Data-intensive Workflow Execution using Distributed Compute Resources

Ashish Pandey, Songjie Wang, Prasad Calyam
Email: apfd6@mail.missouri.edu, {wangso, calyamp}@missouri.edu; University of Missouri-Columbia, USA.

***Abstract*—Cloud computing has become a necessary utility for scientific and technical applications. Many diverse web services are published and subscribed using cloud data centers. It has become fairly easy to use services from Cloud Service Providers (CSPs) for computation and data processing. However, even with all their benefits, commercial cloud resources are not economical when large data processing is required. It is not always feasible for educators and researchers to use commercial cloud resources to run large data processing workflow applications due to budget limitations. In this paper, we propose a framework to help users to leverage distributed compute resources to execute data-intensive application workflows, while minimizing their budget concerns. It also enables users who may have access to small-scale compute resources in-house, to seamlessly interoperate with public cloud resources.**

***Index Terms*—Compute resource utilization, Scientific workflows, distributed computing**

I. Introduction

Scientific and technological applications are becoming increasingly data and compute-intensive. An exemplar case in scientific workflows can be seen in bioinformatics such as gene sequencing/analytics, where applications frequently require diverse multi-cloud resources to execute job flows. Researchers who create these workflow pipelines seek to use large compute and memory resources on a routine basis in an iterative and repeatable manner. Since the nature of research is often iterative and requires repeated workflow execution with varying data, they need to rely on open/commercial cloud resources for their workflows under budget limitations.

To optimally fulfill user requirements, they seek to seamlessly interoperate with any compute resources they can access. A potential scenario could involve the utilization of small-scale resources in-house in conjunction with community cloud resources such as GENI [5] and commercial CSPs such as Amazon Web Services. However, the diversity in the resources offered from different commercial and community cloud providers and a sparsely documented in-house compute resource configurations can overwhelm users who are not well versed in cyberinfrastructure or are not cloud-experts. In this paper, we describe a brokering framework that we are developing to coordinate application workflow execution by efficiently utilizing a distributed set of commercial/community cloud as well as local computing resources. Our ultimate goal is to seamlessly add any available resources using the framework for executing unique/complex compute and data-intensive application workflows.

II. Motivation

For demonstrating potential use of the framework to non-expert cloud user, we created a set of tool configurations as shown in Figure 1. To execute application workflows, we collect user specified resource requirements for their application through a web portal viz., the Cyneuro portal [4]. This portal is basically a science gateway, where multiple scientific computational workflows can be deployed and executed for neuroscience researchers/educators. We integrated a questionnaire interface on the Cyneuro portal, where the interface presents users with a set of questions, such as required number of processing units, minimum memory, and preferred size of storage. Once user specified requirements for the application workflow are collected, a distributed set of compute resources will be provisioned by our broker framework to fulfill user's demand of resources. These resources could be from commercial CSPs, community CSPs, local resources, or a mixture of them for execution of different jobs in the workflow. Once resource allocation is done by creating a collection of distributed compute nodes, users can then configure and run their application workflows on the allocated resources. During job execution, performance of the application and the status of the deployed resources can be monitored at any time. Figure 2 shows the steps of the collection, provisioning, consumption and monitoring that we have designed for supporting a custom scientific workflow.

III. Framework Components

Our framework is deployed using various tools in synergy to create a distributed pool of resources. The distributed computing resource pool is then used to execute the workflow. The components of the framework are shown in Figure 2.

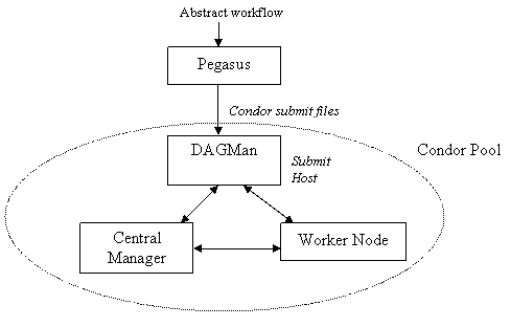

Fig. 1. Overview showing abstract workflow created by an user using Pegasus and a job processing pipeline with HTCondor.

Pegasus WMS: It is a Workflow Management System that can manage large-scale scientific workflows across desktops and computing sites. Pegasus WMS provides a means for representing the workflow of an application in an abstract form that is independent of the resources available to run it,

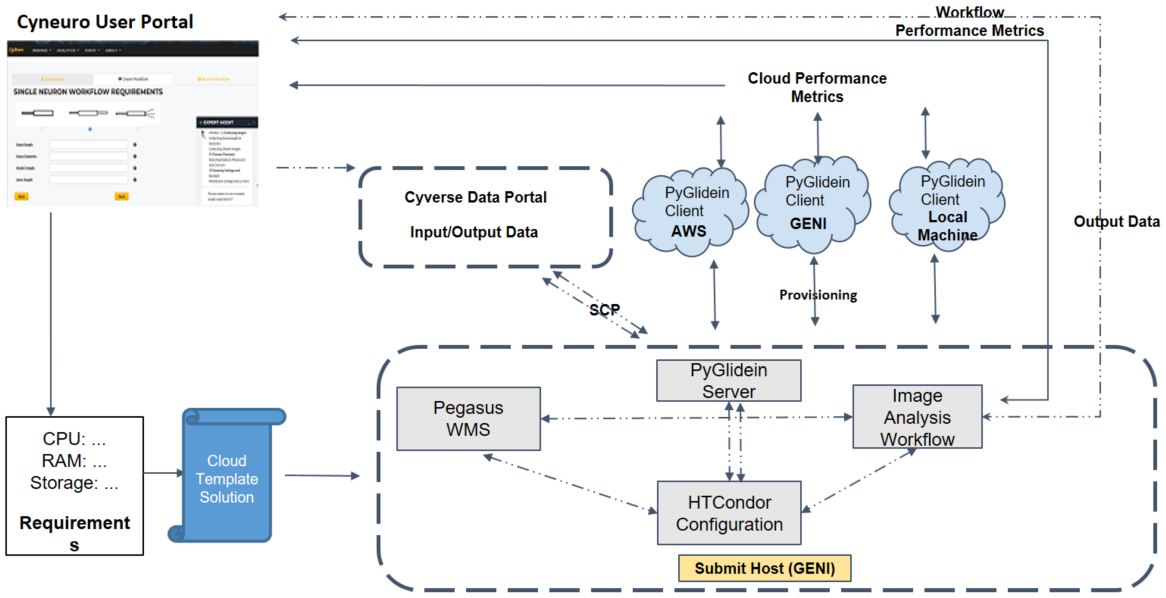

Fig. 2. Architecture design and implementation steps for representing our resource brokering framework components and their interactions.

and the location of data and executables. It compiles abstract workflows into an executable form. The executable workflows are deployed on a local and/or remote distributed resources using the Condor DAGMan workflow engine from HTCondor.

HTCondor: It is an open-source high-throughput computing software framework for coarse-grained distributed parallelization of computationally-intensive tasks. It can be used to manage workload on a dedicated cluster of computers, or to farm out work to idle computer nodes.

PyGlidein: It is an application used to run *glideins* on remote sites, adjusting for pool demand automatically. It consists of a server running on the central HTCondor submit machine and a number of clients on remote submit machines. The client will submit a *glideins*, which connects back to a central HTCondor machine that advertises slots for jobs to run in. Jobs are then run at local on remote machines in the advertised slots.

Cyverse: Cyverse provides infrastructure for storage of large data. Data can be stored remotely on a Cyverse account and can be accessed and downloaded on-demand. Note for the demo that we kept our application data on Cyverse storage.

IV. Workflow Execution

To demonstrate the effectiveness of our proposed framework, a GENI [5] node is instantiated as the submit host machine. Pegasus [1], HTCondor [2] and CyVerse iCommand [7] are installed on the submit host. Further, an image analysis workflow is created in compliance with the Pegasus guidelines. Using Pegasus, our image analysis workflow is configured to run on specific set of machines identified by their IP addresses. Pegasus essentially divides the workflow into subtasks, and the user can pre-configure individual subtask to be run on specified compute nodes. Once the workflow is initialized by the user, pegasus broadcasts the jobs to HTCondor to schedule them on compute resources. Depending on the combined configuration of workflow and condor, the jobs are computed locally or on specified compute machines. HTCondor uses *glidein* services from PyGlidein to connect to remote machines. The *glideins* are submitted to the remote cluster scheduler, and once started up, users perceive that their HTCondor pool extends into a remote cluster. HTCondor can then schedule and monitor the jobs to the remote compute node(s) in the same way it currently schedules jobs on local compute nodes i.e., through the use of the Pegasus APIs.

V. Results and Conclusion

Our novel resource brokering framework configuration allows for the execution of data-intensive application workflows with complex computing demands on remote sites such as AWS, GENI [5], or local organizational resources. Input data is fetched from Cyverse in real-time by the workflow and output data is generated and sent back to the Cyverse storage. Additional compute nodes could be added in real-time to the compute pool to help users process large datasets in a cost-effective and user-friendly manner.

References

[1] Pegasus workflow management system Available: https://pegasus.isi.edu/ [Online][Last Accessed: 9th Aug 2019]

[2] HTCondor Available: https://research.cs.wisc.edu/htcondor/index.html [Online][Last Accessed: 9th Aug 2019]

[3] CyVerse. Available: https://www.cyverse.org/ [Online][Last Accessed: 9th Aug 2019]

[4] CyNeuro Science Gateway. Available: http://cyneuro.org/ [Online][Last Accessed: 9th Aug 2019]

[5] Geni, Exploring Networks of The Future. Available: https://www.geni.net/ [Online][Last Accessed: 9th Aug 2019]

[6] Pyglidein–HTCondor Glidein Service. Available: https://github.com/WIPACrepo/pyglidein [Online][Last Accessed: 9th Aug 2019]

[7] CyVerse iCommand. [Online][Last Accessed: 9th Aug 2019] Available: https://wiki.cyverse.org/wiki/display/DS/Using+iCommands