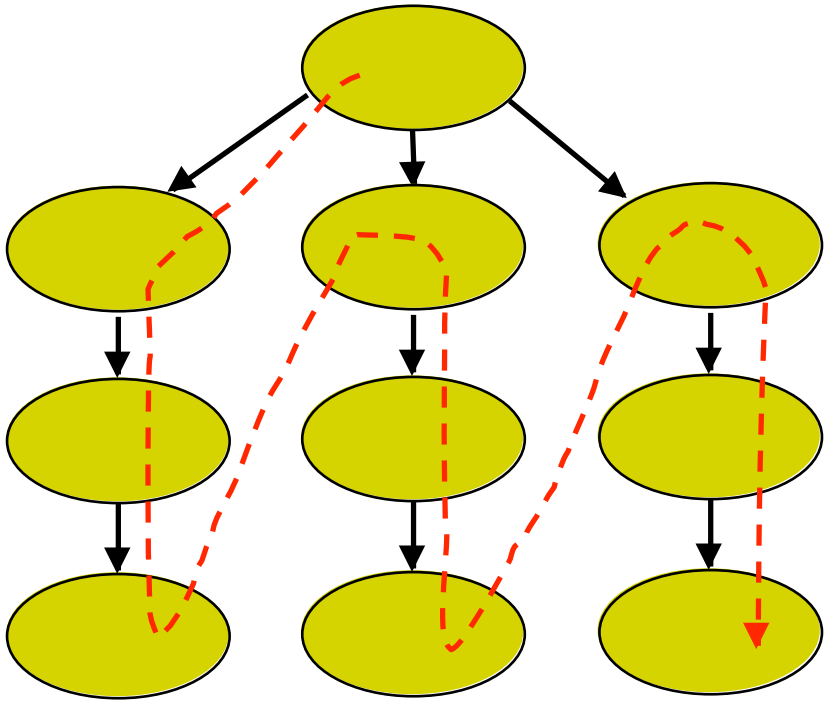
Recovery/bootstrap mode



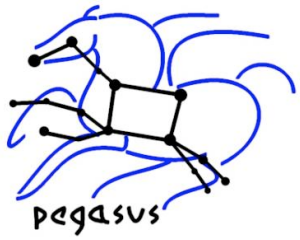
- Most commonly, after condor_hold/ condor_release of DAGMan
- Also after DAGMan crash/restart
- Restores DAG state by reading node job logs

Depth-first DAG traversal

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



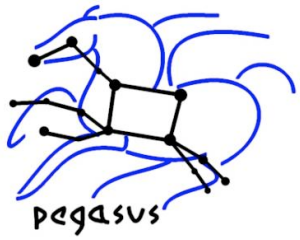
DAG node priorities



- 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+

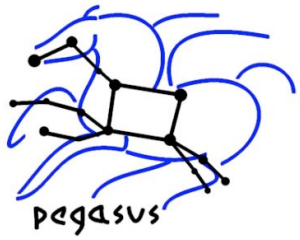
Node priorities can be configured in Pegasus Properties

Pegasus node priority properties



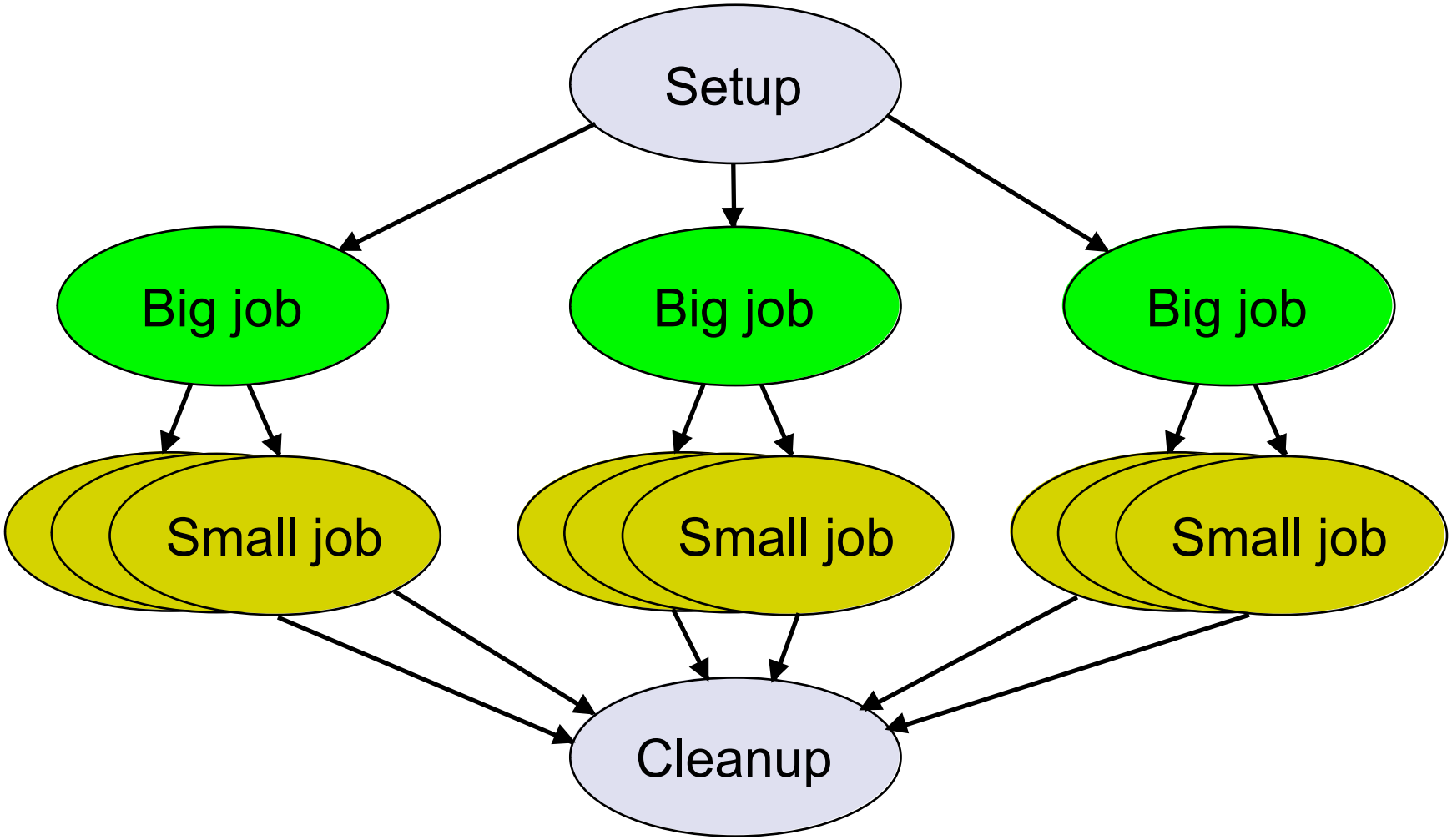
- pegasus.job.priority=<N>
- pegasus.transfer.stagein.priority=N
- pegasus.transfer.stageout.priority=N
- pegasus.transfer.inter.priority=N
- pegasus.transfer.*.priority=N
- For each job in TC or DAX define profile CONDOR::priority=N

Throttling by category

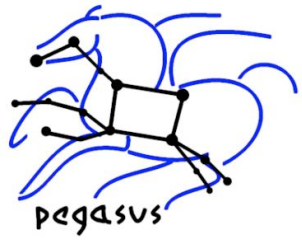


- CATEGORY *JobName CategoryName*
- MAXJOBS *CategoryName MaxJobsValue*
- Applies the maxjobs setting to only jobs assigned to the given category
- Global throttles still apply
- Useful with different types of jobs that cause different loads
- Available in version 6.9.5+

Exercise 3.5 – node categories

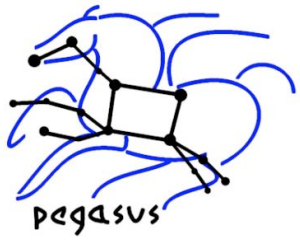


Exercise 3.5, continued




```
% more node_categories.dag
# DAG to illustrate node categories/category throttles.

MAXJOBS BigProc 1
MAXJOBS SmallProc 4

JOB Setup setup.submit

JOB BigProc1 big_proc1.submit
PARENT Setup CHILD BigProc1
CATEGORY BigProc1 BigProc
...
```

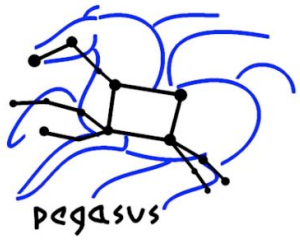
What we're skipping



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

- See the Pegasus and DAGMan manual online!

Relevant Links



- Pegasus: pegasus.isi.edu
- DAGMan: www.cs.wisc.edu/condor/dagman

- Tutorial materials available at: http://pegasus.isi.edu/tutorial/gtgs08/index.php

- For more questions: pegasus@isi.edu