Robust Meta-Workflow Management with Mufasa

Ben Lyons and Douglas Thain
Department of Computer Science and Engineering, University of Notre Dame

***Abstract*—Workflow management systems (WMS) are widely used to describe and execute large computational or data intensive applications. However, when a large ensemble of workflows is run on a cluster, new resource management problems occur. Each WMS itself consumes otherwise unmanaged resources, such as the shared head node where the WMS coordinator runs, the shared filesystem where intermediate data is stored, and the shared batch queue itself. We introduce Mufasa, a meta-workflow management system, which is designed to control the concurrency of multiple workflows in an ensemble, by observing and controlling the resources required by each WMS. We show some initial results demonstrating that Mufasa correctly handles the overcommitment of different resource types by starting, pausing, and cancelling workflows with unexpected behavior.**

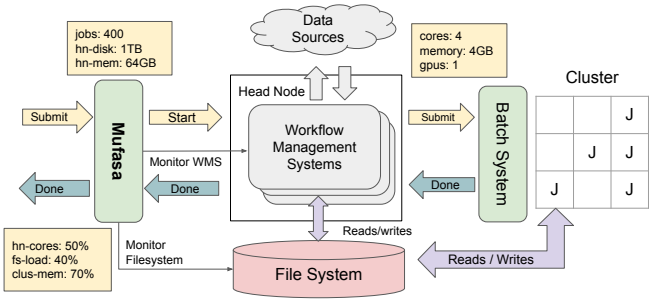

Fig. 1: Architecture of Mufasa
Mufasa is a meta-workflow manager that controls the invocation of workflow management systems (WMS) on the head node of a cluster. By controlling workflow concurrency, it manages cluster reasources, head node resources, and shared filesystem and network resources.

I. Introduction

Workflow management systems (WMS) are widely used in scientific computing to describe and execute large computation or data intensive campaigns. A workflow typically consists of a large graph of tasks with dependencies that must be completed in a specific order, each one run on a node of a cluster. However, a single workflow is rarely the entirety of a computational effort. A single researcher might deploy an large *ensemble* of workflows, each one exploring a different molecular structure, or reducing a different genome, or searching a different area of the sky. In addition, multiple users might submit ensembles to the same cluster at once.

When a large number of workflows are deployed at once, several resource management problems quickly appear. Suppose that a user has 1000 workflows to execute, each one consisting of 10K distinct batch jobs. While each instance of the WMS may be individually careful not to overload the batch system, all of them operating simultaneously may result in a problem. If each of the 1000 instances is careful to only submit 100 jobs to the batch system at a time, the batch queue itself may not be capable of dealing with 100K queued jobs all at once. Each WMS also consumes other resources that are not regulated by the batch system. A WMS may transfer files over the network, store intermediate data in a shared filesystem, and make use of the CPU and RAM on the head node in order to advance the state of the workflow. Without further control, overcommitted resources may result in failure of the whole ensemble.

This problem can be addressed by a *meta-workflow management system*. A meta-WMS accepts requests to execute a large number of workflows, whether an ensemble from a single user, or competing requests from multiple users. The meta-WMS is responsible for starting workflow instances as resources become available, monitoring the resource consumption of those instances, and controlling concurrency so as to avoid over-commitment and failure. While a conventional batch system is responsible for the resource management of tasks on the cluster, the meta-WMS is responsible for the resources consumed by the WMS themselves: the number of jobs submitted to the cluster, the usage of the shared filesystem, network transfers, and head node resources.

We designed Mufasa as a prototype meta-WMS that is responsible for managing these shared resources. The basic structure of Mufasa is shown in Figure 1. When Mufasa is started, the user provides both global and workflow resource limits. The global limits represent an upper bound on the cumulative resource consumption of all WMSs, and the workflow limits represent an