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.**

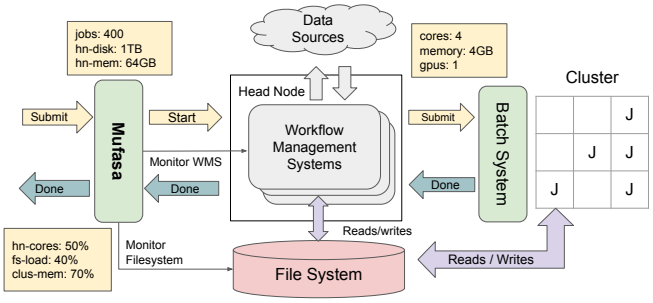

Fig. 1: Architecture of Mufasa
Mufasa is a meta-workflow manager that controls the invocation of workflow management systems (WMS) on the head node of a cluster. By controlling workflow concurrency, it manages cluster reasources, head node resources, and shared filesystem and network resources.

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

estimated limit for a single WMS. Mufasa monitors an inbox directory where users submit workflows to be processed. As each workflow arrives, Mufasa uses the provided limits to determine if there are enough resources to run a WMS to process the workflow. As each WMS runs, Mufasa monitors its resource consumption to ensure that it does not exceed the allocated individual workflow limits. In the event that any of the resources exceed these limits, Mufasa will either kill or pause the workflow, and then reschedule it with a larger resource allocation. Paused workflows are effectively "checkpointed" because the results of completed tasks can remain in shared storage. The overall goal is to ensure that the cumulative resource consumption of all WMSs does not exceed the user provided global limit. In the current implementation, Mufasa can manage the resources of the HTCondor batch system and workflows expressed in the Work Queue [1] framework, such as the Coffea [4] framework for high energy physics.

Managing an entire WMS as a discrete entity is conceptually similar to managing a single application, but has a number of important differences. At a high level, a single WMS can be viewed as a Unix process that has the usual resources of cores, memory, file descriptors, etc. To track external resources, we require the WMS to produce an accurate log of jobs submitted, files transferred, etc. More importantly, a WMS is more malleable [2], [3] than a single job. Generally, a single job requires a specific set of resources, and without splitting up the job it may not be possible to change these resource requirements. However, if a WMS is informed of resource limits, then it may change its own scheduling policy to fit the resource consumption within these bounds. Still, there are lower bounds to the amount of resources required for a WMS to complete, so it may not be possible to process a workflow within arbitrarily small limits.

Figure 2 gives some initial results showing the resource management controls of Mufasa. We established a test system with an upper limit on 1000 queued batch jobs, 5GB head node RAM, and 75GB of shared storage space. Then we constructed benchmark ensembles that stress different resources available in the system. Ensemble 1 consists of 100 workflows, each one consuming 100MB RAM, 6 GB disk, and generating 200 ± 40 batch jobs. Ensemble 2 consists of 100 workflows, each one consuming 1-4GB RAM, 6GB disk, and generating 100 ± 40 batch jobs. The left column shows the jobs, memory, and disk consumed by Ensemble 1, and the right column shows the same consumed by Ensemble 2. In each case, Mufasa controls the concurrency so that the critical resource is not exceeded, and steady progress is made throughout the entire ensemble.

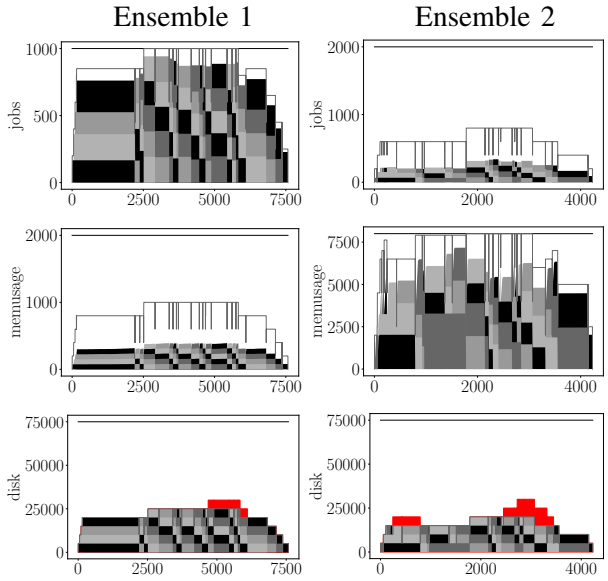

Fig. 2: Workflow Resource Control
Each column shows the cluster jobs, head node memory, and disk consumption of an ensemble of workflows. In the left column, cluster jobs are the limiting resource. In the right column, RAM is the limiting resource. Red highlights shows disk used by paused workflows. Mufasa limits concurrency to control the critical resource.

ACKNOWLEDGEMENTS

We thank Ben Tovar for helping us understand and debug software interfaces. We would also like to thank Barry Sly-Delgado and Thanh Phung Nguyen for brainstorming and discussing conceptual components of the system with the authors. This work was supported in part by NSF grant OCI-1931348.

REFERENCES

[1] P. Bui, D. Rajan, B. Abdul-Wahid, J. Izaguirre, and D. Thain. Work Queue + Python: A Framework For Scalable Scientific Ensemble Applications. In *Workshop on Python for High Performance and Scientific Computing (PyHPC) at the ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (Supercomputing)* , 2011.

[2] R. Dutton and W. Mao. Online scheduling of malleable parallel jobs. *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, 01 2007.

[3] K. Jansen and F. Land. Scheduling monotone moldable jobs in linear time. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 172–181, 2018.

[4] N. Smith, L. Gray, M. Cremonesi, B. Jayatilaka, O. Gutsche, A. Hall, K. Pedro, M. A. Flechas, A. Melo, S. Belforte, and J. Pivarski. Coffea - Columnar Object Framework For Effective Analysis. *CoRR*, abs/2008.12712, 2020.