Hybrid Multi-Cloud Analytics Services Framework

Gregor von Laszewski[1], **Jacques P. Fleischer**[1]
University of Virginia Biocomplexity Institute - Network System Sciences and Advanced Computing Division

Background

High-performance computing (HPC) is for decades a very important tool for science. Scientific applications often consist of multiple tasks/jobs can be leveraging the processing power of requiring considerable computational needs. Often a supercomputer is needed to execute the tasks at high speeds while utilizing the specialized hardware for acceleration that otherwise are not available to the user. However, these systems can be difficult to use when conducting analytic programs that leverage machine learning applied to large data sets to, for example, predict future values or model current states. For such highly complex analytics tasks, there are often multiple steps that need to be run repeatedly either to combine analytics tasks in competition or cooperation to achieve the best results. Although leveraging computational GPUs lead to several times higher performance when applied to deep learning algorithms, may be not possible at the time as the resources are either too expensive or simply not available. The analytics task is to simplify this dilemma and introduce a level of abstraction that focuses on the analytics task while at the same time allowing sophisticated compute resources to solve the task for the scientist in the background. Hence, the scientist should be presented with a function call that automatically puts together the needed resources and stage the task in jobs on the HPC environment without the need of too many details of the HPC environment. Instead, the science user should access analytics REST services that the user can easily integrate into their scientific code as functions or services. To facilitate the need to coordinate the many tasks behind such an abstraction we have developed a specialized analytics Workflow abstraction and service allowing the execution of multiple analytics tasks in a parallel workflow, The workflow can be controlled by the user and is asynchronously executed including the possibility to utilize multiple HPC computing centers via user-controlled services.

Workflow Controlled Computing

The Cloudmesh cc Workflow is enhancing Cloudmesh by integrating an API and service to make using cloud and HPC resources easier. The enhancement is focused on a library called Cloudmesh Controlled Computing (cloudmesh-cc) that adds workflow features to control the execution of tasks and jobs on remote compute resources including clouds, desktop computers, and batch-controlled HPC with and without GPUs. Effectively we access remote, and hybrid resources by integrating cloud, and on-premise resources. The goal is to provide an easy way to access these resources, while at the same time providing the ability to integrate the computational power enabled through a parallel workflow framework Access to these complex resources is provided through easy to use interfaces such as a python API, REST services, and command line tools. Through these interfaces, the framework is universal and can be integrated into the science application or other higher level frameworks and even different programming languages. The software developed is freely available and can easily be installed with standard python tools so integration in the python ecosystem using virtualenv's and anaconda is simple.

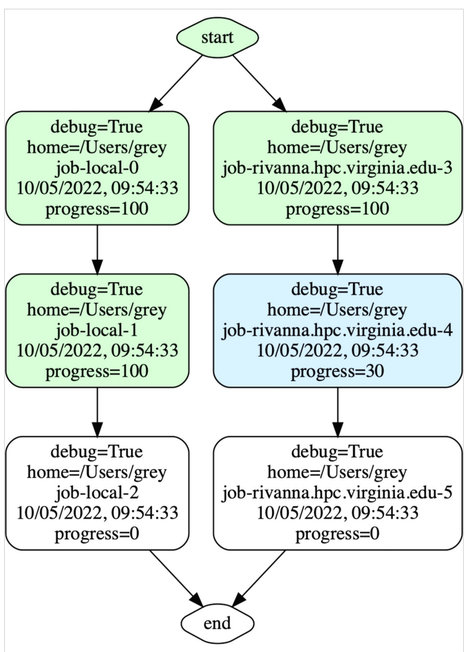

Fig. 1. Execution of a Cloudmesh cc Workflow with display

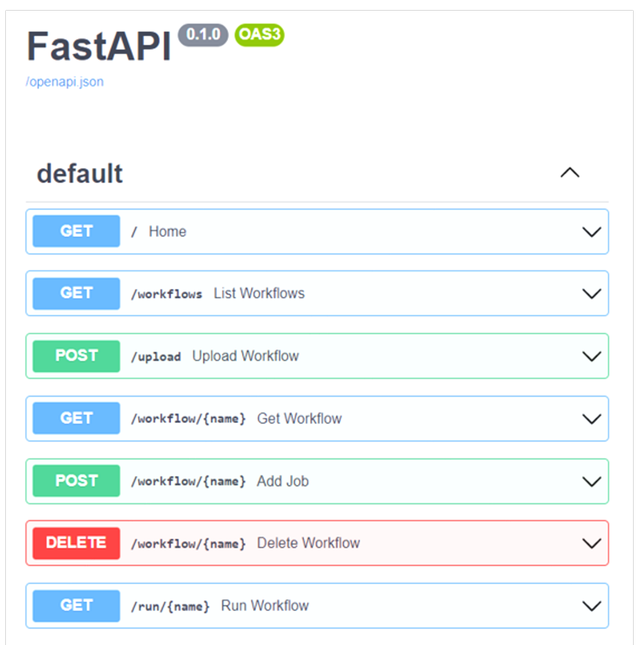

Fig. 2. F Documentation of the REST Cloudmesh cc Workflow interface

Cloudmesh References

- 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)

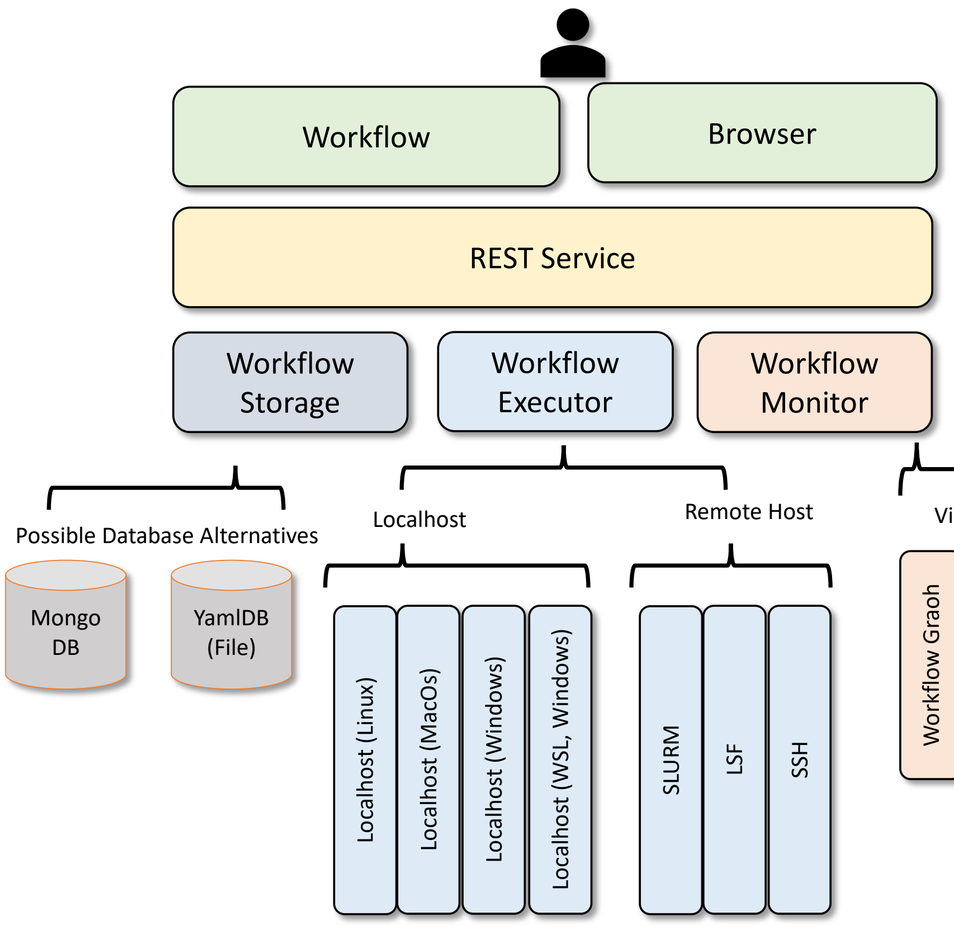

Fig. 3. Architecture of the Cloudmesh cc Workflow Framework

Design

The hybrid multi-cloud analytics service framework ensures running jobs across many platforms. The design (see Fig. 3) includes a small and streamlined number of abstractions so that jobs and workflows can be represented easily. This makes it possible to custom design for each target type a specific job type so that execution on local and remote compute resources including batch operating systems can be achieved. Job types for localhost, ssh, SLURM, and WSL are available. Other job types can easily be added. The design is flexible and new job can be expanded as each job can contain arbitrary arguments. Through this flexibility jobs types can be also run on different operating systems including local job on Linux, macOS, Windows 10, and Windows 11, jobs running in WSL on Windows computers. An important design requirement to display the dependencies of the workflow in a direct acyclic Graph is enabled. This greatly reduced the complexity of the implementation while being able to leverage graphical displays of the workflow, as well as implementing sequential execution of workflows as an alternative to parallel execution while using the build-in topological sort function. It serves as an example that custom schedulers can be designed and easily integrated into the runtime management while executing the tasks and jobs through a straightforward interface. The status of the tasks and jobs is stored in a file database that can be monitored during program execution. The creation of the jobs is done prior to the execution of the workflow, but additional tasks and jobs could be integrated also at runtime. This is possible when using our parallel scheduler that selects tasks and jobs once the parent jobs have been completed. This is important as it allows dynamic workflow execution of long-running workflows, while results from previous calculations can be used in later stages of the workflow and lead to workflow modifications. We have developed a simple-to-use Python API so programs are easy to write. Additionally, we used this API internally to implement a REST. An important feature that we added is the monitoring of the jobs while using progress reports through automated log file mining. This way each job reports the progress during the execution.

Status of the Work

A complete reimplementation that meets our design goals has since been delivered by Gregor von Laszewski and JP. Fleischer. This includes also significant updates and corrections to this poster

Acknowledgement

We like to thank NSF and NIST for supporting this effort
In addition to the authors, the pre-alpha version contains contributions from Jackson Miskill, Alison Lu, Alex Beck. The work conducted by the original student team was a pre-alpha prototype and focus was placed on educating the students in teamwork and python programming. In that prototype many design aspects that we set forward at the beginning of the project were not implemented.