A Cloud-based Dynamic Workflow for Mass Spectrometry Data Analysis

Ashish Nagavaram, Gagan Agrawal,

Michael A. Freitas, Kelly H. Telu

The Ohio State University

Gaurang Mehta, Rajiv. G. Mayani, Ewa Deelman

USC Information Sciences Institute

http://pegasus.isi.edu deelman@isi.edu

The Problem

Execute a compute intensive, data-parallel application within user-given time constraints

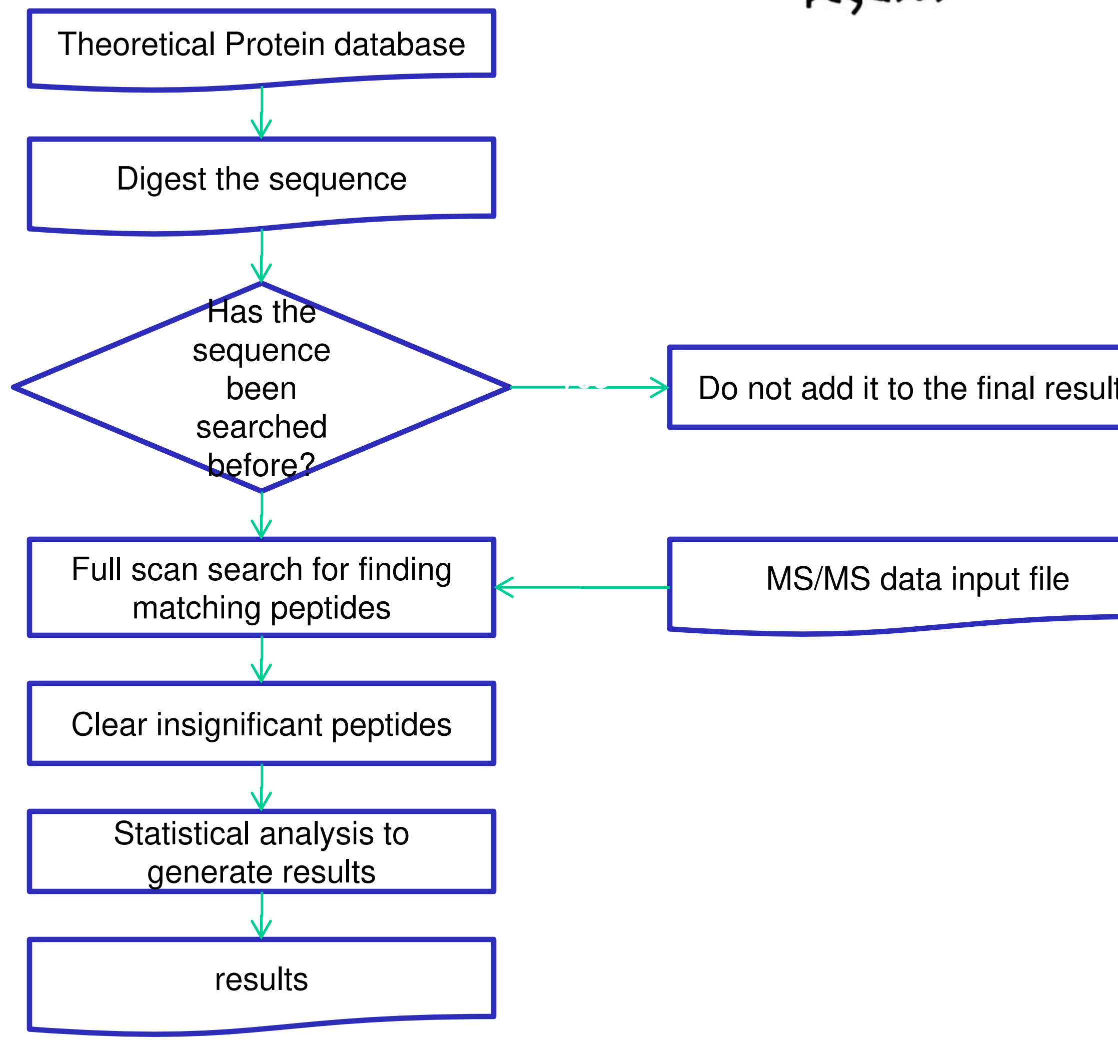
- The application is sequential
- Want to outsource computation to the cloud

Outline

- Application
- Benefits of cloud computing
- Approach
  - Parallelize the application
  - Use of workflow technologies
  - Dynamic resource provisioning
- Evaluation on EC2
- Conclusions

Application: Mass Spectrometry

- Searches proteins and peptides from tandem mass spectrometry data
- Uses Protein DB
- Sensitive probabilistic scoring model
- Noise filtering algorithm



Basic characteristics of the application

- Highly data parallel
- Performance depends on the data set, thus not known *a priory*
- When partitioning the data, need to combine results
- Opportunity/challenge to adapt the execution environment to the specific problem

Workflows and Clouds

Benefits

- User control over environment
- Pay as you go model
- On-demand provisioning / Elasticity
- SLA, reliability, maintenance

Drawbacks

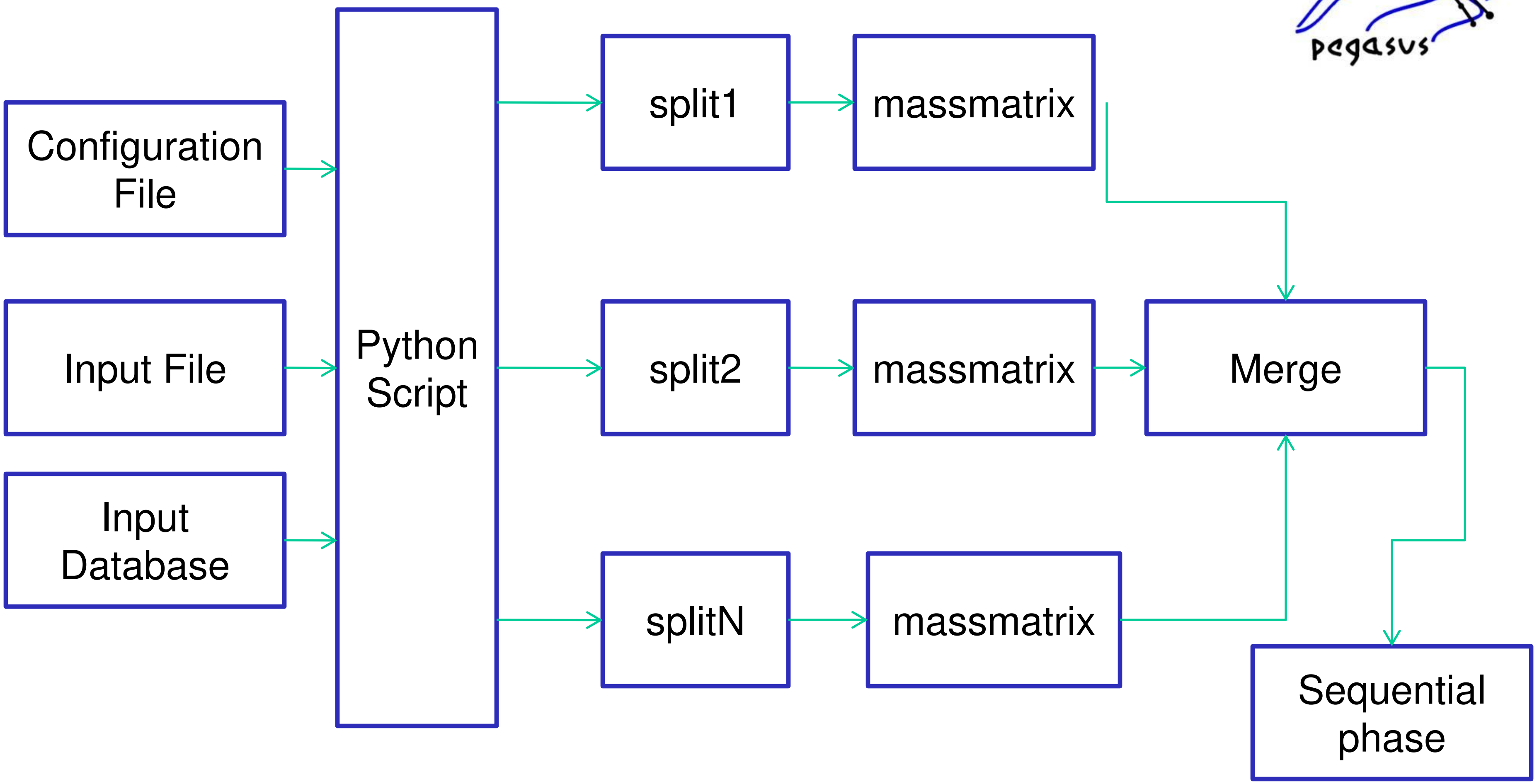
- Complexity (more control = more work)
- Cost
- Performance
- Resource Availability
- Vendor Lock-In

Google's Container-based Data Center in Belgium
http://www.datacenterknowledge.com/

Approach

- Parallelize the application
- Use the workflow paradigm to structure the application
  - Use the Pegasus Workflow Management System to manage the workflow execution
- Use cloud computing for execution
- Adapt the cloud to the application
  - Use a flexible resource provisioning system to acquire the necessary resources (Wrangler)
- Evaluate the parallelization (multi-core machine) and adaptation (EC2 cloud)

Parallelize the application



- Complex data structures (matrix of matrices)
- Need to re-index while maintain both local and global index

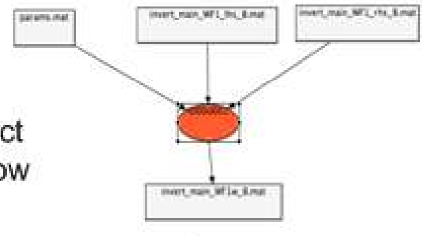
Pegasus
Workflow Management System

- Developed since 2001
- A collaboration between USC and the Condor Team at UW Madison (includes DAGMan)
- Used by a number of applications in a variety of domains
- (astronomy, bioinformatics, earthquake science, gravitational-wave physics, etc.)
- Provides reliability—can retry computations from the point of failure
- Provides scalability—can handle large data and many computations (kbytes-TB of data, 1-10^6 tasks)
- Automatically captures provenance information
- Can run on resources distributed among institutions, laptop, campus cluster, Grid, Cloud

Pegasus Workflow Management System

- Provides a portable and re-usable workflow description
- Enables the construction of complex workflows based on computational blocks
- Can compose workflows using Java, Perl, Python APIs, systems Triana, Wings
- Can be incorporated into portals
- Infers data transfers
- Infers data registrations
- Lives in user-space
- Provides correct, scalable, and reliable execution
  - Enforces dependencies between tasks
  - Progresses as far as possible in the face of failures

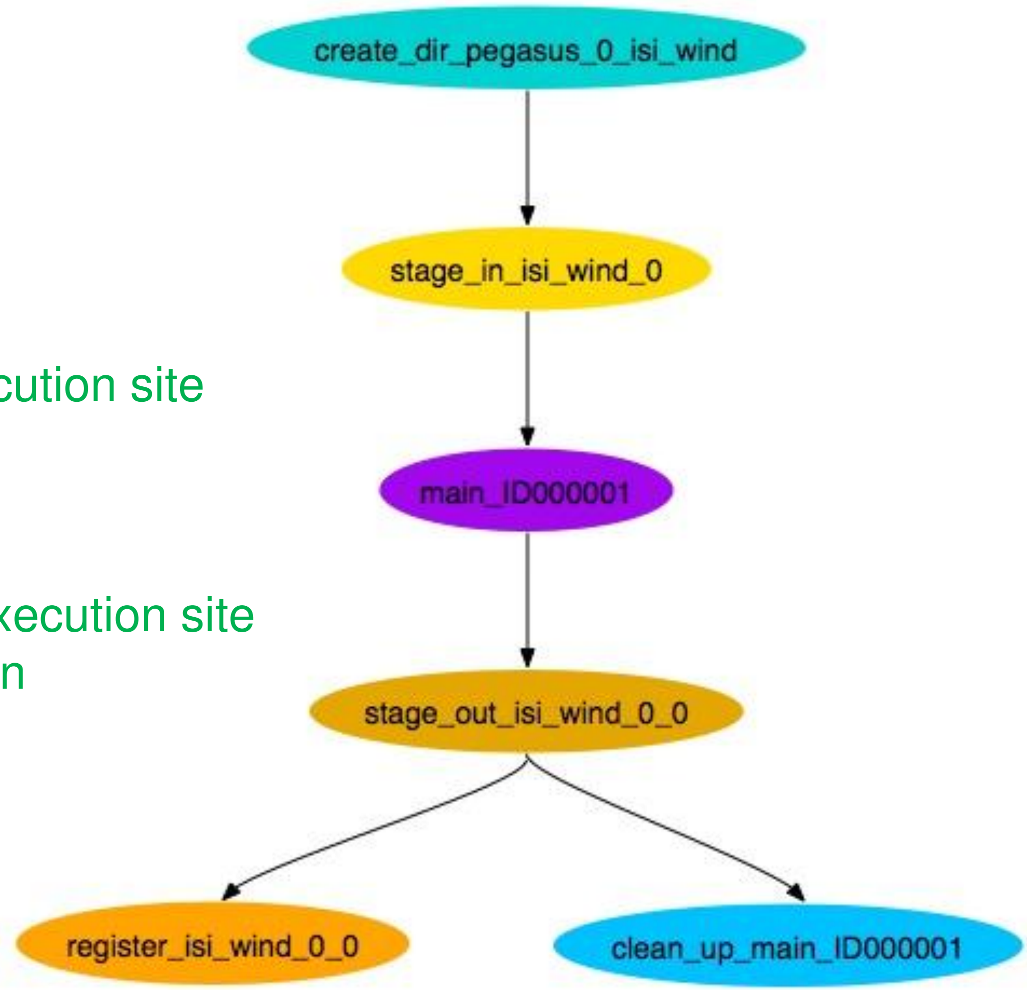
Abstract Workflow

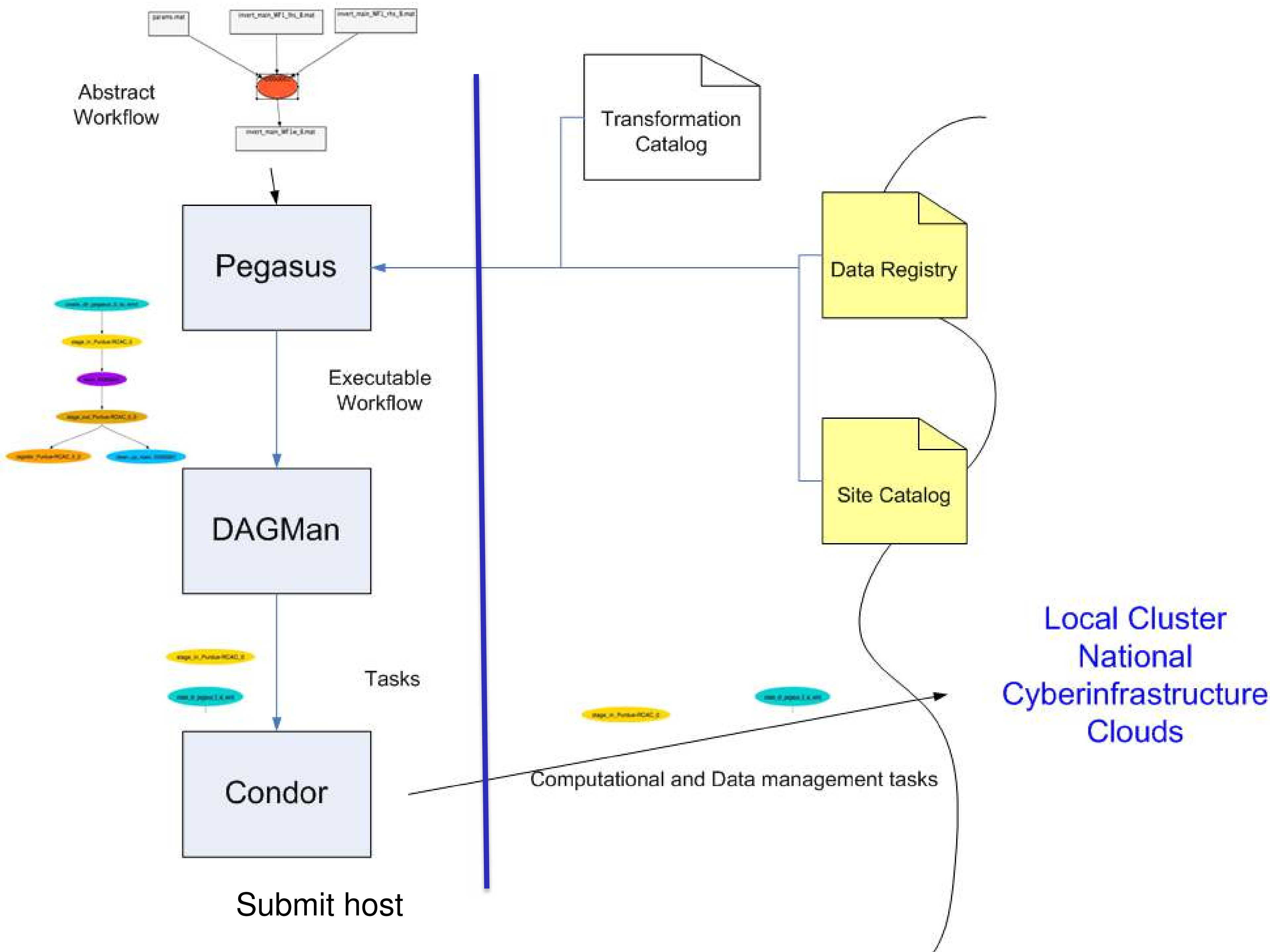


Executable Workflow Generated by Pegasus

Pegasus:
Selects an execution site
Selects a data archive
Creates a workflow that
- Creates a “sandbox” on the execution site
- Stages data
- Invokes the computation
- Stages out data
- Registers data and Cleans up execution site
- Captures provenance information

Performs other optimizations




Abstract Workflow
Pegasus
Executable Workflow
DAGMan
Tasks
Condor
Submit host
Transformation Catalog
Data Registry
Site Catalog
Computational and Data management tasks
Local Cluster
National Cyberinfrastructure
Clouds

Outline

- Application
- Benefits of cloud computing
- Approach
  - Parallelize the application
  - Use of workflow technologies
  - Dynamic resource provisioning
- Evaluation on EC2
- Conclusions

A way to make it work

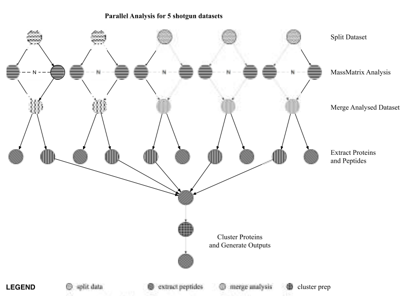
Pegasus makes use of available resources, but cannot control them

Data Storage

data

resources

Work definition



work

Grids and Clouds

Pegasus WMS

Local Resource

A way to make it work

Pegasus makes use of available resources, but cannot control them

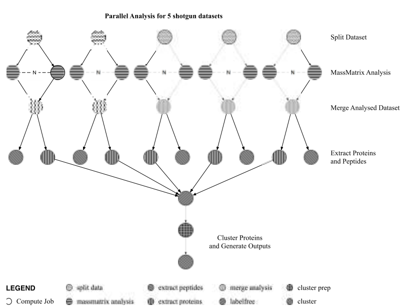
Data Storage

data

resources

Virtual Resource Pool

Work definition



work

Grids and Clouds

Resources requests

Pegasus WMS

Resource Provisioner

Local Resource

Building a Virtual Cluster on the Cloud

- Clouds provide resources, but the software is up to the user
- Running on multiple nodes may require cluster services (e.g. scheduler)
- Dynamically configuring such systems is not trivial
- Workflows need to communicate data—often through files, need filesystems (or stage data in/out for each task)
- Adapt the cluster on demand

Wrangler (Gideon Juve, USC/ISI)

- A service for provisioning and configuring virtual clusters
- User specifies the virtual cluster configuration, and Wrangler provisions the nodes and configures them according to the user’s requirements
- Users can specify custom pluggins for nodes by writing simple scripts
- XML format for describing virtual clusters, support for multiple cloud providers, node dependencies and groups, automatic distribution of configuration files and scripts

Clients-- send requests to the coordinator to launch, query, and terminate, deployments

Coordinator-- a web service that manages application deployments.

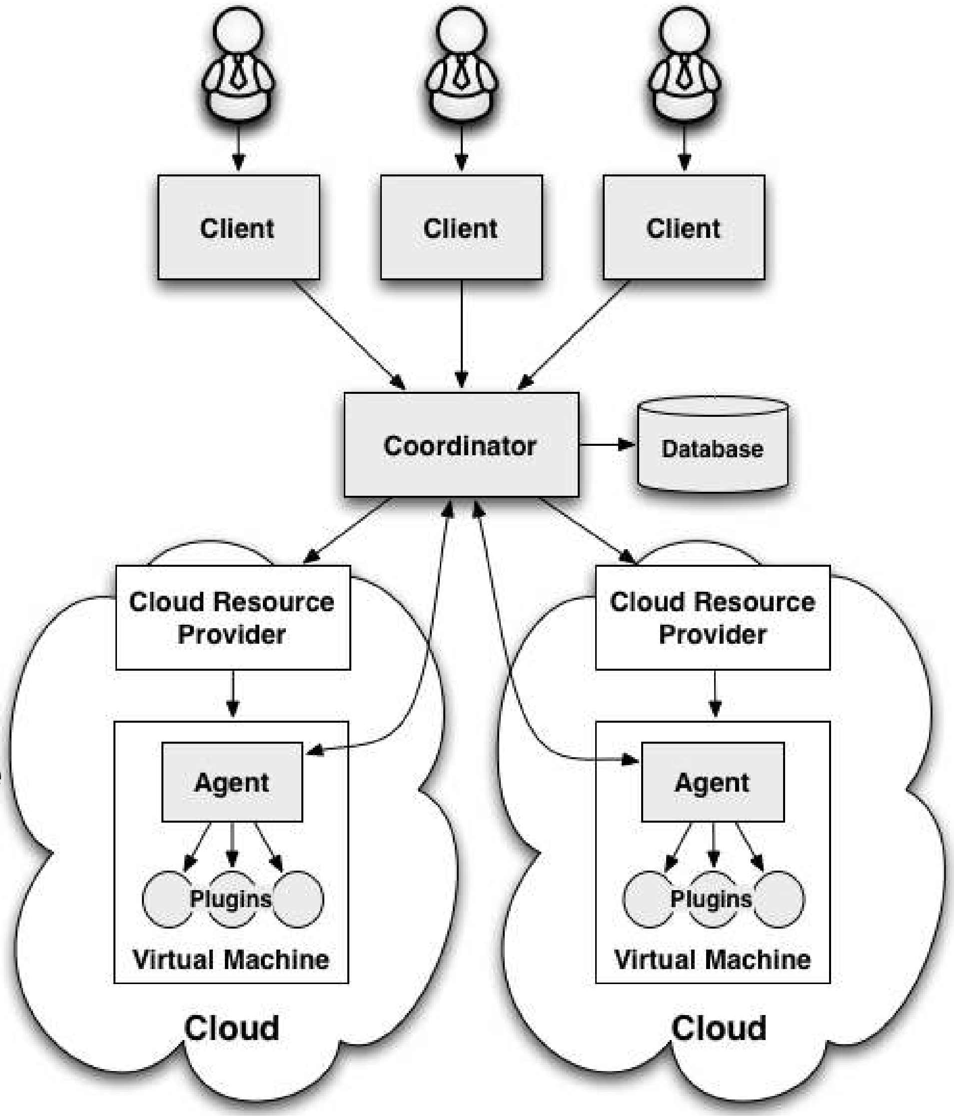
- accepts requests from clients
- provisions nodes from cloud providers
- collects information about the state of a deployment
- acts as an information broker

Agents--run on VMs

- Manage VM configuration and monitors health.
- collect information and reports the state of the node to the collector
- configure the node with the software and services specified by the user
- monitor the node for failures.

Plugins -- user-defined scripts that implement the behavior of a node

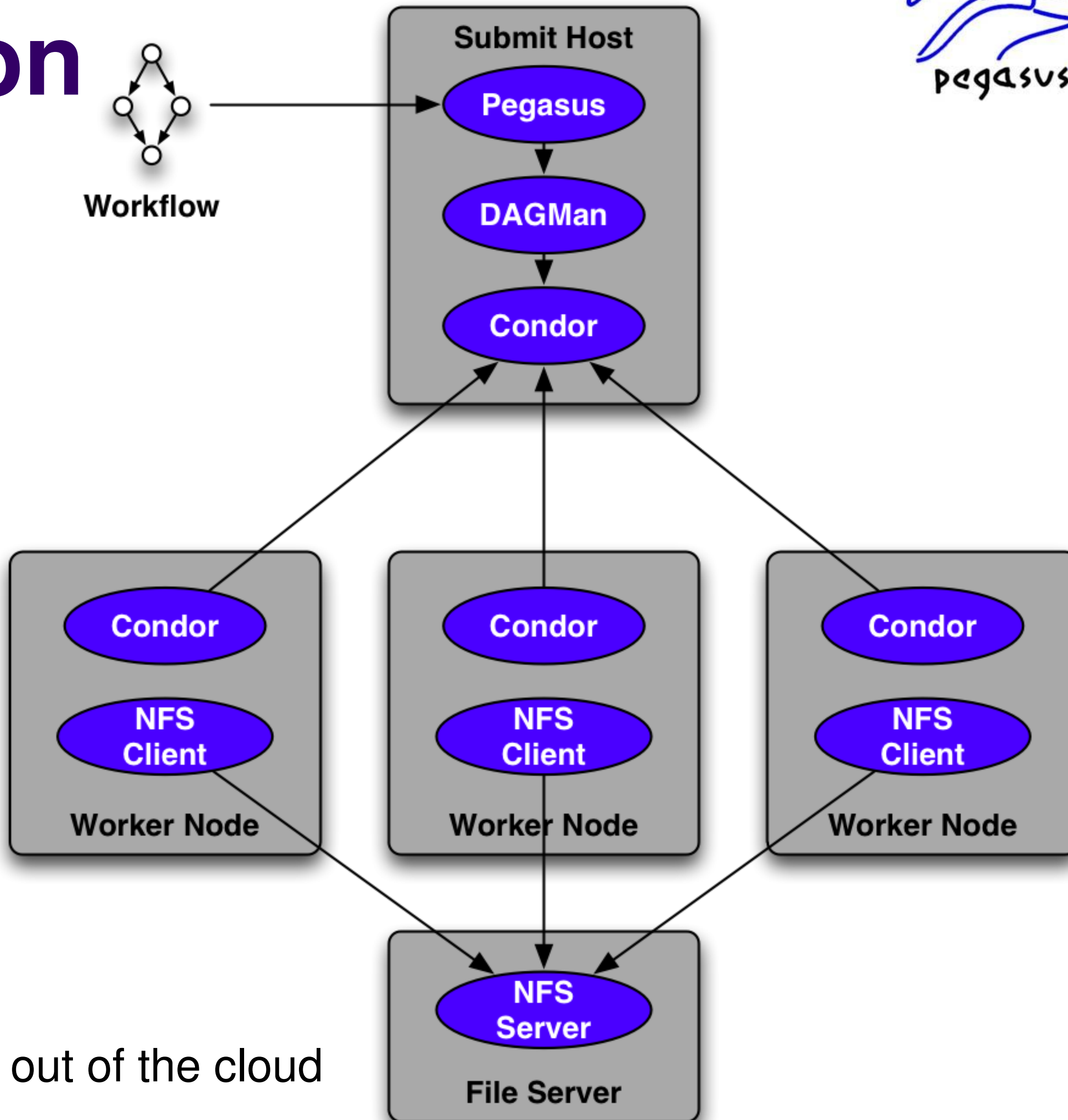
- invoked by the agent to configure and monitor a node
- each node can have multiple plugins.



Wrangler

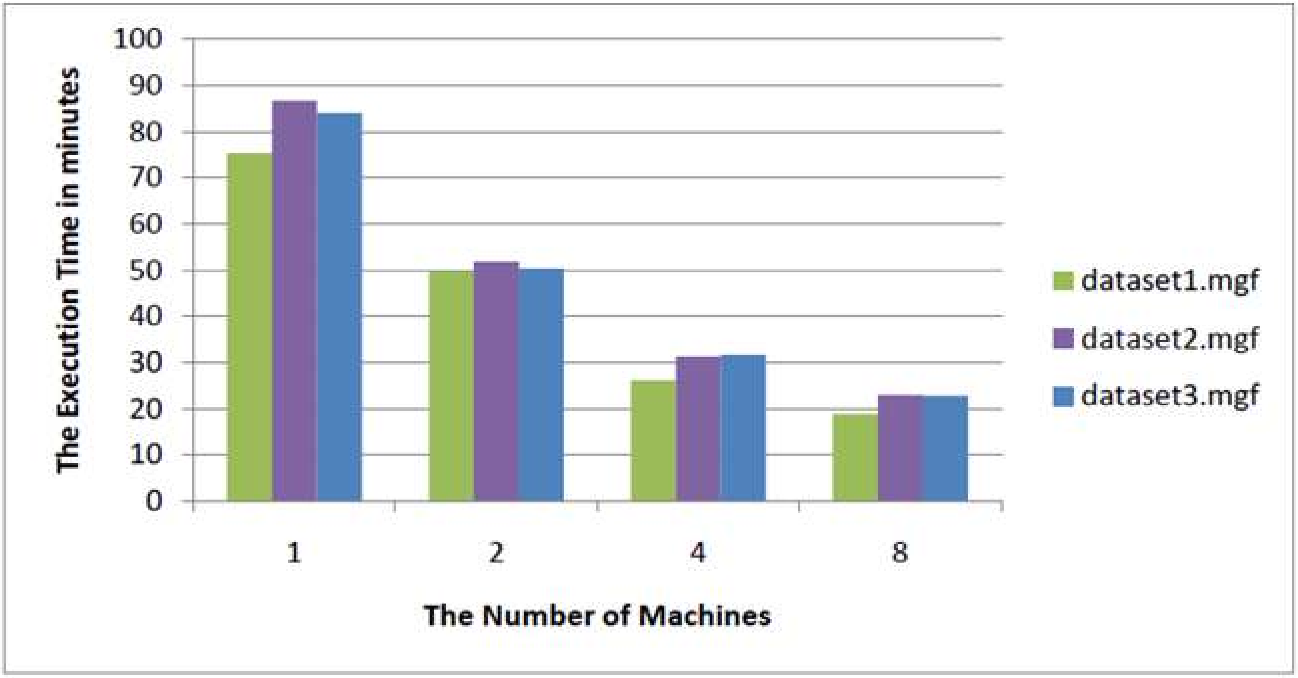
CloudCom 2011

A cloud Condor/NFS configuration



The submit host can be in or out of the cloud

Parallel Execution *-core cluster

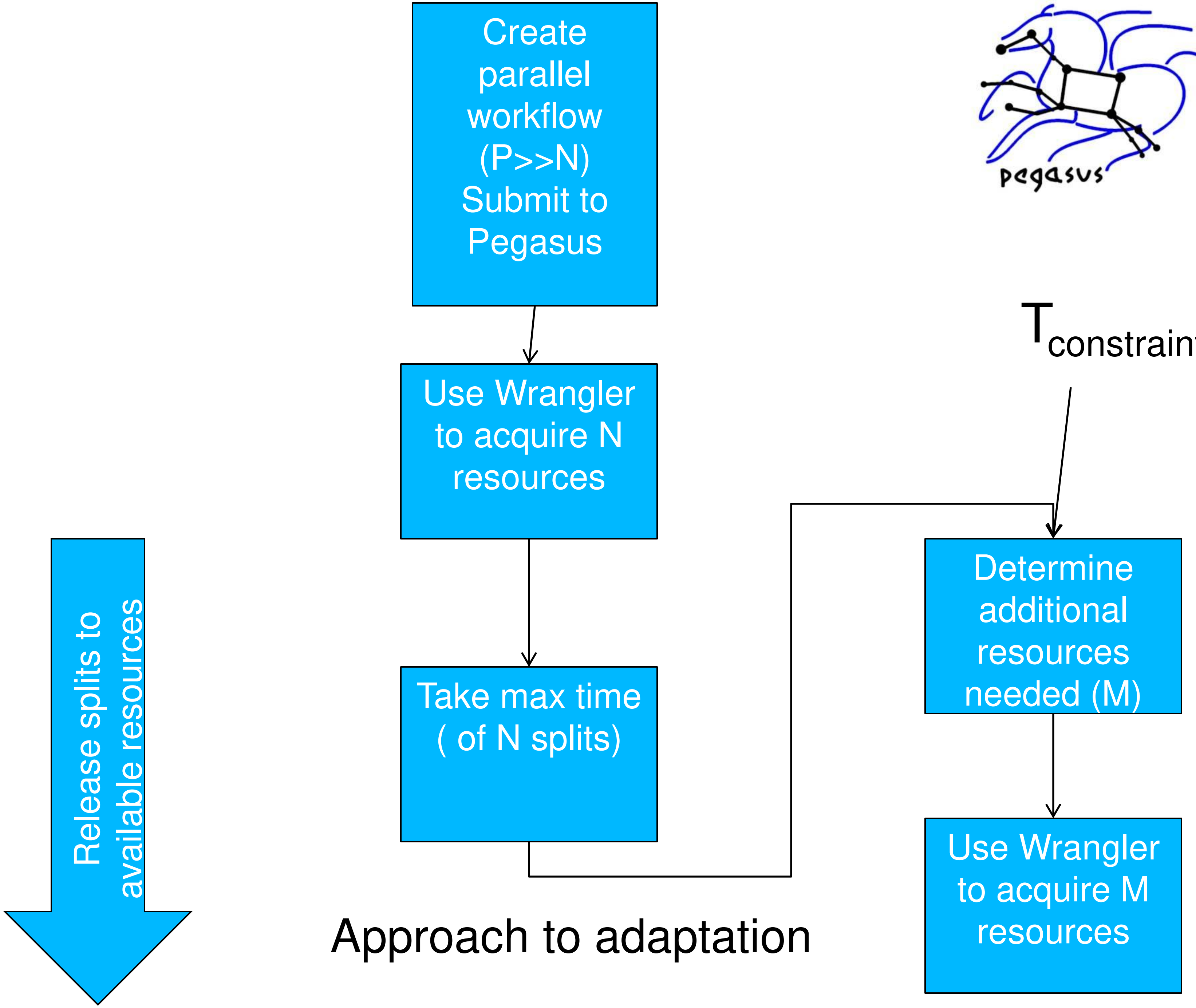


2.9 speedup on 4cores

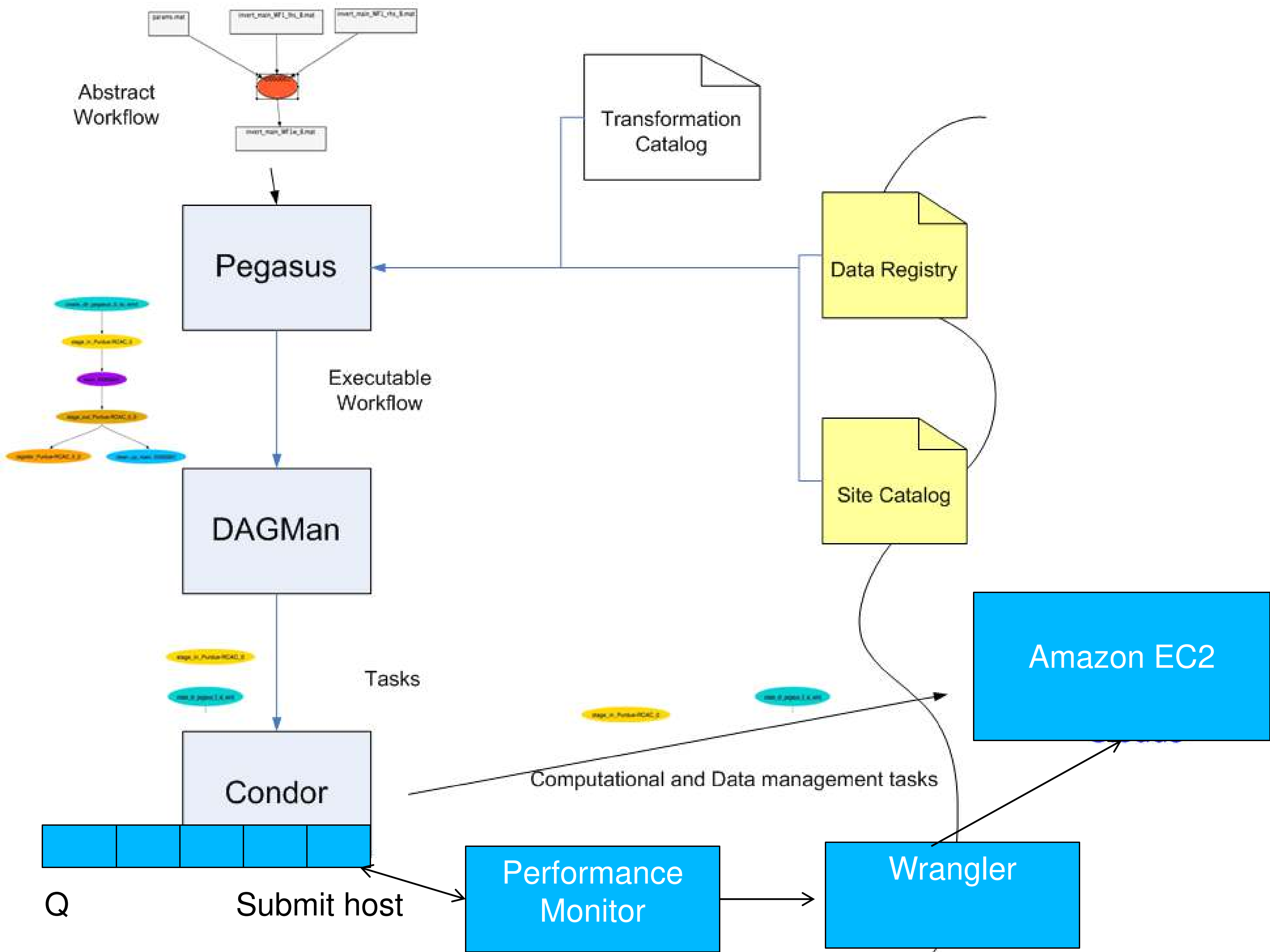
8 core Intel Xeon node with 6GB of RAM

Theoretical database used was of 20 MB

The code was run for 6 different datasets (~50,000 records)



Approach to adaptation


Abstract
Workflow
Transformation
Catalog
Pegasus
Data Registry
Executable
Workflow
Site Catalog
DAGMan
Amazon EC2
Tasks
Computational and Data management tasks
Condor
Performance
Monitor
Wrangler
Q
Submit host

Determining the number of resources needed

User specifies the deadline $T_{constraint}$

Capture performance information in the first *N* executions (T_{per_split})

$$T_{remaining} = T_{constraint} - (2 \times T_{per\_split})$$

Max

Calculate the cumulative remaining execution time

$$T_{execution\_predicted} = T_{per\_split} \times (split\_count - 2N)$$

Estimate needed cores

$$Nodes_{required} = \frac{T_{execution\_predicted}}{T_{time\_constraint}} + 1 - N$$

Execution on EC2

The algorithm chooses how many cores to add

Instance type	Number of cores	Cost (per_hour)	Memory
small	1	0.085	1.7GB
large	2	0.34	7.5GB
extra-large	4	0.68	15GB
high-cpu	8	0.68	7GB

Adaptive algorithm on Dataset 1

Time Constraint	Actual Time	Additional Nodes Launched
60	37	0
30	25.89	2
24	23.45	5
18	17.94	9

Conclusions

- Displayed a framework for dynamic execution of scientific workflows
- User specified time constraint can be used to drive the allocation of resources
- Used real-time performance information for choosing the number of resources
- Possible extensions
  - More dynamic approach that monitors the execution over the lifetime of the application
  - Quality of results vs time
  - Including other criteria for resource acquisition (cost)