Executing Large Scale Scientific Workflow Ensembles in Public Clouds

Qingye Jiang
School of Information Technologies
The University of Sydney
Sydney NSW 2006, Australia
Email: qjiang@ieee.org

Young Choon Lee
Department of Computing
Macquarie University
Sydney NSW 2109, Australia
Email: young.lee@mq.edu.au

Albert Y. Zomaya
School of Information Technologies
The University of Sydney
Sydney NSW 2006, Australia
Email: albert.zomaya@sydney.edu.au

***Abstract*—Scientists in different fields, such as high energy physics, earth science, and astronomy are developing large-scale workflow applications. In many use cases, scientists need to run a set of interrelated but independent workflows (i.e., *workflow ensembles*) for the entire scientific analysis. As a workflow ensemble usually contains many sub-workflows in each of which hundreds or thousands of jobs exist with precedence constraints, the execution of such a workflow ensemble makes a great concern with cost even using elastic and pay-as-you-go cloud resources. In this paper, we address two main challenges in executing large-scale workflow ensembles in public clouds with both cost and deadline constraints: (1) execution coordination, and (2) resource provisioning. To this end, we develop a new pulling-based workflow execution system with a profiling-based resource provisioning strategy. The idea is homogeneity in both scientific workflows and cloud resources can be exploited to remove scheduling overhead (in execution coordination) and to minimize cost meeting deadline. Our results show that our solution system can achieve 80% speed-up, by removing scheduling overhead, compared to the well-known Pegasus workflow management system when running scientific workflow ensembles. Besides, our evaluation using Montage (an astronomical image mosaic engine) workflow ensembles on around 1000-core Amazon EC2 clusters has demonstrated the efficacy of our resource provisioning strategy in terms of cost effectiveness within deadline.**

Keywords*—*workflow ensemble, cloud computing, parallel computing, resource provisioning.

I. Introduction

Many applications in science and engineering are increasingly formed as workflows with many precedence-constrained jobs, e.g., Montage [1], LIGO [2], and CyberShake [3]. Scientists need to run these workflows with different parameters repeatedly, or use a combination of different workflows to achieve an ultimate goal. We use the term **workflow ensemble** to represent an entire scientific analysis as a set of interrelated but independent workflow applications. In modern scientific computing applications, a single scientific workflow often becomes very large in terms of the number of constituting jobs and input data size, which is already a challenge in resource provisioning and scheduling. The situation is further complicated by the number of workflows in a workflow ensemble. For example, an ensemble of 200 6.0 degree Montage workflows spawns over 1.7 million jobs and deals with approximately 7 TB of data. Thus, the efficient execution of a workflow ensemble with multiple workflows is of great practical importance.

While there exist several widely used workflow management systems, designed with grids as the target execution environment, including Condor DAGMan[4], Pegasus [5], and Kepler [6], researchers begin to develop/port workflow management systems with public clouds (e.g., Polyphony [7] and our own DEWE [8]) to take advantage particularly of the elasticity and pay-as-you-go pricing of the cloud. Although all these systems can run on public clouds, their efficacy is limited due to coordination/scheduling overhead and cost efficiency when executing large-scale workflow ensembles in particular.

To this end, we develop DEWE v2[1]—a major overhaul of our preliminary version of DEWE (or simply DEWE v1) [8]—and design a profiling-based resource provisioning strategy. Our solution system, DEWE v2 adopts a pulling-based approach that removes scheduling overhead by exploiting the homogeneity in both scientific workflows and cloud resources. In particular, many scientific workflows feature a large number of nearly identical tasks in terms of their computation and data requirements making the necessity/effectiveness of scheduling less appealing. Our profiling-based resource provisioning strategy, incorporated into DEWE v2, further leverages the exploitation of the abundance of homogeneous resources in clouds in addition to the task homogeneity. The specific contributions of this paper are:

- We demonstrate that the pulling approach has better performance over the scheduling approach in executing large scale workflow ensembles in clouds.
- We propose a profiling-based strategy to provision computing resources in public clouds to meet both cost and deadline constraints.
- We demonstrate that incremental job submission effectively shapes resource utilization pattern, thus achieve better resource utilization than batch submission.

We have extensively evaluated DEWE v2 using Montage (an astronomical image mosaic engine, Figure 1) [1] workflow ensembles with varying sizes and different configurations of Amazon EC2 clusters. In particularly, our large-scale experiments have been conducted using up to 200 6.0 degree Montage workflows in four Amazon EC2 clusters with different instance types (*c3.8xlarge*, *r3.8xlarge* and *i2.8xlarge*) consisting of up to 1,280 vCPUs.

[1]The source code is available from https://github.com/qyjohn/DEWE.v2.

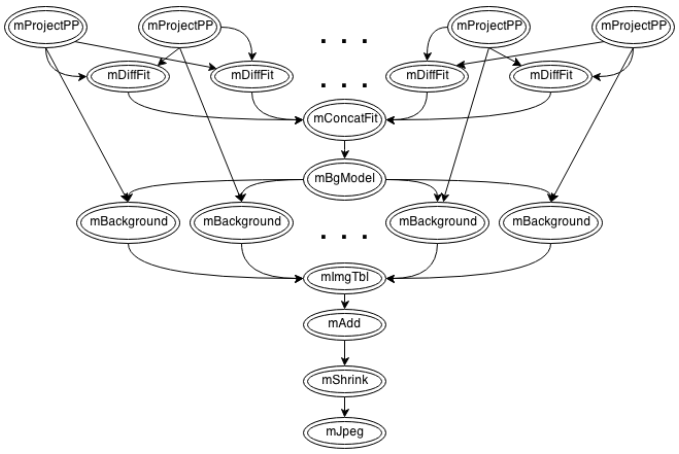


Fig. 1: The structure of the Montage workflow represented by directed acyclic graph (DAG). Vertices/nodes represent tasks and edges represent precedence constraints due primarily to data dependencies.

Our evaluation has been carried out in comparison with Pegasus since DEWE v1 is only capable of running a single workflow at a time. As compared to Pegasus, DEWE v2 can achieve 80% speed-up when running multiple scientific workflows in parallel with the same cluster configuration. The proposed resource provisioning strategy has been incorporated into DEWE v2; and, it clearly demonstrates its capability to identify the most appropriate number of resources to be rented considering both cost and deadline constraints.

The rest of this paper is organized as follows. Section II provides a motivational example focusing on the two main challenges we address in this study, execution coordination and resource provisioning. Section III presents our solution system, DEWE v2 followed by our profiling-based resource provisioning strategy in Section IV. In Section V, we evaluate the performance of DEWE v2, using Pegasus as a comparison, and the efficacy of our profiling-based resource provisioning strategy incorporated into DEWE v2. Section VI reviews related work followed by our conclusions in Section VII.

II. Motivation

In general, there are two approaches to design and implement a workflow management system. The first approach emphasizes scheduling where the master node maintains the state of all participating worker nodes, assigns jobs to worker nodes using various resource scheduling algorithms, as well as stages necessary data files to the worker nodes for job execution. Most existing workflow management systems, from the 'grid era', adopt the scheduling approach, including Condor DAGMan, Pegasus, and Kepler. The second approach emphasizes a stateless design where the master node publishes all pending jobs to a queue, and a number of un-managed worker nodes *pull* the job queue and compete for jobs to execute. The ever-increasing scale of scientific workflows and the availability of abundant homogeneous cloud resources increase the need for cloud-native workflow management systems that cost effectively provision resources and enable the execution of large-scale workflow ensembles with little scheduling overhead.

To motivate the need for pulling-based workflow execution and the possibility of profiling-based resource provisioning, we run a 6.0 degree Montage workflow on four *m3.2xlarge* instances.A 6.0 degree Montage workflow creates a 6-by-6 degree square mosaic centered at a particular region of the sky (e.g., M16). The number of jobs and input data files increases with the number of degrees of the mosaic. A 6.0 degree Montage workflow contains 8,586 jobs, 1,444 input files with a total size of 4.0 GB, and 22,850 intermediate files with a total size of 35 GB. The majority of these 8,586 jobs are copies of a few short-running jobs (`mProjectPP`, `mDiffFit` and `mBackground`).

Figure 1 describes the structure of the Montage workflow. The progress of the workflow has a three-stage pattern (see Figure 2). During the first stage, a large number of `mProjectPP` jobs run in parallel, followed by a large number of `mDiffFit` jobs running in parallel. During the second stage, two jobs `mConcatFit` and `mBgModel` run one after another, during which no other jobs are eligible to run. In this paper, we consider `mConcatFit` and `mBgModel` as blocking jobs because they block the execution of other jobs. During the third stage, a large number of `mBackground` jobs run in parallel, followed by a small number of `mImgTbl`, `mAdd`, `mShrink`, and `mJpeg` jobs.

As shown in Figure 2, the `mProjectPP`, `mDiffFit` and `mBackground` jobs are small jobs with very short execution time within the range of a few seconds. However, they consume and produce a large number of intermediate data files of *similar size* and *number*. Considering the large number of such jobs with many intermediate data files of similar size and number, the decision making of any workflow scheduling algorithm is much complicated, resulting in a significant amount of scheduling overhead. Note that the second instance (the second set of rows) in Figure 2 also acts as the coordinator node in which the initial data resides; hence, significantly less data staging overheads indicated by gaps (short white bars).

Furthermore, the execution time of the second stage is approximately 40% of the makespan. During this stage, among all the available computing resources only one CPU core is being utilized. When the cluster is larger, more computing resources are being wasted during this stage. In a Montage workflow ensemble, resource under-utilization can be worse due to the lack of coordination between individual sub-workflows. Therefore, the Montage workflow ensemble represents a scheduling dilemma requiring trade-offs between cost and performance.

In public clouds like Amazon EC2, a homogeneous environment can be created by launching instances with the same instance type in the same availability zone. This is contrast to the computing resources, in a grid environment, that are considered as heterogeneous; and thus, it is necessary to *schedule* critical jobs to worker nodes with more processing power, and to avoid large data transfer over connections with small bandwidth. The capability and ease of constructing a homogeneous cluster in clouds brings new opportunities in optimizing execution coordination and resource provisioning, which was often not explicitly considered in existing workflow management systems.

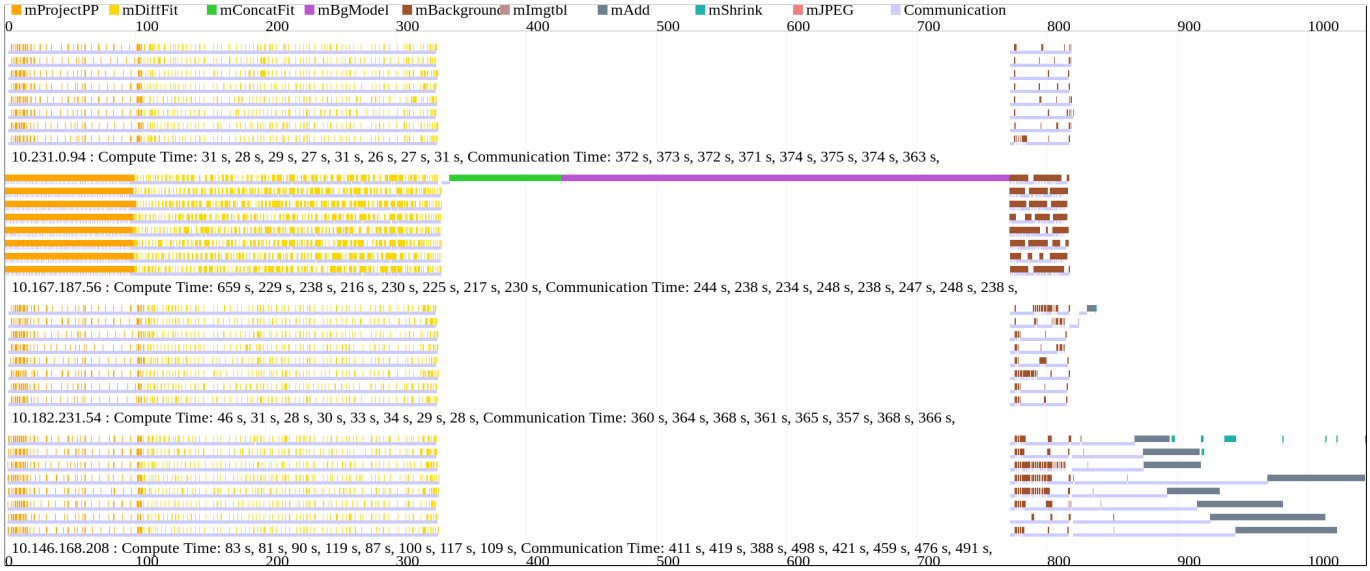


Fig. 2: Detailed visualization of a 6.0 degree Montage workflow running 4 m3.2xlarge instances using DEWE v1. Four sets of 8 rows indicate four EC2 instances, each of which has 8 vCPUs. The horizontal axis represents time in seconds, while the vertical axis represents vCPU slots in the cluster. For each worker node, the graph shows the IP address of the worker node, the time spent on job execution (compute time) and the time spent on data staging (communication time) for each vCPU slot.

III. Design and Implementation of DEWE v2

In this section, we begin the description of DEWE v2 by discussing our design philosophy. We then describe the architecture of DEWE v2 followed by its main components. DEWE v2 shares some fundamental design concepts with DEWE v1; hence the name.

A. Design Philosophy (Scheduling vs. Pulling)

DEWE v2 adopts the pulling approach with the explicit consideration of public clouds, more precisely the availability of abundant tightly coupled homogeneous resources. In particular, with public clouds like Amazon EC2, a homogeneous environment can be achieved by launching all the worker nodes with the same instance type in the same placement group. For critical jobs (i.e., jobs along the critical path of workflow), the computation cost remains the same regardless of the worker node they run on. Furthermore, data transfer between worker nodes can be replaced with a shared file system such as NFS. With a large scale workflow ensemble, the number and size of the input files overwhelm the memory available on the worker nodes. The result is the communication cost becomes the time needed to read the input files from the shared file system, which is statistically the same regardless of the worker node. In this case, the pulling approach has advantages over the scheduling approach because it avoids the scheduling overhead.

B. System Architecture

DEWE v2 consists of three main components: a master daemon, a worker daemon, and a workflow submission application (Figure 3). In a cluster environment, one of the nodes runs the master demon, which can optionally run the worker daemon at the same time. All other nodes run the worker daemon. Using the workflow submission application, scientists/users can submit workflows to the master daemon from any nodes at any time. Figure 3 shows the architecture design of DEWE v2.

The master daemon only manages the progress of the workflow, and publishes jobs that are eligible to run to a message queue. It has no knowledge about the worker nodes, but assumes that the worker nodes are homogeneous in terms of computing capability and communication bandwidth.

On the basis of "first come, first served", the worker nodes actively pull the message queue for jobs to execute. When the job is successfully executed, the worker node sends an acknowledgment message to the master daemon. Based on the acknowledgment messages from the worker nodes, the master daemon publishes new jobs that are eligible to run to the message queue.

A POSIX-compliant shared file system is used to facilitate the data sharing between worker nodes. When an output file is generated by a job on a worker node, it is immediately accessible from other worker nodes and can be used as inputs files for other jobs. This shared file system can be provided by either a centralized storage server (such as a NAS device) or a distributed storage system (such as GlusterFS). We assume that all worker nodes have equal access to the shared file system. A workflow is encapsulated in a folder on the shared file system, including the DAG file, the executable binaries, as well as the input and output files.

To increase the robustness of the system, a timeout mechanism is added to the DAG management module in the master daemon. A job can have a user-defined timeout value or a system-wide default timeout value. If a job has been checked out from the message queue for execution but the corresponding acknowledgment is not received by the master daemon within the timeout setting, the master daemon publishes the

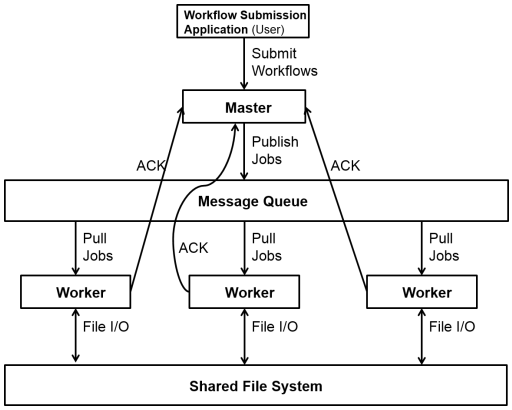


Fig. 3: The architecture of DEWE v2.

job to the message queue again. With this timeout approach, any worker node can fail at any time and the failed jobs will be automatically resubmitted to the message queue for execution by other worker nodes when the timeout occurs.

The master daemon is capable of managing multiple workflows concurrently. When precedence dependencies are met, jobs in different workflows are published to the same message queue for execution. Therefore, multiple workflows can be executed in parallel on the same cluster.

As we can see, DEWE v2 significantly simplifies the workflow execution process. There is no scheduling at any stage during the execution of the workflow. The stateless design of the worker node allows the cluster to scale in or scale out according to the actual workload requirements.

C. Implementation of the Master Daemon

At the core of DEWE v2 is a message queue system based on RabbitMQ. We use three separate topics in the message queue for workflow submission, job dispatching, and job acknowledgment. When the workflow submission application submits a workflow, meta data about the workflow (the name of the workflow, as well as the path to the related folder on the shared file system) is published to the workflow submission topic. The master daemon pulls meta data about the workflow from the workflow submission topic, then parses the DAG file and stores the job dependencies information into a data structure. If a job has no pending dependency precedence requirements, the master daemon publishes meta data about the job (the location of the binary executable with input and output parameters) to the job dispatching topic.

When a job is checked out by a worker node for execution, the worker node sends a message to the job acknowledgment topic indicating the job is now running. When a job is successfully executed on a worker node, the worker node sends another message to the job acknowledgment topic indicating the job is now completed. The master daemon pulls the job acknowledgment topic for such messages. If the message indicates a job is running, the master daemon marks the job as "running" so that the job is no longer visible to other worker nodes. If the message indicates a job is completed, the master daemon marks the job as "completed" and updates the status of all pending jobs that depend on the completed job. When a job has no pending precedence requirements it becomes eligible to run. Then the master daemon publishes meta data about jobs that are eligible to run to the job dispatching topic, where they are pulled by the worker nodes for execution.

The master daemon periodically examines the status of all "running" jobs. If a job is checked out by a worker node for execution but the corresponding acknowledgment indicating the job is completed is not received within its timeout setting, a timeout event is triggered. The master daemon then resubmits meta data about the job to the job dispatching topic so that other worker nodes can execute the job again.

D. Implementation of the Worker Daemon

The worker daemon has a stateless design. The only knowledge it has about the whole workflow execution system is the address of the message queue. It has no knowledge about the master node, other worker nodes in the system, or the jobs that have been executed on the worker node itself. It reads input files from, and writes output files to, the shared file system, just like using a local file system.

The worker daemon pulls the job dispatching topic for jobs to execute. Upon receiving a job from the message queue, the worker daemon sends a message to the job acknowledgment topic indicating the job is now running. A separate thread is launched by the worker daemon to handle each individual job. Upon completion of the job, the worker daemon sends another message to the job acknowledgment topic indicating the job is now completed. The thread associated with a job is terminated when the job is completed.

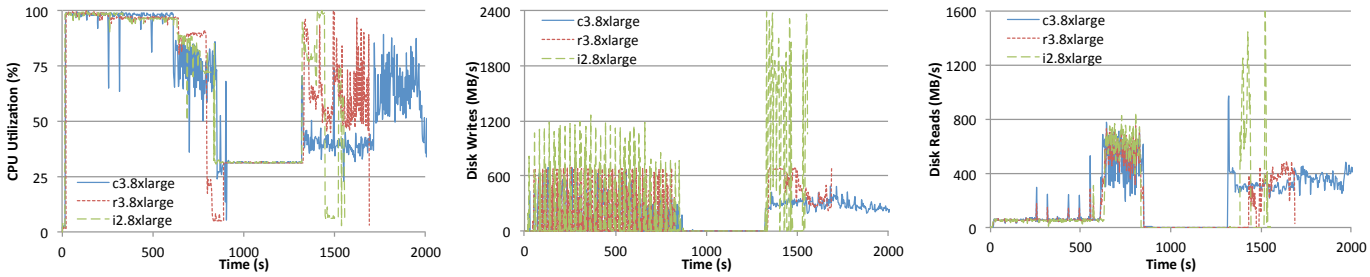
To avoid resource competition among concurrently running jobs, we put an upper limit on the number of concurrent job execution threads. The worker daemon stops pulling the job dispatching topic when the number of concurrent job execution threads equals to the number of CPUs available on the worker node. However, the worker daemon does not bind a job to a particular CPU. If a job is implemented in a way that can leverage multiple CPUs (for example, OpenMP), the desired behavior is preserved. This feature can significantly speed up the execution of a workflow when the blocking jobs (e.g., `mConcatFit` and `mBgModel` in Montage workflow) are implemented as parallel code.

E. Implementation of the Workflow Submission Application

The workflow submission application accepts two parameters from the user—workflow name and the path to the related folder on the shared file system. The workflow submission application publishes this information to the workflow submission topic in the message queue system, where it is checked out by the master daemon for further processing.

IV. Profiling-Based Resource Provisioning

In this section, we propose a profiling-based strategy to provision computing resources for large scale scientific workflow ensembles with both cost and deadline constraints. As many scientific workflows show homogeneity particularly in the sense that a majority of tasks in a workflow have similar resource consumption pattern, we begin with small scale experiments to profile the resource consumption patterns



(a) CPU Utilization (b) Disk Write Operations (c) Disk Read Operations

Fig. 4: Resource consumption patterns of ten 6.0 degree Montage workflows running on a single node cluster with DEWE v2.

of the workflow ensemble. Based on the small scale testing results we derive the performance index of a worker node. Then we use the performance index to determine the number of worker nodes needed for the actual large scale experiments. To simplify our discussions, all the tests presented in this section are carried out with batch submission.

A. *Profiling*

To come up with a resource provisioning strategy for executing large scale scientific workflow ensembles in public clouds, we use both single-node tests and multi-node tests to profile the resource consumption pattern of multiple workflows running in parallel. In the single-node tests, we run up to ten 6.0 degree Montage workflows on a single-node cluster with DEWE v2. The largest workload contains 85,860 jobs, 14,440 input files with a total size of 40 GB, and 228,500 intermediate files with a total size of 350 GB. In the meantime, our multi-node tests deal with 20 6.0 degree Montage workflows with varying numbers of nodes.

We use the c3.8xlarge, r3.8xlarge, and i2.8xlarge instances on AWS EC2 in our profiling. Table I lists the specifications of the selected instance types. On all of the selected instance types, the storage devices are SSD-backed instance store volumes (storage from disks that are physically attached to the host computer). To achieve the best disk I/O performance, we combine all the instance store volumes available on the instance in a RAID 0 configuration. All the workflow related disk I/O operations are configured to occur on the RAID 0 device. The file system being used on all worker nodes is ext4. In the multi-node tests, all nodes share their storage using NFS. While all three instance types have similar CPU and memory performance, there exists significant difference in the disk I/O performance of the RAID 0 device, as shown in Table II. During the experiments we run a background monitoring process on all worker nodes to collect operating system level metrics every 3 seconds using mpstat and iostat. The metrics collected include the number of concurrent threads, CPU utilization rate, I/O operations per second, as well as disk read and disk write throughputs. This profiling technique allows us to understand and compare the actual resource consumption during the execution of a workflow under various conditions for example with different workflow management system or using different cluster configurations.

Figure 4 shows the resource consumption pattern of ten 6.0 degree Montage workflows running on a single-node cluster with DEWE v2. During the first stage, the workflow is CPU intensive, as evidenced by the 100% CPU utilization rate on all three instance types (Figure 4a). If we look at the disk write operations alone (Figure 4b), we would think that the workflow is I/O intensive during this stage. However, this stage takes approximately the same amount of time on all three instance types, regardless of the significant difference in their write throughput. This indicates that CPU is the real bottleneck during this stage. The operating system caches the disk writes and flushes them to the disk in batches, resulting in the intermittent disk writes at full capacity. During the second stage, the workflow is neither CPU intensive nor I/O intensive, as evidenced by the low CPU utilization rate and zero disk writes. The progress of the workflow is controlled by the single-thread `mConcatFit` and `mBgModel` jobs. During the third stage, the workflow is I/O intensive. The i2.8xlarge instance, with the highest I/O capacity, finishes executing this stage first, following by the r3.8xlarge and the c3.8xlarge instances, according to their I/O capacities.

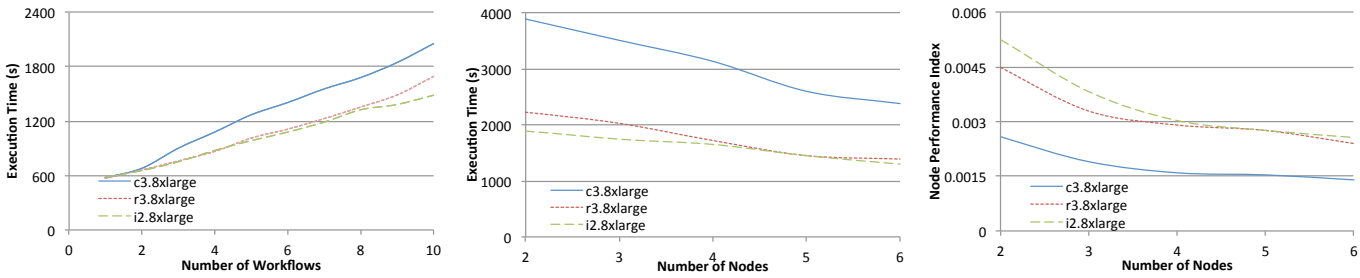
TABLE I: EC2 Instance Types

Model	vCPU	Memory (GB)	Storage (GB)	Network (Gbps)	Price (USD/hour)
c3.8xlarge	32	60	2 x 320	10	1.68
r3.8xlarge	32	244	2 x 320	10	2.80
i2.8xlarge	32	244	8 x 800	10	6.82

TABLE II: Disk I/O Capacity of EC2 Instance Types

Model	Sequential Read (MB/s)	Sequential Write (MB/s)	Random Read (MB/s)	Random Write (MB/s)
c3.8xlarge	250	800	400	600
r3.8xlarge	350	1000	700	800
i2.8xlarge	2200	3800	1800	3600

Figure 5 shows the impact of workload and cluster size on the performance of the cluster. On the single-node cluster, the size of the cluster remains the same. As the number of workflows increases, the execution time increases linearly (Figure 5a). On the multi-node cluster, the size of the workload remains the same, i.e., 20 6.0 degree Montage workflows. As the number of worker nodes increases, the execution time decreases linearly (Figure 5b); however, the slope is quite smooth being flattened out as the number of worker nodes increases.



(a) Single-Node Cluster (b) Multi-Node Cluster (c) Performance Degradation

Fig. 5: The impact of workload and cluster size on the performance of the cluster. On the single-node cluster, we run up to 10 6.0 degree Montage workflows. On the multi-node clusters, we run 20 6.0 degree Montage workflows regardless cluster size.

B. Node Performance Index

The node performance index P of the worker nodes in a multi-node cluster can be defined as the execution speed of the workflow on a single node (or workflow per second per node). More formally,

$$P = \frac{W}{N * T} \tag{1}$$

where W is the number of workflows running on the cluster, N is the number of worker nodes in the cluster, and T is the execution time needed for N workflows. A simple way to read this is how much of a workflow can be completed by one worker node in one second. As shown in Figure 5c, as the number of worker nodes increases, the node performance index decreases. The phenomenon is commonly observed in clusters, and is referred to as clustering performance degradation. In our test case, the observed clustering performance degradation gradually converges when the number of worker nodes is greater than 4. Based on Figure 5c, we estimate that the node performance indexes for large scale clusters are 0.0015, 0.0024, and 0.0026 for clusters with c3.8xlarge, r3.8xlarge, and i2.8xlarge instance types.

Based on Equation 1, we can estimate the number of worker nodes needed to execute a large scale scientific workflow ensemble with deadline constraints using the following formula:

$$N = \frac{W}{P * T} \tag{2}$$

where N is the desired number of worker nodes in the cluster.

V. Evaluation

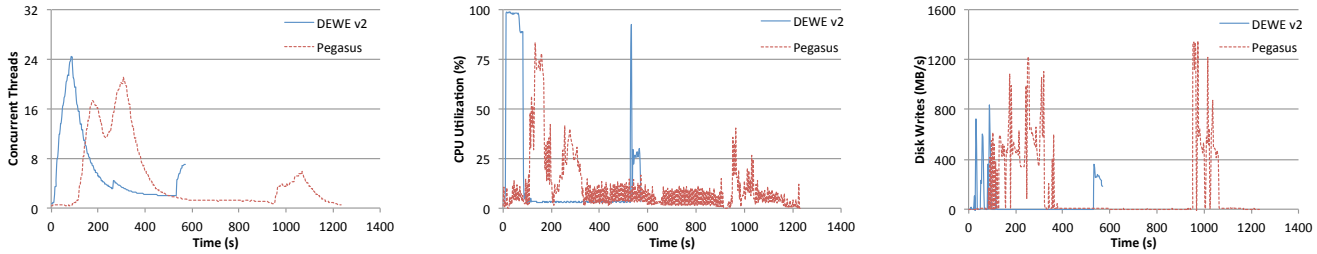
In this section, we evaluate the performance of DEWE v2 in terms of execution coordination and resource provisioning.

A. Performance of DEWE v2

The evaluation on the performance of DEWE v2 is conducted in comparison with Pegasus, a well-known scheduling-based workflow management system using up to five 6.0 degree Montage workflows. We choose Pegasus because (a) the Montage workflow was previously thoroughly studies in Pegasus; and (b) the Pegasus team provided a shared AMI on AWS with the Montage workflow as an example. It should be mentioned that Pegasus is only responsible for workflow planning, and it uses DAGMan for job scheduling and Condor for job execution. In this sense, DEWE v2 is the equivalent of Pegasus + DAGMan + Condor. Within the scope of this paper, the term "Pegasus" actually refers to a system including Pegasus, DAGMan and Condor. We verify that the results obtained from DEWE v2 and Pegasus are identical by comparing the size and MD5 check sum of the final output images produced by job `mJpeg` (see Figure 1). The experiments are carried out on AWS EC2 in its us-east-1 region. The instance type being used is c3.8xlarge. The storage being used is the instance-store SSD volumes with RAID 0 configuration. To eliminate the impact of network latency, the required input files are downloaded to the storage device before the experiments.

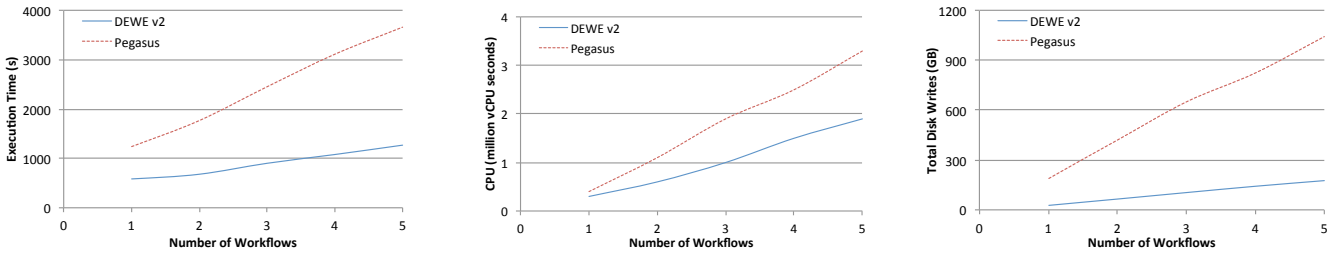
1) Scheduling vs Pulling: Figure 6 shows the resource consumption patterns of one 6.0 degree Montage workflow running on a single-node cluster with DEWE v2 and Pegasus. Although the c3.8xlarge instance has 32 vCPU, the maximum number of concurrent threads observed is 25 for DEWE v2 and 20 for Pegasus. The maximum CPU utilization observed is 100% for DEWE v2 and 80% for Pegasus. This indicates that DEWE v2 is more efficient in utilizing CPU resources. The observed disk write operations for Pegasus are much more than DEWE v2, indicating that Pegasus carries out more disk I/O activities than DEWE v2. As a result, the average makespan for DEWE v2 is 600 seconds whereas that for Pegasus is 1240 seconds, which is significantly longer.

Figure 7 shows the resource consumption of multiple 6.0 degree Montage workflows running on a single node cluster with DEWE v2 and Pegasus. The instance being used is c3.8xlarge on AWS EC2 in the us-east-1 region. A desired number of workflows are submitted to the workflow management system in one batch. The required input files are downloaded to the storage devices on the instance before the experiments. Total execution time (Figure 7a) refers to the time needed to finish the execution of the workflows, regardless of the actual resource utilization rate on the worker nodes. When the number of workflows increases, the total execution time increases linearly. Total CPU time (Figure 7b) refers to the actual CPU time spent on job execution activities, which is calculated by integrating the actual CPU utilization rate over the entire workflow execution period on all CPUs. Total disk writes (Figure 7c) refers to the amount of data being written to the file system, which is calculated by integrating the



(a) Concurrent Threads (b) CPU Utilization (c) Disk Write Operations

Fig. 6: Resource consumption patterns of one 6.0 degree Montage workflow on a single c3.8xlarge instance.



(a) Total Execution Time (b) Total CPU Time (c) Total Disk Writes

Fig. 7: Resource consumption of multiple 6.0 degree Montage workflow on a single c3.8xlarge instance.

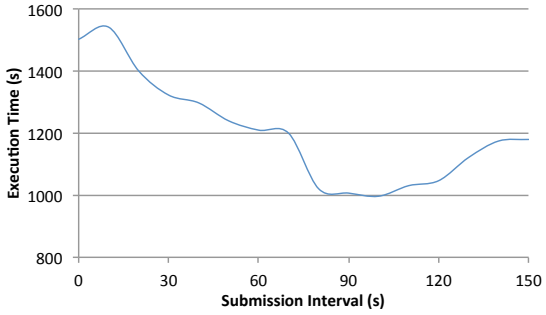


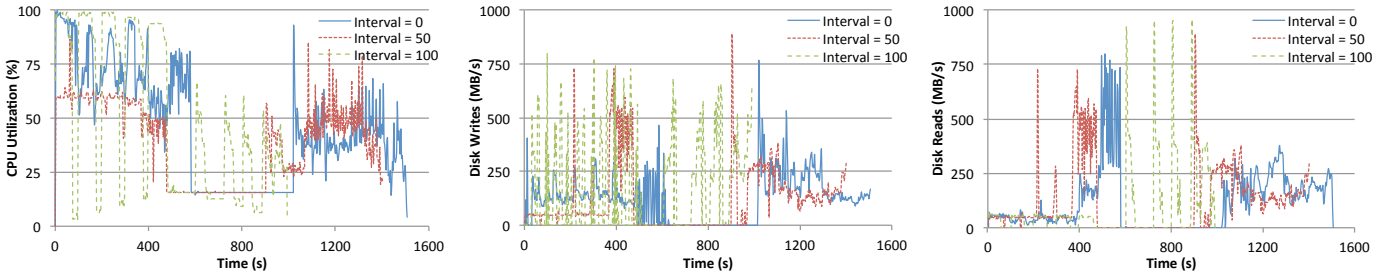
Fig. 8: Impact of submission intervals on execution time.

actual disk write throughput over the entire workflow execution period. When the number of workflows increases, both total CPU time and total disk writes increase linearly. In general, Pegasus consumes a lot more computing resource (such as CPU time and disk writes) than DEWE v2, resulting in much longer execution time. For example, the execution time of five 6.0 degree Montage workflows being run with DEWE v2 is approximately the same as the execution time of one 6.0 degree Montage workflow being run with Pegasus. In other words, DEWE v2 can achieve 80% speed-up when running multiple workflows in parallel with the same cluster configuration.

2) Workflow Submission Intervals: A 6.0 degree Montage workflow demands different computing resources in different stages. When executing a large scale workflow ensemble with many workflows on the same cluster, it is possible to optimize resource utilization by controlling the workflow submission intervals so that different workflows in the workflow ensemble do not demand the same computing resource at the same time. In this test, we run a workflow ensemble with five 6.0 degree Montage workflows with DEWE v2 on a single-node cluster with both the master daemon and worker daemon on the same node. The test includes submitting all five workflows in one batch (batch submission), or submitting the five workflows one after another at fixed intervals (incremental submission). Batch submission can be considered as a special case of incremental submission where the submission interval is zero. As shown in Figure 8, the time needed to execute all five workflows decreases when the submission interval increases, then increases again when the submission interval is greater than 100 seconds. In this particular test case, 34% speed up can be achieved by setting the submission interval to 100 seconds.

Figure 9 shows the resource consumption patterns of the test workflow ensemble with five 6.0 degree Montage workflows running on a single node cluster with DEWE v2. Due to space limits we only show results with workflow submission intervals of 0, 50 and 100 seconds. As shown in Figure 9a, when we increase the workflow submission interval, the CPU utilization pattern in the system changes. When submission interval is 0 second (batch submission), the CPU utilization exhibits a clear three-stage pattern, with significant resource under-utilization in the second stage. This is very similar to the CPU utilization pattern in a single workflow. When submission interval is 100 seconds, such three-stage pattern is no longer obvious. This is because different types of jobs from different workflows can be executed in parallel, resulting in an increase in average CPU utilization across the whole execution time. The same result is also observed in disk I/O activities, which is reflected in both disk writes (Figure 9b) and disk reads (Figure 9c). Due to the increase in resource utilization, shorter execution time can be achieved with well-designed incremental submission techniques. The investigation of more sophisticated submission strategies is beyond the scope of this paper and we plan to do this as our future work.

3) System Robustness: We carry out two tests to examine the robustness of DEWE v2. In one test, we run one 6.0 degree Montage workflow with DEWE v2 on a single-node cluster with both the master daemon and worker daemon on the same node. During the execution of the workflow, we introduce interruptions to the system by killing the worker daemon and then starting it again 5 seconds later. In the other test, we run



(a) CPU Utilization (b) Disk Write Operations (c) Disk Read Operations

Fig. 9: Resource consumption patterns of five 6.0 degree Montage workflows on a single c3.8xlarge instance with DEWE v2.

one 6.0 degree Montage workflow with DEWE v2 on a two-node cluster with NFS as the shared file system. One of the nodes has both the master daemon and the worker daemon, while the other node has only the worker daemon. However, at any time there is only one worker daemon running. During the execution of the workflow, we introduce interruptions to the system by killing the worker daemon on one node, then starting the worker daemon on the other node 5 seconds later.

In both tests, interrupted jobs are automatically resubmitted for execution after timeouts. DEWE v2 is capable of completing the execution of the workflow, regardless of number of interruptions. When the interruptions are introduced during the execution of non-blocking jobs, such as `mProjectPP` and `mDiffFit`, the increase in makespan roughly equals to the total duration of the interruptions. This is because DEWE v2 can resume execution of the workflow as soon as the worker daemon restarts, without the need to wait for the timeout of the interrupted jobs. When the interruptions are introduced during the execution of blocking jobs, such as `mConcatFit` and `mBgModel`, the increase in makespan roughly equals to the sum of the timeout settings of the interrupted jobs. This is because DEWE v2 must wait for the timeout of the interrupted jobs to resume execution of the workflow.

DEWE v2's capability of resuming workflow execution after interruption of the worker daemon opens the door for dynamic resource provisioning. During the execution of large scale workflow ensembles, researchers can dynamically adjust the number of worker nodes in a cluster to meet both deadline and cost constraints. When there are a large number of non-blocking jobs in the queue, more worker nodes can be added to the cluster to speed up the execution. When there are a limited number of blocking jobs in the queue, some worker nodes can be removed from the cluster to reduce cost. Such dynamic resource provisioning strategy might not be effective for public clouds with a charge-by-hour model (such as AWS), but can be useful for public clouds with a charge-by-minute model (such as Google Compute Engine). In this paper, we carry out all our experiments on AWS, therefore we are not able to explore further on this topic.

B. *Evaluation of Profiling-based Resource Provisioning*

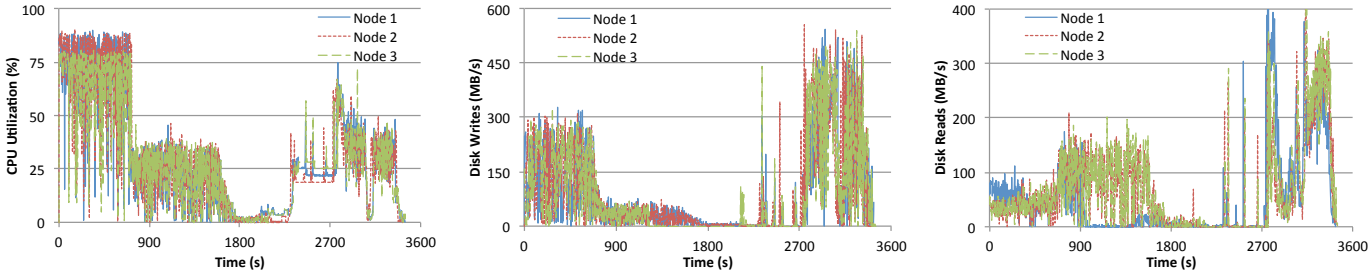
We use large scale experiments to evaluate our resource provisioning strategy. The largest workflow ensemble includes 200 6.0 degree Montage workflows, which contains 1,717,200 jobs, 288,800 input files, and 4,570,000 intermediate files. Approximately 7.0 TB data is written to the underlying storage during the execution.

TABLE III: Cluster Configurations

Cluster	Nodes	vCPU	Memory (TB)	Storage (TB)	Price (USD/hr)
c3.8xlarge	40	1280	2.40	25.6	67.2
r3.8xlarge	25	800	6.10	16.0	70.0
i2.8xlarge	23	768	5.61	147.2	156.7
i2.8xlarge B	10	320	2.44	64.0	68.2

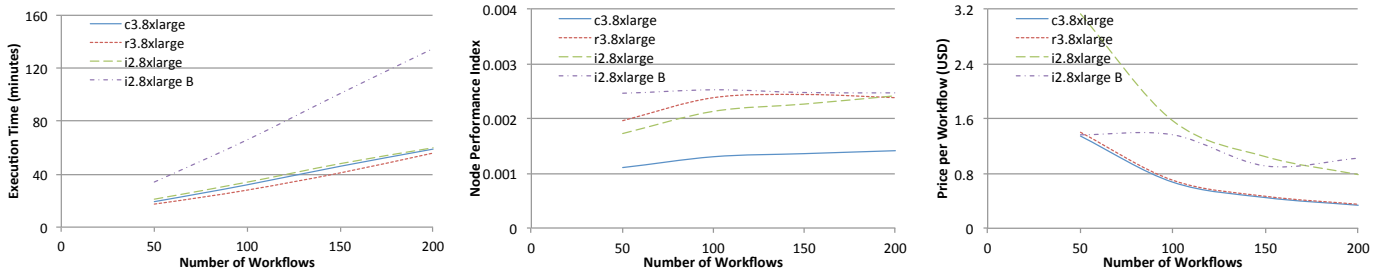
In order to meet both cost and deadline constraints, we design our clusters with the goal to complete the largest workload ensemble (W = 200) within an hour. This is because users pay for EC2 instances by the hour, and any partial hour usage will be charged as a full hour. The time constraint T is set to 3300 seconds (55 minutes) because we would like to have some flexibility in the execution time. Based on Equation 2, the estimated number of worker nodes are 40, 25, and 23 for c3.8xlarge, r3.8xlarge and i2.8xlarge instance types. An additional cluster i2.8xlarge B with the i2.8xlarge instance type and 10 nodes is also tested as a comparison. We use 10 nodes for the i2.8xlarge B cluster because it has approximately the same hourly price as the c3.8xlarge and r3.8xlarge clusters. Table III summarizes the experimental setup.

In our previous experiments, we use NFS as the share file system between worker nodes. This requires each and every worker node to share its local storage via NFS, and mount the NFS shares from other nodes, resulting in an N-to-N mapping between worker nodes. As the size of the cluster grows, the configuration of the cluster becomes increasingly complex, resulting in unbalanced utilization. In the large scale experiments, we use MooseFS (http://www.moosefs.org/) as the shared file system between worker nodes. All the worker nodes are configured to be a MooseFS trunk server. Using cloud-init scripts, the worker nodes automatically join the storage pool and mount the MooseFS file system when the instances are being launched. Furthermore, the required input files are copied to the shared file system before the experiments. Therefore, the test results only include the execution time of the workflow ensemble. As a distributed file system, MooseFS has the option to store one file with multiple copies on different storage devices. In order to save storage space, each file has only one copy in our experiments. The worker nodes mount the shared file system as a POSIX-compliant file system. When executing a particular job, the worker daemon has no knowledge about the actual location of the input and output files. Although data locality can not be assumed in our experiments, it is safe to assume that statistically all worker nodes have equal access to the underlying shared file system.



(a) CPU Utilization (b) Disk Write Operations (c) Disk Read Operations

Fig. 10: Resource consumption patterns of 200 6.0 degree Montage workflows running on a 25-node r3.8xlarge cluster.



(a) Execution Time (b) Performance Index (c) Price per Workflow

Fig. 11: Evaluation of profiling-based resource provisioning strategy using large scale experiments.

Figure 10 shows the resource consumption pattern of 200 6.0 degree Montage workflows running on the r3.8xlarge cluster. The cluster includes 25 worker nodes, but we only present data from three worker nodes. As shown in the figure, all three worker nodes have the same the resource consumption patterns, which is similar to the resource consumption pattern on a single-node cluster (Figure 4). This is also true for other worker nodes not shown in the figure. This indicates that the workload is evenly distributed across the cluster. The cluster behaves in a way that is similar to a supercomputer.

Figure 11 shows the execution time, node performance index, and price per workflow for workflow ensembles with different number of 6.0 degree Montage workflows. For all clusters, the execution time increases linearly as the number of workflows in the workflow ensemble increases (Figure 11a). On clusters c3.8xlarge, r3.8xlarge and i2.8xlarge, the workflow ensemble with 200 6.0 degree Montage workflows is completed within 60 minutes, meeting the designed deadline constrain. On cluster i2.8xlarge B, the workflow ensemble with 200 6.0 degree Montage workflows takes 135 minutes to complete, far exceeding the designed deadline constraint.

Figure 11b shows the node performance index for different clusters. The i2.8xlarge B cluster has the highest node performance index. This is because the cluster has the smallest number of nodes, resulting in the highest resource utilization rate. For clusters c3.8xlarge, r3.8xlarge and i2.8xlarge, the node performance index grows when the workload ensemble grows. When the number of workflows is small, the clusters are not fully utilized. In this case, the observed node performance index is lower than the designed node performance index. When the number of workflows is large, the clusters become fully utilized. In this case, the observed node performance index is very close to the designed node performance index.

Figure 11c shows the average price of executing a single workflow on different clusters under different workloads. For clusters c3.8xlarge, r3.8xlarge and i2.8xlarge, all the tests are completed in one hour. With the hourly pricing model, the cost is the same for different workloads. As a result, the price per workflow decreases as the workload increases. This suggests that the size of the cluster should be carefully designed based on the target workload to achieve the best price performance. For cluster i2.8xlarge B, the price per workflow fluctuates because the costs of running different workload are different. However, for the designed workload with 200 6.0 degree Montage workflows, all three clusters designed with the proposed resource provision model (c3.8xlarge, r3.8xlarge and i2.8xlarge) achieve lower price per workflow than cluster i2.8xlarge B, which is not designed with the proposed resource provision model. This indicates that the proposed resource provision strategy is effective in designing clusters to meet both cost and deadline constraints.

VI. Related Work

While there have been many efforts put on executing scientific workflows, they are mostly focusing on scheduling at small scale, with single workflow at a time or using simulations. These efforts range from workflow management systems, [5], and Kepler [6], [7], to scheduling and resource allocation algorithms, [9], [10], [11], [12], [13], [14], [15].

As scientific workflows become increasingly large-scale and complex, their distributed execution across multiple resources is far beyond an average task. Coinciding with this increase in scale and complexity have been efforts on developing workflow management systems, including Condor DAGMan[4], Pegasus [5], and Kepler [6]. These frameworks tend to be heavy-weight and are inaccessible to scientists who lack dedicated hardware and support staff. Moreover, many

of these workflow management systems are designed for grid environment and focus on providing independence from the underlying execution environment.

Early pulling-bases systems such as Celery (http://celery.readthedocs.org/) and Work Queue (http://ccl.cse.nd.edu/software/workqueue/) require the end users to develop an application for each and every workflow, which is not convenient for large-scale scientific workflows. Apache Crunch (https://crunch.apache.org) and Apache Falkon (http://falcon.apache.org) are built around Apache Hadoop, and can only be used to handle workflows running on top of Apache Hadoop. AWS ElasticBeanstak environment tier[2] and Windows Azure worker role[3] can pull jobs from a queue for execution, but the end users still need to develop an application to handle precedence requirements between different jobs and data staging across multiple worker nodes.

Polyphony [7] was designed and developed with AWS as the target execution environment, but the software is not accessible to the workflow researcher community. Furthermore, Polyphony uses the AWS Simple Queue Service (SQS) as the message queue, which is not intended for high performance computing applications. The work in [10] deals with scheduling scientific workflows across multiple geographically distributed resource sites; however, the scale of workflows is still limited to small, e.g., 255 tasks per workflow. All of the above-mentioned workflow management systems exhibit inefficiency in scheduling a large number of short-life jobs across multiple worker nodes.

In order to execute large scale scientific workflow ensembles in a cost effective way, the computing resources needed for the execution must be carefully planned. Such planning usually involves cost and performance trade-off for scientists. Most of existing literature on resource provisioning for scientific workflows use grid as the target execution environment [15], [12], [14]. Malawski et al. [13] developed a set of algorithms to increase the efficiency of executing scientific workflow ensembles on public clouds under cost and deadline constraints. The authors evaluated the effectiveness of these algorithms in simulated environments

VII. Conclusion

In this paper, we address two main challenges in executing large-scale workflow ensembles in public clouds: (1) execution coordination, and (2) resource provisioning. We present our solutions to these challenges with the development of DEWE v2, a pulling-based workflow management system, and its effective resource provisioning strategy. By adopting the pulling approach in our solution system, we have demonstrated that much of scheduling overhead when executing a large-scale workflow ensemble particularly in public clouds can be removed as a majority of tasks in scientific workflows often exhibit homogeneity in their resource consumption pattern and acquiring a large number of homogeneous public cloud resources is easily possible. We compare the performance of DEWE v2 with Pegasus showing DEWE v2 is capable of achieving 80% speed-up. We have also demonstrated that provisioning cloud resources using our profiling-based strategy based on node performance index is very effective in terms of both cost and deadline compliance.

[2]http://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features-managing-env-tiers.html.

[3]http://azure.microsoft.com/en-us/documentation/articles/fundamentals-introduction-to-azure/

Acknowledgment

This work is supported by the AWS in Education Research Grants. Dr. Young Choon Lee would like to acknowledge the support of the Australian Research Council Discovery Early Career Researcher Award (DECRA) Grant DE140101628.

References

[1] J. C. Jacob and D. S. e. a. Katz, "Montage: a grid portal and software toolkit for science-grade astronomical image mosaicking," *Int'l J. Computational Science and Engineering*, vol. 4, no. 2, pp. 73–87, 2009.

[2] A. Abramovici, W. E. Althouse, and et. al., "LIGO: The laser interferometer gravitational-wave observatory," *Science*, vol. 256, no. 5055, pp. 325–333, 1992.

[3] R. Graves, T. H. Jordan, and et. al., "Cybershake: A physics-based seismic hazard model for Southern California," *Pure and Applied Geophysics*, vol. 168, no. 3-4, pp. 367–381, 2010.

[4] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger, "Workflow management in condor," in *Workflows for e-Science*, 2007, pp. 357–375.

[5] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny, "Pegasus: Mapping scientific workflows onto the grid," in *Proceedings of IEEE International Conference on Grid Computing*, 2004, pp. 11–20.

[6] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: an extensible system for design and execution of scientific workflows," in *Proceedings. 16th International Conference on Scientific and Statistical Database Management*, 2004, pp. 423–424.

[7] K. S. Shams, M. W. Powell, and et al., "Polyphony: A workflow orchestration framework for cloud computing," in *Proceedings of 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, 2010, pp. 606–611.

[8] L. M. Leslie, C. Sato, Y. C. Lee, Q. Jiang, and A. Y. Zomaya, "DEWE: A framework for distributed elastic scientific workflow execution," in *Proceedings of 13th Australasian Symposium on Parallel and Distributed Computing (AusPDC)*, 2015, pp. 3–10.

[9] M. Tanaka and O. Tatebe, "Disk cache-aware task scheduling for data-intensive and many-task workflow," in *Proceedings of IEEE International Conference on Cluster Computing*, 2014, pp. 167–175.

[10] K. Maheshwari, E.-S. Jung, J. Meng, V. Vishwanath, and R. Kettimuthu, "Improving multisite workflow performance using model-based scheduling," in *Proceedings of 2014 43rd International Conference on Parallel Processing (ICPP)*, 2014, pp. 131–140.

[11] Y. C. Lee and A. Y. Zomaya, "Stretch out and compact: Workflow scheduling with resource abundance," in *Proceedings of 2013 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2013, pp. 219–226.

[12] R. Duan, R. Prodan, and T. Fahringer, "Performance and cost optimization for multiple large-scale grid workflow applications," in *Proceedings of the ACM/IEEE Conference on Supercomputing (SC)*, 2007, pp. 1–12.

[13] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," in *Proceedings of Int'l Conference on High Performance Computing, Networking, Storage and Analysis*, 2012, pp. 22:1–22:11.

[14] G. Juve and E. Deelman, "Resource provisioning options for large-scale scientific workflows," in *Proceedings of IEEE Fourth International Conference on eScience*, 2008, pp. 608–613.

[15] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R. Graves, N. Gupta, V. Gupta, T. H. Jordan, C. Kesselman, P. Maechling *et al.*, "Managing large-scale workflow execution from resource provisioning to provenance tracking: The cybershake example," in *Proceedings of IEEE Second International Conference on eScience*, 2006, pp. 14–14.