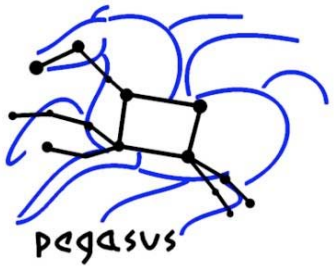


Managing LIGO Workflows on OSG with Pegasus

Karan Vahi

USC Information Sciences Institute

vahi@isi.edu

USC
VITERBI
SCHOOL OF
ENGINEERING

Pegasus: Planning for Execution in Grids

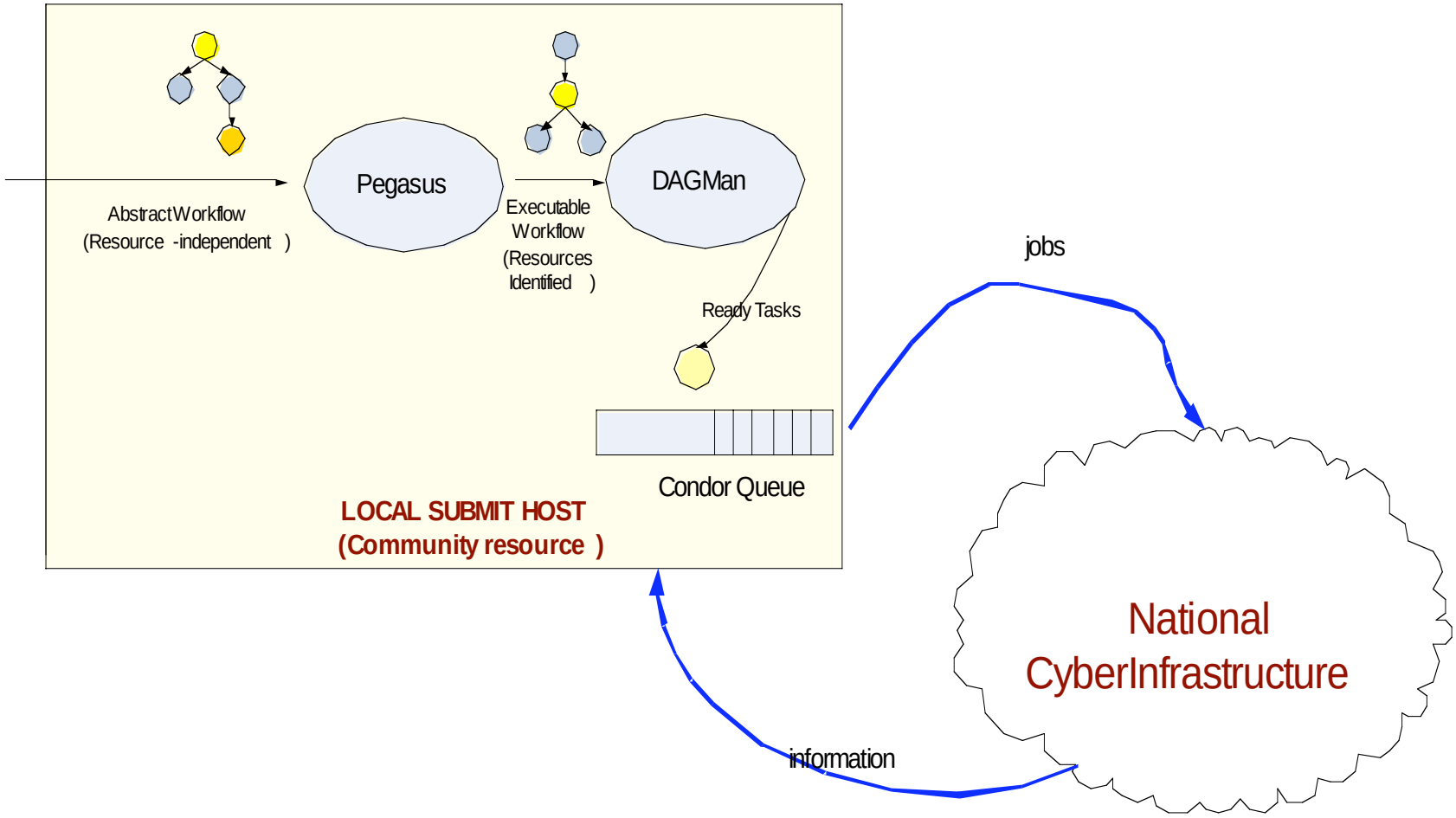
- Abstract Workflows - Pegasus input workflow description
  - workflow “high-level language”
  - only identifies the computations that a user wants to do
  - devoid of resource descriptions
  - devoid of data locations

- Pegasus
  - a workflow “compiler”
  - target language - DAGMan’s DAG and Condor submit files
  - transforms the workflow for performance and reliability
  - automatically locates physical locations for both workflow components and data
  - finds appropriate resources to execute the components
  - provides runtime provenance

Typical Pegasus Deployment

Pegasus and DAGMan are client tools

- No special requirements on the infrastructure



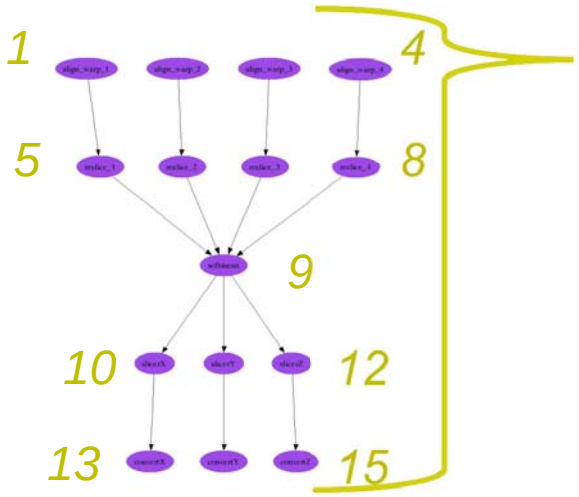
Basic Workflow Mapping

- 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

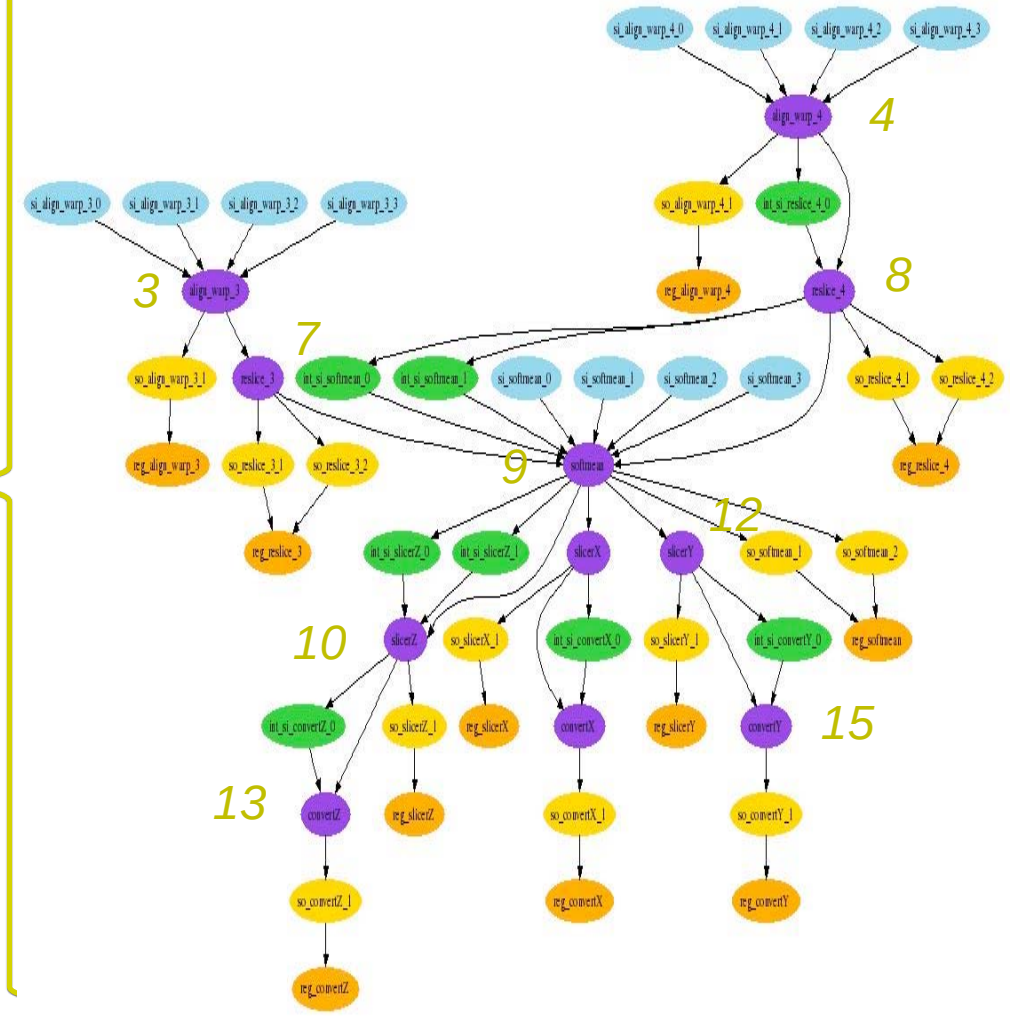
Pegasus Workflow Mapping



Original workflow: 15 compute nodes devoid of resource assignment

Resulting workflow mapped onto 3 Grid sites:

- 13 data stage-in nodes
- 11 compute nodes (4 reduced based on available intermediate data)
- 8 inter-site data transfers
- 14 data stage-out nodes to long-term storage
- 14 data registration nodes (data cataloging)



60 tasks

Catalogs used for discovery

- To execute on the OSG Pegasus needs to discover
  - Data (the input data that is required by the LIGO workflows)
  - Executables (Are there any LIGO executables installed before hand)
  - Site Layout (What are the services running on an OSG site)

Discovery of Data

- Replica Catalog stores mappings between logical files and their target locations.
- LIGO tracks it's data through LDR (based on Globus RLS). Pegasus queries LDR
  - discover input files for the workflow
  - track data products created
  - data reuse
- Pegasus also interfaces with a variety of replica catalogs
  - File based Replica Catalog
    - useful for small datasets
    - cannot be shared across users.
  - Database based Replica Catalog
    - useful for medium sized datasets.
    - can be used across users.

How to: *A single client* ***rc-client*** *to interface with all type of replica catalogs*

Discovery of Site Layout

- Pegasus queries a site catalog to discover site layout
  - 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

- For the OSG, Pegasus interfaces with VORS (Virtual Organization Resource Selector) to generate a site catalog for OSG

How to: *A single client* ***pegasus-get-sites*** *to generate site catalog for OSG, Teragrid*

Discovery of Executables

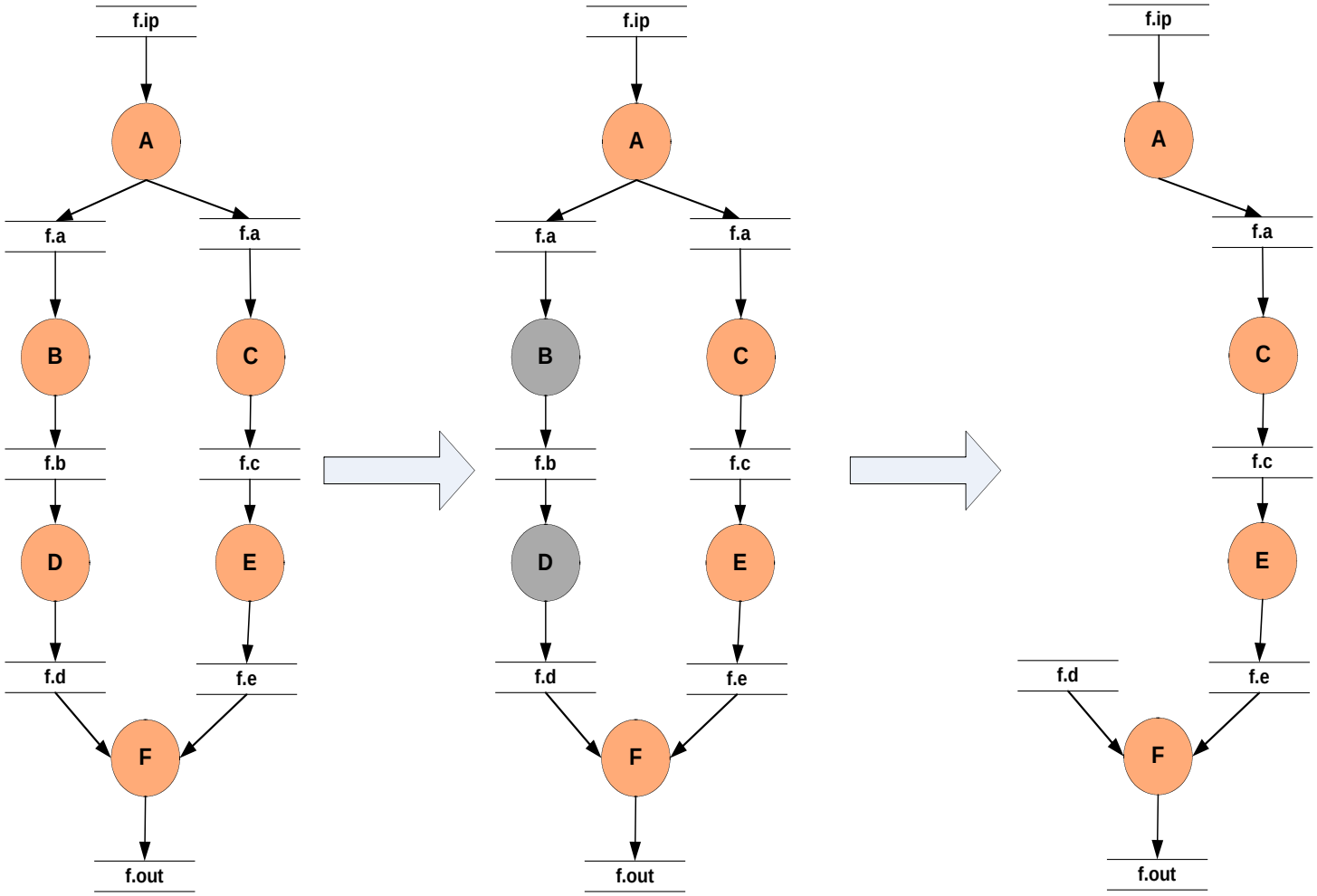
- Transformation Catalog maps logical transformations to their physical locations
- Used to
  - discover application codes installed on the grid sites
  - discover statically compiled codes, that can be deployed at grid sites on demand
- LIGO Workflows
  - Do not rely on preinstalled application executables
  - These executables are staged by Pegasus, at runtime to the remote OSG sites.

How to: *A single client* ***tc-client*** *to interface with all type of transformation catalogs*

Simple Steps to run on OSG

1. Specify your computation in terms of DAX
   - Write a simple DAX (abstract workflow)generator
   - Java based API provided with Pegasus
   - Details on http://pegasus.isi.edu/doc.php
2. Set up your catalogs
   - Use *pegasus-get-sites* to generate site catalog and transformation catalog for OSG
   - Record the locations of your input files in a replica client using *rc-client*
3. Plan your workflow
   - Use *pegasus-plan* to generate your executable workflow that is mapped to OSG
4. Submit your workflow
   - Use *pegasus-run* to submit your workflow
5. Monitor your workflow
   - Use *pegasus-status* to monitor the execution of your workflow

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

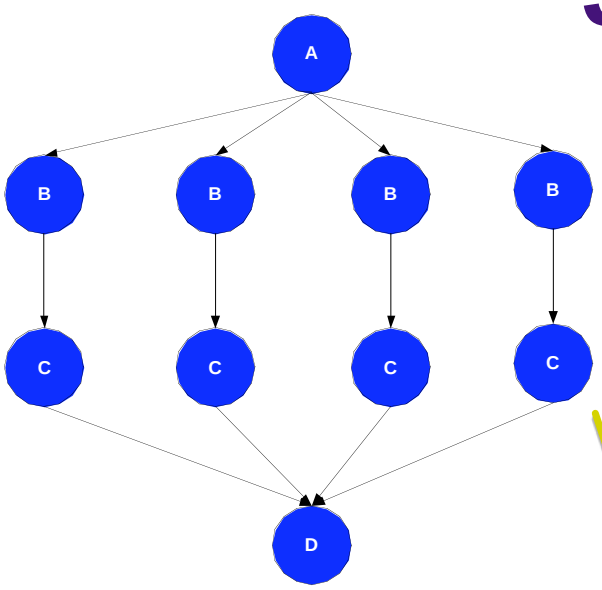
How to: *Files need to be cataloged in replica catalog at runtime. The registration flags for these files need to be set in the DAX*

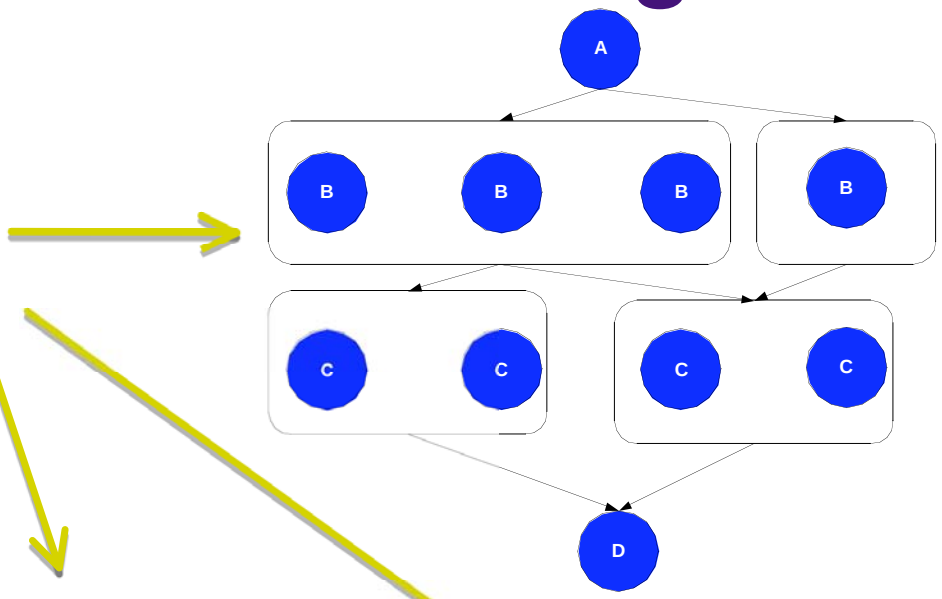
Pegasus: Efficient Space Usage

- Input data is staged dynamically, new data products are generated during execution
- For large workflows 10,000+ input files
  - Similar order of intermediate/output files
  - Not enough space-failures occur
- Solution:
  - Determine which data are no longer needed and when
  - Add nodes to the workflow do cleanup data along the way
- Benefits: can significantly decrease the amount of space needed to run a workflow
- Next steps: Work on a Storage-Aware Scheduling Algorithm

How to: *Dynamic cleanup by default is on. To turn it off, pass* ***--nocleanup*** *to* ***pegasus-plan***

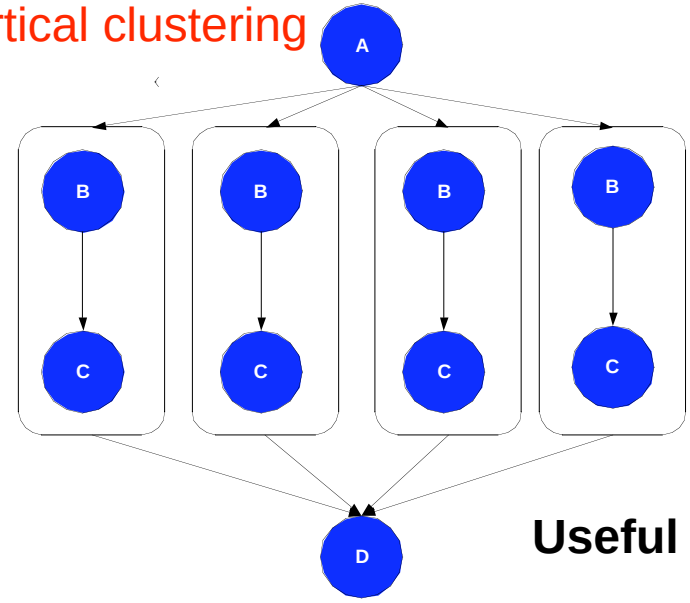
Job clustering



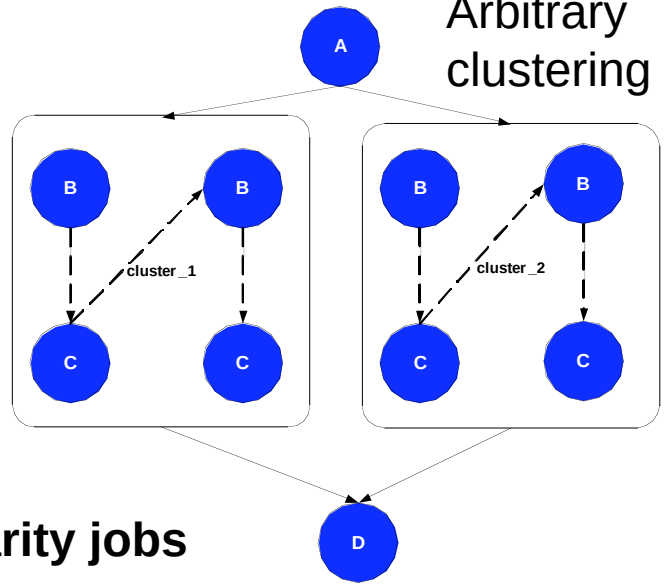


Level-based clustering

Vertical clustering



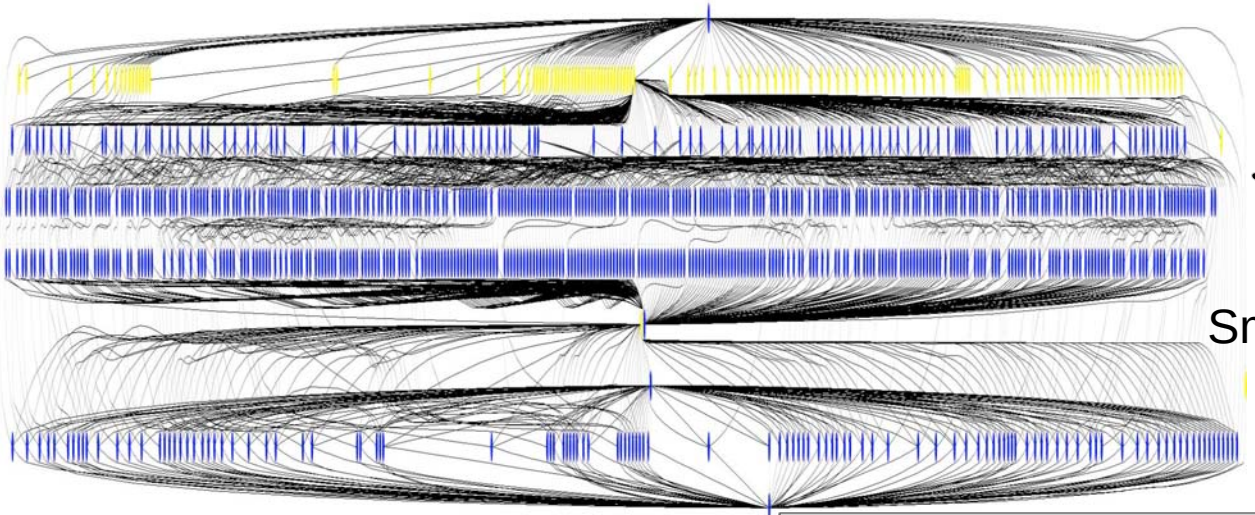
Arbitrary clustering



Useful for small granularity jobs

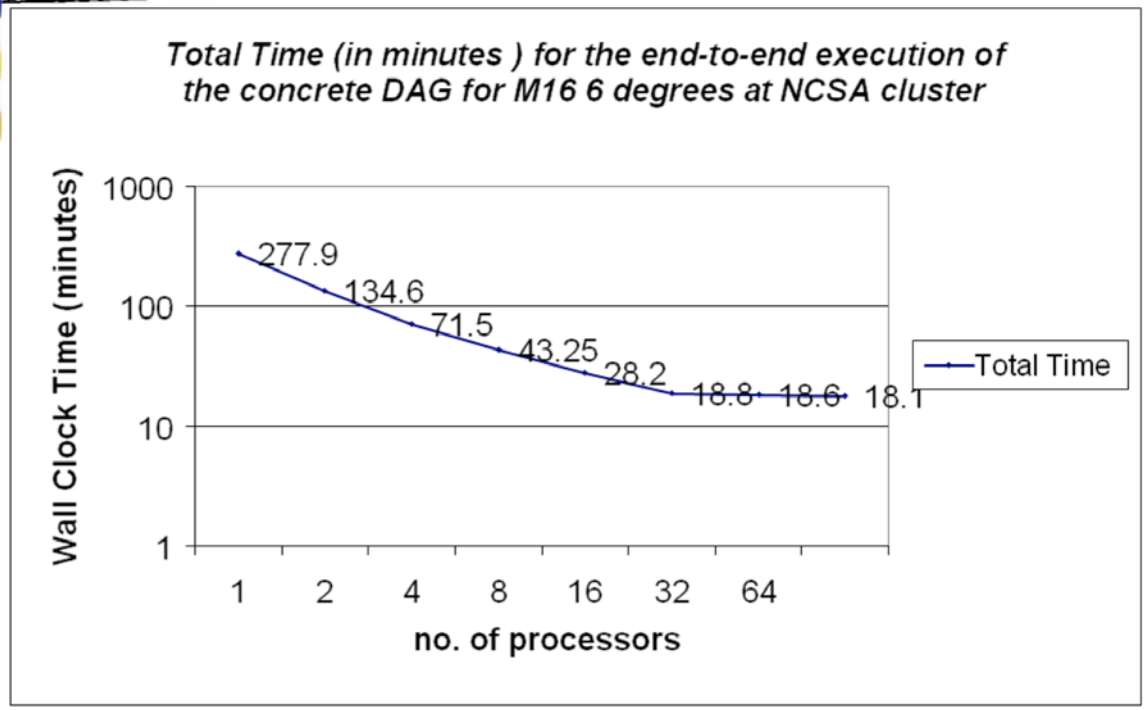
How to: *To turn job clustering on, pass* ***--cluster*** *to* ***pegasus-plan***

Performance optimization through workflow restructuring

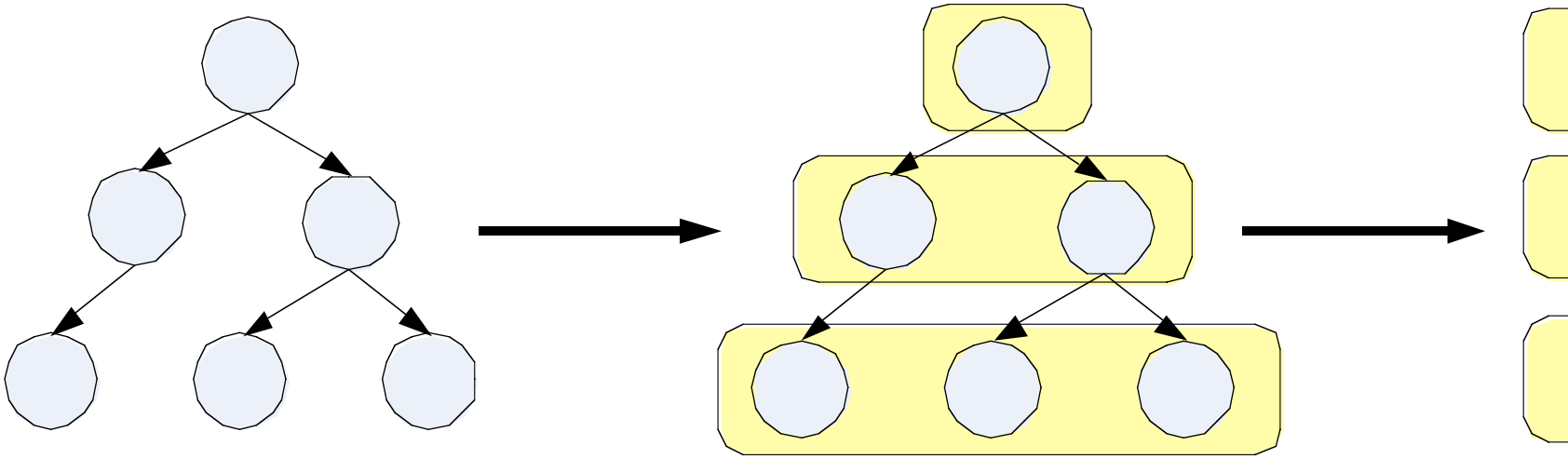


Small 1,200 Montage Workflow

Montage application
~7,000 compute jobs in instance
~10,000 nodes in the executable workflow
same number of clusters as processors
speedup of ~15 on 32 processors



Managing execution environment changes through partitioning



Original Abstract Workflow

A Particular Partitioning

New Abstract Worfklow

Provides reliability—can replan at partition-level
Provides scalability—can handle portions of the workflow at a time

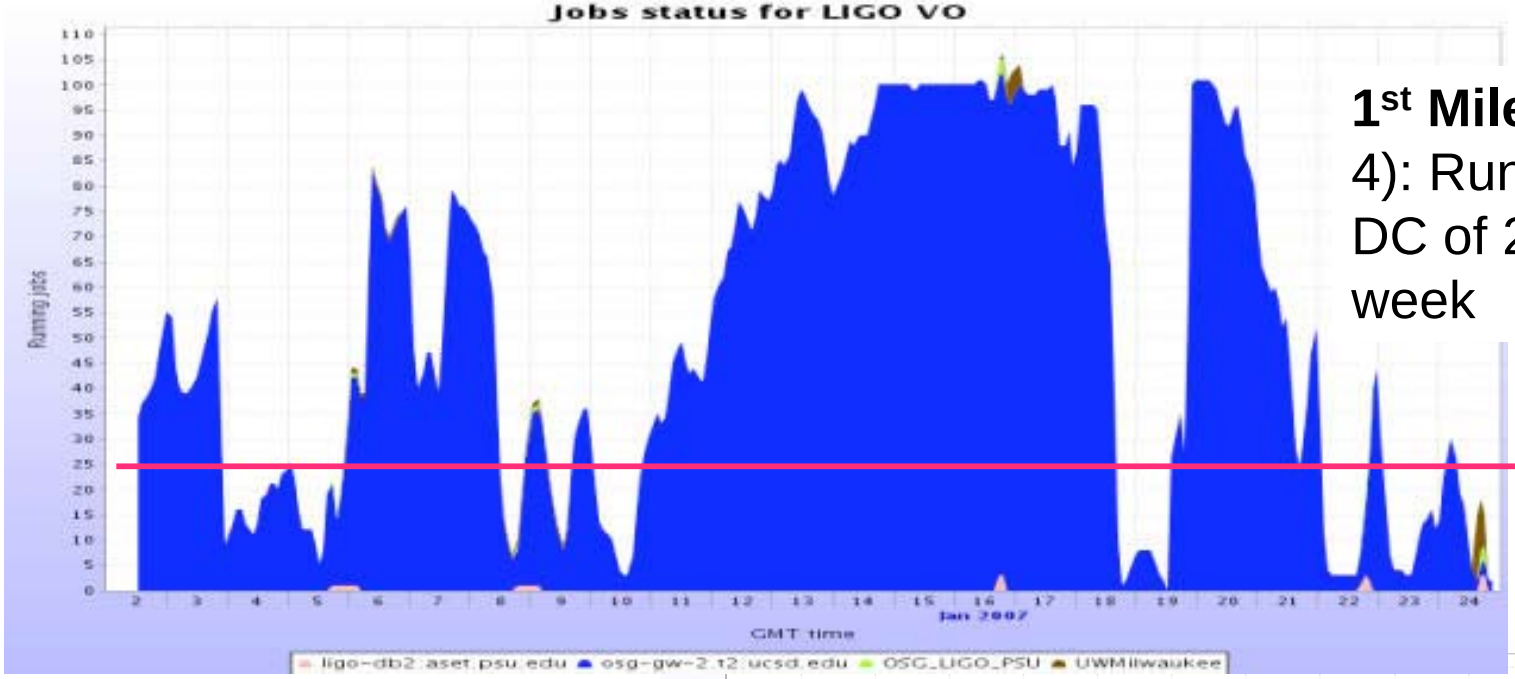
How to: *1) Partition the workflow into smaller partitions at runtime using* ***partitiondax*** *tool.*

2) Pass the partitioned dax to ***pegasus-plan*** *using the* ***--pdax*** *option.*

Reliability

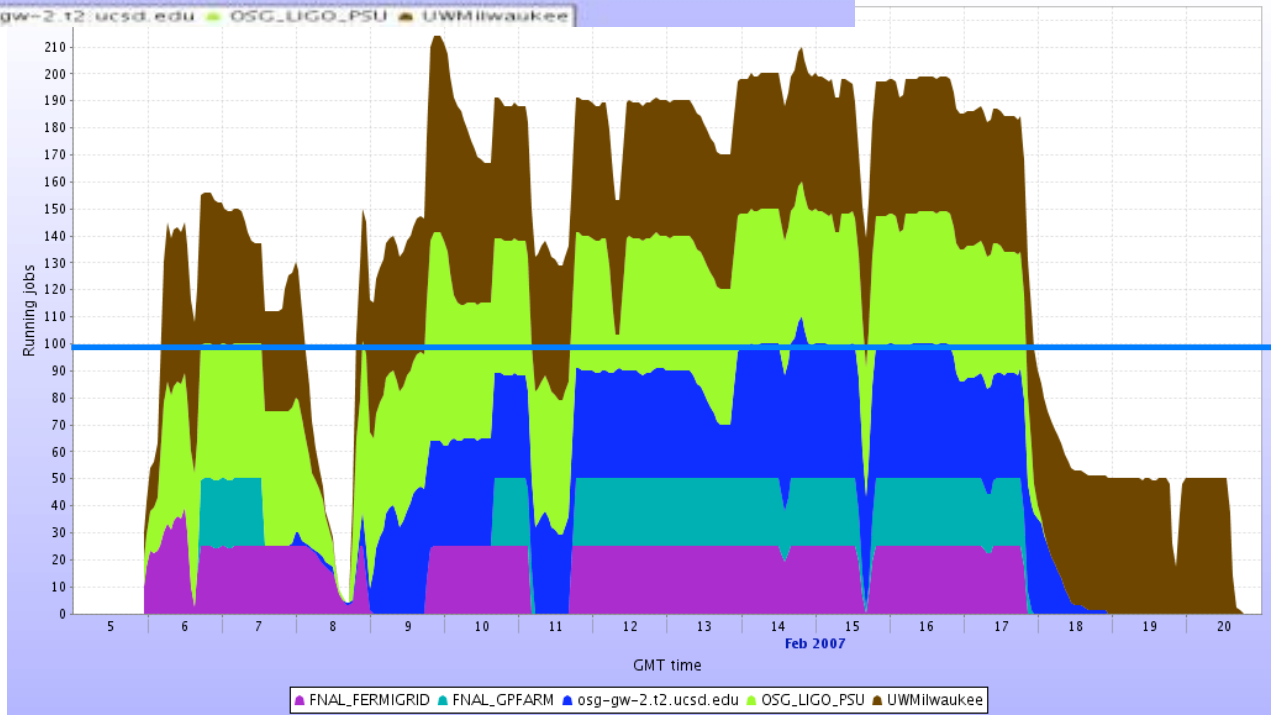
- Job Level Retry
  - Leverages DAGMAN retry capabilities.
  - Allows us to overcome transient grid failures.
- Re-planning
  - Partitions can be re-planned in case of errors.
  - Only happens after retries have been exhausted.
- Try alternative data sources for staging data
- Provide a rescue-DAG when all else fails
- Clustering of data transfers and execution tasks, alleviates:
  - GridFTP server overloads
  - High load on head node.

Helping to meet LIGO/OSG milestones



1st Milestone (month 4): Run at UCSD with DC of 25 slots for one week

2nd Milestone (month 8): Run on OSG with DC of 100 slots for one week



Work done by Kent Blackburn, David Meyers, Michael Samidi, Caltech

Future OSG-focused Developments

- Working towards 3rd Milestone: (month 12): reach 1000 slots peak in MonaLisa on OSG
  - *Limit use of shared file systems / job manager fork / total number of jobs*.
    - Use of node clustering techniques
    - Support for the placement of the data and jobs in temporary OSG directories.
    - Support for local disk I/O and vertical clustering of jobs

- *SRM/dcache support for Pegasus workflows*

What does Pegasus do for an application?

- Provides an OSG-aware workflow management tool
  - Queries OSG services (like VORS) to get information about sites in the OSG
  - Deploys user executables as part of the workflow.
  - Reduced Storage footprint. Data is also cleaned as the workflow progresses.
- Data Management within the workflow
  - Interfaces with the variety of Replica Catalog's (including RLS) to discover data
  - Does replica selection to select replicas.
  - Manages data transfer by interfacing to various transfer services like RFT, Stork and clients like globus-url-copy.
  - No need to stage-in data before hand. We do it within the workflow as and when it is required.
- Improves application performance and execution
  - Job clustering
  - Support for condor glidein's
  - Techniques exist to minimize load on remote Grid resources during large scale execution of workflows .
  - Data Reuse
    - Avoids duplicate computations
    - Can reuse data that has been generated earlier.

Pegasus developments in the next 2 years

- New releases will continue to be included in VDT
- Continued improvements in documentation
- Development of accounting capabilitities
- Improvements in the monitoring capabilities
  - Providing command-line and gui tools to view the progress of the workflow
- Development of new debugging and diagnostic tools
  - Debugging across Pegasus/DAGMan/Grid
- Research areas:
  - Management of multiple workflows
  - Automatic resource provisioning
- Welcome community input on future developments: pegasus@isi.edu

Relevant Links

- Pegasus: http://pegasus.isi.edu
  - Distributed as part of VDT (*Standalone version in VDT 1.7 and later*)
  - Can be downloaded directly from
    - http://pegasus.isi.edu/code.php

- Interested in trying out Pegasus
  - Do the tutorial
    - http://pegasus.isi.edu/tutorial/tg07/index.html
    - Send email to pegasus@isi.edu, to do tutorial on ISI cluster.
  - Quickstart Guide
    - Available at http://pegasus.isi.edu/doc.php
    - More detailed documentation appearing soon.
  - Relevant Papers
    - http://pegasus.isi.edu/publications.php

- Support lists
  - pegasus-support@mailman.isi.edu

- **We are looking for new applications and developers interested in using our tools**