Workflow Submit Nodes as a Service on Leadership Class Systems

George Papadimitriou
georgpap@isi.edu
University of Southern California

Karan Vahi
vahi@isi.edu
University of Southern California

Jason Kincl
kincljc@ornl.gov
Oak Ridge National Laboratory

Valentine Anantharaj
anantharajvg@ornl.gov
Oak Ridge National Laboratory

Ewa Deelman
deelman@isi.edu
University of Southern California

Jack Wells
wellsjc@ornl.gov
Oak Ridge National Laboratory

ABSTRACT

DOE scientists, today, have access to high performance computing (HPC) facilities with very powerful systems that enable them to execute their computations faster, more efficiently, and at greater scales than ever before. To further their knowledge and produce new discoveries, scientists rely on workflows - sometimes very complex - that provide them with an easy way to automate, reproduce and verify their computations. However, historically, creating workflow submission environments in large HPC facilities has been cumbersome, requires expertise and many man-hours of effort due to the peculiarities, policies, and the restrictions that these systems present. In this paper we discuss the approach a large DOE facility (OLCF) is taking in order to provide containers as a service to its users. This capability is used to create Pegasus workflow management system submit nodes as a service (WSaaS) at the Oak Ridge Leadership Computing Facilities (OLCF), targeting the Summit supercomputer. This deployment builds upon the Kubernetes/Openshift cluster (Slate) that exists within OLCF's DMZ and its automation triggers. Additionally, we evaluate our approach's overhead and effort to deploy the solution as compared to previous solutions, such as setting up a Pegasus submission environment on OLCF's login nodes or submitting jobs remotely via the rvGAHP.

CCS CONCEPTS

• **Networks** → **Cloud computing**; • **Computer systems organization** → **Cloud computing**; **Grid computing**.

KEYWORDS

Pegasus, Kubernetes, Scientific Workflows, Summit Supercomputer

ACM Reference Format:
George Papadimitriou, Karan Vahi, Jason Kincl, Valentine Anantharaj, Ewa Deelman, and Jack Wells. 2020. Workflow Submit Nodes as a Service on Leadership Class Systems. In *Practice and Experience in Advanced Research Computing (PEARC '20), July 26–30, 2020, Portland, OR, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3311790.3396671

PEARC '20, July 26–30, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6689-2/20/07…$15.00
https://doi.org/10.1145/3311790.3396671

1 INTRODUCTION

In the past decade, there has been a push for building state of the art exascale and petascale systems such as Summit at Oak Ridge Leadership Computing Facility (OLCF) [29], Cori at National Energy Research Scientific Computing Center (NERSC) [28], and Bluewaters at the National Center for Supercomputing Applications (NCSA) [27] that provide computational scientists access to thousands of cores and terabytes of memory, that are backed with reliable and scalable networks and parallel filesystems. Because of the value of these resources to science and the increasing cyber-threats, these state of the art systems have been designed with both performance and security in mind. As a result, accessing them is strictly controlled with a variety of measures, such as two-factor authentication to the login nodes, keeping the compute nodes behind a firewall, and using only designated data transfer nodes to transfer data in and out of the facilities.

For computational scientists, such systems provide an attractive option to scale up their simulations and analysis. Historically, these systems have been geared towards supporting large parallel (often MPI-based) monolithic applications. While the need to support such applications still exists, the computing landscape is also shifting towards using scientific workflows to manage the execution of these codes, the associated data transfers of the inputs, and often complex post-processing of the outputs. Scientists often setup their analysis as a workflow containing thousands of jobs that are a mix of multi-node and single node jobs, with some of them being responsible for the data movement associated with the analysis.

To manage these workflows, scientists usually rely on a workflow management system such as Pegasus [14], Makeflow [5], and Apache Airflow [17]. Ideally, scientists would prefer to submit these workflows from their desktops using remote job submission capabilities. However, the adoption of two-factor authentication on the modern HPC systems has made remote job submission extremely hard. While some techniques, such as rvGAHP [10], exist for doing remote submissions, they are hard to setup and can be difficult to debug. As a result, scientists find themselves deploying their workflow middleware stack within the science DMZ of the HPC systems, often on the login nodes. However, this approach is also not ideal, as restrictions exist that prevent easy deployment of services on login nodes. These include the lack of root access to install software and its dependencies, and also limitations on long running user processes on the login nodes. While it's possible for users to deploy and configure the workflow system of their choice in user space, it increases the difficulty for computational scientists to

use these resources effectively, since processes can be terminated by the administrators without notifying them, interrupting their experiments.

A natural solution to this is to allow users to deploy containers that have their software and workflow middleware pre-configured. However, to execute workflows from these containers, users need access to both the underlying shared filesystem and the batch scheduler. From a system administrator's perspective, allowing users to run containers on the login nodes carries risks, such as the potential of user privilege escalation. Hence, in some cases invocation of containers on the login nodes is strictly monitored with limited access to the resources, and in others it's completely prohibited. The operators of these resources recognize these challenges and have recently taken steps to make it easier for users to configure their own workflow submit nodes within the science DMZ of the HPC systems. They have setup Kubernetes/OpenShift-based clusters within their DMZ that allow users to start containers and configure their workflow middleware. These containers have access to both the shared filesystem on the cluster, and the underlying batch system that enables users and workflow systems to submit jobs to the compute resources.

In this paper, we describe our experiences of setting up and creating a Pegasus Workflow Submit Node as a service (WSaaS) at the Oak Ridge Leadership Computing Facilities (OLCF), targeting the Summit supercomputer and other OLCF resources such as RHEA, and the DTNs. This service allows computational scientists to easily spin up a fully configured Pegasus workflow management system submit node and to submit their analysis pipelines to OLCF resources. The paper is organized as follows. In Section 2, we provide an overview of the underlying technologies used by the site operations team at OLCF, and by our deployment. Section 3 describes our approach to building this service and how we bootstrapped the various components together. In Section 4, we evaluate our approach's overhead and effort to deploy this solution as compared to previous approaches, such as setting up a Pegasus workflow submission environment directly on a login node, or submitting remotely via the rvGAHP. Finally, we discuss related work in Section 5 and summarize our conclusions in Section 6.

2 BACKGROUND

2.1 Pegasus WMS

Pegasus [14] is a popular workflow management system that enables users to design workflows at a high-level of abstraction. The workflow descriptions are independent of the resources available to execute the workflow tasks and are also independent of the location of data and executables. Pegasus transforms these abstract workflows into executable workflows that can be deployed onto distributed and high-performance computing resources such as DOE Leadership Computing Facilities (e.g., NERSC [28] and OLCF [29]), shared computing resources (e.g., XSEDE [37], OSG [32]), local clusters, and academic and commercial clouds. During the compilation process, Pegasus performs data discovery, locating input data files and executables. Data transfer tasks are automatically added to the executable workflow and perform two key functions: (1) stage in input files to staging areas associated with the computing resources, and (2) transfer the generated outputs back to a user-specified location. Additionally, data cleanup (removes data that is no longer required at the execution site) and data registration tasks (that catalog the output files) are also added to the workflow. To manage user's data, Pegasus interfaces with a wide variety of backend storage systems that use different data access and transfer protocols.

Pegasus relies on HTCondor [36] DAGMan as its worklfow executor to run and manage the generated executable workflows. DAGMan in turn, submits the workflow jobs, as they become ready to run (when all parent jobs have completed successfully) to the underlying job queue managed by HTCondor. During workflow execution, provenance information from workflow and job logs is automatically parsed and stored in a relational datastore by a monitoring daemon called *pegasus-monitord* [20].

2.2 HTCondor

HTCondor [36] is a comprehensive job management system. In contrast to other batch systems such as PBS [7] and SLURM [34], it is particularly suited for distributed high throughout computing (HTC) environments, where one can setup a compute pool of nodes connected over a local area network or a wide area network. HTCondor provides users with a local job queue managed by a daemon *HTCondor Schedd* to which users submit jobs. In addition to submitting jobs to HTCondor managed compute resources, HTCondor also provides a component HTCondor-G [18] that allows users to submit jobs to grid resources. Whenever, a grid job is detected in the local job queue, HTCondor-G spawns a daemon called *GridManager* that manages the submission and monitoring of the job against the grid resource via the Grid ASCII Helper Protocol (GAHP) [19]. While developed initially to support submission the Globus GRAM [13] frontends, HTCondor-G and GridManager have been updated to support submission to other local or remote (via ssh) batch schedulers.

2.3 rvGAHP

Historically, scientists have used push-based remote job submissions for submitting resource provisioning jobs or direct submission of long running compute tasks from their local workflow submit hosts to DOE compute resources, using SSH or GRAM. With the rollout of two-factor authentication, traditional SSH based push mechanisms are no longer possible as they entail doing two-factor authentication for each SSH connection, potentially on a per job basis. Globus GRAM once a standard for remote job submission using X509 credentials is no longer supported, and as a result it's no longer deployed on newer systems. One possible solution to the two-factor authentication, is the pull-based job provisioning [26, 35] done at the remote resource. Users start pilot jobs at the remote end, which connect to a job queue on the user's submit node (outside of the DOE compute resources) and pull a list of jobs that need to be executed. The pull-based job provisioning approach has the advantage of reducing queuing delays from the job submitter perspective. However, these approaches can result in a resource mismatch between the resources required and the current set of jobs that are ready to execute on the workflow submission node. This mismatch can be of paramount concern when executing thousands of jobs

that collectively use hundreds of thousands of hours of computing resources.

In order to achieve push-based remote job submissions a new technique, called "reverse GAHP" (rvGAHP), was presented in [10]. It uses a reverse SSH connection from the remote resource to connect to an HTCondor GridManager process on the user workflow submit node. This allows HTCondor-G GridManager process to automatically connect to a GAHP process running on the remote resource, whenever it detects a job in the local HTCondor job queue on a workflow submit node. This approach achieves push-based job submissions, without requiring DOE resources to accept SSH connections and effectively circumvents the two-factor authentication. While this approach achieves high efficiency of resource usage similar to normal push-based job submissions, it is a complicated setup for a normal user to do. It requires users to compile and run HTCondor in user space, on the DOE resources. If we compare it to the direct job submission to the batch scheduler, it suffers longer latencies between the jobs finishing on the remote resource and them being detected by the HTCondor daemon on the workflow submit node.

2.4 Kubernetes

Kubernetes is a second generation descendent of Borg, the container management system used by Google to manage long-running latency-sensitive services as well as resource-intensive batch jobs [9, 38]. Kubernetes is described as "a portable, extensible, open-source platform for managing containerized workloads and services" [23].

Traditionally, scientific workflows and application workloads were deployed on physical servers that often resulted in inefficient utilization of resources, poor performance, and resource contention. Further, the workloads were not easy to scale, and expensive to deploy and maintain. The advent of virtual machines (VM) and support for them in hardware allowed multiple virtual machines to be hosted on the same hardware with the applications being securely isolated among the VMs [1]. However, this deployment approach is also relatively resource-intensive involving the VMs hosting the complete operating system and the associated software stack and components.

Lately, container technologies have been widely adopted with Kubernetes being one of the available platforms for managing the containerized workloads. Containers have several advantages over VMs. Containers are lightweight. They are relatively easy to create, deploy and manage. Containers embrace runtime isolation while running an application image on an operating system. Kubernetes provides a platform for the deployment of containers across the cluster with a workload that can be scaled in response to demand.

Kubernetes can be described and explained in terms of a set of useful abstractions that represent the state of the system, including what containerized applications are running and the network and storage resources associated with the workloads. A *Pod* is the most basic execution unit that could be created and deployed in Kubernetes. All containers in a *Pod* are guaranteed to be scheduled onto a single node. A single *Pod* typically runs only one application container, however multiple application containers can co-exist inside the same *Pod*, which controls their execution. The *Pod* ceases to exist when the containers exit.

A Kubernetes *Service*, on the other hand, 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 application containers via a common way. There are multiple *Service* types. One is the *Nodeport* which exposes the *Service* on each host node's IP at a static port.

One of the main design goals of Kubernetes is easier deployment and management of complex distributed systems while taking advantage of the optimal resource utilization facilitated by containerization that has "transformed the data center from being machine-oriented to being application oriented"[9].

2.5 Project Slate

Slate is a cluster resource for deploying and running user-managed persistent application services at the OLCF. It is built on top of OpenShift [21], an open source container application platform developed by Red Hat based on Kubernetes, and provides container orchestration services to OLCF users, allowing them to run services that do not fit into a batch job, such as workflow management systems.

Slate is integrated with the OLCF's existing HPC environment, including the GPFS file system (Alpine) and Summit's batch scheduler, which are available inside the container. Access to the shared file systems (Alpine and NFS) is offered through the host executing the containers, while access to the batch schedulers (Summit, Rhea, DTN) is provided over SSH to the DTNs, since they have cross cluster job submission capabilities to Summit's and Rhea's batch queues. To offer this level of integration transparently to the users, custom triggers and hooks have been built into the cluster that make the HPC environment resources available via annotations in the deployment specification of the *Pods*. Services running on the cluster are available externally to all project users and can be reached from the compute clusters at OLCF, via direct network access.

Within the Slate cluster, user-invoked *Pods* and services run under the automation user assigned to a project. This is a unique type of user within OLCF that has some privileges in addition to those of the regular user accounts. The automation user comes with long lasting ssh-keys, generated by the facility, which are white-listed on OLCF's DTN nodes, and allow password-less SSH authentication from the Slate cluster IP range without the use of two-factor authentication. This functionality is fundamental to the way job submissions are supported, since SSH access to the DTNs in an automated, but secure, way is necessary.

3 APPROACH

As described in Section 2.5, the Slate cluster allows OLCF users to deploy persistent application services with access to major HPC computing resources, such as the shared file systems and the batch schedulers of Summit and Rhea. This integration makes the deployment of a fully functional workflow management system, like Pegasus, feasible on cutting edge HPC infrastructure. However, spawning such a service is not a trivial task if starting from scratch. It requires a good understanding of the underlying infrastructure, of the workflow management system and its dependencies, and a lot of attention has to be given to the details in order to make

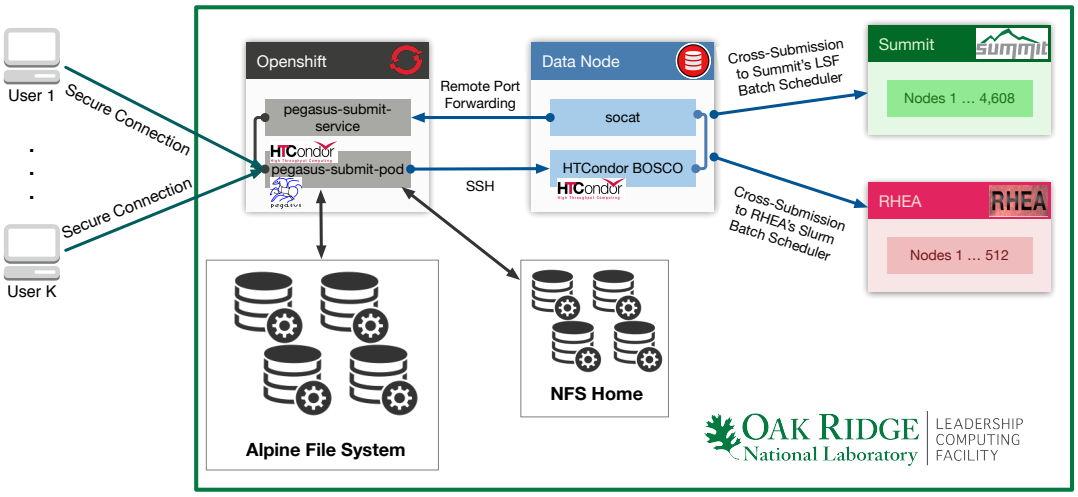

Figure 1: Pegasus Kubernetes (OpenShift) Deployment at OLCF.

the deployment reliable and easy to use. Our approach builds on top of the infrastructure and the automation introduced with the Slate Kubernetes (OpenShift) cluster, and we developed Pegasus Workflow Submit Nodes as a Service (WSaaS) that can leverage OLCF's resources, while bypassing the two-factor authentication for workflow jobs and keeping the deployment simple to use.

Figure 1 illustrates our Pegasus Kubernetes deployment at OLCF. In this deployment users authenticate themselves against the Slate cluster using two-factor authentication and can bring up a Pegasus workflow submission Pod, within OLCF's DMZ. The Pod is configured for SSH style job submissions with HTCondor BOSCO[40] on the DTNs. Access to both the GPFS filesystem (Alpine) and NFS is provided from within the Pod, and since Pods are ephemeral, a persistent state of the workflow execution is maintained there. Finally, as described in Section 2.5, the cross-submission functionality of the DTNs is used to submit and monitor the batch jobs to OLCF's compute clusters.

Four main components contribute to the realization of this deployment. The *pegasus-submit-pod*, *pegasus-submit-service*, *socat* and *HTCondor BOSCO*.

pegasus-submit-pod. This Pod runs a container image with Pegasus and HTCondor installed, and is configured to submit jobs to OLCF resources. To create the container image we have developed a generalized Dockerfile recipe that prepares it for an OLCF automation user on the Slate cluster. An account for the automation user is created in the image, with the same user id and group id assigned by OLCF administrators. Since some configuration steps need to take place after the initialization of the Pod, we have automated the process with a use of a container entrypoint script. In this script the following actions take place:

- SSH keys of the automation user are soft-linked to a location where BOSCO expects to find them
- HTCondor binaries become aware of the Pod's hostname, to avoid errors in daemon processes
- BOSCO becomes aware of the address the Slate cluster exposes HTCondor's GridManager service, via an environmental variable

Additionally, inside the Pod exists a script that configures the environment on OLCF's DTNs, which has to be invoked once, the very first time the service is used.

HTCondor BOSCO. One of the requirements that the Slate cluster imposes to the users, is that all the job submissions have to be done through a DTN node, which is decoupled from the container environment spawned in Kubernetes. The expectation is that users connect to the DTN over SSH to submit and monitor jobs. In the case of Pegasus, we can achieve SSH based job submissions using a technology called BOSCO [40]. BOSCO is a tool shipped with HTCondor that allows users to submit HTCondor jobs over SSH to remote batch clusters. To achieve job submissions over SSH, BOSCO connects the local HTCondor GridManager on the workflow submit node, to a Remote GAHP process on a node (in our case the DTNs) that can submit and monitor jobs to a batch scheduler. Traditionally, in order to connect the two processes, BOSCO uses SSH remote port-forwarding to forward packets from the Remote GAHP process back to the GridManager of the submit node. However, this functionality is not available to the automation user accounts on the DTNs. To work around this in our deployment we introduced the *pegasus-submit-service* on the Slate cluster and the *socat* process on the DTNs.

pegasus-submit-service. This Kubernetes *Service* exposes HTCondor's GridManager port to the rest of the OLCF computing resources (e.g., the DTNs). This is a requirement for the Remote GAHP, since it needs to contact the GridManager service directly. The pegasus-submit-service is configured to use Kubernetes' "Nodeport" service type, which binds the GridManager's port to a port on the Slate cluster between 30000 and 32767. Our deployment requires one pegasus-submit-service for each pegasus-submit-pod, since each one of them runs its own GridManager. The assignment of the "Service" to a "Pod" is done dynamically by the Slate cluster, via annotations in the deployment configuration scripts.

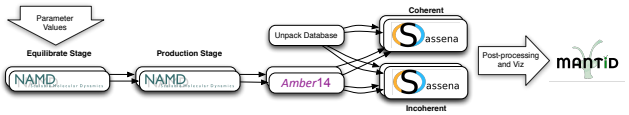

Figure 2: A diagram of a branch of the SNS workflow.

socat. Socat [15] is a command line-based utility that establishes two bidirectional byte streams and transfers data between them. It is deployed on the DTNs in order to allow the Remote GAHP to communicate with the HTCondor GridManager service, by forwarding TCP packets destined for the data nodes loopback device on a specific port, to the pegasus-submit-service. Every time the Remote GAHP needs to communicate with the GridManager service, a socat process is started and it is terminated after the end of the communication.

3.1 Produced Artifacts

Since we started this work, we had the vision of providing the OLCF users with a generic way to bring up the entire setup presented in Figure 1, without the need to dive into all the configuration details. Thus, we have developed a templated approach[30] that is publicly available on GitHub and can be used by any OLCF user interested in spawning a Pegasus workflow submit node on the Slate cluster. It aids in the preparation of a custom "pegasus-submit-pod" container image for any OLCF automation user and provides deployment specification files in YAML format, which spawn the "pegasus-submit-service" and "pegasus-submit-pod" on the Slate cluster, using OpenShift's origin client [22]. More specifically this GitHub repository includes the following files:

- **bootstrap.sh** generates the personalized Dockerfile and Kubernetes specifications for a deployment.
- **pegasus-submit-build.yml** contains Kubernetes build specifications for building the container image.
- **pegasus-submit-service.yml** contains Kubernetes *Service* specification that can be used to spawn a Nodeport service that exposes the HTCondor Gridmanager running in a pegasus submit Pod, to outside world.
- **pegasus-submit-pod.yml** contains Kubernetes *Pod* specification that can be used to spawn a pod with Pegasus and HTCondor that has access to OLCF's GPFS filesystem and the batch schedulers.

Additionally, while developing this approach we discovered that the HTCondor scripts monitoring jobs on the LSF batch queues weren't compatible with the available environment on the OLCF DTNs. We enhanced the LSF monitoring scripts, with logic that queries the status of all jobs and persistently stores the results in a cache that reduces polling overheads. Currently, we are in touch with the HTCondor developers to contribute these enhancements back to HTCondor's main branch.

4 EVALUATION

To evaluate the Pegasus Kubernetes (OpenShift) deployment at OLCF, we consider four aspects: 1) ease of deployment, 2) job submission delays, 3) functionality, and 4) limitations of the possible

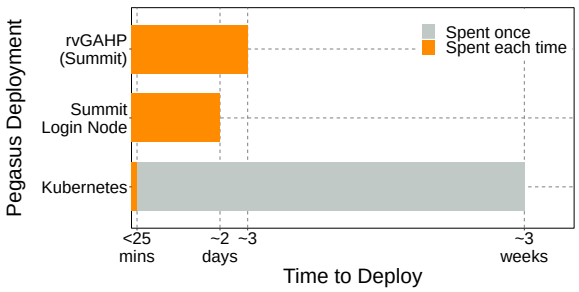

Figure 3: Time to Deploy on a New System

approaches. Throughout these categories we compare the three Pegasus deployment options: local deployments on the login nodes, deployments that use the rvGAHP technique for remote submissions, and the Kubernetes solution. Moreover, to assess overheads in the job submission delays present in the three approaches, we configured the Nanodiamond Pegasus Workflow to run on Summit and Rhea, in order to collect statistics.

Nanodiamond Pegasus Workflow. The Nanodiamond Pegasus workflow is a material science workflow developed by scientists at the Spallation Neutron Source (SNS) [25], a DOE research facility at Oak Ridge National Laboratory. The SNS workflow executes an ensemble of molecular dynamics (MD) and neutron scattering intensity calculations to optimize a model parameter value, for example, to investigate temperature and hydrogen charge parameters for models of water molecules. The workflow takes as input a set of temperature values and four additional parameters: (1) type of material, (2) the number of required CPU cores, (3) the number of timesteps in the simulation, and (4) the frequency at which the output data is written. Figure 2 shows a branch of the workflow that analyzes a single temperature value. First, each set of parameters is fed into a series of parallel molecular dynamics simulations using NAMD [31]. The first simulation computes an equilibrium, which is used by the second simulation to compute the production dynamics. The output from the MD simulations has the global translation and rotation removed using AMBER's [33] *cpptraj* utility, which is passed into Sassena [24] to compute coherent and incoherent neutron scattering intensities from the trajectories. The final outputs of the workflow are transferred to the user's desktop and loaded into Mantid [6] for analysis and visualization.

4.1 Ease of Deployment

To assess the difficulty of configuring these three Pegasus deployments at OLCF, we initially deployed Pegasus and HTCondor on Summit's login nodes, and alongside HTCondor we installed the rvGAHP server-side binaries. To test the rvGAHP setup we used a Pegasus submit node at the USC Information Sciences Institute (USC/ISI), where we installed rvGAHP's client-side binaries. Figure 3 presents the approximate time each of the deployments took us to setup, and assumes that the user has an average level of system deployment knowledge and experience. The deployment on Summit's login node took us about 2 days. Configuring the rvGAHP took us about 3 days, since it involves additional steps, while testing and debugging take longer due to its complexity. On the other

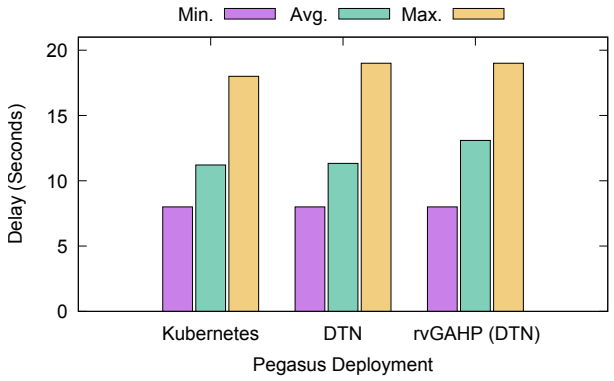

Figure 4: HTCondor Queue Time to Submission

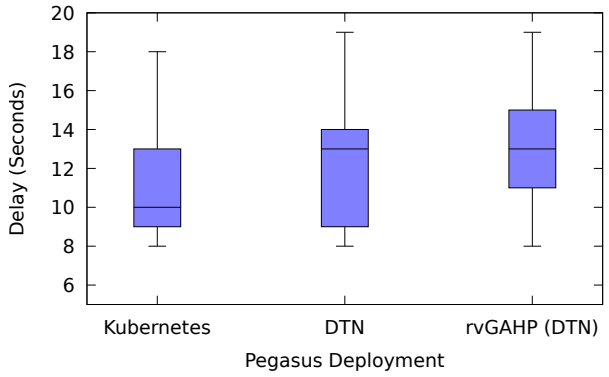

Figure 5: HTCondor Queue Time Distribution

hand, the Pegasus Kubernetes deployment took us about 3 weeks to create, test, and harden the solution. However, this was a cost that we paid once designing the deployment, and now we can set it up reliably in less than 25 minutes. This time accounts for building the container image, spawning the service and doing the one time required configuration on the DTNs.

All the deployments after their initial configuration require a few seconds to be initialized. The main difference is how much time it takes to create the environment for the first time, test and debug it. With the Kubernetes deployment, all the steps are well documented and there are very few places where a user error can occur (less than six commands are required to deploy). Whereas, the login node and the rvGAHP deployments require more complicated steps of compiling the source code and configuring the environment manually, which is more prone to errors and requires a certain level of expertise. Additionally, the steps for the login node and the rvGAHP deployments have to be done for each new machine and for each user that's interested in running the Pegasus WMS on that machine. Thus, compared to the other two approaches, the Pegasus Kubernetes deployment is far easier to maintain and keep up-to-date, and because it needs minimal effort to deploy, it is more friendly to the users, and as a result more likely to be deployed.

4.2 Job Submission Delays

To evaluate job submission delays, we configured comparable environments to the Pegasus Kubernetes approach for both the "Login Node" and rvGAHP deployments. All of the deployments submit to the compute clusters (Summit and RHEA) via the cross submission functionality of the DTNs, and the data transfers are handled by the DTNs or the workflow submit node. In the first scenario we used the Pegasus Kubernetes deployment as seen in Figure 1. In the second scenario we deployed Pegasus and HTCondor on one of OLCF's DTNs. In the third scenario we installed the rvGAHP server-side binaries on the same DTN and used a Pegasus submit node located at USC/ISI to conduct the workflow runs.

In our experiments, we used a version of the Nanodiamond Pegasus workflow (Figure 2) with two branches, for two different temperatures. This resulted in a workflow DAG that contained 11 compute jobs (8 MPI jobs and 3 single core jobs). Originally, the Nanodiamond workflow ingests a few MBs of configuration files and outputs over 15GBs of data, while having a runtime of over 10 hours. For our tests we reduced the output to a few MBs and the runtime to about 20 minutes, since we were interested only in studying job submission delays. We configured the workflow to execute Equilibrate and Production stages on Summit, since NAMD can offload work to the GPUs. To execute *cpptraj*, as well as the Coherent and Incoherent neutron scattering intensities calculations, we used RHEA. Finally for unpacking the Sassena DB and the data movement we used the DTNs. For each deployment scenario we did *30 workflow runs* and in total we submitted *990 compute jobs* to the batch queues at OLCF.

Figure 4 presents the time each compute task spent in HTCondor's queue, as observed by HTCondor, for the different deployment scenarios. This delay quantifies the time it took for the job to be successfully submitted to the batch queues (remote) since it was submitted to the HTCondor queue (local). This statistic is retrieved from HTCondor logs, where it is logged with a precision of a second. For all the deployments we observe a minimum delay of 8 seconds across the batch queues (violet color), with the maximum observed delay (orange color) for the Kubernetes deployment being 1 second lower than the rest (18 vs 19 seconds). Moreover, the average HTCondor queue delay (green color) for the Kubnernetes and the DTN (login node) deployments, are comparable at ~11.20 seconds, while the average queue delay for the rvGAHP is ~13.10 seconds, approximately 2 seconds higher. In Figure 5, the distribution of the HTCondor queue time is depicted. Minimum and maximum delays are mainly outliers, and while median delay for the "Login Node" and rvGAHP approaches are comparable, the median delay for Kubernetes is smaller by ~3 seconds.

Having Figures 4 and 5 in mind, the Kubernetes deployment doesn't impose additional overheads over the "Login Node" approach and overall it provides faster submission turnaround times, due to the isolation the Kubernetes cluster provides. The additional delays observed in rvGAHP's deployment can be attributed to rvGAHP's client and server overhead, and even though our submit node at ISI had a higher round-trip time than that of the local OLCF machines, it was only measured at 75*ms*, which shouldn't have affected the average rvGAHP queue delay by much.

4.3 Functionality

The Pegasus Kubernetes deployment provides the same functionality as having Pegasus deployed on a login node. Multiple users assigned to a project can access the Slate cluster and spawn their own "pegasus-submit-pod" under the project's automation user. Additionally, the deployment leverages all the major OLCF compute resources. Auxiliary tasks, such as data movement, can be carried out locally on the submit node or at the batch queues of the DTNs, and compute tasks, such as simulations, can be submitted to both Summit and RHEA using the LSF and Slurm batch scheduling systems.

A challenge when using virtual infrastructures is that workflow-level checkpointing provided by Pegasus may not work. Pegasus generates *rescue worfklows* while the workflow is executing–this is done by saving the intermediate data products and the state of the workflow. This information can be used to restart a workflow execution from the last known state in case of a failure. Even though Pods are ephemeral, this functionality is still offered to the users, since we maintain all the workflow generated files and logs in the shared filesystem.

Finally, due to Pegasus' versatile job mapping capabilities, a single workflow can be mapped to all the aforementioned resources (Summit, Rhea) transparently for the user. Therefore, scientists can configure a single analysis workflow that automatically executes transfers on the DTNs, runs the simulations and the heavy processing on Summit and then does the lightweight post processing steps on Rhea.

4.4 Limitations

Despite its positive contributions, such as minimizing the time to deploy, our approach also comes with its limitations. First of all, unlike the rvGAHP approach, this deployment cannot be used for remote job submissions, as it requires the Pegasus submit node within OLCF's DMZ, running on the Slate cluster. While we don't consider this to be a significant drawback, it can be a deal-breaker for some user operational scenarios. Furthermore, it imposes restrictions on the number of running jobs a single *project allocation* can have on Summit and Rhea. OLCF imposes per-user limits to the number of *eligible to run jobs* and the maximum number of *queued jobs* that exist in the batch queues. All the jobs submitted from our "pegasus-submit-pod" are done under a project's automation user. Because of this, our deployment sets the limit of the number of jobs a project can submit to the batch queues equal to the limits of a single user. This limitation doesn't exist in the "Login Node" and rvGAHP deployments, since each user can submit jobs under their own user id.

5 RELATED WORK

Until recently, there has not been much work in allowing users to setup their own workflow submit nodes within the science DMZ of large, high performance computing facilities. One of the earlier efforts, is the virtual clusters framework [39] developed at SDSC that provides virtual HPC clusters to projects using the NSF sponsored Comet supercomputer. Another related project is VC3 [8] that allows users to install custom software environments and automates deployment of cluster frameworks from different resources to access diverse computing resources for collaborative science teams.

However, now, HPC facilities are starting to see the benefits that containerized applications as services can have for their users and their operations. They are actively working on integrating systems like OpenShift [21] and Rancher [2] into their infrastructure. The National Energy Research Scientific Computing Center (NERSC) is offering a service that can be used to spawn Docker containers, called Spin [11]. Spin uses the Rancher-cli to create containers and users can thus deploy workflow managers, databases, science gateways and more. Spin is not as tightly integrated with the main HPC resources at NERSC, as project Slate at OLCF (Section 2.5). However, NERSC, in the past, has accommodated long running workflows by providing ssh-keys that lift the two-factor authentication requirement.

In terms of getting jobs submitted from a remote workflow or workload management system to OLCF resources, others have explored pull-based solutions, in order to work-around the two-factor authentication barrier. Projects such as IceCube [4] and ATLAS [3] have used pull-based pilot job resource provisioning to fetch and execute jobs on OLCF Titan using pyGlidein and Panda [12, 26, 35]. The Southern California Earthquake Center (SCEC) uses the rvGAHP [10] approach to do push-based job submissions to OLCF Summit. On the other hand, other facilities have opted to provide rest endpoints to satisfy their users that rely on remote submissions for their science. The Texas Advanced Computing Center (TACC) provides an Agave [16] REST API front-end that allows users to submit jobs to TACC resources from their science gateways.

6 CONCLUSION

In this paper we presented how we leveraged the Slate cluster within OLCF's DMZ, in order to create a Workflow Submit Node as a Service (WSaaS) that uses Pegasus as its workflow management system. Our approach creates a fully functional Pegasus submit node in less than 25 minutes, compared to other solutions that require experience with the workflow management system software and its dependencies, and need days of testing and debugging. We compared our newly proposed approach for submitting jobs to OLCF's Summit and Rhea systems with a deployment on a "Login Node" and remote submissions that use the rvGAHP technique. We found that it introduces fewer delays than the rvGAHP solution and has a slight edge over deployments on login nodes (such as the DTNs), due to the resource isolation Kubernetes provides. Additionally, we developed and made publicly available on GitHub, a templated approach that can be used by any OLCF user that's interested in deploying a Pegasus submit node on the Slate cluster.

In the future we would like to explore how Kubernetes' persistent storage volumes can be used across Pods, and what would be their benefits in the context of workflow execution environments. Additionally, since Kubernetes' deployments support automatic restart of Pods that died suddenly, we are planning to enhance our Pegasus Kubernetes deployment to automatically salvage running workflows as Pods become available again and reduce the time scientists need to check on the status of their workflow runs.

We believe that as more OLCF users become aware of the simplicity and of the benefits that this approach has on executing workflows on OLCF's resources, more scientists will begin adopting it and focus more on the science aspects of their research, instead of worrying about how to access the computing resources available to them.

ACKNOWLEDGMENTS

This work was funded by DOE contract number #DESC0012636, "Panorama—Predictive Modeling and Diagnostic Monitoring of Extreme Science Workflows", and by the U.S. Department of Energy, Office of Science under contract DE-AC02-06CH11357. Also, this research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. Finally, we would like to thank Scott Callaghan from SCEC for his help in debugging the rvGAHP deployment on Summit's login nodes and OLCF's data transfer nodes (DTNs).

REFERENCES

[1] Containers vs. Virtual Machines (VMs): What's the Difference? https://blog.netapp.com/containers-vs-vms/.
[2] Rancher. https://rancher.com/.
[3] G. Aad et al. 2008. The ATLAS Experiment at the CERN Large Hadron Collider. *JINST* 3 (2008), S08003. https://doi.org/10.1088/1748-0221/3/08/S08003
[4] M. G. Aartsen et al. 2017. The IceCube Neutrino Observatory: Instrumentation and Online Systems. *JINST* 12, 03 (2017), P03012. https://doi.org/10.1088/1748-0221/12/03/P03012 arXiv:astro-ph.IM/1612.05093
[5] Michael Albrecht, Patrick Donnelly, Peter Bui, and Douglas Thain. 2012. Makeflow: A Portable Abstraction for Data Intensive Computing on Clusters, Clouds, and Grids. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies* (Scottsdale, Arizona, USA) *(SWEET '12)*. Association for Computing Machinery, New York, NY, USA, Article Article 1, 13 pages. https://doi.org/10.1145/2443416.2443417
[6] Owen Arnold, Jean-Christophe Bilheux, JM Borreguero, Alex Buts, Stuart I Campbell, L Chapon, M Doucet, N Draper, R Ferraz Leal, MA Gigg, et al. 2014. Mantid—data analysis and visualization package for neutron scattering and μ SR experiments. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 764 (2014), 156–166.
[7] A. Bayucan, R. L. Henderson, C. Lesiak, B. Mann, T. Proett, and D. Tweten. 1999. Portable batch system: External reference specification. In *Technical report, MRJ Technology Solutions*, Vol. 5.
[8] Lincoln Bryant, Jeremy Van, Benedikt Riedel, Robert W. Gardner, Jose Caballero Bejar, John Hover, Ben Tovar, Kenyi Hurtado, and Douglas Thain. 2018. VC3: A Virtual Cluster Service for Community Computation. In *Proceedings of the Practice and Experience on Advanced Research Computing* (Pittsburgh, PA, USA) *(PEARC '18)*. Association for Computing Machinery, New York, NY, USA, Article Article 30, 8 pages. https://doi.org/10.1145/3219104.3219125
[9] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, Omega, and Kubernetes. *ACM Queue* 14 (2016), 70–93. http://queue.acm.org/detail.cfm?id=2898444
[10] Scott Callaghan, Gideon Juve, Karan Vahi, Philip J. Maechling, Thomas H. Jordan, and Ewa Deelman. 2017. rvGAHP: Push-Based Job Submission Using Reverse SSH Connections. In *Proceedings of the 12th Workshop on Workflows in Support of Large-Scale Science* (Denver, Colorado) *(WORKS '17)*. Association for Computing Machinery, New York, NY, USA, Article Article 3, 8 pages. https://doi.org/10.1145/3150994.3151003
[11] National Energy Research Scientific Computing Center. Spin. https://docs.nersc.gov/services/spin/.
[12] CHEP 2018. *Big Panda Workflow Management System on OLCF Titan for HENP and extreme scale applications.* https://indico.cern.ch/event/587955/contributions/2937286/attachments/1683059/2705499/Klimentov_CHEP-Jul2018.pdf
[13] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. 1998. A resource management architecture for metacomputing systems. In *Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson and Larry Rudolph (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 62–82.
[14] Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip J Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and Kent Wenger. 2015. Pegasus: a Workflow Management System for Science Automation. *Future Generation Computer Systems* 46 (2015), 17–35. https://doi.org/10.1016/j.future.2014.10.008 Funding Acknowledgements: NSF ACI SDCI 0722019, NSF ACI SI2-SSI 1148515 and NSF OCI-1053575.
[15] Dest-Unreach. Socat. http://www.dest-unreach.org/socat/.
[16] Rion Dooley, Steven R. Brandt, and John Fonner. 2018. The Agave Platform: An Open, Science-as-a-Service Platform for Digital Science. In *Proceedings of the Practice and Experience on Advanced Research Computing* (Pittsburgh, PA, USA) *(PEARC '18)*. Association for Computing Machinery, New York, NY, USA, Article Article 28, 8 pages. https://doi.org/10.1145/3219104.3219129
[17] Apache Software Foundation. Apache Airflow. https://airflow.apache.org/.
[18] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steve Tuecke. 2001. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC)*. San Francisco, California, 7–9.
[19] GAHP. Grid ASCII Helper Protocol. https://research.cs.wisc.edu/htcondor/gahp/.
[20] Dan Gunter, Ewa Deelman, Taghrid Samak, Christopher Brooks, Monte Goode, Gideon Juve, Gaurang Mehta, Priscilla Moraes, Fabio Silva, Martin Swany, and Karan Vahi. 2011. Online Workflow Management and Performance Analysis with Stampede. In *7th International Conference on Network and Service Management (CNSM-2011)*.
[21] Red Hat. OpenShift. https://www.openshift.com/.
[22] Red Hat. Origin Client. https://github.com/openshift/origin.
[23] Kubernetes. https://kubernetes.io/.
[24] Benjamin Lindner and Jeremy C Smith. 2012. Sassena—X-ray and neutron scattering calculated from molecular dynamics trajectories using massively parallel computers. *Computer Physics Communications* 183, 7 (2012), 1491–1501.
[25] TE Mason, D Abernathy, I Anderson, J Ankner, T Egami, G Ehlers, A Ekkebus, G Granroth, M Hagen, K Herwig, et al. 2006. The Spallation Neutron Source in Oak Ridge: A powerful tool for materials research. *Physica B: Condensed Matter* 385 (2006), 955–960.
[26] F H Barreiro Megino, K De, A Klimentov, T Maeno, P Nilsson, D Oleynik, S Padolski, S Panitkin, and T Wenaus and. 2017. PanDA for ATLAS distributed computing in the next decade. *Journal of Physics: Conference Series* 898 (oct 2017), 052002. https://doi.org/10.1088/1742-6596/898/5/052002
[27] NCSA. National Center for Supercomputing Applications (NCSA). http://www.ncsa.illinois.edu/.
[28] NERSC. National Energy Research Scientific Computing Center (NERSC). https://www.nersc.gov.
[29] OLCF. Oak Ridge Leadership Computing Facility. https://www.olcf.ornl.gov.
[30] George Papadimitriou. 2020. *Pegasus Deployment at OLCF Kubernetes.* https://doi.org/10.5281/zenodo.3825253
[31] James C Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert D Skeel, Laxmikant Kale, and Klaus Schulten. 2008. Scalable molecular dynamics with NAMD on the IBM Blue Gene/L system. *IBM Journal of Research and Development* 26, 1.2 (2008), 1781–1802. https://doi.org/10.1147/rd.521.0177
[32] Ruth Pordes, Don Petravick, Bill Kramer, Doug Olson, Miron Livny, Alain Roy, Paul Avery, Kent Blackburn, Torre Wenaus, Frank Würthwein, Ian Foster, Rob Gardner, Mike Wilde, Alan Blatecky, John McGee, and Rob Quick. 2007. The open science grid. *Journal of Physics: Conference Series* 78 (jul 2007), 012057. https://doi.org/10.1088/1742-6596/78/1/012057
[33] Romelia Salomon-Ferrer, David A Case, and Ross C Walker. 2013. An overview of the Amber biomolecular simulation package. *Wiley Interdisciplinary Reviews: Computational Molecular Science* 3, 2 (2013), 198–210.
[34] SchedMD. Simple Linux Utility for Resource Management. http://slurm.schedmd.com/.
[35] D Schultz, B Riedel, and G Merino. 2017. Pyglidein – A Simple HTCondor Glidein Service. *Journal of Physics: Conference Series* 898 (oct 2017), 092018. https://doi.org/10.1088/1742-6596/898/9/092018
[36] Douglas Thain, Todd Tannenbaum, and Miron Livny. 2005. Distributed computing in practice: the Condor experience. *Concurr. Comput.* 17, 2-4 (Feb. 2005), 323–356.
[37] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. Scott, and N. Wilkins-Diehr. 2014. XSEDE: Accelerating Scientific Discovery. *Computing in Science & Engineering* 16, 05 (sep 2014), 62–74. https://doi.org/10.1109/MCSE.2014.80
[38] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*. Bordeaux, France.
[39] Rick Wagner, Philip Papadopoulos, Dmitry Mishin, Trevor Cooper, Mahidhar Tatineti, Gregor von Laszewski, Fugang Wang, and Geoffrey C. Fox. 2016. User Managed Virtual Clusters in Comet. In *Proceedings of the XSEDE16 Conference on Diversity, Big Data, and Science at Scale* (Miami, USA) *(XSEDE16)*. Association for Computing Machinery, New York, NY, USA, Article Article 24, 8 pages. https://doi.org/10.1145/2949550.2949555
[40] D Weitzel, I Sfiligoi, B Bockelman, J Frey, F Wuerthwein, D Fraser, and D Swanson. 2014. Accessing opportunistic resources with Bosco. *Journal of Physics: Conference Series* 513, 3 (jun 2014), 032105. https://doi.org/10.1088/1742-6596/513/3/032105