Experiences Using Cloud Computing for A Scientific Workflow Application

Jens-S. Vöckler, Gideon Juve, Ewa Deelman, Mats Rynge
Information Sciences Institute
University of Southern California
{voeckler,gideon,deelman,rynge}@isi.edu

G. Bruce Berriman
Infrared Processing and Analysis Center
California Institute of Technology
gbb@ipac.caltech.edu

ABSTRACT

Clouds are rapidly becoming an important platform for scientific applications. In this paper we describe our experiences running a scientific workflow application in the cloud. The application was developed to process astronomy data released by the Kepler project, a NASA mission to search for Earth-like planets orbiting other stars. This workflow was deployed across multiple clouds using the Pegasus Workflow Management System. The clouds used include several sites within the FutureGrid, NERSC's Magellan cloud, and Amazon EC2. We describe how the application was deployed, evaluate its performance executing in different clouds (based on Nimbus, Eucalyptus, and EC2), and discuss the challenges of deploying and executing workflows in a cloud environment. We also demonstrate how Pegasus was able to support sky computing by executing a single workflow across multiple cloud infrastructures simultaneously.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems*

General Terms

Experimentation

Keywords

Pegasus, Periodograms, Cross Cloud Computing, Nimbus, Eucalyptus, FutureGrid, EC2, Magellan, Workflow, Experience, Sky Computing

1. INTRODUCTION

Many important scientific applications can be expressed as workflows, which describe the relationship between individual computational tasks and their input and output data in a declarative way. When developing workflow applications it is convenient to use an abstract workflow definition that is devoid of resource-specific details. This enables workflows to be automatically adapted to run across different environments. For complex workflows, abstraction also helps scientists to express their workflows at a higher level without being concerned about the details of how individual jobs are invoked or how data is transferred between jobs. Such abstract workflows can be processed by the Pegasus Workflow Management System [3], which automatically converts abstract workflow descriptions into concrete execution plans containing resource-specific information. In order to generate an executable workflow, Pegasus analyzes the abstract workflow, adds auxiliary data transfer and cleanup jobs, performs workflow optimizations such as task clustering and workflow reduction, and generates resource-specific job descriptions. The concrete workflow is passed to DAGMan and Condor[1] for execution on distributed resources.

Clouds are being investigated as a platform for the execution of scientific applications such as workflows. In contrast to grids and other traditional HPC systems, clouds provide a customizable infrastructure where applications can provision the desired resources ahead of the execution and deploy the required software environment on virtual machines (VMs).

Custom virtual machine images containing specific operating systems, services, and configuration can be constructed for each application. These images can be deployed on VMs in a number of different cloud platforms, including commercial clouds such as Amazon EC2[2] and science clouds built on top of open source cloud management software such as Eucalyptus[3], Nimbus[4], Open Nebula[5], and others.

Cloud management systems provide a service-oriented model for provisioning and managing computational resources. In this model, scientists can request virtual machine resources on-demand for their application. The ability to provision resources, however, is not sufficient to run a workflow application. The computational resources provided by clouds are basic and in many cases only the base OS, networking and simple configuration is included. What is missing for scientific workflows are job and data management services. Our approach is to use Pegasus and Condor to provide these services.

For large applications it may be necessary to leverage a number of different clouds to achieve scientifically meaningful results in a reasonable amount of time. Provisioning re-

[1]http://www.cs.wisc.edu/condor/
[2]http://aws.amazon.com/ec2/
[3]http://open.eucalyptus.com/
[4]http://www.nimbusproject.org/
[5]http://opennebula.org/

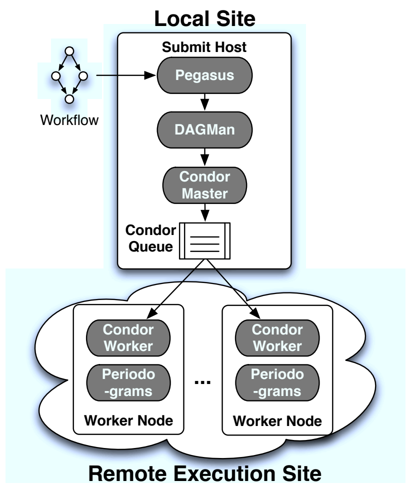

Figure 1: Execution Environment.

sources across different clouds (also referred to as *Sky Computing* [14]), provisions a distributed environment, where issues of remote job submission, data management, and security, among other things, need to be addressed.

In this paper we describe how we used Pegasus and Condor to execute an astronomy workflow on virtual machine resources drawn from multiple cloud infrastructures. Below we will describe how the application was deployed, evaluate its performance executing in different clouds (based on Nimbus, Eucalyptus, and EC2), describe the necessary system tuning needed to scale up the application and the system, and discuss the challenges of deploying and executing workflows in cloud environments. We also demonstrate how Pegasus was able to support sky computing by executing a single workflow across multiple cloud infrastructures simultaneously.

2. APPROACH

In our work we view the process of running an application in a distributed environment as a four-phase process:

1. provision the needed resources and configure the services,
2. map the workflow onto the resources,
3. run the application, and
4. de-provision the resources.

Steps 2 and 3 can also be iterative if the execution environment is dynamic.

2.1 Provisioning resources

In order to run on or across heterogeneous clouds, one needs to build virtual machine images that can run in these different environments. Usually, each cloud provides basic images that can be customized with additional software and services. One issue that comes up when building images for scientific applications is the need for adequate storage within the VM. In addition to the OS and application software, this storage is used to hold a subset of input, intermediate, and output data that are operated on by workflow jobs running inside the VM. In the case of data-intensive workflows, this storage often needs to be on the order of hundreds of gigabytes. If this storage is not available, applications need to be able to perform remote I/O, a shared file system needs to be deployed across the VMs, or the workflow management system needs to manage the storage as the application is progressing, performing careful staging and cleanup operations to ensure that the workflow footprint does not exceed the available storage space. In most of our work we first provision the necessary resources using client tools provided by the various clouds. This provisioning process can be carried out by hand (as we did in this paper), or by automated tools such as Nimbus Context Broker [13], Wrangler [9] and others. These tools also help configure the environment with services necessary for remote job submission, data transfers, job scheduling, etc. For our workflows, the goal of the provisioning step is to construct a virtual Condor pool, where the VMs act as Condor worker nodes and report to a Condor Master node that runs on a *submit host (SH)* outside the cloud. The submit host also runs Pegasus. Figure 1 shows the basic deployment of the system.

It is possible to place the submit host within the cloud, however, there are disadvantages to this approach. For example, the network address for the Condor Master would be dynamic and would need to be discovered by the user and by the Condor worker nodes. Additionally, the user needs to have a machine from which the provisioning commands can be executed. This machine can also be used for hosting the services required by Pegasus and Condor.

2.2 Mapping workflows onto cloud resources

Pegasus maps the resource-independent, user-provided, workflow description (*abstract workflow*) onto the available resources. The abstract workflow is expressed as a DAG (directed acyclic graph) where nodes represent computational tasks. These tasks describe the computations to be performed in logical terms (logical name, logical input and output data files). Edges in the DAG represent dependencies between the nodes. The workflow can be constructed hierarchically, with nodes representing entire sub-workflows.

Pegasus maps abstract workflows onto the available resources to generate an executable workflow. The mapping involves several steps, including:

1. pruning nodes from the abstract workflow which would produce outputs that already exist
2. transforming abstract workflow nodes into executable jobs that can be executed on remote resources by identifying specific executables, environment variables, input and output file paths, etc.
3. adding tasks to create execution directories on remote resources
4. adding tasks to stage data and executables into and out of the execution site
5. adding tasks to delete temp. files from execution sites
6. adding tasks to register output files and meta-data in information catalogs
7. performing workflow-level performance optimizations.

Once the executable workflow is constructed, Pegasus can cluster several computational tasks together into a single batch job [18]. Such clustering is often helpful in the case where individual tasks are of short duration and may encounter large overheads during execution. These overheads include scheduling delays in Condor and DAGMan, the latency of sending jobs over the network to a remote site for execution, and long wait times in remote scheduling queues.

When running on heterogeneous grids, Pegasus schedules tasks onto specific sites and ensures that input data is staged before the computations are scheduled onto the resources. For the application described in this paper, Pegasus maps the workflow onto a virtual Condor pool that does not use a shared file system. Instead of adding transfer jobs to the executable workflow, Pegasus uses Condor's file transfer features to transfer input and output files for each job. Thus step 4 of the mapping process is omitted.

2.3 Executing the workflow

DAGMan, the workflow engine used by Pegasus WMS, manages dependencies in the executable workflow and releases jobs into a Condor queue when their dependencies have been successfully executed (Figure 1). Condor sends these jobs to the available Condor workers. In addition to Pegasus, DAGMan and Condor also provide a number of failure recovery mechanisms.

2.4 De-provision resources

After the workflow finishes execution, the virtual resources are released either manually, using the cloud client interfaces, or using automated tools.

3. RELATED WORK

Researchers have been advocating for the use of virtual machines for scientific computing for some time [1][12][5][7]. This early work describes the potential benefits of cloud computing in the sciences, but does not relate the practical issues and challenges of running science applications in the cloud.

Much research has been done to quantify the performance of virtual machines and cloud platforms for scientific applications [15][16][22][23][24]. These efforts have focused mainly on micro benchmarks and benchmark suites such as the NAS parallel benchmarks[6] and the HPC Challenge[7]. These studies are good sources of information about the performance limitations of cloud platforms, but they do not convey the overall experience of running real applications in the cloud.

A few studies have investigated the performance of real science applications on clouds [8][4], but with few exceptions these studies have focused on tightly-coupled MPI applications, which are sensitive to network latency – an area in which clouds have struggled to compete with HPC systems. In comparison, our work focuses on scientific workflows, which are loosely coupled parallel applications that are more amenable to the relatively low-performance networks used in commercial and academic clouds.

Vecchiola, et al. have conducted research on workflow applications in the cloud [19]. They ran an fMRI workflow on Amazon EC2 using S3 for storage, compared the performance to Grid'5000, and analyzed the cost on different numbers of nodes. In our own previous work we have studied the cost and performance of workflows in the cloud via simulation [2], using an experimental Nimbus cloud [6], using individual EC2 nodes [11], and using a variety of different intermediate storage systems [10].

[6]http://www.nas.nasa.gov/Resources/Software/npb.html
[7]http://icl.cs.utk.edu/hpcc/.

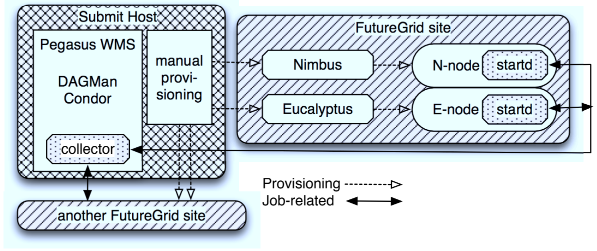

Figure 2: Provisioning Architecture in FutureGrid.

In this study we relate the practical experience of trying to run a real scientific workflow application on an experimental cloud test bed that is dedicated to science applications. In contrast to previous workflow studies, our application represents a non-trivial amount of computation performed over many days. This allows us to evaluate the scalability of our application on the cloud as well as the performance and stability of the cloud over time. In addition, this work was conducted on a distributed cloud test bed, the FutureGrid, which enables us to evaluate our application across different computing sites using different cloud management systems.

4. ENVIRONMENT SETUP

In this section, we describe the experimental cloud setup, the workflow application, and the testbed we chose for the experiments.

Figure 2 shows the provisioning architecture that was used for the cloud setup. The cross-hatched box shows the submit host (SH), an entity located at the USC Information Sciences Institute (ISI). The diagonally hatched resources are examples of FutureGrid remote sites at various places across the country (California, Midwest, Texas).

The dashed lines with white arrows indicate resource provisioning. In our experiments, we manually allocated resources, typically out of band, before running the workflow. Manual provisioning permits us to dynamically adjust resources, i.e. adding or subtracting resources. The provisioning uses either the Nimbus or Eucalyptus clients to allocate resources and start Virtual Machines (VMs). In its current form, FutureGrid has a fixed total number of resources allocated to each management infrastructure, varying from site to site, as shown in Table 1.

Our base virtual image contains CentOS 5.5 upon which the latest Pegasus WMS runs. Its boot configuration starts a Condor *startd* daemon on the resource. Part of this boot process dynamically retrieves the latest configuration to report back to the Condor *collector* at ISI. The solid lines with black arrows show the Condor-I/O between remotely started VMs and the SH.

Once the resource properly reports back to the Condor *collector* on the SH, the remote resource becomes available to execute Condor jobs. Jobs are scheduled by Condor itself and dependencies are managed by DAGMan, all according to a plan that is generated by Pegasus.

The file staging in this scenario depends on Condor-I/O. We are working to include more efficient means of staging data and support for encrypted data channels, i.e. GridFTP.

While this paper focuses on our experiments in the FutureGrid testbed, a very similar deployment architecture was

Resource	CPUs	Eucalyptus	Nimbus
IU *india*	1,024 2.9GHz Xeon	400	-
UofC *hotel*	512 2.9GHz Xeon	-	336
UCSD *sierra*	672 2.5GHz Xeon	144	160
UFl *foxtrot*	256 2.3GHz Xeon	-	248
Total	3,136	544	744

Table 1: FutureGrid Available Nimbus- and Eucalyptus Resources in November 2010.

used both on the NERSC Magellan cloud, and on Amazon EC2. The differences include the client software used to provision the resources, and the format of the VM images.

4.1 Periodograms

The workflow application that this paper focuses on has been developed to process astronomy data collected by the Kepler satellite[8]. Kepler is a NASA mission that uses high-precision photometry to search for transiting exoplanets around main sequence stars. In May 2009, the Kepler team began a photometric transit survey of 170,000 stars that has a nominal mission lifetime of 3.5 years. By the end of 2010, the project had released light curves of 210,664 stars. These light curves, which record the brightness of a star over time, contain measurements made over 229 days, with between 500 and 50,000 epochs per light curve.

Analyzing light curves to identify periodic signals, such as those that arise from transiting planets and from stellar variability, requires the calculation of periodograms, which reveal periodic signals in time-series data and estimate their significance.

Generating periodograms is a computationally-intensive process that requires high-performance, distributed computing. We have developed a periodogram service, written in C, that implements three common algorithms and that takes advantage of the "brute force" nature of periodograms. The processing of each frequency sampled in a periodogram is performed independently of all other frequencies, and so periodogram calculations are easily performed in parallel. Examining all 210K light-curves using the 3 periodogram algorithms would otherwise require approximately 41 days of sequential computation. In order to manage the parallelization process in a scalable, efficient manner, we have developed a workflow application for computing periodograms and used it to generate an initial atlas of periodograms for the light curves released by the Kepler mission.

4.2 FutureGrid

The FutureGrid project[9] is designed to enable researchers to address the computer science research challenges related to the cloud computing systems. These include topics such as authentication, authorization, scheduling, virtualization, middleware design, interface design, and cybersecurity in cloud platforms, as well as the optimization of grid- and cloud-enabled applications from a variety of domain sciences. FutureGrid provides novel computing capabilities designed to enable reproducible experiments while simultaneously supporting dynamic provisioning [20].

[8]http://kepler.nasa.gov/

[9]https://portal.futuregrid.org/about

The FutureGrid testbed makes it possible for researchers to conduct computer science experiments at scale in a customizable environment. It includes a geographically distributed set of heterogeneous computing systems, a data management system, and a dedicated network. It supports virtual machine-based environments, as well as native operating systems for experiments aimed at minimizing overhead and maximizing performance. Project participants integrate existing open-source software packages to create an easy-to-use software environment that supports the instantiation, execution and recording of grid and cloud computing experiments.

Using FutureGrid researchers are able to measure the overhead of cloud technology by requesting linked experiments on both virtual and bare-metal systems. FutureGrid enables US scientists to develop and test new approaches to parallel, grid and cloud computing, and compare and collaborate with international efforts in this area. The FutureGrid project provides an experimental platform that accommodates batch, grid and cloud computing, allowing researchers to attack a range of research questions associated with optimizing, integrating and scheduling the different service models. These are only examples of possible uses of FutureGrid, however such a testbed can host a wide variety of computer science and domain science research.

Table 1 shows the locations and available resources of five clusters at four FutureGrid sites across the US in November of 2010 (note that the sum of cores is larger than the sum of the remaining columns due to cores used for management and for other purposes like *bare metal* scheduling using Moab). We used the *Eucalyptus* and *Nimbus* resources, and constrained our resource usage to roughly a quarter of the available resources in order to allow other users to proceed with their experiments.

5. EXPERIMENTS

For the full periodograms atlas we processed all 210K light-curves using 3 different algorithms. For this paper, we applied only the most computationally intensive algorithm *plavchan* [17]. In some cases we also used a smaller, 16K light curve subset of the data.

The workflow generator, the component that creates the abstract workflow that is given to the Pegasus planner, estimates the runtime of each light curve based on the number of data points, the algorithm being used, and the parameters supplied for the algorithm. Each computational task is categorized according to its estimated runtime as being:

very fast sub-second runtime,
fast runtime in seconds, or
slow runtime about an hour.

Based on our experiences with different application workflows [18], we decided to cluster the workflow tasks in order to reduce both the scheduling overhead and the total number of Condor jobs. The majority of the experiments used a simple numeric clustering scheme where very fast tasks were clustered into groups of 100 tasks per job, fast tasks were clustered into groups of 20 tasks per job and slow tasks were not clustered at all. For the experiments in section 5.2 we specified a target runtime for the jobs of 60 minutes and clustered the tasks based on their estimated runtime using the simple first-fit bin packing algorithm.

Site	RAM Swap	Cores per VM	Requ. Res.	Avail. Hosts	Actv. Hosts	Jobs	Tasks	Cumul. Dur.
Euca *india*	6GB (0.5GB)	2	30	30	8	19	1,900	0.4 h
Euca *sierra*	6GB (0.5GB)	2	30	29	28	162	7,080	119.2 h
Nimbus *sierra*	2GB (0.0GB)	2	20	20	20	140	7,134	86.8 h
Nimbus *foxtrot*	2GB (0.0GB)	2	20	20	17	126	6,678	77.5 h
Nimbus *hotel*	2GB (0.0GB)	2	50	50	50	352	10,290	250.6 h
Total			150	149	123	799	33,082	534.5 h

Table 2: Statistics about Sites, Jobs and Tasks.

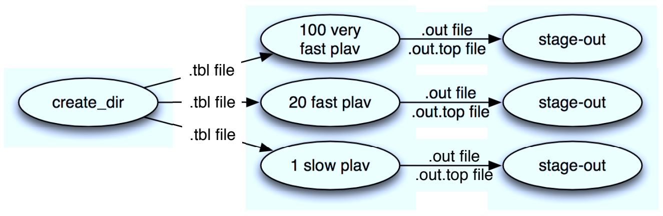

Figure 3: Workflow stub examining light-curves using the *plavchan* algorithm.

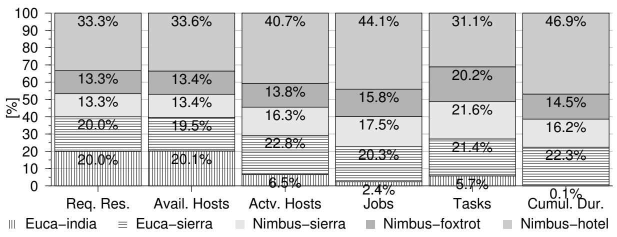

Figure 4: Table 2 represented as percentages.

A simplified version of the executable workflow generated by Pegasus is shown in Figure 3. The initial root job `create_dir` is an auxiliary job created by Pegasus to create a temporary directory on the submit host where outputs can be staged. Following the create_dir job are many branches containing a periodogram computation followed by a stage-out job. Although Figure 3 only shows one branch per category, in the actual workflow there could be many branches per category depending on the total number of light curves to examine. The periodogram jobs are where the majority of time is spent. The `stage-out` jobs associated with each branch execute on the submit machine to move files between directories on the submit host. Stage-out jobs have only a small contribution to the total runtime.

Input and output data for the workflow reside on the submit host. For these experiments we used a machine at ISI in Marina del Rey, CA as the submit host. Condor-I/O was used to stage input files before executing jobs on remote resources, and to retrieve output files after each job finishes.

The input data was pre-compressed using gzip and uncompressed on the remote resource before executing the computation. The outputs generated were compressed on the remote resource before being staged back to the submit host. Compressing the data resulted in space savings of more than 80 percent. This compression not only reduces the transfer time, but also reduces the cost of data transfers to commercial clouds such as Amazon EC2.

5.1 Experiments on FutureGrid

This section describes two experiments. In the first experiment, we ran 33K light curves on FutureGrid. This experiment computed periodograms for the 16K light curve dataset twice, for a total of around 33K periodograms. The second experiment ran the entire 210K light curve dataset on FutureGrid using an improved software configuration based on our experiences running the smaller dataset.

5.1.1 33k Periodograms

The 33K workflow contained 799 parallel branches as illustrated in Figure 3, for a total of 1,599 Condor jobs, including 1 setup job, 799 computational jobs, and 799 staging jobs. The workflow included 217 *very fast* branches, 572 *fast* branches, and 10 *slow* branches.

Column *requested resources* (Req. Res.) in Figure 4 and Table 2 shows the number of resources requested from the FutureGrid testbed before starting the workflow. Automatic provisioning, including de-provisioning, is projected for the FutureGrid *Experiment Management* framework, but is not part of experiments shown in this paper.

The ratio between Nimbus and Eucalyptus resources was 3 to 2. One third of the resources were requested from site *hotel* (50 Nimbus nodes), one third from site *sierra* (20 Eucalyptus + 30 Nimbus), and the final third shared between sites *foxtrot* (20 Nimbus) and *india* (30 Eucalyptus). Only site *sierra* was running both Nimbus and Eucalyptus. Running across multiple cloud management systems in separate administrative domains–effectively using multiple clouds–is an example of *Sky Computing* [14].

Table 2 shows the runtime information collected from the workflow execution. All jobs were invoked using the kickstart wrapper [21]. Kickstart invokes the native application code, measures its runtime performance, and records execution environment information.

Comparing the columns *requested resources* and *available hosts* in Table 2 (and Figure 4), one will realize that not all provisioned resources reported back properly. The discrepancy in this case is that only 29 of the 30 resources requested from the *sierra* Eucalyptus site were able to start. This failure was no problem for our execution environment, which can handle variable numbers of resources without requiring configuration changes.

The column *active hosts* in Table 2 shows the number of hosts from each site that actively participated in the workflow. Not all hosts that were requested and reported back to the submit host were able to be utilized by the Condor

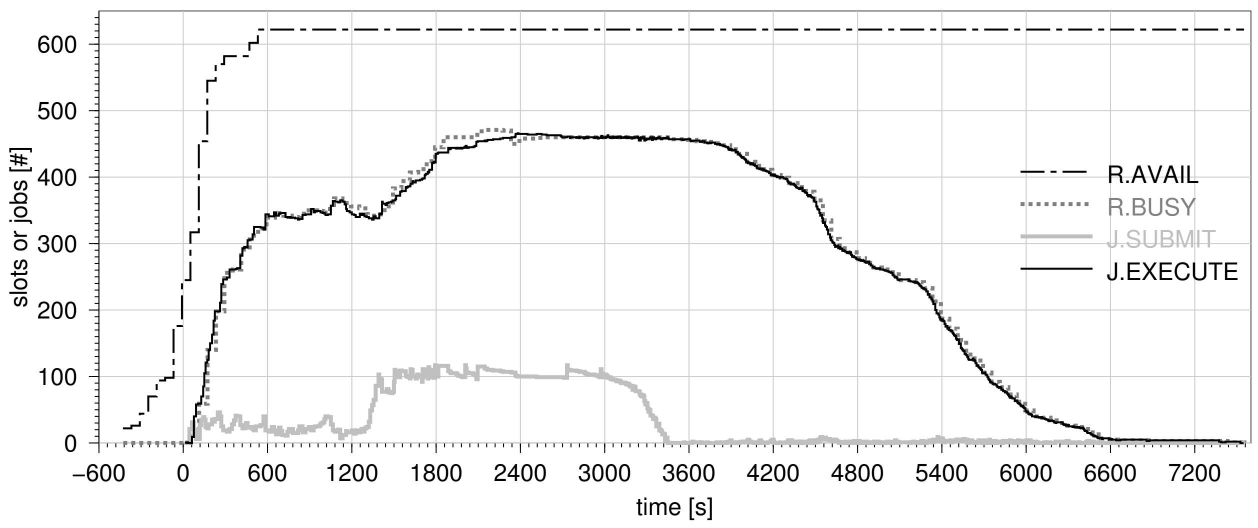

Figure 5: Resources and Job States of 2×16k Light-Curves over Time.

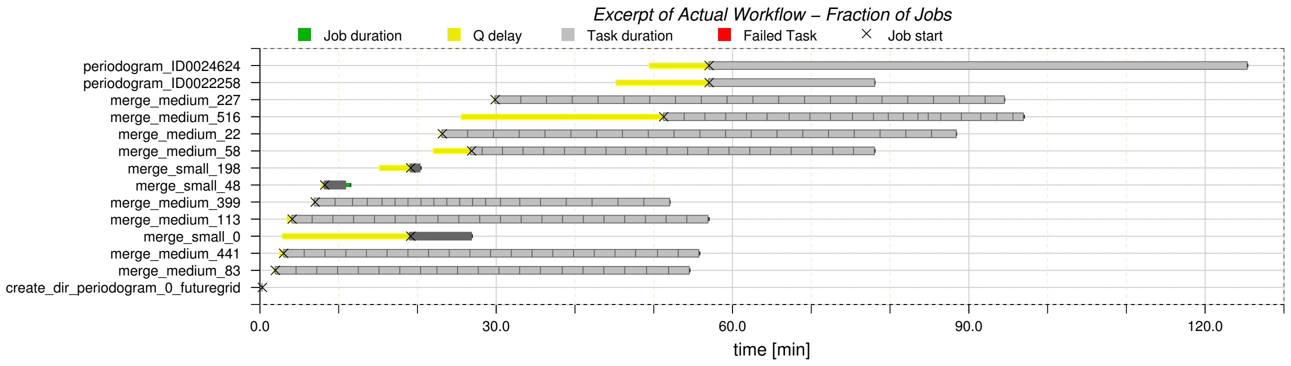

Figure 6: Fraction of Jobs over Time.

scheduler. As Figure 5 illustrates, we were unable to fully saturate the given resources with jobs due to inefficiencies in our initial configuration that limited the throughput of our scheduler.

The distribution of jobs and tasks across the sites is shown by the columns *jobs* and *tasks*. With the exception of site *india*, which had only 8 of 30 hosts active, we were able to get a good balance of jobs/tasks across all the clouds with about 6-7 jobs completed per host per cloud.

The total workflow wall time duration (the elapsed clock time from the start of the workflow to the end) amounted to 7,541 seconds, or a little over two hours. Table 2 shows that the cumulative sequential runtime of the computational tasks amounts to over 534 hours. This is equivalent to a speed-up of 254.8. Although the actual number of physical cores in our virtual pool was 298, we were over-subscribing the pool to create over 600 job slots. Considering the actual number of CPU cores our observed speedup was good.

Figure 5 shows the resource utilization and jobs over time. The x-axis origin was placed at the start of execution of the DAGMan workflow. However, since resources were provisioned prior to workflow execution, the x-axis starts in the negative area, 10 minutes before the workflow started.

The first curve *R.AVAIL* in Figure 5 shows the slots as resources come online and report back to the Condor *collector* on the SH. The number of slots top out at 622 resources, of which 596 were remote FutureGrid resources. Of the remaining 26 slots, 10 are attributed to the submit host, and 16 slots to 4 experimental VMs reporting from elsewhere, which did not participate in the experiment. Each slot can receive one Condor job.

The curve *R.BUSY* shows the slots marked as "busy" by Condor. We oversubscribed the resources so that each core had two slots, for a total of 4 slots per 2-core node.

The curve *J.EXECUTE* shows the Condor jobs marked as "running" in the Condor queue. This curve closely mirrors the *R.BUSY* curve, as it should. The two curves provide the same data point but from a different perspective, one from the view point of resources, and one from the view point of jobs.

The curve *J.SUBMIT* shows the number of Condor jobs marked as "idle" in its queue. We were carefully tailoring the

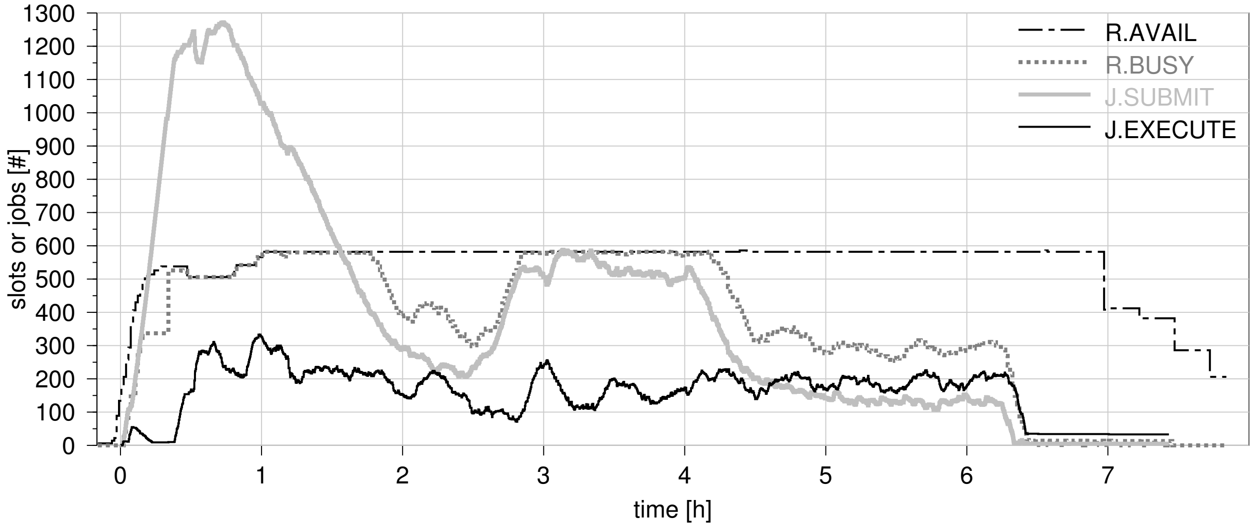

Figure 7: Resources and Job States of 210k Light-curves over Time.

performance configuration of DAGMan to not release anymore "ready" jobs into the Condor queue when the amount of "idle" jobs exceeded 100 jobs. This allows Condor to have enough jobs to schedule to idle resources without leaving thousands of jobs in the queue that would only increase the time spent matchmaking.

Finally, Figure 6 shows a *small subset* of jobs in a Gantt chart for illustrative purposed. A complete Gantt chart for this workflow would have a y-axis showing 1,599 jobs. The chart shows the `create_dir` job at the bottom and left-most, 3 *very fast* jobs `merge_small*`, 8 *fast* jobs `merge_medium*`, and 2 jobs from the *slow* category `perio*`. It does not show any staging jobs. It does show, however, that the separate staging jobs only contribute negligibly to the workflow duration, because the right boundary is determined by the end of the workflow.

The yellow bar (solid color) preceding each job shows the time the job spent in the Condor queue before being matched to a resource. The gray boxes show the start and end times of each task, as observed by `kickstart` [21]. In case of *very fast* tasks, these are 100 extremely small, overlapping boxes. The cross marks the point in time when the task started executing according to Condor.

5.1.2 Scaling Up

After collecting experience running 16k sets of light-curves in the previous experiments, we scaled up the workflow to experiment with periodograms from all 210k light-curves. The scalability issues required a delicate balance between the ability to saturate remote resources without overloading the submit host with too many transfers/connections.

Figure 5 helped us start tuning Condor for performance to improve the resource saturation and to examine other system components. Based on the initial experiments, we came up with the following guidelines for scaling up the system.

- **Workflow Optimizations:** Pegasus' clustering should be used for short-running jobs as it is useful for significantly reducing the communication overhead for workflow tasks. Files to be staged should be compressed because it provides benefits in shorter transfer times and lower costs on commercial clouds.

- **Submit host Unix settings:** To scale-up, the number of file descriptors available to Condor should be raised. Also, a sufficiently large port range needs to be assigned to Condor to facilitate its communications.

- **Submit host Condor configuration:** This configuration controls how Condor daemons interact, i.e. to use TCP for the *collector*, delays between *negotiator* cycles, socket cache sizes, file descriptors per daemon, and port ranges. The latter are limited resources, and careful tuning is required for large scale operations.

  We configured the submit host to use the Condor *shared port daemon*, an entity to reduce the number of Internet ports used by Condor.

- **Condor DAGMan configuration:** DAGMan settings control how many jobs it will submit in a row, and to prefer depth-first search over breadth-first search to traverse the workflow graph. DAGMan runtime settings give us a runtime ceiling to limit the number of jobs DAGMan releases to Condor. We found the *idle* limit more useful than the job count ceiling because it takes into account the number of jobs that Condor needs to manage at any time rather than having an ever growing Condor queue.

- **Remote host Condor configuration:** This configuration controls how the Condor started on the VM is configured. Most of these settings are similar to the submit host settings.

  We activated the Condor *connection broker* (CCB) on the remote entity to enable VMs with a private network address to be able to talk to the submit host. As a by-product, it also reduces the number of Internet

Site	CPU	RAM Swap	Workflow Walltime	Cumul. Duration	Speed Up
Magellan	8 x 2.67GHz Nehalem	19GB (0.0GB)	5.2 h	226.6 h	43.6
Amazon	8 x 2.3GHz Xeon	7GB (0.0GB)	7.2 h	295.8 h	41.1
FutureGrid	8 x 2.5GHz Xeon	29GB (0.5GB)	5.7 h	248.0 h	43.5

Table 3: Comparison of 6 node (48 core) runs.

Site	Req. Res.	Actv. Hosts	Jobs	Tasks	Cumul. Dur.
Euca *india*	30	26	1,135	28,761	102.3
Euca *sierra*	30	29	1,074	24,600	125.8
Nimbus *sierra*	20	20	810	21,540	85.5
Nimbus *foxtrot*	20	20	1,428	34,480	164.1
Nimbus *hotel*	50	50	4,012	101,283	356.5
Total	150	145	8,459	210,664	834.2

Table 4: Scaling up to 210k periodograms.

ports required, and thus the number of file descriptors on the submit host.

- **Condor slots:** They should equal the physical cores available for computationally intensive tasks like the periodogram experiments.

We ran a number of large-scale experiments. Here we present the results of one run. When running at scale, both system and application failures occur. In the case of the run presented here, 99.8% of jobs completed successfully, while 20 jobs failed without possibility of automatic recovery, possibly due to problems with the data or the application. In this run on FutureGrid, we were still over-subscribing the CPU resources during the experiment.

Figure 7 shows the workflow execution. As before, we requested the same set of resources as shown in the first data column of Table 4. However, this time only 145 hosts reported back to our submit host. Five Eucalyptus resources did not start as shown in column *active hosts* in Table 4.

This means the curve *R.AVAIL* in Figure 7 tops out at 582 slots, including 6 slots for the submit machine. Curve *R.BUSY* shows the number of slots that Condor thinks are busy running jobs.

Unlike before, the *J.EXECUTE* curve in Figure 7 does not follow the *R.BUSY* curve. The *R.BUSY* curve was obtained from the Condor logs while the *J.EXECUTE* curve was computed from the state transition of each Condor job as reported by DAGMan's debug log. A machine will reach state "busy" before the job is started, and remains so for a short while after the job exits.

Condor-I/O staging in of data files will make a job appear to be running in the Condor queue, but not in the DAGMan information we tracked. Condor-I/O staging out will let the job remain in the running state until staging is finished. Scalability constraints limit the number of simultaneous Condor-I/O streams, which will have a dampening effect on the number of short-running jobs.

Taking all effects into consideration still fails to explain the large discrepancy between the two curves, however, we trust *R.BUSY* to be the correct measurement. Based on that, we were able to saturate the remote resources with jobs from Condor's resource perspective.

Curve *J.SUBMIT* shows an interesting outlier. We thought we had tuned the number of idle jobs for DAGMan, but the constraint was not propagated. This caused the submitted jobs to grow in an unbounded way. About half an hour into the workflow, after a manual configuration update was signaled to DAGMan, we saw a drop in the number of submitted jobs. We suspect this signal to be partly responsible for problems determining the proper count of *J.EXECUTE*.

Running the workflow through Pegasus provides us with retries in case of job failures. However, after a finite number of failing retries, it is better to give up than keep going. A *rescue* workflow comprises the jobs to be run – after human intervention.

Due to failed jobs in the workflow, for which we do not have runtimes, and which artificially extend the workflow's wall time, it does not make sense to compute a speed-up. However, the approximate speed-up is in line with the low *J.EXECUTE* graph, and thus tells us that we have not found the right balance in our tunings.

We were also able to successfully run the 210K dataset in a grid environment using configurations not shown in this paper. We used more conservative settings in that environment.

5.2 Cloud Comparison

We also investigated the performance of the 16K light curve workflow on different cloud infrastructures including academic and commercial clouds. To facilitate the comparison, we used Eucalyptus (with it's EC2 interface) and Amazon's EC2. We ran the 16k light curve sample set on other cloud infrastructure:

1. NERSC's Magellan cloud (Eucalyptus),
2. Amazon's cloud (EC2), and
3. FutureGrid's *sierra* cloud (Eucalyptus).

The setup is identical to the one shown in Figure 2. The resources are either Eucalyptus or EC2. In these separate experiments, the number of nodes and core-selection was constrained, because in the Amazon case, the computations have a monetary cost associated with them.

The experiments allocated 6 nodes with 8 cores each in all cases. Table 5 shows the various *reported* machine attributes. Due to differences in virtually every aspect, differences on the host systems notwithstanding, the workflow task durations differ for each major cloud, despite the identical setup.

This experiment uses one Condor slot per physical CPU. Given that 48 physical cores were available, a speed-up over 43 for two experiments is good.

Table 3 reports the fluctuations in runtime of the various tasks on each site. The columns *qt* designate a quartile. The column *CD* is the cumulative duration of tasks. It is interesting to see how close the cumulative runtimes are to each other. The longer runtime on the Amazon cloud can

Host	**Jobs**	**Tasks**	**CD** [h]	**min** [s]	**qt1** [s]	**med** [s]	**qt3** [s]	**max** [s]	**avg** [s]	**stddev**
192.168.3.194	55	2,345	35.8	1.0	1.1	1.2	135.5	4,022.1	55.0	109.9
192.168.3.195	57	2,577	38.3	1.0	1.1	3.5	135.3	143.7	53.5	65.1
192.168.3.196	62	2,778	39.1	1.0	1.0	1.1	135.2	4,033.2	50.7	99.3
192.168.3.197	61	2,904	37.4	1.0	1.0	1.1	135.1	144.0	46.4	63.4
192.168.3.198	63	2,965	38.8	1.0	1.0	1.8	135.1	4,018.0	47.1	96.3
192.168.3.199	62	2,972	37.1	1.0	1.0	3.5	134.9	4,020.5	44.9	95.7
TOTAL Magellan	360	16,541	226.6	1.0	1.0	1.2	135.2	4,033.2	49.3	89.8
10.125.81.3	59	2,419	52.4	1.3	1.4	4.7	178.6	192.0	78.0	86.9
10.125.81.43	62	2,816	49.7	1.3	1.4	4.4	171.3	5,614.1	63.6	136.0
10.125.81.166	61	3,179	46.8	1.3	1.3	1.5	169.8	5,535.2	53.0	125.5
10.125.82.156	57	2,347	50.3	1.3	1.3	1.6	178.3	5,571.8	77.1	142.9
10.125.82.197	59	2,708	49.4	1.3	1.3	1.5	171.5	5,580.9	65.7	135.2
10.125.82.233	62	3,072	47.2	1.3	1.3	1.4	170.3	189.5	55.3	81.3
TOTAL Amazon	360	16,541	295.8	1.3	1.3	1.6	172.7	5,614.1	64.4	120.4
10.0.4.130	61	2,794	41.6	1.1	1.2	3.9	149.2	4,362.7	53.6	107.4
10.0.4.131	63	3,017	41.4	1.1	1.2	1.2	149.2	155.4	49.3	69.2
10.0.4.132	60	2,734	40.8	1.1	1.2	1.4	149.2	4,394.4	53.8	110.9
10.0.4.133	53	2,250	41.4	1.2	1.2	3.9	149.4	4,390.5	66.2	116.7
10.0.4.134	58	2,557	42.0	1.2	1.2	3.9	149.4	156.8	59.1	72.1
10.0.4.135	65	3,189	40.8	1.2	1.2	1.2	149.0	4,413.3	46.1	102.4
TOTAL Sierra	360	16,541	248.0	1.1	1.2	1.3	149.2	4,413.3	54.0	97.8

Table 5: Comparison of runtimes on Magellan, Amazon and Sierra.

be explained by two factors: A lower CPU speed, and by poor WAN performance. The Magellan cloud, and FutureGrid *sierra* are both located in the same state as the submit machine, but the Amazon cloud is much further away.

If you subtract the submit host tasks from the total tasks per site, and recall that the first experiment from section 5.1 was running every periodogram from the same set twice, the number of tasks match up. This makes the cumulative durations comparable, if one assumes $\frac{534.5}{2} = 267.3$ hours, in line with the experiments in this section.

6. CONCLUSIONS AND FUTURE WORK

We have demonstrated that sky computing is a viable and effective solution for at least some scientific workflows. The micro benchmark differences between the cloud infrastructures do not have a major effect on the overall user experience, and the convenience of being able to add and remove resources at runtime outweighs the networking and system management overheads. Some of the guidelines and findings for scaling the setup, such as over subscription of CPU resources are broadly applicable. Other findings, such as Condor configuration and tuning, are specific to the use of Pegasus / Condor, but should still be considered valuable lessons learned and possibly applicable to similar setups using other tooling.

For further experiments, especially when employing periodograms as the target application, we would like to investigate a number of issues:

- Better utilization of remote resources: explore increasing the number of running jobs to saturate the given resources and the tradeoffs involved with communication overheads associated with remote job execution.
- Different clustering strategies: explore the benefits of different task cluster sizes.
- Submit host management: evaluate the tradeoffs between various system parameters and load on the submit host, which is related to the overall workflow performance.
- Alternative data staging mechanisms, explore different protocols, and storage solutions.

The above research will be greatly facilitated by computer science testbeds such as FutureGrid, which aims to provide a customizable environment that can be repeatedly instantiated in the same form over time.

7. ACKNOWLEDGMENTS

This material is based upon work supported in part by the National Science Foundation under Grant No. 0910812 to Indiana University for "FutureGrid: An Experimental, High-Performance Grid testbed." Partners in the FutureGrid project include U. Chicago, U. Florida, San Diego Supercomputer Center - UC San Diego, U. Southern California, U. Texas at Austin, U. Tennessee at Knoxville, U. of Virginia, Purdue I., and T.-U. Dresden.

G. B. Berriman is supported by the NASA Exoplanet Science Institute at the Infrared Processing and Analysis Center, operated by the California Institute of Technology in coordination with the Jet Propulsion Laboratory (JPL).

8. REFERENCES

[1] J. S. Chase, D. E. Irwin, L. E. Grit, J. D. Moore, and S. E. Sprenkle. Dynamic virtual clusters in a grid site manager. In *In Proceedings of the Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, 2003.

[2] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: the Montage example. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, SC '08, pages 50:1–50:12, Piscataway, NJ, USA, 2008. IEEE Press.

[3] E. Deelman, G. Singh, M.-H. Su, J. Blythe, and Y. e. a. Gil. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Sci. Program.*, 13:219–237, July 2005.

[4] C. Evangelinos and C. Hill. Cloud Computing for Parallel Scientific HPC Applications: Feasibility of Running Coupled Atmosphere-Ocean Climate Models on Amazon's EC2. In *Proceedings of the first workshop on cloud Computing and its Applications (CCA08)*, 2008.

[5] R. J. Figueiredo, P. A. Dinda, and J. A. B. Fortes. A Case For Grid Computing On Virtual Machines. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*, ICDCS '03, pages 550–, Washington, DC, USA, 2003. IEEE Computer Society.

[6] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, et al. On the Use of Cloud Computing for Scientific Workflows. In *Proceedings of the 2008 Fourth IEEE International Conference on eScience*, pages 640–645, Washington, DC, USA, 2008. IEEE Computer Society.

[7] W. Huang, J. Liu, B. Abali, and D. K. Panda. A case for high performance computing with virtual machines. In *Proceedings of the 20th annual international conference on Supercomputing*, ICS '06, pages 125–134, New York, NY, USA, 2006. ACM.

[8] K. R. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, et al. Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud. In *CloudCom*, pages 159–168. IEEE, 2010.

[9] G. Juve and E. Deelman. Wrangler: Virtual Cluster Provisioning for the Cloud. In *HPDC*, 2011.

[10] G. Juve, E. Deelman, K. Vahi, and G. Mehta. Data Sharing Options for Scientific Workflows on Amazon EC2. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–9, Washington, DC, USA, 2010. IEEE Computer Society.

[11] G. Juve, E. Deelman, K. Vahi, G. Mehta, et al. Scientific workflow applications on amazon ec2. In *E-Science Workshops, 2009 5th IEEE International Conference on*, pages 59 –66, Dec. 2009.

[12] K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual workspaces: Achieving quality of service and quality of life in the Grid. *Sci. Program.*, 13:265–275, October 2005.

[13] K. Keahey and T. Freeman. Contextualization: Providing One-Click Virtual Clusters. In *eScience, 2008. eScience '08. IEEE Fourth International Conference on*, pages 301 –308, dec. 2008.

[14] K. Keahey, M. Tsugawa, A. Matsunaga, and J. Fortes. Sky Computing. *IEEE Internet Computing*, 13:43–51, 2009.

[15] J. Napper and P. Bientinesi. Can cloud computing reach the top500? In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, UCHPC-MAW '09, pages 17–20, New York, NY, USA, 2009. ACM.

[16] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, et al. A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing. In *1st International Conference on Cloud Computing*. ICST Press, 2009.

[17] P. Plavchan, M. Jura, J. D. Kirkpatrick, R. M. Cutri, and S. C. Gallagher. Near-Infrared Variability in the 2MASS Calibration Fields: A Search for Planetary Transit Candidates. *The Astrophysical Journal Supplement Series*, 175(1):191, 2008.

[18] G. Singh et al. Workflow task clustering for best effort systems with Pegasus. In *Proceedings of the 15th ACM Mardi Gras conference.*, MG '08, pages 9:1–9:8, New York, NY, USA, 2008. ACM.

[19] C. Vecchiola, S. Pandey, and R. Buyya. High-Performance Cloud Computing: A View of Scientific Applications. In *International Symposium on Parallel Architectures, Algorithms, and Networks*, 2009.

[20] G. von Laszewski, G. C. Fox, F. Wang, A. Younge, A. Kulshrestha, et al. Design of the FutureGrid Experiment Management Framework. In *The International Conference for High Performance Computing, Networking, Storage and Analysis (SC10)*, pages 1–10, New Orleans, LA, 11/2010 2010. IEEE.

[21] J. Vöckler, G. Mehta, Y. Zhao, E. Deelman, and M. Wilde. Kickstarting Remote Applications. The 2nd Workshop on Grid Computing Environments (OGCE06) [unpublished], 11 2006.

[22] E. Walker. Benchmarking Amazon EC2 for High-Performance Scientific Computing. In *USENIX Login*, pages 18–23, 2008.

[23] L. Youseff, R. Wolski, B. Gorda, and R. Krintz. Paravirtualization for HPC Systems. In *In Proc. Workshop on Xen in High-Performance Cluster and Grid Computing*, pages 474–486. Springer, 2006.

[24] W. Yu and J. S. Vetter. Xen-Based HPC: A Parallel I/O Perspective. In *"Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid"*, pages 154–161, Washington, DC, USA, 2008. IEEE Computer Society.