A Physical and Virtual Compute Cluster Resource Load Balancing Approach to Data-Parallel Scientific Workflow Scheduling*

Jianwu Wang[1], Prakashan Korambath[2], Ilkay Altintas[1]
[1] San Diego Supercomputer Center, UCSD
{jianwu, altintas}@sdsc.edu
[2] Institute for Digital Research and Education, UCLA
ppk@ats.ucla.edu

***Abstract*— To execute workflows on a compute cluster resource, workflow engines can work with cluster resource manager software to distribute jobs into compute nodes on the cluster. We discuss how to interact with traditional Oracle Grid Engine and recent Hadoop cluster resource managers using a dataflow-based scheduling approach to balance compute resource load for data-parallel workflow execution. Our experiments show that: 1) The presented approach can balance computational resource load well by interacting with the resource managers and provides good execution performance on both physical and virtual clusters; 2) Oracle Grid Engine outperforms Hadoop for CPU-intensive applications on small-scale clusters.**

Keywords- data-parallel workflow scheduling, virtual cluster, load balancing, cluster resource manager comparison

I. INTRODUCTION

Scientific workflow management systems have demonstrated their ability to help domain scientists solve scientific problems by synthesizing different data and computing resources. Data parallelism in scientific workflows describes parallel execution of workflow on multiple parts of input data if these input parts can be processed independently [1]. To speed up workflow execution, many data-parallel workflows are run on compute cluster resources where cluster resource managers are usually employed to distribute jobs on the clusters. Currently, two kinds of resource manager software are commonly used: (i) traditional resource manager software, such as Oracle Grid Engine (OGE)[1] and Load Sharing Facility (LSF)[2], to manage job distribution on the cluster, and (ii) the relatively new Hadoop software[3] to manage data distribution along with job distribution on a cluster.

Through a natural scheduling approach and its application to scientific use cases, this paper discusses how to work with resource managers to schedule data-parallel scientific workflows and compare their performance differences.

II. A DATA-PARALLEL SCIENTIFIC WORKFLOW SCHEDULING APPROACH

Scheduling data-parallel workflows on limited nodes before knowing the execution times of the tasks in the workflows is a challenge. A simple example of this challenge is illustrated in Figure 1. The workflow in Figure 1 has three tasks (Tasks A, B and C) and three input parts (Inputs 1, 2 and 3). The tasks have to be executed sequentially; yet the inputs can be processed independently. Suppose two homogenous nodes are available to run the workflow. The task execution time for each input on one node is listed in the execution time table in Figure 1. From the table, we can see the execution times of the same task for different inputs are very different. For many real workflow applications, the number of inputs and tasks could be too large to get all execution time information before scheduling the workflow.

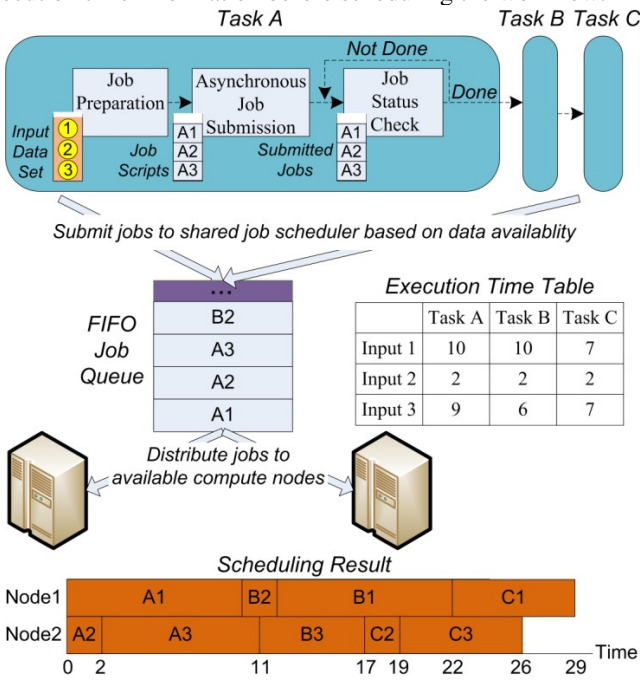

Figure 1. A data-parallel workflow scheduling approach.

Most task scheduling problems in distributed environments have no optimal solutions in polynomial time [2], especially when workflow task execution times are unknown beforehand. In our approach, we only use a natural way to schedule tasks, namely putting executable tasks with their inputs in a first-in, first-out (FIFO) queue and scheduling them to nodes that are available. In this approach (shown in Figure 1), task executions for each input are represented as jobs. Once an input is available, a corresponding job will be created and submitted to a job scheduler asynchronously. All the tasks in the workflow will share one job scheduler which manages submitted jobs in a FIFO

*This work was supported by NSF SDCI Award OCI-0722079, NSF CEO:P Award No. DBI 0619060, DOE SciDac Award No. DE-FC02-07ER25811, and UCGrid project. We also thank the AWS in Education Research Grants from Amazon.com, Inc for EC2 usage credit.

[1] http://www.oracle.com/us/products/tools/oracle-grid-engine-075549.html
[2] http://www.platform.com/workload-management
[3] http://hadoop.apache.org/core/

978-0-7695-4461-8/11 $26.00 © 2011 IEEE
DOI 10.1109/SERVICES.2011.50

queue. The shared job scheduler distributes jobs to available compute nodes. Here, we use notation A1 to present the job for task A processing input 1, and use the similar way for other jobs. The characteristics of our approach are explained as follows using the example in Figure 1.

Data-driven Workflow Execution: In data-driven workflow, i.e., dataflow, execution semantics, tasks are executed based on input data availability. For each task, inputs will be processed in order and be automatically buffered in a queue if the task is currently busy. In Figure 1, job A3 is submitted after job A2, and B2 is the first job in task B since A2 finishes earlier than other jobs in task A and then input 2 is the first input for task B.

Pipeline Parallelism: Pipeline parallelism describes how a set of data is processed simultaneously among a group of sequential tasks, each task processing one or more data elements of the set [1]. With pipeline parallelism, if an input finishes its processing in one task, it does not need to wait for the completion of the other inputs in the same task before being processed by the downstream tasks in the workflow. For instance, input 2 can be processed in task B before all inputs finish their processing in task A in Figure 1.

Asynchronous Job Submission: Since execution time may vary for different jobs, asynchronous job submission and status checking enables the workflow execution engine to get the information that a job is done without waiting for the completion of other jobs. In a pipeline, this finished job will immediately trigger downstream tasks. In this way, job B2 in the example can get started before all jobs in task A finishes.

Shared Job Scheduler: A job scheduler manages multiple computing nodes and distributes jobs to available nodes. In our approach, jobs in different tasks are managed by the same job scheduler to achieve overall load balancing on distributed nodes.

Job Partition: If a job in a task can be further split into multiple small independent sub jobs, especially for downstream tasks, it would help to balance the load of compute nodes through the shared job scheduler. In the example, if job C1 can be evenly split to 7 sub jobs, the 6th sub job will be distributed into node 2, and then the whole workflow can be finished at time unit 28 instead of 29.

III. WORKFLOWS INTERACTING WITH CLUSTER RESOURCE MANAGERS

We demonstrate our approach to interact with OGE and Hadoop via a Kepler workflow use case in Computational Chemistry. Kepler workflow system[4] satisfies the characteristics of the approach.

The use case is an enzyme design process that goes through three computational steps, namely RosettaMatch, RemoveBChain and RosettaDesign, before validation by experiments [3]. The execution of these programs takes around 300 MB memory on one machine, and these programs are CPU intensive since over 99.9% of their execution time is taken by CPU. The entire process can be performed independently for different inputs, i.e., scaffolds. The total computation time for each input varies greatly, ranging from half a minute to five hours.

A. Kepler Workflow Interacting with Oracle Grid Engine

Like other traditional cluster resource managers, OGE is used to submit, schedule, distribute, and manage the execution of large numbers of jobs on the nodes of a cluster. A shared file system is normally employed for data and program access from the nodes.

Figure 2 shows a workflow that interacts with OGE on one compute cluster. The connection links between the components, called *actors* in Kepler, describe their dependencies, which determine that each input has to go through the three processing tasks sequentially. Once one actor generates its output, the output will trigger the execution of the downstream actors, e.g., the RemoveBChain actor will start processing once it gets data from the RosettaMatch actor.

Inside a composite actor, such as RosettaMatch, the sub-workflow dynamically creates job scripts according to workflow inputs and submits them asynchronously to the OGE job scheduler on the cluster using Kepler actors.

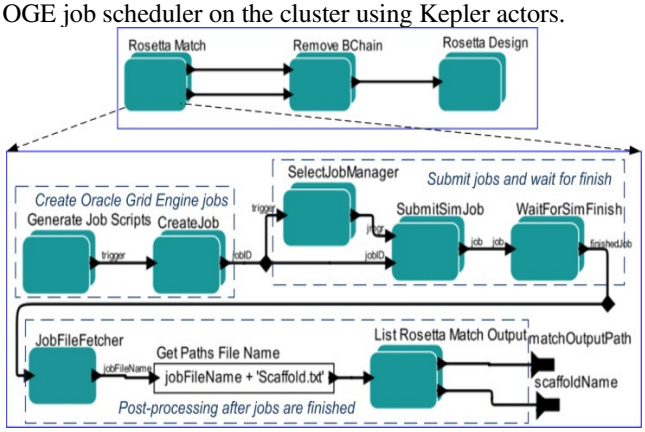

Figure 2. Kepler workflow for enzyme design process using OGE.

B. Kepler Workflow Interacting with Hadoop

The Hadoop software is composed of a MapReduce runtime system and a distributed file system, called HDFS. Input data is automatically partitioned into chunks and stored on compute nodes. User programs are distributed and executed in parallel on the partitioned data blocks. HDFS supports MapReduce execution model [4] with the capability of automatic data redundancy and diffusion among each node in the Hadoop cluster. A Hadoop node dispatches tasks and manages the executions of the other Hadoop nodes.

The workflow in Figure 3 is built using the MapReduce actor in Kepler. Since Map and Reduce are two separate functions in the MapReduce programming model, Map and Reduce are treated as two independent sub-workflows in Kepler MapReduce actor [5]. However, since the use case only requires "embarrassingly parallel execution" for different inputs, only a Map sub-workflow is needed here.

In this workflow, job files to be executed in compute nodes are put into HDFS once they are created and asynchronously submitted to Hadoop scheduler via an AsynMapReduce actor. In the Map sub-workflow of each AsynMapReduce actor, job information is gotten from HDFS and the

[4] http://kepler-project.org/

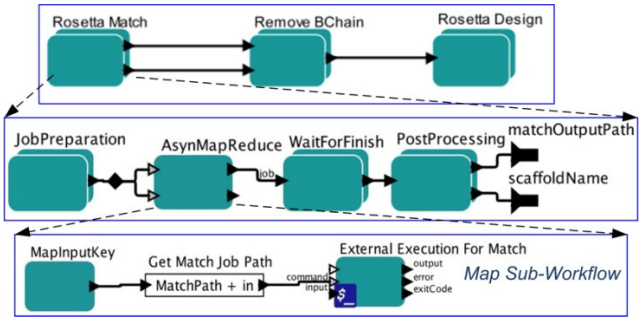

Figure 3. Kepler workflow for enzyme design process using Hadoop.

job is executed by an ExternalExecution actor. This workflow does not take much advantage of data distribution in HDFS since legacy domain-specific programs, like Rosetta programs in our use case, cannot directly process data in HDFS. Thus in this workflow, Rosetta programs and database are still accessed from the standard shared file system in the cluster, not HDFS. We will discuss possible solutions to the problem in Section 4.

IV. Experiments and Discussions

The above developed workflows are tested in three kinds of physical and virtual compute cluster environments. A typical compute cluster consists of compute nodes, a control node, a storage file system and network connection. The control node will manage the job execution on the compute nodes through OGE or Hadoop. Users will access the environment through the control node, and run their applications on the compute nodes using the data from the external storage server which normally hosts a shared file system for the nodes.

In our experiments, we create two kinds of virtual clusters based on different virtualization hosts: *Private Cloud Cluster* and *Public Cloud Cluster*. A private cloud cluster is created based on users' existing physical cluster. We only virtualize compute nodes and share the control node, the external storage server and the resource manager with the host cluster. A public cloud cluster is created from a public cloud provider, such as Amazon Elastic Compute Cloud (EC2)[5], where all the cluster components are built based on the public cloud environment.

Our experiments on the physical cluster and the private cloud cluster are carried out in a dedicated Beowulf type cluster, which consists of commodity x86_64 hardware and 1 Gb Ethernet switch for networking. Storage is provided through RAID array servers, which are NFS servers mounted on the compute nodes. Its default scheduler is OGE. The nodes used in this experiment have 8 GB memory, 2 dual core AMD 2.0 GHz CPUs and CentOS 5.5 Linux installed on the local hard drives. The private cloud cluster environment is built using Xen as the hypervisor. Each virtual compute node only has bare CentOS 5.5 with 6 GB memory. In both environments, Kepler, the Rosetta database and the Rosetta programs for the enzyme design process use case are accessed from the mounted NFS server.

The public cloud clusters on EC2 are built by StarCluster[6] toolkit, which automatically instantiates instances based on provided Amazon machine image (AMI), attaches Amazon Elastic Block Storage (EBS) volumes, installs OGE job scheduler, and installs NFS to build a shared file system on the EBS volumes. We use the large instance type for all the nodes in the cluster. Each node has 7.5 GB memory and 2 virtual CPU cores. In this environment, Kepler, the Rosetta database and the Rosetta programs are accessed from an EBS volume that are mounted and shared by the NFS service on the control node.

For the execution of the OGE workflow, we start the execution daemon of OGE on the physical and virtual compute nodes so that we can submit jobs via the control node where these jobs will be distributed to the compute nodes. Similarly, Hadoop is configured and started to manage and distribute jobs for the Hadoop workflow in Section 3. Simple FIFO queue based scheduling is used in both the OGE and Hadoop. Each physical or virtual compute node only runs one job at a time. Since the resources are dedicated and each workflow is executed separately in our experiments, there is no queue wait time for each job other than waiting for the preceding jobs in the workflow to complete.

Figure 4 lists the experiment results for the workflow runs using two input sets. The first input set includes 5 scaffolds which has 5, 5 and 50 jobs for the RosettaMatch, RemoveBChain and RosettaDesign task respectively. The second one includes 10 scaffolds and has 10, 10 and 286 jobs respectively. The execution of the RosettaDesign task is split into smaller jobs for better resource load balancing.

These results indicate that: 1) On all execution environments, the execution on four nodes for both the OGE and Hadoop workflows have good execution acceleration rates compared to the execution on one node; 2) The execution on the private cloud cluster only brings very little overhead (around 1%) over the physical cluster; 3) The Hadoop workflow execution takes longer time than the OGE workflow execution. Moreover, our execution monitoring shows the compute nodes are evenly balanced until remaining job number is less than the compute node number.

Experimental results in Figure 4 also indicate that the execution performance on the public cloud cluster is always better than that on other environments, mainly due to the newer and faster CPU models on the public cloud cluster. Yet it is hard to measure the exact CPU difference since the CPU models might be different for the virtual instances.

We think the overhead for the Hadoop workflow execution is due to several reasons. First, jobs are not directly executed in Hadoop nodes but wrapped by Kepler MapReduce sub-workflows (as shown at the bottom of Figure 3), which brings additional overhead compared to the direct job execution on compute nodes in the OGE workflow. In addition, scheduling overhead of Hadoop is larger since it needs extra effort for HDFS management.

Although the Hadoop workflows take longer execution time than the OGE ones in the experiments, we argue using Hadoop still has potential advantages in some aspects.

[5] http://aws.amazon.com/ec2/

[6] http://web.mit.edu/stardev/cluster/

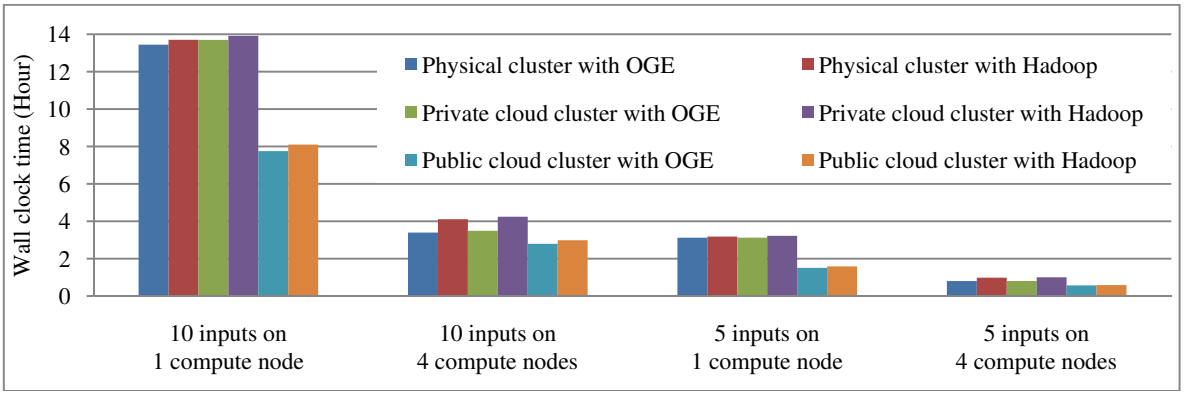

Figure 4. Execution time comparison for the enzyme design process.

The first aspect is data-intensive computation. HDFS supports automatic data partition among Hadoop nodes, and user programs can be distributed and executed in parallel on the partitioned data blocks. If stored in the HDFS, the data and programs can be easily shared among and accessed from compute nodes, providing data locality, and reducing data connection traffic on shared file system. With HDFS, Hadoop can potentially outperform traditional job schedulers for data-intensive application execution on large-scale compute clusters. One challenge here is that legacy domain-specific programs, like Rosetta programs in our use case, cannot directly process data in HDFS. Possible solutions for it are: 1) Staging data out from HDFS to standard file system on the local node before program execution; 2) Extending legacy programs to be able to follow MapReduce programming model and access data in HDFS; 3) Using toolkits like FUSE to allow HDFS mounted as a standard file system[7].

The second aspect is workflow description for execution logic on compute nodes. It is intuitive to describe the execution logic to be run on Hadoop nodes through sub-workflows, as shown at the bottom of Figure 3. In addition, since the execution logic is explicitly specified in a sub-workflow rather than an external script, provenance information can be easily captured by the workflow engine.

V. CONCLUSIONS

To validate workflow engines interacting with cluster resource managers for efficient workflow execution on cluster resources, two compute cluster resource managers, namely OGE and Hadoop, are used and compared via scientific workflow applications. Our experiments show dataflow-based workflow scheduling approach can have good resource load balancing and performance speedup for data-parallel workflow applications on physical and virtual compute clusters. The experiments also show running CPU intensive jobs with moderate I/O access hardly have performance deterioration on a virtual environment.

There are increasing efforts for workflow research [6, 7] and systems such as Oozie[8], Azkaban[9] and Cascading[10] to work with the MapReduce programming model or Hadoop environment. Yet none of the above work compares their performance with traditional resource manager like OGE. Our experiments show that traditional resource manager can have better performance for those data-parallel workflow applications that can be executed using both Hadoop and traditional resource managers on small-scale clusters. The performance comparison result is the same when testing on a physical compute cluster and two virtual ones. We also analyze the reasons for the performance difference and potential advantages using Hadoop.

For future work, we plan to improve the HDFS utilization and compare the execution performance for data-intensive workflow applications on large-scale clusters.

REFERENCES

[1] C. Pautasso, G. Alonso. "Parallel Computing Patterns for Grid Workflows". Proc. of Workshop on Workflows in Support of Large-Scale Science (WORKS06), 2006

[2] H. El-Rewini, T. Lewis, H. Ali. "Task Scheduling in Parallel and Distributed Systems". ISBN: 0-13-099235-6, PTR Prentice Hall, 1994.

[3] J. Wang, P. Korambath, S. Kim, S. Johnson, K. Jin, D. Crawl, I. Altintas, S. Smallen, B. Labate, K. N. Houk. "Theoretical Enzyme Design Using the Kepler Scientific Workflows on the Grid". Proc. of the 5th Workshop on Computational Chemistry and Its Applications (5th CCA) at Int Conf. on Computational Science (ICCS 2010).

[4] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". Proc. of the 6th Symp. on Operating Systems Design and Implementation (OSDI 2004), pp. 137-150. USENIX Association, 2004.

[5] J. Wang, D. Crawl, I. Altintas. "Kepler + Hadoop : A General Architecture Facilitating Data-Intensive Applications in Scientific Workflow Systems". Proc. of the 4th Workshop on Workflows in Support of Large-Scale Science (WORKS09) at Supercomputing 2009 (SC2009) Conference, 2009.

[6] C. Zhang, H. D. Sterck. "CloudWF: A Computational Workflow System for Clouds Based on Hadoop". Proc. of the 1st Int. Conf. on Cloud Computing (CloudCom 2009).

[7] X. Fei, S. Lu, C. Lin. "A MapReduce-Enabled Scientific Workflow Composition Framework". Proc. of 2009 IEEE Int. Conf. on Web Services (ICWS 2009), pp. 663-670, 2009.

[7] http://wiki.apache.org/hadoop/MountableHDFS
[8] http://yahoo.github.com/oozie/index.html
[9] http://sna-projects.com/azkaban/
[10] http://www.cascading.org/