Pegasus Workflows on OLCF - Summit

George Papadimitriou
georgpap@isi.edu

http://pegasus.isi.edu
OUTLINE

Introduction
- Scientific Workflows
- Pegasus Overview
- Success Stories

Demo
- Executing a Pegasus workflow on Summit Supercomputer

Kubernetes/OpenShift
- What is Kubernetes (Specs, Pod, Services)
- Why use Kubernetes in HPC
- OpenShift at OLCF
- Pegasus Deployment on OpenShift at OLCF

How To Deploy
- Prerequisites
- Instructions

Acknowledgements
Introduction
Compute Pipelines

Building Blocks

- Allows scientists to connect different codes together and execute their analysis
- Pipelines can be very simple (independent or parallel) jobs or complex represented as DAG’s
- Helps users to automate scale up

However, it is still up-to user to figure out

Data Management

- How do you ship in the small/large amounts data required by your pipeline and protocols to use?

How best to leverage different infrastructure setups

- OSG has no shared filesystem while XSEDE and your local campus cluster has one!

Debug and Monitor Computations

- Correlate data across lots of log files
- Need to know what host a job ran on and how it was invoked

Restructure Workflows for Improved Performance

- Short running tasks? Data placement
**Why Pegasus?**

**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**

---

NSF funded project since 2001, with close collaboration with HTCondor team

---

https://panorama360.github.io
Key Pegasus Concepts

Pegasus WMS == Pegasus planner (mapper) + DAGMan workflow engine + HTCondor scheduler/broker

Pegasus maps workflows to infrastructure
DAGMan manages dependencies and reliability
HTCondor is used as a broker to interface with different schedulers

Workflows are DAGs
Nodes: jobs, edges: dependencies
No while loops, no conditional branches
Jobs are standalone executables

Planning occurs ahead of execution

Planning converts an abstract workflow into a concrete, executable workflow
Planner is like a compiler
DAX
DAG in XML

Portable Description
Users do not worry about low level execution details

logical filename (LFN)
platform independent (abstraction)

transformation
executables (or programs)
platform independent

DAG
directed-acyclic graphs

stage-in job
Transfers the workflow input data

stage-out job
Transfers the workflow output data

cleanup job
Removes unused data

registration job
Registers the workflow output data

executable workflow

https://panorama360.github.io
Pegasus also provides tools to generate the abstract workflow...

```python
#!/usr/bin/env python

from Pegasus.DAX3 import *
import sys
import os

# Create a abstract dag
dax = DAXG("hello_world")

# Add the hello job
hello = Job(namespac="hello_world",
            name="hello", version="1.0")
b = File("f.b")
hello.uses(b, Link=Link.INPUT)
hello.uses(b, Link=Link.OUTPUT)
dax.addJob(hello)

# Add the world job (depends on the hello job)
world = Job(namespac="hello_world",
            name="world", version="1.0")
c = File("f.c")
world.uses(c, Link=Link.INPUT)
world.uses(c, Link=Link.OUTPUT)
dax.addJob(world)

# Add control-flow dependencies
dax.addDependency(Dependency(parent=hello, child=world))

# Write the DAX to stdout
dax.writeXML(sys.stdout)
```

DAG in XML

```xml
<!-- generator: Python -->
<edg xmlns="https://pegasus.isi.edu/schemas/DAX" version="1.4" name="hello_world">
  <!-- describe the jobs making up the hello world pipeline -->
  <job id="100000001" namespace="hello_world"
       name="hello" version="1.0">
    <uses name="f.b" Link=Input/>
  </job>

  <job id="100000002" namespace="hello_world"
       name="world" version="1.0">
    <uses name="f.b" Link=Output/>
  </job>

  <!-- describe the edges in the DAG -->
  <child ref="100000001"/>
  <parent ref="1000000001"/>
</dax>
```
Success Stories
60,000 compute tasks
Input Data: 5000 files (10GB total)
Output Data: 60,000 files (60GB total)
executed on LIGO Data Grid, Open Science Grid and XSEDE

Advanced LIGO – Laser Interferometer Gravitational Wave Observatory
Southern California Earthquake Center’s CyberShake

Builders ask seismologists: What will the peak ground motion be at my new building in the next 50 years?

Seismologists answer this question using Probabilistic Seismic Hazard Analysis (PSHA)

CPU jobs (Mesh generation, seismogram synthesis):
1,094,000 node-hours

GPU jobs: 439,000 node-hours

AWP-ODC finite-difference code
5 billion points per volume, 23000 timesteps
200 GPUs for 1 hour

Titan:
421,000 CPU node-hours, 110,000 GPU node-hours

Blue Waters:
673,000 CPU node-hours, 329,000 GPU node-hours

286 sites, 4 models
each workflow has 420,000 tasks
Impact on DOE Science

Enabled cutting-edge domain science (e.g., drug delivery) through collaboration with scientists at the DoE Spallation Neutron Source (SNS) facility

A Pegasus workflow was developed that confirmed that nanodiamonds can enhance the dynamics of tRNA.

It compared SNS neutron scattering data with MD simulations by calculating the epsilon that best matches experimental data.

Ran on a Cray XE6 at NERSC using 400,000 CPU hours, and generated 3TB of data.

Water is seen as small red and white molecules on large nanodiamond spheres. The colored tRNA can be seen on the nanodiamond surface.

(Image Credit: Michael Mattheson, OLCF, ORNL)

Demo Workflow

We will follow the tutorial:  
https://pegasus.isi.edu/tutorial/summit/tutorial_submitting_wf.php
Kubernetes
Kubernetes: Brief Overview

- **Kubernetes** is an open-source platform for running and coordinating containerized application across a cluster of machines.
- It can be useful for:
  - Orchestrating containers across multiple hosts
  - Control and automate deployments
  - Scale containerized applications on the fly
  - And more...

- **Key objects** in the Kubernetes architecture are:
  - **Master**: Controls Kubernetes nodes – assign tasks
  - **Node**: Perform the assigned tasks
  - **Pod**: A group of one or more containers deployed on a single node
  - **Replication Controller**: Controls how many copies of a pod should be running
  - **Service**: Allow pods to be reached from the outside world
  - **Kubelet**: Runs on the nodes and starts the defined containers

Reference: [https://www.redhat.com/en/topics/containers/what-is-kubernetes](https://www.redhat.com/en/topics/containers/what-is-kubernetes)
Kubernetes: Configuring Objects

- Within Kubernetes, **specification** files describe the applications, services and objects being deployed.

- Specification files can be written in **YAML** and **JSON** formats and can be used to:
  - Deploy Pods
  - Create and mount volumes
  - Expose services etc.

Reference:
https://kubernetes.io/docs/tasks/configure-pod-container/
Kubernetes: Pods

- A Pod is the basic execution unit of a Kubernetes application
- Pods represent processes running on the cluster
- One can have one or multiple containers running within a Pod.

- Networking: Each Pod is assigned a unique IP address within the cluster

- Storage: A Pod can specify a set of shared storage Volumes. Volumes persist data and allow Pods to maintain state between restarts.

- Lifecycle: A Pod starts running on its assigned cluster-node until the container(s) exit or it is removed for some other reason (e.g. user deletes it).

References:
https://kubernetes.io/docs/concepts/workloads/pods/pod-overview/
https://kubernetes.io/docs/concepts/workloads/pods/pod/
https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/
https://kubernetes.io/docs/concepts/storage/volumes/
Kubernetes: Services

• A **Service** provides an abstract way to expose an application running on a set of Pods as network service to the rest of the world.

• Since Pods are ephemeral, services allow users to access the backend applications via a common way.

• Service types are:
  • **ClusterIP**: Exposes the service on a cluster-internal IP.
  • **NodePort**: Exposes the service on each Node’s IP at a static port.
  • **LoadBalancer**: Exposes the service externally and loadbalances it.
  • **ExternalName**: Maps the service to a name, returns a CNAME record.

Reference: [https://kubernetes.io/docs/concepts/services-networking/service/](https://kubernetes.io/docs/concepts/services-networking/service/)
Kubernetes: Why it can be useful in HPC

• Running services on login nodes can be cumbersome (build from scratch, compile all dependences etc.) and sometimes prohibited by the system administrators.
• Maintaining an application/service up to day is easier

• **Assist workflow execution**
  • Create submission environments
  • Handle data movement and job submissions
  • Automation and Reproducibility

• **Create collaborative web portals**
  • Jupyter Notebooks
  • Workflow Design (e.g. Wings)

• **Streaming Data**
  • Consuming
  • Publishing
Kubernetes (OpenShift) at OLCF

- OLCF has deployed OpenShift, a distribution of Kubernetes developed by RedHat

- OpenShift provides a command line and a web interface to manage your Kubernetes objects (pods, deployments, services, storage etc.)

- OLCF’s deployment has automation mechanisms that allow users to submit jobs to the batch system and access the shared file systems (NFS, GPFS)

- All containers run as an automation user that is tied to a project

Reference:
Kubernetes (OpenShift) at OLCF: Pegasus Deployment
Kubernetes at OLCF: Pegasus Deployment - Advantages

• Pegasus workflow *environments* at OLCF have been *simplified*.

• Using the Kubernetes cluster at OLCF, we can deploy Pegasus submit nodes as services, within a few seconds.

• The deployment uses HTCondor’s BOSCO SSH style submissions on the DTNs and achieves submissions to the SLURM and LSF batch schedulers.

• This approach allows a single workflow to be configured to use **all** of OLCF’s resources. E.g. Execute transfers on the DTNs, run simulations and heavy processing on Summit and then do lightweight post processing steps on RHEA.
Kubernetes at OLCF: Overhead Evaluation (PEARC’20)


Best Student Paper Award
in “Advanced research computing environments – systems and system software” Track
Kubernetes at OLCF: Overhead Evaluation (PEARC’20)

Statistics from 990 compute jobs to the batch queues at OLCF!
How to Deploy

We will follow the tutorial:  [https://pegasus.isi.edu/tutorial/summit/tutorial_setup.php](https://pegasus.isi.edu/tutorial/summit/tutorial_setup.php)
How to Deploy: Prerequisites

• Pegasus Kubernetes Templates for OLCF:
  • https://github.com/pegasus-isí/pegasus-olcf-kubernetes

• OpenShift’s Origin Client:
  • https://github.com/openshift/origin/releases

• A working RSA Token to access OLCF’s systems

• An automation user for OLCF’s systems

• Allocation on OLCF’s OpenShift Cluster (https://marble.ccs.ornl.gov)
How to Deploy: Useful Origin Client Commands

- `oc login`: acquires an access token, authenticate against a cluster
- `oc status`: returns/prints the status of your deployments
- `oc describe`: shows details of a specific resource
- `oc create`: creates a Kubernetes resource from specification
- `oc start-build`: initiates the creation of a container image
- `oc logs`: returns/prints the Kubernetes log for a resource
- `oc exec`: executes a command in a container
- `oc delete`: deletes a resource
How to Deploy: Pegasus - Kubernetes Templates

- **bootstrap.sh** Generates customized Dockerfile and Kubernetes pod and service specifications for your deployment.

- **Specs/pegasus-submit-build.yml** Contains Kubernetes build specification for the pegasus-olcf image.

- **Specs/pegasus-submit-service.yml** Contains Kubernetes service specification that can be used to spawn a Nodeport service that exposes the HTCondor Gridmanager Service running in your submit pod, to outside world.

- **Specs/pegasus-submit-pod.yml** Contains Kubernetes pod specification that can be used to spawn a pegasus/condor pod that has access to Summits's GPFS filesystem and its batch scheduler.
How to Deploy: Customize Templates

In bootstrap.sh update the section "ENV Variables For User and Group" with your automation user's name, id, group name, group id and the Gridmanager Service Port, which must be in the range 30000-32767.

Replace the highlighted text:

- **USER:** with the username of your automation user (eg. csc001_auser)
- **USER_ID:** with the user id of your automation user (eg. 20001)
- **USER_GROUP:** with the project name your automation user belongs to (eg. csc001)
- **USER_GROUP_ID:** with the project group id your automation user belongs to (eg. 10001)
- **GRIDMANAGER_SERVICE_PORT:** with the Kubernetes Nodeport port number the Gridmanager Service should use (eg. 32752)

Execute Script:  

```
$ bash bootstrap.sh
```
How to Deploy: Acquire an Access Token (Step 1)

```bash
$ oc login -u YOUR_USERNAME https://marble.ccs.ornl.gov/
```

Username: olcf_user
Password:
Login successful.

You have one project on this server: "csc001"

Using project "csc001".
Create a new build and build the image:

1. $ oc create -f Specs/pegasus-submit-build.yml
   buildconfig.build.openshift.io/pegasus-olcf created

2. $ oc start-build pegasus-olcf --from-file=Docker/Dockerfile
   Uploading file "Docker/Dockerfile" as binary input for the build ...
   Uploading finished
   build.build.openshift.io/pegasus-olcf-1 started
How to Deploy: Build the Container Image (Step 2)

Trace the progress of the build:

```
$ oc logs -f build/pegasus-olcf-1

...  
Step 30/30 : LABEL "io.openshift.build.name" "pegasus-olcf-1" "io.openshift.build.type" "container"  
--- Using cache  
--- ed0f4341ff43  
Successfully built ed0f4341ff43  
PUSHING image docker-registry.default.svc:5000/cscXXX/pegasus-olcf:latest  
Pushed 2/14 layers, 14% complete  
Pushed 3/14 layers, 21% complete  
Pushed 4/14 layers, 29% complete  
Pushed 5/14 layers, 36% complete  
Pushed 6/14 layers, 43% complete  
Pushed 7/14 layers, 50% complete  
Pushed 8/14 layers, 57% complete  
Pushed 9/14 layers, 64% complete  
Pushed 10/14 layers, 71% complete  
Pushed 11/14 layers, 79% complete  
Pushed 12/14 layers, 86% complete  
Pushed 13/14 layers, 93% complete  
Pushed 14/14 layers, 100% complete  
Push successful  
```
How to Deploy: Start the Kubernetes Service (Step 3)

Start a Kubernetes Service that will expose your pod’s services:

```
$ oc create -f Specs/pegasus-submit-service.yml
service/pegasus-submit-service created
```

**Note:** In case this step fails, go back to the bootstrap.sh change the service port number and execute it again. Proceed from this step, there is no need to rebuild the container.
How to Deploy: Start the Pegasus Pod (Step 4)

Start a Kubernetes Pod with Pegasus and HTCondor:

```bash
$ oc create -f Specs/pegasus-submit-pod.yml
pod/pegasus-submit created
```

Logon to the Pod:

```bash
$ oc exec -it pegasus-submit /bin/bash
[csc001_auser@pegasus-submit /]$
How to Deploy: Configuring for Batch Submissions (Step 5)

*If this is the first time you bringing up the Pegasus container in Kubernetes we need to configure it for batch submissions.*

In the shell you got on the previous step execute:

```
$ bash /opt/remote_bosco_setup.sh
```

**Note:** This script installs some additional files needed to operate on OLCF, and prepares the environment on the DTNs, by installing BOSCO.
How to Deploy: Check the status of the deployment

If all goes well you should see something similar to this in your terminal:

```
$oc status
In project cscXXX on server https://marble.ccs.ornl.gov:443

svc/pegasus-submit-service (all nodes):32753 -> 11000
  pod/pegasus-submit runs docker-registry.default.svc:5000/cscXXX/pegasus-olcf:latest

bc/pegasus-olcf docker builds Dockerfile on istag/centos:centos7
  -> istag/pegasus-olcf:latest
  build #1 succeeded 15 minutes ago

1 info identified, use 'oc status --suggest' to see details.
```
How to Deploy: Deleting the Pod and the Service

Deleting the Pod:

$ oc delete pod pegasus-submit

Deleting the Service:

$ oc delete svc pegasus-submit-service

Deleting the container image:

$ oc delete bc pegasus-olcf
Acknowledgements

Special thanks to the OLCF people that helped us make this deployment happen!

Jason Kincl
kincljc@ornl.gov

Valentine Anantharaj
anantharajvg@ornl.gov

Jack Wells
wellsjc@ornl.gov

This work was funded by DOE contract number DESC0012636, "Panorama---Predictive Modeling and Diagnostic Monitoring of Extreme Science Workflows", and U.S. Department of Energy, Office of Science under contract DE-AC02-06CH11357.

This work used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.
George Papadimitriou
Computer Science PhD Student
University of Southern California
email: georgpap@isi.edu
Pegasus est. 2001
Automate, recover, and debug scientific computations.

Get Started

Pegasus Website
http://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*