Hybrid Multi-Cloud Analytics Services Framework

Alison Lu, Jackson Miskill, Alex Beck, JP Fleischer, Gregor von Laszewski
University of Virginia Biocomplexity Institute - Network System Sciences and Advanced Computing Division

Background

High performance computing (HPC) has recently become a very important part of science. Through processing on a supercomputer, programs can run at previously unobtainable high speeds. HPC is typically used for analytic programs that use large data sets and machine learning to predict future values or to model current states.

For such high-complexity projects, there are often multiple complex programs that may be ran several times for higher performance, like in deep learning algorithms. With such projects, program execution is submitted as a job to a typically remote HPC centers such as UVA's Rivanna, where time is billed as node-hours. It is necessary for such projects to have a service that lets the user manage and execute without supervision. We have created a service that lets the user run jobs across multiple platforms in a dynamic queue with visualization and data storage.

Current Work

This software was developed using Cloudmesh, a repository that provides numerous methods of interfacing the local system with cloud services. Much of the software from the hybrid multi-cloud analytics service framework was developed from Cloudmesh repositories.

Workflow

- A hybrid multi-cloud analytics service framework was created to manage heterogeneous and remote workflows, queues, and jobs.
- It was designed for access through both the command line and browser GUI REST services to simplify the coordination of tasks on remote computers.
- In addition, this service supports multiple operating systems like MacOS, Linux, and Windows 10 and 11, on various hosts: the computer's localhost, remote computers, and the Linux-based virtual image WSL.
- Jobs can be visualized and saved as a YAML and SVG data file.
- This workflow was extensively tested for functionality and reproducibility.

MNIST

- The Modified National Institute of Standards and Technology Database is a machine learning database based on image processing
- Various MNIST files involving different machine learning cases were modified and tested on various local and remote machines
- These cases include Multilayer Perceptron, LSTM (Long short-term memory), Auto-Encoder, Convolutional & Recurrent Neural Networks, Distributed Training, and PyTorch training

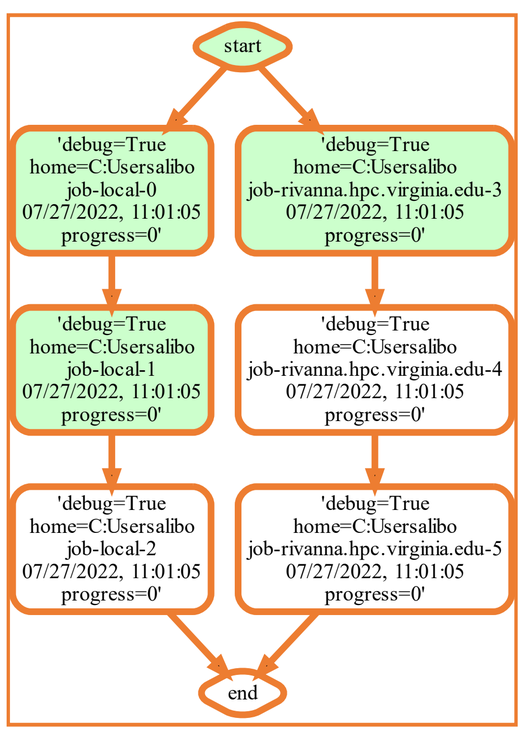

Fig 1. Sample Workflow

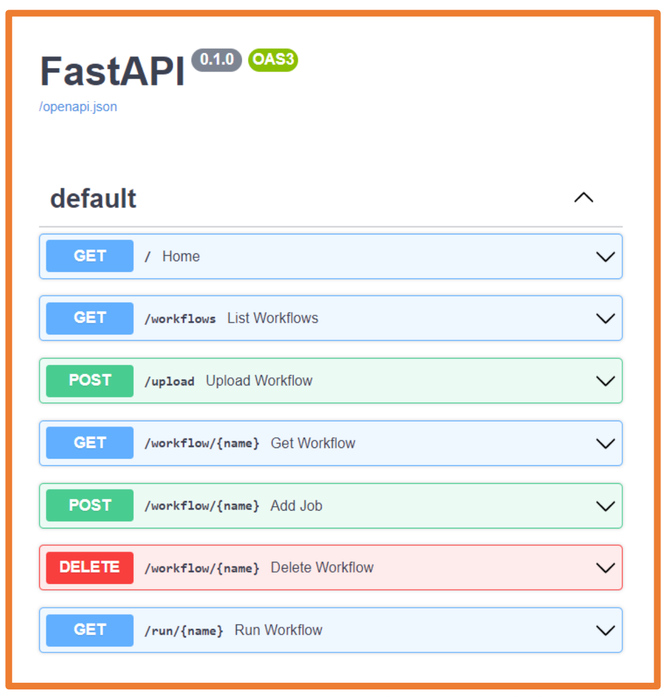

Fig 2. FastAPI Docs for Workflow

Cloudmesh: Code Used / Developed

- cloudmesh/cloudmesh-cc (github.com)
- cloudmesh/cloudmesh-common: Common methods that make programming in python easier and used in cloudmesh (github.com)
- cloudmesh/cloudmesh-vpn (github.com)
- cybertraining-dsc/reu2022 (github.com)

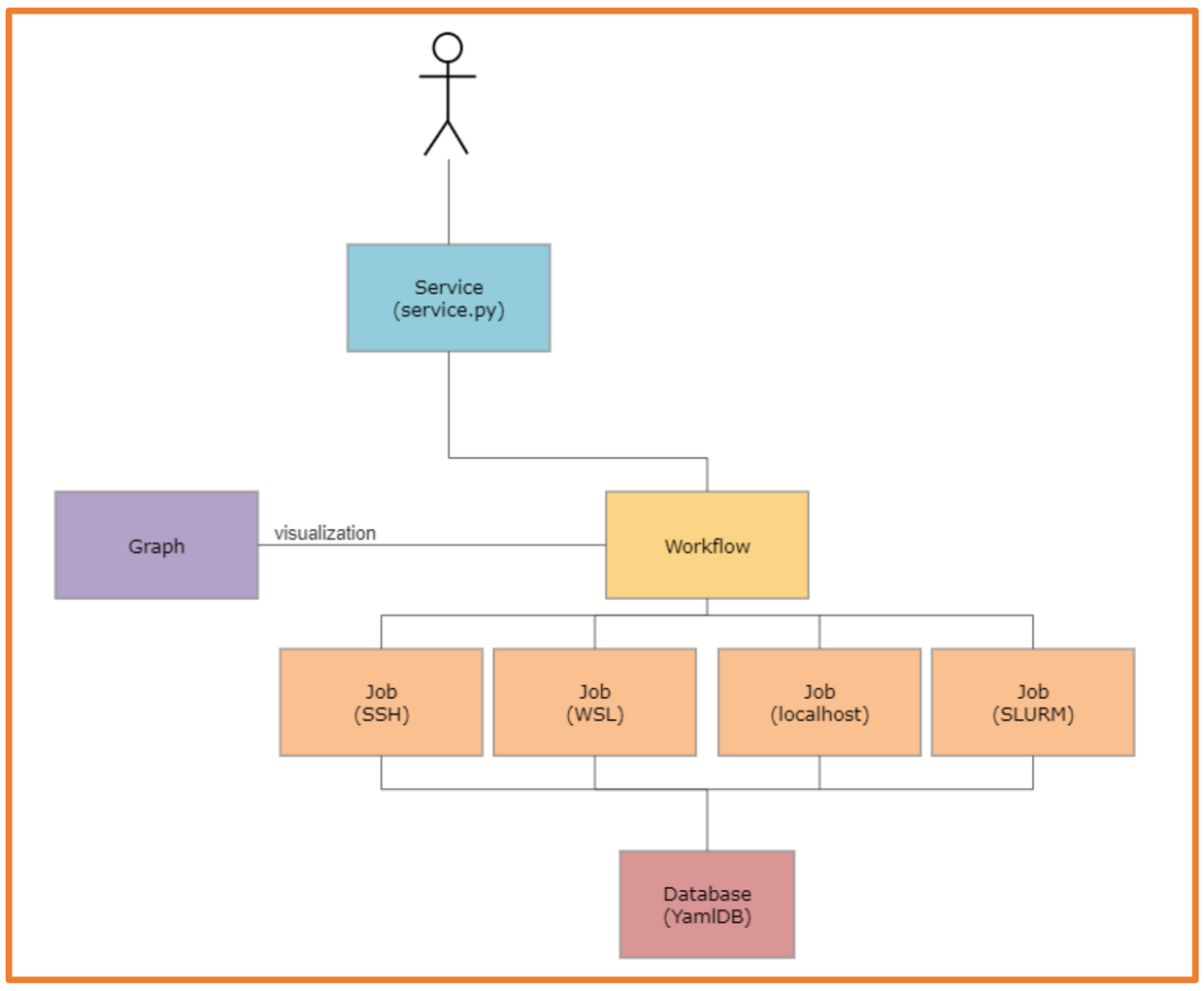

Fig 3. Workflow Layout

Design

The hybrid multi-cloud analytics service framework was created to ensure the optimal user experience for running jobs across many platforms. From the bottom up, there are a plethora of Python objects that were created to ensure execution of the jobs. These are the database, job, graph, and workflow objects along with the frontend service itself. The databases that were produced relied on a previously implemented service called YamlDB: it allows for persistent storage of data that users can call upon at the time of their choosing, rather than simply storing in a machine's memory. Therefore, when the jobs are run, the updated status and progress fields of the job are saved. These saved values can then be used to create graphical visuals as desired. Next, job objects were created so that programs could be run on different machines. These objects allow the users to be able to execute their jobs on different machines. For instance, we have a local job, a WSL job, a Slurm job, and an SSH job (which was primarily used for Rivanna). When a workflow is created, there is an internal structure that identifies the type of machine that is being operated on and creates the correct job based on that machine. After the job, a graph class was created to mesh with the workflow. The graph creates a node and edge graph with a corresponding visual representation of the graph. This was to allow the user to visualize where they are in the workflow. Next the workflow itself allows users to add jobs, remove jobs, add edges, and then run each of the jobs in the order that was specified. This allows the user to interact with each of the previous objects. Finally, the service itself is a FastAPI implementation of the workflow. Users can access this through their own devices, they can add, remove jobs, and run jobs through a graphical user interface. This interface can be run through command line commands as created in the cloudmesh-cc repository.

In summary, users interact with a graphical user interface to create workflows of jobs, which are scripts that run specific experiments. This graphical user interface shows users where their jobs are in the process of completion and create output files for each job. With this framework, researchers and scientists should be able to create jobs on their own, place them in the workflow, and run them on various types of computers.

Future Work

There are a few aspects of the service that are unfinished. For instance, the FastAPI section of the service could be improved to have a more user-friendly graphical interface. This could be accomplished by integrating what we learned through FastAPI with a different frontend framework, such as Django or flask.

Regarding the workflow, major cleanup is needed. This could be done by streamlining variables and creating better documentation so future users and programmers can use the service.

With these changes, the service could be vastly improved. In the future, we would like researchers to be able to easily access the code and to have success stories with their experiments. Additionally, we would like external users to report what they found useful and not useful about the service so that we can more craft better solutions in the future.