



Pegasus



Custom Execution Environments with Containers in Pegasus-enabled Scientific Workflows

Karan Vahi*, ***Mats Rynge****, ***George Papadimitriou****, ***Duncan Brown[¶]***,
Rajiv Mayani*, ***Rafael Ferreira da Silva****, ***Ewa Deelman****,
Anirban Mandal[§], ***Eric Lyons[§]***, ***Michael Zink[§]***

**USC Information Sciences Institute*

[¶]Syracuse University

[§]RENCI

[§]University of Massachusetts Amherst

USC Viterbi
School of Engineering
Information Sciences Institute



renci

**University of
Massachusetts
Amherst**

Outline

Motivation *Reproducibility for Workflows*

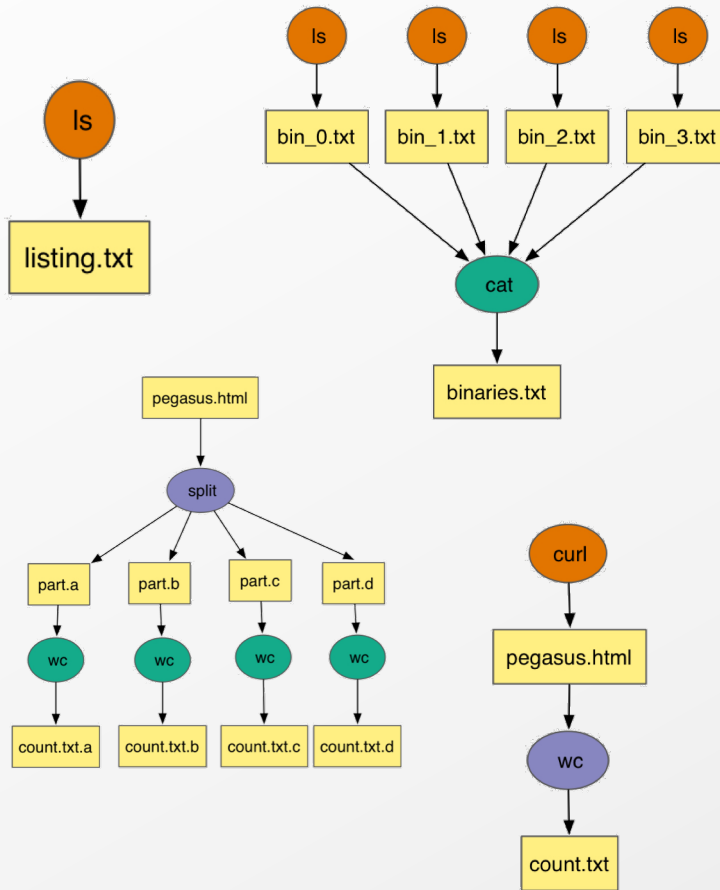
Containers *Solution for Reproducibility*
Challenges deploying for
Distributed Workflows
Design Considerations

Pegasus *Introduction*
Container Support

Experiments *Setup*
Results

What are workflows?

- Allows scientists to connect different codes together and execute their analysis
- Workflows can be very simple (independent or parallel) jobs or complex represented usually as DAG's
- Workflows are DAGs
 - Nodes: jobs, edges: dependencies
 - No while loops, no conditional branches
 - Jobs are standalone executables
- Helps users to automate scale up



Reproducibility in Scientific Workflows

- Why?

- Ease of Use and Portability

- Don't **limit** the **execution** environments
 - Ideally, users can reliably recreate your analysis on varied execution environments
 - Local Desktop (Windows, Linux, MACOS)
 - Local HPC Cluster (Mainly Linux oriented)
 - Computing Grids (Collection of University HPC clusters, such as OSG)
 - Leadership Class HPC Systems (Linux variants like Cray)
 - Cloud Environments (Choice of OS and architectures available)

Challenges to Reproducibility?

Custom Execution Environments

- When you start using shared resources you loose control over the hardware and OS
- Hard to ensure **homogeneity**: Users will run your code on same platform/OS it was developed on.
- Some dependent libraries required for your code may conflict with system installed versions
 - **TensorFlow** requires specific python libraries and versions.
 - Some libraries maybe easy to install on latest Ubuntu, but not on EL7
- If **running** on shared computing resources such as computational grids
 - you run on a site with **heterogeneous** nodes and your job lands on a node where OS is **incompatible** with your executable

Outline

Motivation *Reproducibility for Workflows*

Containers *Solution for Reproducibility
Challenges deploying for
Distributed Workflows
Design Considerations*

Pegasus *Introduction
Container Support*

Experiments *Setup
Results*

Solutions: Containers

- Virtualizes the OS instead of the Hardware
 - Sits on top of the physical server and the host OS
 - Each container shares the Host kernel and binaries and libraries
- Separates the application from the node OS.
- Lightweight
 - Instead of GB's size is on order of MB's
 - Take seconds to start instead of minutes
 - Can pack more applications on the same node compared to Virtual Machines

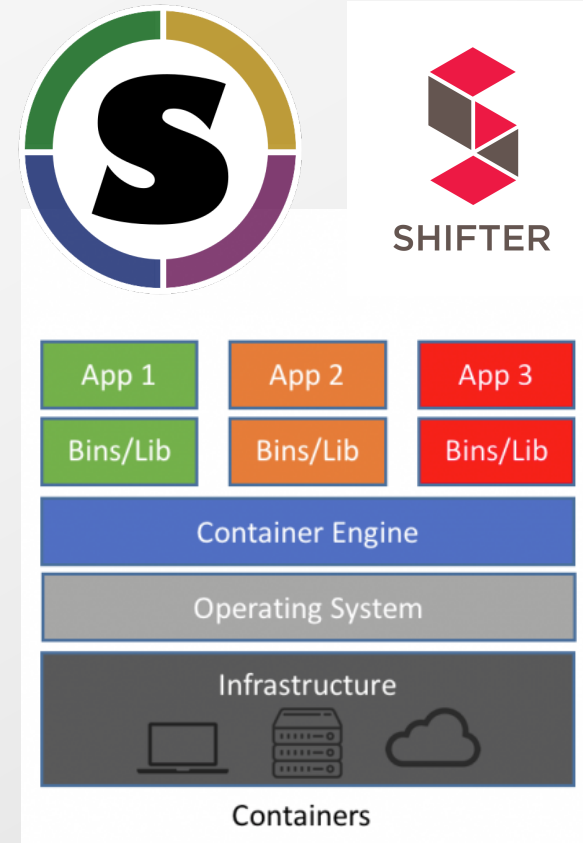
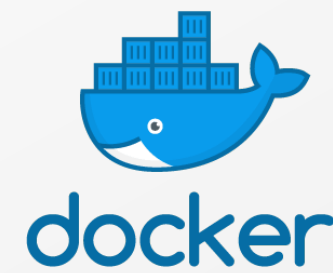


Image Source: <https://blog.netapp.com/wp-content/uploads/2016/03/Screen-Shot-2018-03-20-at-9.24.09-AM-935x500.png>

Solutions: Why Containers?

- Reproducibility
 - Supply a **fully defined** and **reproducible** environment
 - Usually described as a **recipe** file that captures the steps to configure and setup the container
- Ability to provide a flexible user controlled environment that underlying compute cluster cannot
 - Administrators **main goal** is to provide a **stable, slow moving, multi-user** environment
 - **Cannot provide** all combinations of development **libraries** and tools for their user community
- Perfect for deploying on demand.
 - Also **seamlessly transfer** to another compute environment

However: Challenges deploying Containers for Distributed Workflows

- How to distribute container images and make them available to compute jobs
 - Pegasus workflows contain thousands or millions of jobs simultaneously running
- Container Technologies are fragmented
 - One size fits all approach does not work

Design Considerations

- Support for different container technologies
 - **Docker** popular in traditional **corporate** computing environment.
 - By default jobs run as root!
 - **Singularity** preferred in **HPC** as allows jobs to run in user space
 - Some HPC centers support **custom solutions** such as **Shifter** to run Docker images
- Work in Distributed Environments
 - Users don't know a-priori which node or cluster a job lands on.
 - OSG is **dynamic** computing environment
- Easy Configuration and Representation
 - Easy for users to configure which container and type of container required by their jobs
- Support for Public Registries
 - Lot of popular images available. Have ability to retrieve them

Outline

Motivation *Reproducibility for Workflows*

Containers *Solution for Reproducibility*
Challenges deploying for
Distributed Workflows
Design Considerations

Pegasus *Introduction*
Container Support

Experiments *Setup*
Results

Pegasus Workflow Management System

Automates complex, multi-stage processing pipelines

Enables parallel, **distributed computations**

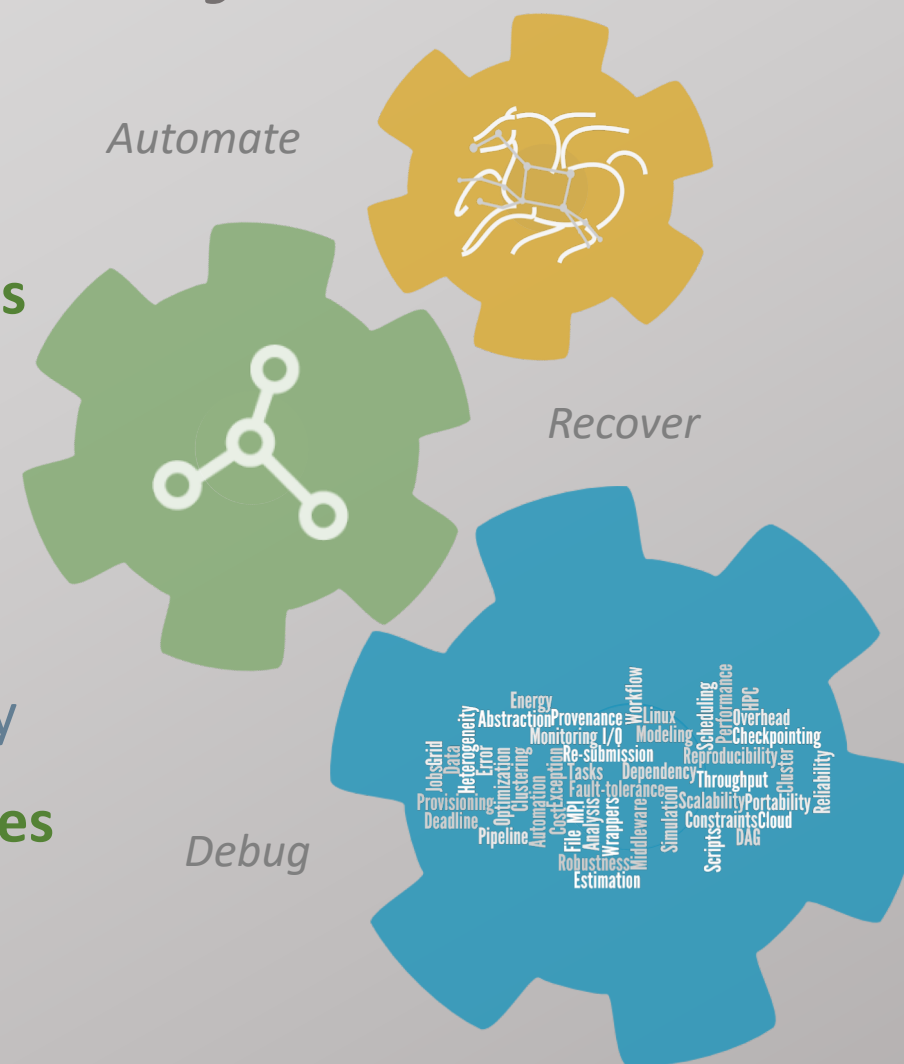
Automatically executes data transfers

Reusable, aids **reproducibility**

Records how data was produced (**provenance**)

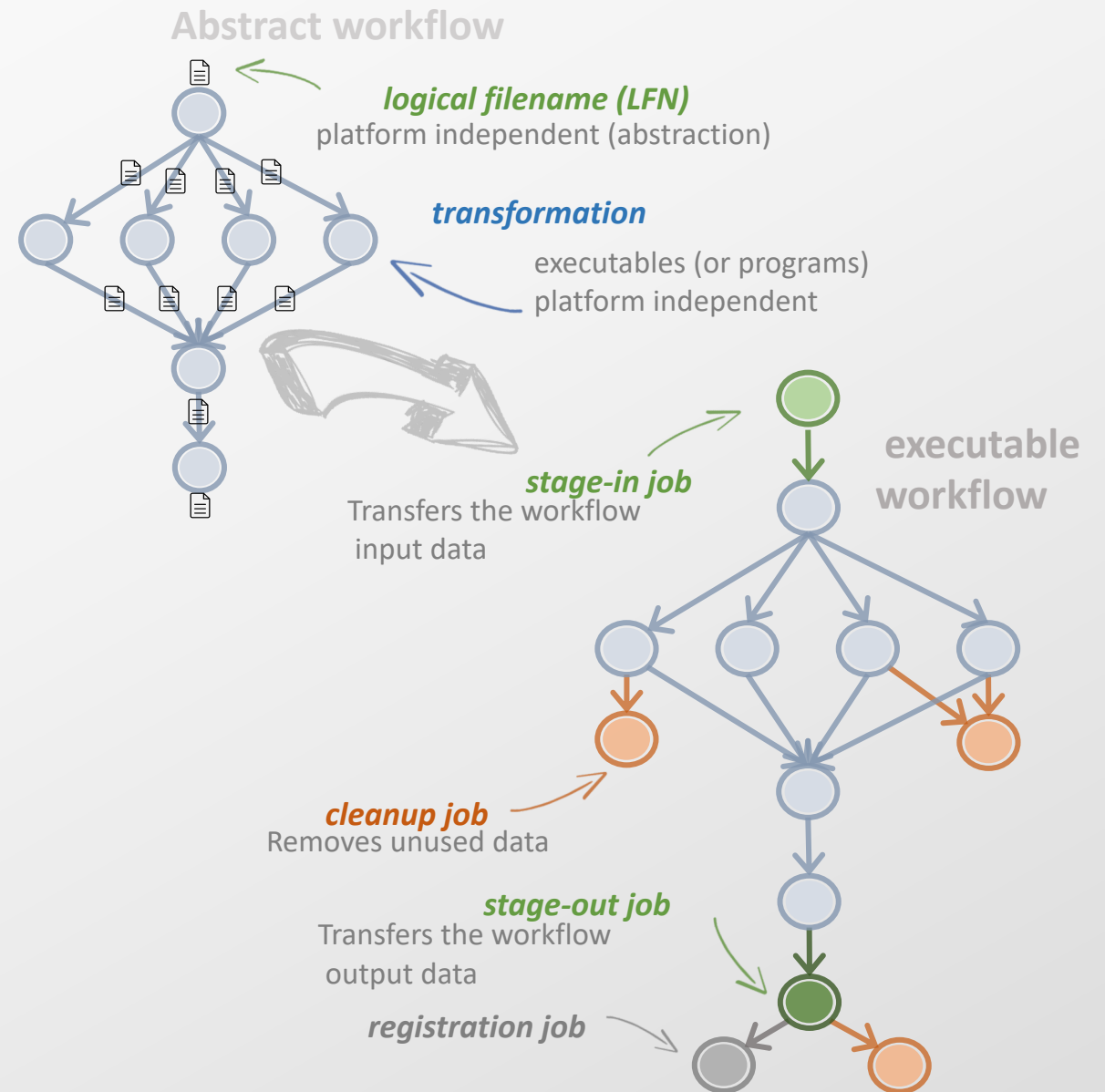
Handles **failures** with to provide reliability

Keeps track of data and **files**



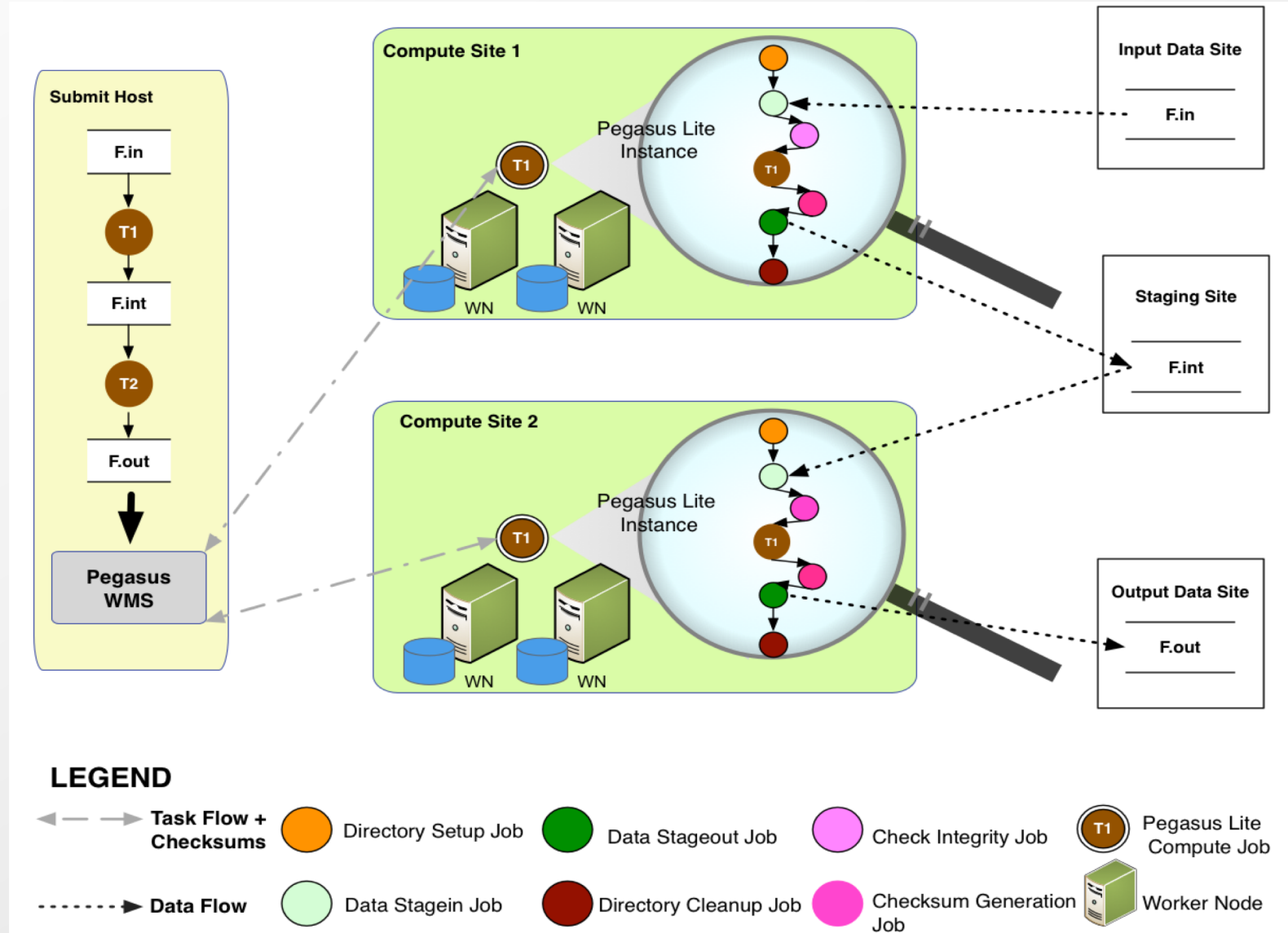
Pegasus

- Users describe their pipelines in a **portable** format called Abstract Workflow, **without worrying** about **low level execution** details.
- Pegasus takes this and **generates an executable workflow** that
 - has **data management** tasks added
 - **transforms** the workflow for **performance** and **reliability**



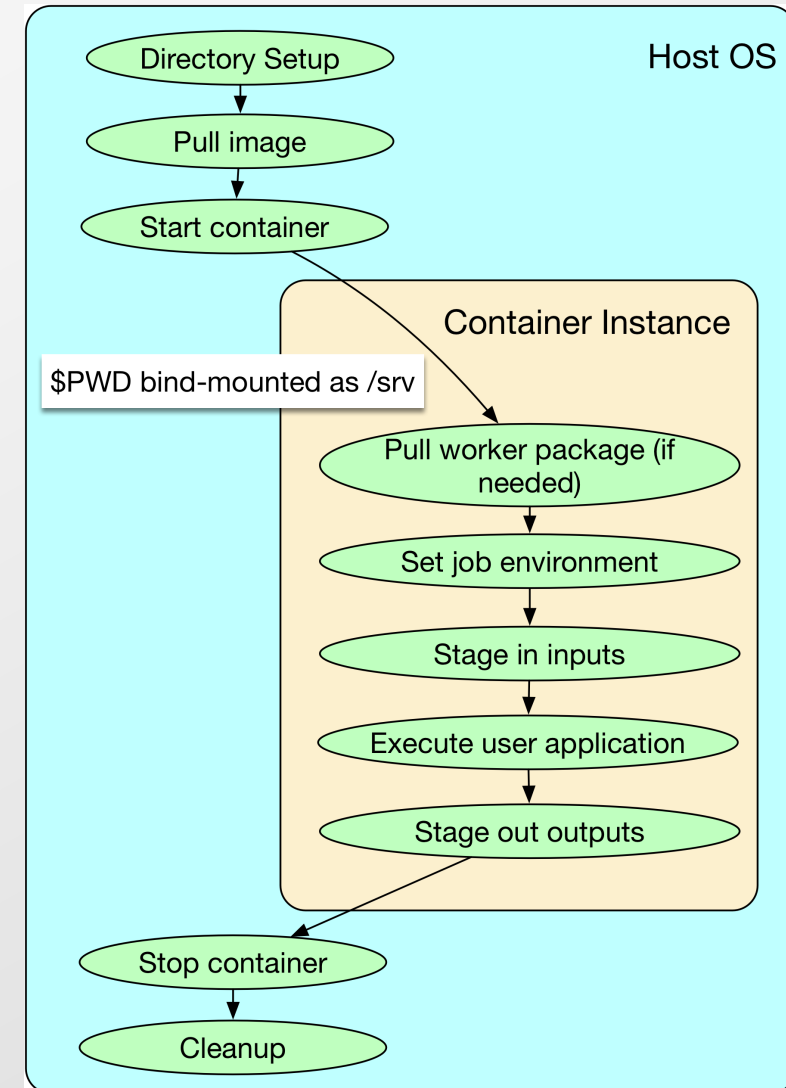
Pegasus Deployment

- Workflow Submit Node
 - Pegasus WMS
 - HTCondor
- One or more Compute Sites
 - Compute Clusters
 - Cloud
 - OSG
- Input Sites
 - Host Input Data
- Data Staging Site
 - Coordinate data movement for workflow
- Output Site
 - Where output data is placed



Pegasus: Container Execution Model

- Containerized jobs are launched via Pegasus Lite
 - Container image is put in the job directory along with input data.
 - Loads the container if required on the node (applicable for Docker)
 - Run a script in the container that sets up Pegasus in the container and job environment
 - Stage-in job input data
 - Launches user application
 - Ship out the output data generated by the application
 - Shut down the container (applicable for Docker)
 - Cleanup the job directory



Pegasus: Data Management

- Treat containers as input data dependency
 - Needs to be staged to compute node if not present
- Users can refer to container images as
 - Docker Hub or Singularity Library URL's
 - Docker Image exported as a TAR file and available at a server , just like any other input dataset.
- If an image is specified to be residing in a hub
 - The image is pulled down as a tar file as part of data stage-in jobs in the workflow
 - The exported tar file is then shipped with the workflow and made available to the jobs
 - Motivation: Avoid hitting Docker Hub/Singularity Library repeatedly for large workflows
- Symlink against a container image if available on shared filesystem
 - For e.g. CVMFS hosted images on Open Science Grid

Pegasus: Container Representation

Described in Transformation Catalog

- Maps logical transformations to physical executables on a particular system

container

Reference to the container to use.

Multiple transformation can refer to same container

type

Can be either docker or singularity or shifter

image

URL to image in a docker|singularity hub OR
to an existing docker image exported as a
tar file or singularity image

mount

Mount information to mount host directories
into container



Pegasus

- transformations

```
- namespace: "example"
  name: "keg"
  version: 1.0
site:
- name: "isi"
  arch: "x86"
  os "linux"
  pfn "/usr/bin/pegasus-keg"
  container "centos-pegasus"
```

```
# INSTALLED means pfn refers to path in the container.
# STAGEABLE means the executable can be staged into the container
type "INSTALLED"
```

- cont:

```
- name: "centos-pegasus"
```

```
# can be docker, singularity or shifter
type: "docker"
```

```
# URL to image in docker|singularity hub or shifter repo URL or
# URL to an existing image exported as a tar file or singularity image file
image: "docker:///centos:7"
```

```
# mount information to mount host directories into
# container format src-dir:dest-dir[:options]
mount:
```

```
- "/Volumes/Work/lfs1:/shared-data/:ro"
```

```
# environment to be set when the job is run in the container
# only env profiles are supported
profile:
```

```
- env:
  "JAVA_HOME" "/opt/java/1.6"
```



Outline

Motivation *Reproducibility for Workflows*

Containers *Solution for Reproducibility*
Challenges deploying for
Distributed Workflows
Design Considerations

Pegasus *Introduction*
Container Support

Experiments *Setup*
Results

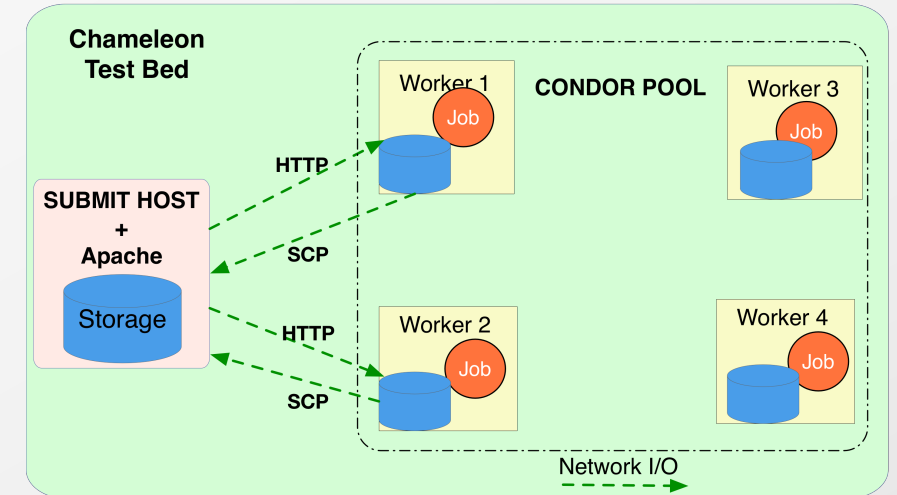
Experiments: Setup

- Used Chameleon Testbed in TACC

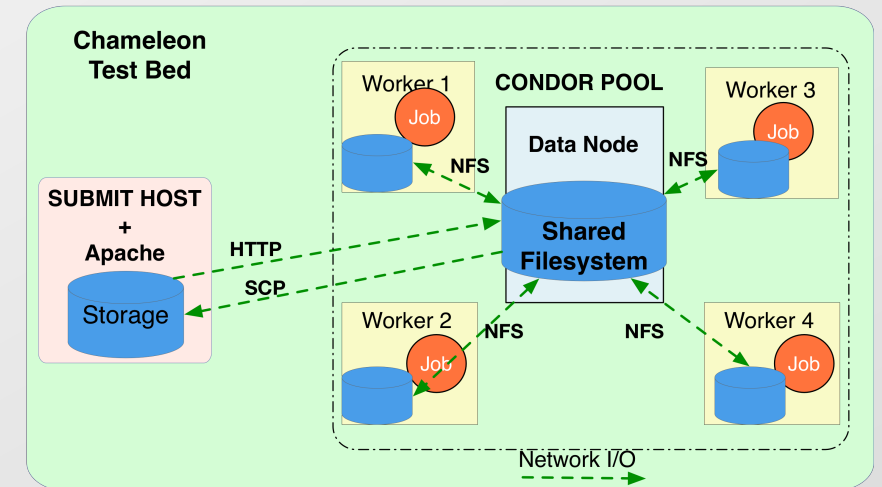
- 1 workflow submit node
- 1 NSF server node
- 4 worker nodes
- All nodes were bare metal with 24 physical cores, 128GB RAM
- 10 Gbps network connection
 - Network capped at 1Gbps

- Test Workflow

- CASA workflow with 63 compute jobs and 10 additional data transfer and auxiliary tasks



Non Shared Filesystem Setup



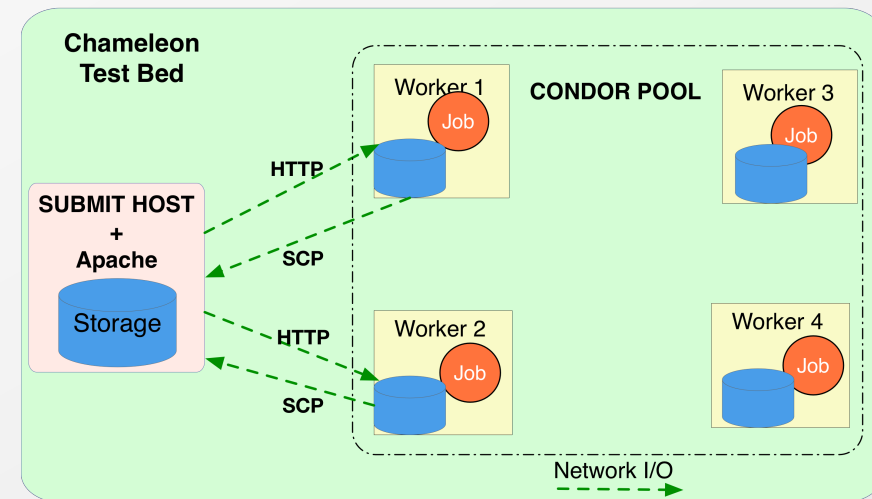
Shared Filesystem Setup

Experiments:

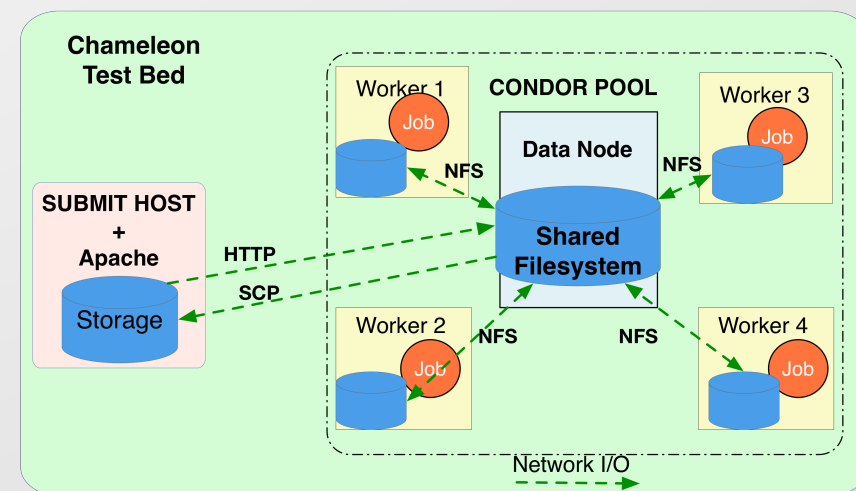
- Base experiment
 - Run CASA workflow **without** any containers in the **non shared filesystem** setup
- Experiment 2
 - Executing workflow **with Docker** and **Singularity** containers in non shared filesystem setup
- Experiment 3
 - Staged input data to **NFS** and have compute jobs **symlink** against it

Goals

- Demonstrate **increase** in **walltime** due to staging of containers and how job clustering helps
- Show staging of containers can **saturate network** and **disk IO**



Non Shared Filesystem Setup

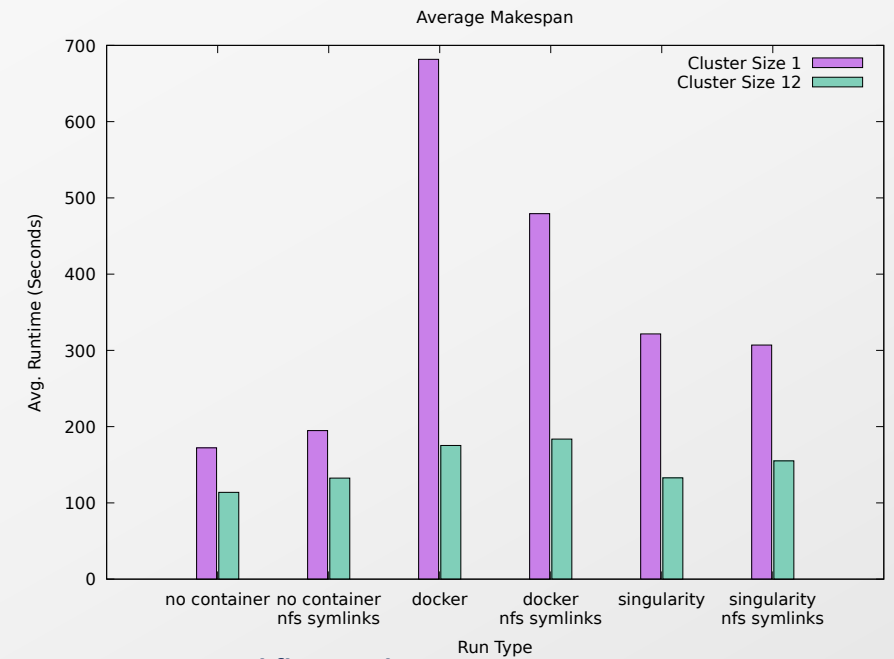


Shared Filesystem Setup

Results:

• Workflow Makespan Per Execution Setup

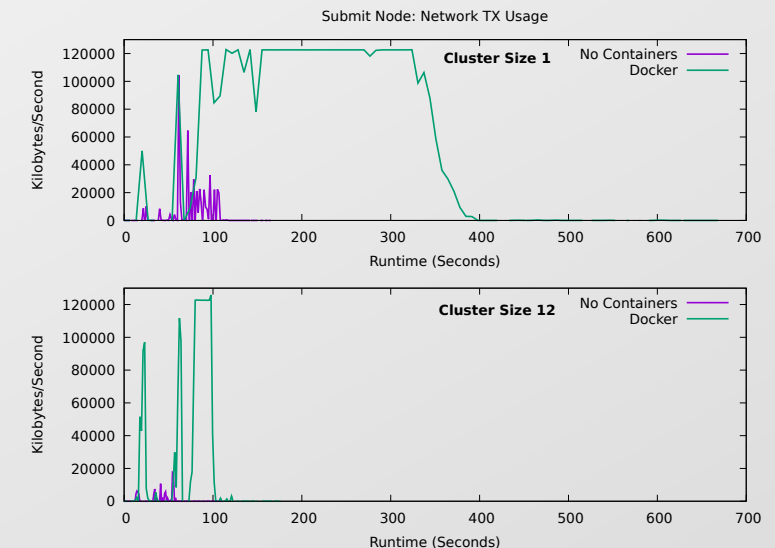
- Increase from 172.2 seconds to 681.7 and 321.6 for Docker and Singularity Containers with no job clustering.
- Clustering decreases the overhead, as container is staged once per 12 tasks.
- Docker image size 488MB vs 153 MB for Singularity image file.



Average Workflow Makespan per execution environment setup

• Egress Traffic on the Submit Node

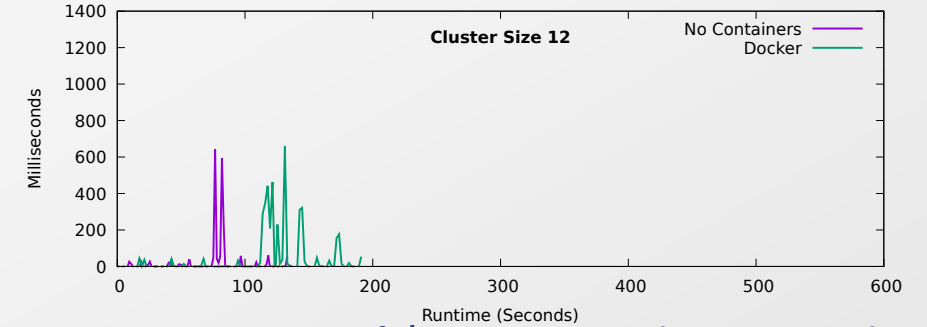
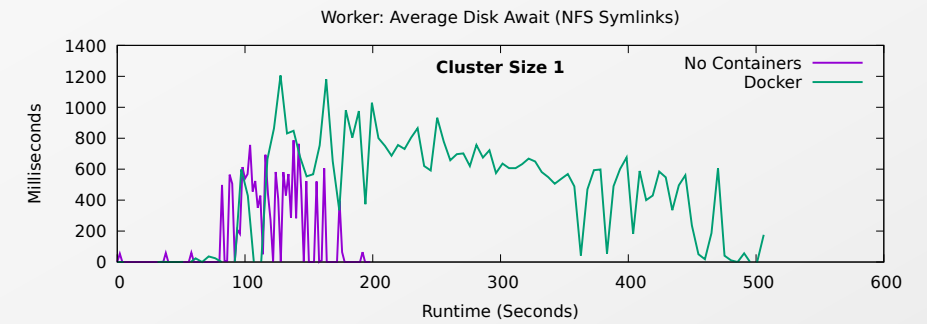
- Submit host is data staging site for the non shared filesystem setup.
- High because of transfer of associated data transfers of containers per job.



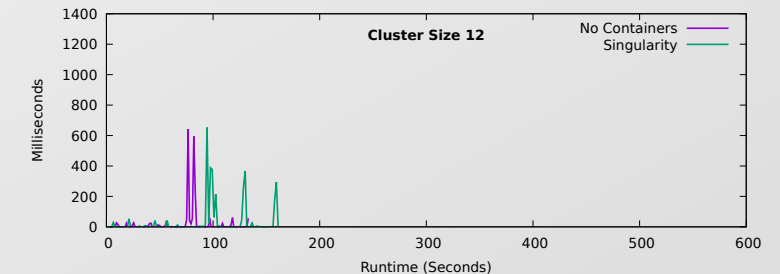
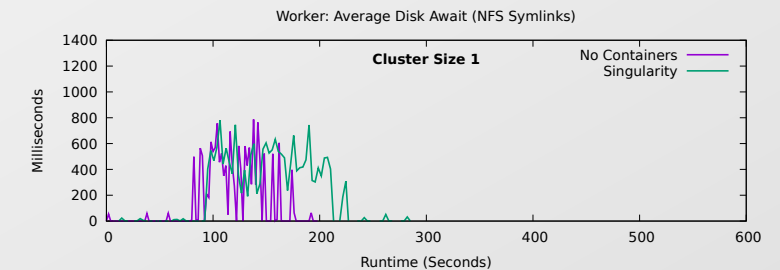
Egress network traffic on submit node , without use of containers and using Docker. NO NFS

Results:

- Average Service time I/O Requests using Docker with NFS symlinking
 - Negligible effect in case of NO containers
 - Using Docker, leads to significant increase even when symlinking.
 - Docker image still **needs to be un-tarred** on local node and **loaded** to local registry.
- Average Service time I/O Requests using Singularity with NFS symlinking
 - Singularity images are read directly
 - And are much smaller in size



Average service time of I/O requests on worker 4 using Docker containers with NFS symlinking



Average service time of I/O requests on worker 4 using Singularity containers with NFS symlinking

Case Study: LIGO PyCBC Workflows

- PyCBC

- Python based software package for exploring astrophysical sources of gravitational waves
- Used in discoveries of gravitational waves from binary black holes and binary neutron stars.

- Complex Runtime Environment

- Call functions from both Python libraries (third party and PyCBC both) and also compiled code from shared object libraries
- Requires **build** and **runtime** environments are compatible (compatible versions of glibc, gcc, python)
- For LIGO managed clusters can be solved using **virtualenv** and **standard** software installation
- However does not work for OSG and XSEDE
- Tried building bundled executables using PyInstaller. Not completely static and requires dynamically linked glibc

- Containers via Pegasus

- Deployment of containers managed by Pegasus
- Mount CVMFS inside the container for access to existing data on the site

Pegasus Container Support: Experiences

- Direct Access to Singularity Images via CVMFS
 - On OSG, singularity images distributed using CVMFS available on all nodes
 - Pegasus opted to pull image once to data staging site and pull to the compute node at runtime.
 - Disadvantage of not being able to use out of band caching and distribution made available by CVMFS
 - We updated Pegasus to **enable bypass** of container staging, and **symlink** directly against images on CVMFS
- Moved Data Staging inside of the container
 - Earlier **the data staging happened outside** of the container on the HOST OS.
 - Allowed us to **rely on infrastructure provided** tools on the HOST OS.
 - However, left user **no control** to using their **own choice** of transfer tools.
 - In Pegasus 4.9.1 moved **data staging** to occur **inside** the container
- Loading multiple Docker image tar files.
 - Adverse affect on local disk performance if multiple jobs try loading an image on the same node in a short period of time.

Questions?



Pegasus

est. 2001

Automate, recover, and debug scientific computations.

Get Started

Pegasus Website

<https://pegasus.isi.edu>

Users Mailing List

pegasus-users@isi.edu

Support

pegasus-support@isi.edu

Pegasus Online Office Hours

<https://pegasus.isi.edu/blog/online-pegasus-office-hours/>

Bi-monthly basis on second Friday of the month, where we address user questions and also apprise the community of new developments