2015 IEEE 11th International Conference on eScience

Cost-Aware Cloud Provisioning

Ryan Chard*, Kyle Chard†, Kris Bubendorfer*, Lukasz Lacinski†, Ravi Madduri† and Ian Foster†
*School of Engineering and Computer Science, Victoria University of Wellington
†Computation Institute, University of Chicago & Argonne National Laboratory

***Abstract*—Cloud computing is often suggested as a low-cost and scalable model for executing and scaling scientific analyses. However, while the benefits of cloud computing are frequently touted, there are inherent technical challenges associated with scaling execution efficiently and cost-effectively. We describe here a cost-aware elastic provisioner designed to dynamically and cost-effectively provision cloud infrastructure based on the requirements of user-submitted scientific workflows. Our provisioner is used in the Globus Galaxies platform—a Software-as-a-Service provider of scientific analysis capabilities using commercial cloud infrastructure. Using workloads from production usage of this platform we investigate the performance of our provisioner in terms of cost, spot instance termination rate, and execution time. We demonstrate cost savings across six production gateways of up to 95% and 12% improvement in total execution time when compared to a worst case scenario using a single instance type in a single availability zone.**

I. Introduction

The lure of cost-effective elastic computing capacity has resulted in a move towards the hosting of e-Science applications on commercial clouds [1]. However, the use of commercial clouds does not inevitably result in cost-effective and efficient computing–various aspects, such as instance types, availability zones, and pricing models, must be considered when provisioning resources [2]. In fact, naïve resource provisioning can often result in inefficient and expensive execution [3].

We describe here a multi-faceted cost-aware provisioner that we have developed to dynamically and cost-effectively provision cloud resources in the Globus Galaxies platform. The Globus Galaxies platform [4] is a cloud-based scientific workflow system that is currently used by more than 300 researchers across 30 institutions. This platform enables the deployment of cloud-hosted Software-as-a-Service (SaaS) science gateways. Users of Globus Galaxies-based gateways are presented with an integrated suite of tools that make it easy to run complex scientific workflows on large datasets. The Globus Galaxies platform is designed to operate on Amazon Web Services (AWS) Elastic Compute Cloud (EC2); it provides for the automatic, on-demand provisioning of resources for workflow execution. It underlies, in particular, Globus Genomics [5], a service which facilitates the development and execution of highly customized bioinformatics workflows and is currently used by over 200 users across 23 active gateways with 4.2 million compute hours in 2014. The platform is also used in medical imaging [6], cosmology [7], materials science, and medical research [8].

The Globus Galaxies platform, like other workflow systems, executes user defined workflows composed of various tools. Upon execution workflows are decomposed into a series of computational jobs, each with varying resource requirements (e.g., compute, storage, memory). The cost-aware provisioner aims to dynamically create new cloud instances on which these jobs are executed while minimizing cost and/or execution time. Our approach builds upon tool profiles that describe the requirements of individual workflow jobs, leverages different cloud computing acquisition and pricing models (e.g., spot and on-demand instances), considers real-time pricing information across instance types and availability zones, migrates spot requests to satisfy waiting jobs, and can use on-demand instances when spot requests are not fulfilled within a customizable period of time.

We demonstrate, via analysis of actual usage of the cost-aware provisioner in various Globus Galaxies-based gateways, that our approach provides substantial reductions in execution cost and increases in execution performance when compared to a naïve provisioning approach that simply acquires on-demand instances. By expanding the scope of instance types and availability zones considered we achieve a substantial reduction in cost. In addition, we show that our approach can lower spot instance termination rates and at the same time decrease execution time for individual jobs. Collectively, these capabilities provide considerable improvements to the cost and performance efficiency of the Globus Galaxies platform.

The rest of this paper is structured as follows. We present related work in Section II and describe the Globus Galaxies platform in Section III. In Section IV we analyze production usage of the platform before presenting the cost-aware provisioner in Section V. In Section VI we describe the approach taken to gather representative usage data. In Section VII we evaluate the provisioner using real-world scientific workloads. Finally we summarize our contributions in Section VIII.

II. Related Work

Commercial clouds are now used to host many scientific computing applications [9], [10] and gateways [11], [12]. Although it is often shown that clouds can effectively scale to host scientific workflows, the methods used to efficiently acquire and maintain resources are frequently overlooked.

Voorsluys and Buyya [13] propose a resource allocation policy to reliably execute compute-intensive applications over pools of EC2 spot instances. They present three approaches to improve fault tolerance: checkpointing, task duplication, and migration. They also investigate various provisioning strategies, such as utilizing the highest and lowest bid price, and show substantial improvements in cost and reliability. We

978-1-4673-9325-6/15 $31.00 © 2015 IEEE
DOI 10.1109/eScience.2015.67

extend this approach by investigating the effect of using multiple instance types and availability zones in conjunction with workload profiles to reduce the monetary cost of computing workloads.

Andrzejak et al. [14] investigate the effect of using different bid prices on EC2. The authors present a probabilistic decision model to select bid prices for specific resources. They use configurable parameters to tune the balance of reliability and monetary savings, producing levels of confidence in the ability to meet quality of service objectives and deadlines. In this paper we focus on using workload profiles and on-demand provisioning in addition to spot requests in order to reduce cost and improve the reliability of execution.

Mao and Humphrey [15] present a cost-aware auto-scaling scheduler to meet application deadlines in cloud workflows. They compare their scheduling approach with two utility computing cost-based approaches and demonstrate cost-savings of 8%-40% and improved utilization over other approaches. They primarily focus on optimizing cost based on deadline constraints. Our work compliments this approach by evaluating the effect of different provisioning approaches on cost.

Malawski et al. [16] investigate deadline-constrained and cost-minimization provisioning. They compare five provisioning algorithms, using priority, static, and dynamic resource selection approaches. Using a cloud workflow simulator they evaluate these provisioning approaches and show improved performance and reduced delays and failures, they also investigate the ability to accurately estimate run times. In comparison with our approach, they base cost efficiency on resource utilization rather than the operation of provisioned resources.

III. Globus Galaxies platform

The Globus Galaxies platform enables the creation of SaaS-based scientific gateways that can be easily customized to meet the needs of different domains. The platform includes: Galaxy [17], for the construction, execution, and management of parallel end-to-end workflows; Globus [18] for reliable and secure big data transfers; Globus Nexus [19] to support integrated identity management, authentication, and access control; the Swift parallel scripting language [20] for the parallelization of individual workflow components; and finally—the focus of this article—a cost-aware elastic cloud provisioner, to dynamically provision cloud infrastructure on demand. The high level architecture is shown in Fig. 1.

We briefly describe in the following the Galaxy system; the Globus Galaxies platform's use of cloud infrastructure; and our elastic provisioning model.

A. Galaxy

Galaxy [17] is an end-to-end workflow engine developed for large-scale, interactive, bioinformatics research. It simplifies the creation of complex workflows and their execution on various compute resources. Like most scientific workflow systems, Galaxy allows arbitrary tools to be added to the platform by "wrapping" them to describe input and output

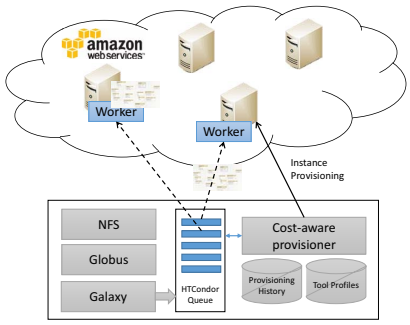


Fig. 1: Globus Galaxies platform architecture.

parameters, as well as their respective data types. Workflows can then be created by linking tools.

Galaxy provides a set of job runners to enable the execution of workflows (and their tools) over different infrastructures. Job runners abstract computational resources from Galaxy. A number of job runners are provided for common HPC schedulers. However, domain-specific job runners can also be developed and used to deploy workloads to unsupported compute environments. The Globus Galaxies platform uses the HTCondor [21] scheduler to allocate workflows across computational resources. Worker nodes—provisioned on cloud resources—are configured to connect to the HTCondor pool in order to obtain jobs.

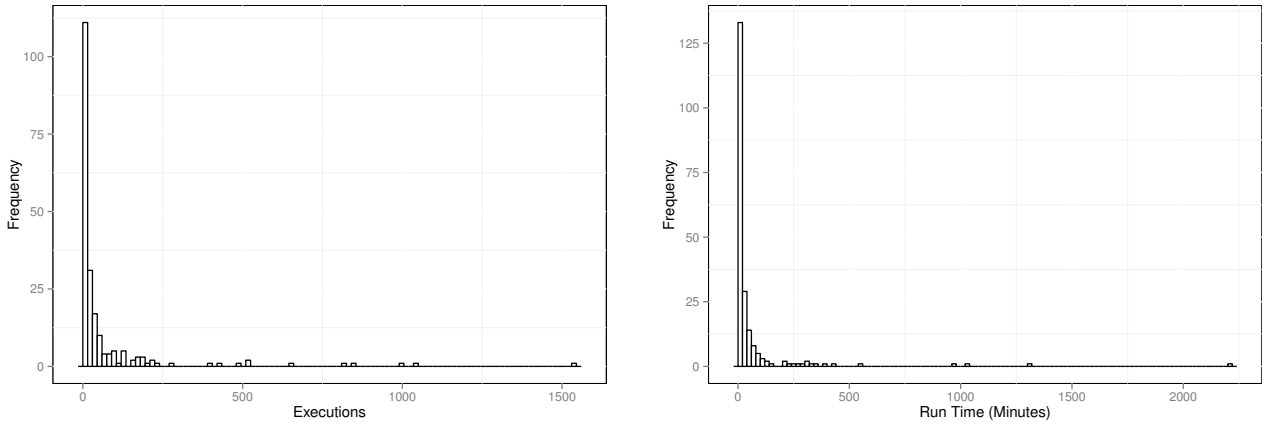
B. Cloud Integration

The Globus Galaxies platform is designed to be deployed on the cloud. Each gateway deployment relies on a single virtual machine—the *head node*—responsible for operating the Galaxy Web service, Galaxy database, NFS shared file system, Globus endpoint, and HTCondor master. The gateway restricts which jobs can be run on the head node, allowing only simple data transfer jobs to be executed locally. Computationally demanding jobs are sent to HTCondor and are queued while waiting for resources. Cloud instances are dynamically provisioned to fulfill the requirements of idle jobs waiting in the HTCondor queue. HTCondor supports the specification of job requirements, such as CPU and memory demands, which are used to assist instance type selection.

Depending on the gateway, cloud instances are provisioned using a predefined Amazon Machine Image (AMI) which includes a number of pre-installed tools (e.g., bioinformatics tools) or by dynamically configuring a bare-bone image with appropriate software using CloudInit.

C. Resource Provisioning

Cloud computing providers typically offer various instance types and acquisition models. For example, EC2 includes over 28 instance types and supports three acquisition models. Instance types encapsulate a set of resources including CPU, memory, storage and networking capacity. Instance types are designed to meet different usage requirements and present



(a) tool execution

(b) job execution time

Fig. 2: Tool usage frequency distribution.

users with flexibility to select appropriate resource combinations. Acquisition models provide different prices for the same instance types based on different quality of service guarantees.

The acquisition models supported by EC2 are: on-demand, reserved and spot. On-demand instances have a static advertised hourly cost and are readily available for fulfillment. Reserved instances provide a reduced hourly rate in exchange for long term reservations, upfront payment and predicted usage requirements. Spot instances provide the most cost-effective provisioning method by allowing reservations at a fluctuating price based on demand (which is often substantially lower than on-demand and reserved prices). To obtain a spot instance, users bid a maximum price at which they are willing to pay for the instance. If the bid price is exceeded during the lifetime of the instance, the instance is terminated and any work executing is lost. Due to the volatile nature of spot prices, the spot price can sometimes exceed the on-demand price for short, or even prolonged, periods of time. The demand for different instance types also changes over time. Instances with high demand, such as the m2.4xlarge type, frequently see high spot price volatility. Thus, spot instances lack the reliability and predictability of on-demand and reserved instances.

The Globus Galaxies platform permits the use of different instance types and acquisition models. We leverage this capability in our cost-aware provisioner to minimize cloud computing costs and application run times, by requesting different instance types using different acquisition models.

IV. Globus Genomics platform usage

To reach a better understanding of (prior to building the provisioner) the properties of workflows and their tools, we analyzed the usage data from six production Globus Genomics gateways. We focused on Globus Genomics gateways as the Globus Genomics application is the longest running application using the Globus Galaxies platform and therefore the richest source of usage data. In the analysis we integrated usage data with Galaxy logs and HTCondor execution traces to find tool execution frequencies and execution times. Over 308 days, we collected data on more than 14,500 completed jobs and 25,500 hours of execution.

A. Tool Usage

Globus Genomics provides its users with access to several thousand tools. In many cases, several tools are provided that perform the same (or a comparable) task. Each has advantages and disadvantages in different settings and are selected based on user preferences when constructing workflows. The frequency of number of executions and execution time are shown in Fig. 2a and Fig. 2b, respectively. These figures illustrate the long tail nature of tool execution in Globus Genomics. A small number of tools are executed many times while many are executed only a small number of times. Of the 212 tools analyzed, the 10 most frequently executed account for 52.3% of all executions, while the 20 most frequently executed account for 68.1%. Similarly, many tools execute for short periods of time, with 75 tools executing for less than one minute on average, while a small number execute for long periods of time—over 2200 minutes (35 hours) on average.

B. Tool Requirements

Each tool offered by Globus Genomics has different memory and compute requirements, which may furthermore vary with input datasets and tool settings. It is important to understand these requirements as they directly impact the required cloud instance types. Rather than consistently launching large instance types—capable of fulfilling all jobs—we analyzed individual tool requirements to determine the optimal instance type.

To model tool requirements we have constructed profiles for a set of prominent, frequently run, tools as a three-tuple of compute, memory, and disk requirements. This information

TABLE I: Tool execution requirements

Tool	Executions	Avg Run Time	CPUs	Memory	Disk	Instance
fastq_groomer_parallel	1047	5:48:25	32	High	High	r3.4xlarge
bwa_mem	994	4:34:49	32	High	High	r3.4xlarge
fastqc	820	0:34:53	4	Medium	Medium	c3.2xlarge
rgPicardMarkDups	659	0:43:53	32	High	Medium	r3.4xlarge
picard_ARRG	512	0:27:13	32	Medium	Low	r3.4xlarge
sickle	511	1:56:27	4	Medium	High	m2.4xlarge
samtools_flagstat	489	0:21:20	4	Low	Low	c3.2xlarge
picard_SortSam	421	0:15:26	8	Medium	Low	m2.4xlarge
picard_BuildBamIndex	394	0:23:12	4	Low	Low	c3.2xlarge
bowtie2	273	0:15:11	32	Medium	Low	r3.4xlarge

was collected by executing tools with different input datasets to determine accurate instance type requirements.

Table I shows, for ten commonly used tools, both the tool profile and the smallest instance type that can be used to fulfill requirements. These simple profiles highlight the range of requirements of commonly used tools, with execution time ranging from 15 minutes to almost 6 hours, number of CPUs from 4 to 32, and memory and disk requirements from low and high.

Using these tool profiles we are able to specify approximate requirements when submitting jobs to the HTCondor queue using ClassAds [22]. This information is in turn used by the cost-aware provisioner to restrict the set of eligible instances to those that can satisfy job requirements. In the case of small jobs it increases the search scope and enables a wider range of instances to be considered which can decrease cost and improve efficiency. It also allows several instances of the same tool to be run concurrently on the same large instance.

V. Cost-Aware Provisioning

Our cost-aware elastic provisioner combines job requirements with real time cloud resource pricing to cost-effectively provision cloud resources. Once instances are requested, a record of the request is stored in an Amazon Relational Database Service (RDS) instance to enable further optimizations to cost and execution time.

Algorithm 1 outlines our algorithm used to provision cloud instances [2]. The algorithm periodically monitors an HTCondor queue for unallocated idle jobs whose wait time exceeds a pre-defined value. It filters and ranks viable instance types (based on profiles) from each availability zone to select the most cost-effective instance for a job. A timeout threshold prevents starvation by acquiring on-demand instances when spot requests are not readily fulfilled. Spot request migration aims to reduce overall execution time by reusing unneeded spot requests. The following sections describe the key features of the provisioner.

Algorithm 1 EC2 Instance Provisioning

```
1:  timeout = /* configurable */
2:  threshold = /* configurable */
3:  while true do
4:      /* periodically run */
5:      idleJobs = jobs in HTCondor queue
6:      for job in idleJobs do
7:          if job not yet allocated then
8:              if job queue time > timeout then
9:                  launch on-demand instance
10:                 cancel or migrate outstanding spot requests for job
11:             else
12:                 eligibleIns = instance types that meet job profile
13:                 for instance in eligibleIns do
14:                     onDemandP = on-demand price for instance
15:                     minZoneSpotP = min zone spot price for instance
16:                     if minZoneSpotP > threshold × onDemandP then
17:                         instancePrice = onDemandP
18:                     else
19:                         instancePrice = minZoneSpotP
20:                     end if
21:                 end for
22:                 sort eligibleIns by instancePrice
23:                 select instance with lowest instancePrice
24:                 launch instance
25:             end if
26:         else
27:             cancel or migrate outstanding spot requests for job
28:         end if
29:     end for
30: end while
```

A. Selecting Viable Instance Types

Before submitting instance requests the provisioner first consults predefined job profiles to determine which instance types can satisfy the requirements of the job. Generally, we have found that the use of profiles greatly increases the number of instance types that can be potentially used to host a job. It therefore increases the potential for cost reduction as additional instance types can be surveyed. We currently support 14 different instance types and restrict selection based on compute, memory, and disk requirements. In the absence of a job profile, the provisioner uses a system-wide default instance type that satisfies the requirements of all jobs.

B. Cost-Aware Instance Selection

Having identified a set of suitable instances, the provisioner requests real-time spot pricing information using AWS APIs. Spot prices are collected for each viable instance type from each availability zone. The resulting prices are compared with published on-demand prices for each instance type to determine the cheapest real-time price. Depending on predefined policies (such as the maximum time to wait for a spot request) for each gateway, the cheapest instance type and availability zone is generally selected. It may therefore

select a faster instance than is necessary if the cost is lower than the best match. Individual gateways are also configured with maximum bid prices for each instance type ensuring that potential costs can be limited at the expense of execution time. As some applications are more sensitive to termination and gateway administrators may have different levels of risk aversion the provisioner can be pre-configured with default bidding policies that determine what acquisition model to use: on-demand, spot, or a combination of both. Further work is required to optimally determine bids based on current and historical spot pricing information. This information may also be used to predict termination risk associated with spot instances.

C. Reverting to On-Demand Instances

Demand for spot instances and thus their prices can fluctuate over time. Indeed, popular instance types can sometimes have much higher spot prices than their on-demand equivalents. For example, the price of the m2.4xlarge instance type frequently exceeds $6, more than six times the on-demand price of $0.98, behavior that we attribute to legacy applications that were configured to use this instance type when it was first introduced and have not yet been migrated to newer instance types. As the spot price increases, so too does the time required to fulfill for spot requests, especially if the bid price is lower than the current spot price. Moreover, during volatile periods, the risk of having instances terminated is much higher. In response to these challenges, the cost-aware provisioner includes functionality to revert to using on-demand instances when spot prices begin to increase (based on analysis of spot price history). The decision to revert to on-demand instances is based on a pre-defined wait time threshold. If spot requests exceed this threshold, the cost-aware provisioner will request the cheapest on-demand instance capable of satisfying the job's requirements. The threshold is configurable for each gateway and is typically set to 20 minutes. This strategy can reduce the cost, improve the execution time, and minimize the risk of having instances terminated during execution.

D. Over-provisioning Instance Requests

We do not know, when making a spot request, whether and when the request will be fulfilled. The cost-aware provisioner includes functionality to configure how frequently new spot requests are made and how many different instance types are requested simultaneously for an individual job. This strategy allows multiple instances of different types to be requested for each job (either immediately or over regular intervals). The reason for this approach is that spot requests can take considerable time to be fulfilled and it is often faster to request a different instance type while waiting for a request to be fulfilled. Once a request is fulfilled, the provisioner either cancels outstanding requests (if possible) or uses them to satisfy requirements of other jobs waiting to be allocated.

In future work we aim to investigate the use of what we call *permissive instance provisioning*, whereby we make a spot request for each instance type that meets the requirements in each availability zone immediately after a job is queued. Through continuous probing to evaluate the state of spot requests, the first fulfilled request can initiate the termination of unnecessary requests.

E. Instance Request Migration

Once an instance has been provisioned and a job has been allocated, unnecessary spot requests can either be terminated or used to satisfy other idle jobs. In situations with many jobs and high throughput, instance request migration presents an opportunity to improve the overall execution time. That is, rather than canceling requests, the provisioner can assign unnecessary requests to other waiting jobs, assuming that job requirements are met.

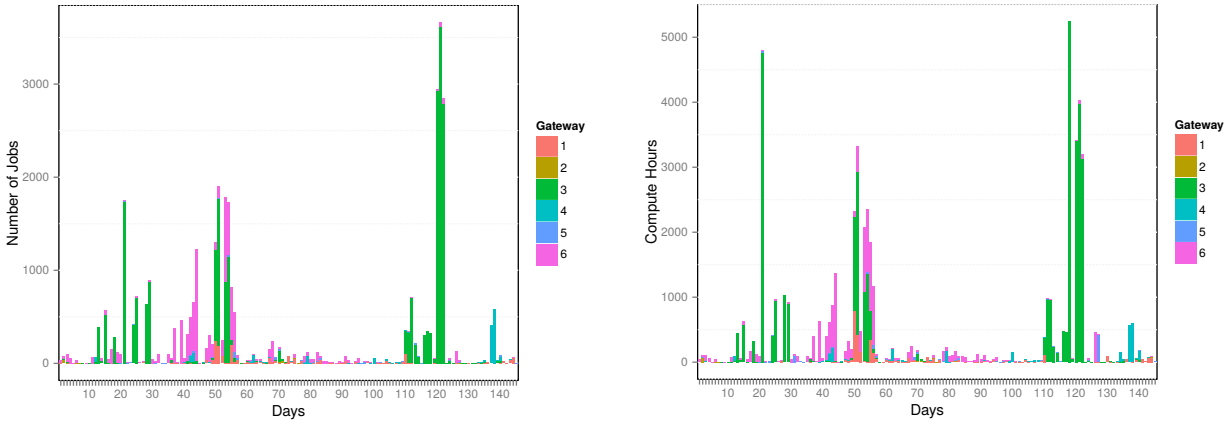
VI. Data Collection

The fact that the Globus Galaxies platform is used extensively for large-scale computing puts us in a unique position to gather usage information on which to base our evaluation. We describe here the methods by which this information is recorded and integrated to establish a representative workload.

We instrumented the cost-aware provisioner to record information about provisioning decisions, such as the selected instance type and availability zone, in a shared RDS database. The provisioner logs, for each request, when the request was made, when the request was satisfied, and when any new instance(s) are started. We store this information alongside Galaxy records for each execution, including the tool, input datasets, parameters, submitting user, and the identifier (condor ID) assigned to the task by HTCondor. HTCondor records, for each job that is submitted, its identifier, queue time, execution time, and AWS instance used.

We use the Galaxy database, HTCondor logs, and shared RDS provisioner database to compute the cost of every job that has has been executed by a Globus Galaxies gateway. The AWS API provides historic spot price for each EC2 instance and availability zone for the previous 90 days. We use that information to calculate the cost of each instance for each hour that it operates, both during the actual execution and during other hypothetical counter-factual provisioning schedules. Using the unique identifier assigned to every job and HTCondor logs we map every job to the instance it was executed on and compute the cost of executing the job. As HTCondor supports a multi-job "slot" model we divide the calculated cost across jobs that share a single instance.

We collected usage data from six Globus Genomics gateways over a period of 145 days to provide a representative production workload for evaluating the cost-aware provisioner. Fig. 3a shows the number of jobs executed and Fig. 3b the total compute time consumed by each gateway over this period. The difference between the two figures shows the distinct usage characteristics of the gateways. For example, it is evident that gateway 1 is primarily used for long running jobs: it has higher compute hours than job executions. Within this period there have been over 35,000 job executions requiring over 2,200 days of compute time. These figures highlight the



(a) number of jobs executed

(b) compute hours used

Fig. 3: Number of jobs and compute hours used per day from six Globus Genomics Gateways over a 145 day sample period

sporadic nature of Globus Genomics usage, where individual gateways are often used extensively for a period of days or weeks at a time (such as gateway 3). The workload includes both infrequently used and frequently used Globus Genomics gateways to evaluate the provisioner across a range of scenarios.

VII. Evaluation

Our evaluation investigates the use of the cost-aware provisioner in terms of execution time and cost when executing scientific workflows in the cloud. To better evaluate the performance of the provisioner, we compare the effect of broadening the search scope in terms of instance types and availability zones. The scopes considered are:

- **single instance, single availability zone (SI-SAZ)**: A baseline approach in which the provisioner requests a single instance type (m2.4xlarge) in a single availability zone (us-east-1c).
- **multi-instance, single availability zone (MI-SAZ)**: The provisioner requests different instance types in a single availability zone (us-east-1c) and selects the instance with the lowest price.
- **single instance, multi-availability zone (SI-MAZ)**: The provisioner requests a single instance type (m2.4xlarge) across all availability zones and selects the instance with the lowest price.
- **multi-instance, multi-availability zone (MI-MAZ)**: The provisioner requests different instance types across all availability zones and selects the instance type with the lowest price.

Based on analysis of the spot history for various instance types over the past three months we use a bid price of $6.56 to avoid termination and establish a worst-case scenario for each of the provisioning scopes. While individual gateways may use lower bid prices this will result in some reduction in cost at the expense of execution delays incurred while waiting for resources to be provisioned. It is important to note that users are not charged their bid price but rather the spot price at the time of use. The following results apply this same method by re-calculating spot prices every hour.

A. Cost

The total projected cumulative cost for each search scope across six production Globus Genomics gateways over a 145 day period is shown in Fig. 4. The figure shows, using a naïve approach that considers only a single instance and availability zone (SI-SAZ), costs over $27,000. Whereas, expanding the scope to multiple instance types (MI-SAZ) reduces the cost to approximately $2,250. Overall the potential savings are $25,486.75, or 92.1% from the worst-case SI-SAZ to the MI-MAZ scope. In comparison to the naïve approach, incorporating multiple availability zones provides a 15.8% improvement. Unexpectedly, the use of multiple availability zones and instance types (MI-MAZ) provides little gain over simply using multiple instance types (MI-SAZ). The results show that even simple improvements can substantially reduce the cost of executing scientific workloads in the cloud.

Fig. 5 shows the cumulative cost for each of the six Globus Genomics gateways. The figure shows the total cost when using the naïve, SI-SAZ, approach (solid) as well as the reduced cost achieved by the cost-aware, MI-MAZ, provisioner (dashed) over the 145 day period. The results show that substantial reductions in cost can be achieved for each of the gateways irrespective of their workload requirements. A breakdown of each of the provisioning scopes for each gateway is presented in Table II. This shows, on average, that an individual gateway can reduce its operating costs by 75.9% (Max 95.0%, Min 43.0%) when using multiple instance types and availability zones over the baseline SI-SAZ approach.

B. Spot Instance Termination

Applying real spot price traces (obtained from AWS APIs) to each of the provisioning scopes enables analysis of when

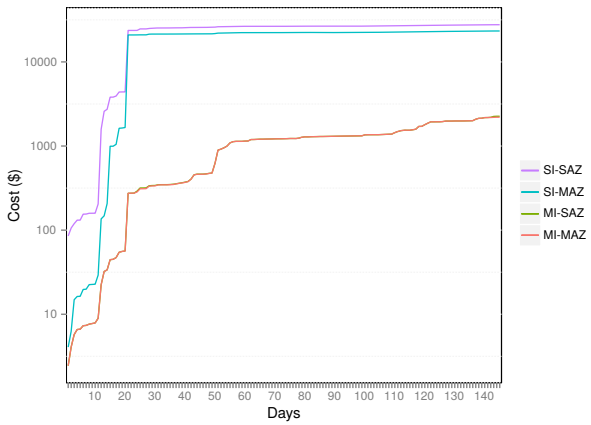


Fig. 4: The total cumulative cost of operating six Globus Genomics gateways with different provisioner search scopes over a 145 day period on a logarithmic scale.

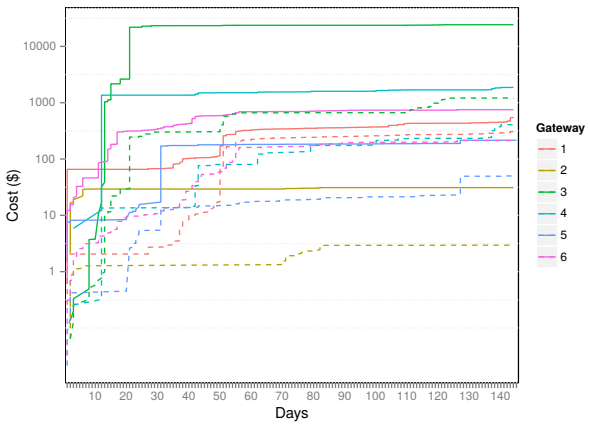


Fig. 5: The cumulative cost of the naïve SI-SAZ (solid line) and MI-MAZ (dashed line) search scopes for each of the six Globus Genomics gateways over a 145 day period on a logarithmic scale.

TABLE II: Total 145 day cost comparison between gateways

Gateway	SI-SAZ	SI-MAZ	MI-SAZ	MI-MAZ
1	551.38	341.20	371.57	314.16
2	31.08	4.86	2.95	2.95
3	24269.89	22080.71	1219.03	1213.44
4	1867.33	499.92	410.07	406.55
5	216.08	55.00	46.48	50.08
6	751.25	339.59	209.26	213.05
Total	**27686.99**	**23321.29**	**2259.36**	**2200.24**

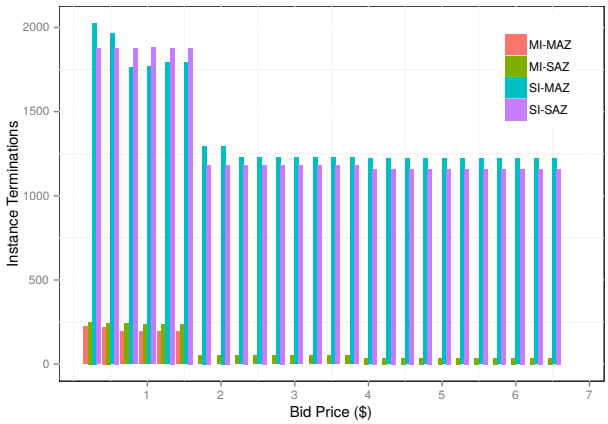


Fig. 6: The number of spot instance terminations for each search scope with $0.25 bid price increments.

the spot price exceeds the bid price and therefore when an instance will be terminated. Fig. 6 shows the frequency of instance termination for each of the provisioning scopes with varying bid prices (in $0.25 increments). The results show that the single-instance scopes (utilizing m2.4xlarge instances) are highly susceptible to instance termination, as demand for a specific instance type will frequently effect the spot price in all availability zones. The figure also demonstrates a lack of sensitivity to bid prices as the rate of termination does not change significantly with different bid prices. This implies that when an instance type experiences an increase in demand (which increases the spot price) it will often increase substantially. Thus, when configuring bid prices, we can define them to be high (to avoid termination) or low if we can easily migrate workloads. Similarly, when provisioning new instances we should use on-demand pricing if the spot price increases as it is likely to increase significantly.

C. Reverting to On-demand Instances

As shown in Fig. 6 there is potential for spot instances to be terminated. Our cost-aware provisioner is configured to revert to on-demand instances in some situations to minimize execution time. However, when reverting to on-demand instances there is a trade-off between execution time and cost. This trade-off is shown in Fig. 7 and Fig. 8. These figures show the total cost and execution time when four timeout values (5, 10, 15, and 20 minutes) are used and the search scope is set to MI-MAZ. The timeout value specifies how long a job is allowed to wait for a spot instance before reverting to an on-demand instance. The results indicate that short timeout periods can improve overall execution time at the expense of substantially increasing the operating cost of a gateway (a 5 minute timeout results in increased cost over eight times the MI-MAZ price). The substantial increase in cost is reflective of the competitive spot pricing market in which spot prices across availability zones rarely remain higher than on-demand prices. We also evaluated the effect of timeouts using the SI-SAZ scope and found that the total cost was lower when a 5 minute timeout value was used. This was due to a prolonged increased spot price in the selected availability zone and a lack of alternative availability zones in which to provision new instances.

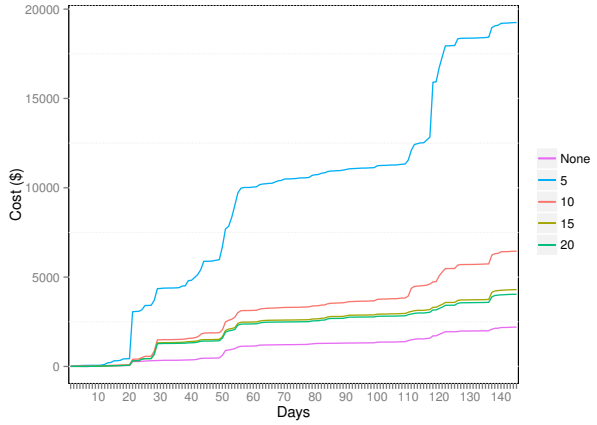


Fig. 7: Cumulative cost when using on-demand instances with different timeout periods using MI-MAZ scope.

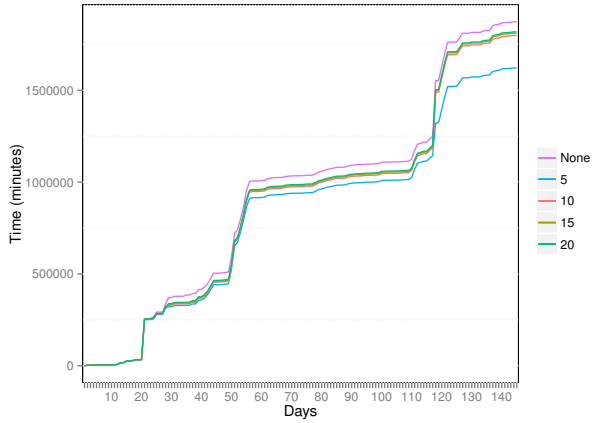


Fig. 8: Cumulative execution time when using on-demand instances with different timeout periods using MI-MAZ scope.

D. Production Usage

To evaluate the performance of the cost-aware provisioner in its entirety we compare the performance of three live production Globus Genomics gateways with and without it. Without the provisioner we employ a simple approach that acquires an m2.4xlarge instance in a single availability zone. This provisioner operates periodically by polling the HTCondor queue to find idle jobs. If the job can not be scheduled to idle resources, the provisioner acquires an instance to fulfill the workload. With the cost-aware provisioner deployed we select the most cost-effective instance type in any availability zone using request migration and the ability to revert to on-demand instances when needed. In both cases we record the length of time that jobs must wait before being allocated to a provisioned resource. We only include jobs that wait for a resource to be acquired rather than those that may be allocated to existing resources. We use logs from a 168 day period where the simple provisioner was used and then over the subsequent 80 days we analyze the cost-aware provisioner. In total 1775 jobs are allocated with the simple provisioner, and 2302 are allocated using the cost-aware provisioner. Using the simple provisioner jobs wait, on average, 650 seconds before being executed. When using the cost-aware provisioner the average wait time is reduced to 570 seconds, a reduction of 80 seconds per job. This presents a 12.3% reduction in the time spent waiting for an instance to be provisioned. While some of this improvement may be attributed to different workloads and economic conditions at the time of analysis it may also be due to the ability to migrate requests. A comparison of the two workloads in Table III shows that the average submit and execution time is relatively consistent.

TABLE III: Jobs allocated with and without the provisioner

Provisioner	Jobs	Avg Wait	Avg Exec	Avg Submit
Simple	1775	0:10:50	3:54:07	13:38:59
Cost-aware	2302	0:09:30	3:37:01	13:58:05

VIII. Conclusion

We described the multi-faceted, cost-aware, provisioner used by the Globus Galaxies platform to elastically provision cloud resources on demand. We demonstrated, via analysis of production logs from six Globus Genomics gateways, that the cost-aware provisioner was able to reduce cost, execution time, and spot instance terminations.

Our results show that cost-aware provisioning can result in substantial economic advantages over naïve provisioning approaches. Increasing the provisioner search scope to consider additional instances and availability zones results in a 92.1% reduction in overall cost between the worst-case SI-SAZ and best-case MI-MAZ scopes. In addition, adaptive cost-aware provisioning approaches were shown to provide lower instance termination rates and decreased execution time. In the situation that the cost-aware provisioner is able to revert to on-demand instances when spot instances are difficult to acquire, we showed that while execution time is improved, it may do so at the expense of cost. Finally, based on live production deployment of the cost-aware scheduler, we observed that total execution time and wait times are reduced, potentially as a result of lower instance termination rates and request migration.

These results further motivate the value in establishing rich tool profiles. We therefore aim to further develop profiles to guide and restrict the selection of instance types for different workloads. In addition to the use of compute, memory, and disk requirements, we also intend to examine the role that IO requirements and network performance can have when selecting instance placement, which we have previously shown to affect workload performance [23].

Acknowledgment

This work was supported in part by the U.S. Department of Energy under contract DE-AC02-06CH11357 and the NIH under contracts 1U54EB020406-01 Big Data for Discovery

Science Center and R24HL085343 Cardiovascular Research Grid. We acknowledge generous research credits provided by Amazon Web Services that helped support our work. We also appreciate the support of the Globus and Galaxy teams.

REFERENCES

[1] D. Lifka, I. Foster, S. Mehringer, M. Parashar, P. Redfern, C. Stewart, and S. Tuecke, "XSEDE cloud survey report," Technical report, National Science Foundation, USA, Tech. Rep., 2013.

[2] R. Chard, K. Chard, K. Bubendorfer, L. Lacinski, R. Madduri, and I. Foster, "Cost-aware elastic cloud provisioning for scientific workloads," in *Proceedings of the 8th IEEE International Conference on Cloud Computing (CLOUD)*, 2015.

[3] S. Yi, A. Andrzejak, and D. Kondo, "Monetary cost-aware checkpointing and migration on Amazon cloud spot instances," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 512–524, 2012.

[4] R. Madduri, K. Chard, R. Chard, L. Lacinski, A. Rodriguez, D. Sulakhe, D. Kelly, U. Dave, and I. Foster, "The Globus Galaxies platform: delivering science gateways as a service," *Concurrency and Computation: Practice and Experience*, vol. Preprint, pp. –, 2015.

[5] R. K. Madduri, D. Sulakhe, L. Lacinski, B. Liu, A. Rodriguez, K. Chard, U. J. Dave, and I. T. Foster, "Experiences building Globus Genomics: a next-generation sequencing analysis service using Galaxy, Globus, and Amazon Web Services," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 13, pp. 2266–2279, 2014.

[6] K. Chard, R. Madduri, X. Jiang, F. Dahi, M. W. Vannier, and I. Foster, "A cloud-based image analysis gateway for traumatic brain injury research," in *Proceedings of the 9th Gateway Computing Environments Workshop*, ser. GCE '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 13–16.

[7] R. Chard, S. Sehrish, A. Rodriguez, R. Madduri, T. D. Uram, M. Paterno, K. Heitmann, S. Cholia, J. Kowalkowski, and S. Habib, "PDACS: a portal for data analysis services for cosmological simulations," in *Proceedings of the 9th Gateway Computing Environments Workshop*. IEEE Press, 2014, pp. 30–33.

[8] R. Winslow, J. Saltz, I. Foster *et al.*, "The cardiovascular research grid (CVRG) project," *Proceedings of the AMIA Summit on Translational Bioinformatics*, pp. 77–81, 2011.

[9] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, and J. Good, "On the use of cloud computing for scientific workflows," in *Proceedings of the 4th International Conference on eScience*. IEEE, 2008, pp. 640–645.

[10] R. Chard, R. Madduri, N. Karonis, K. Chard, K. Duffin, C. Ordonez, T. Uram, J. Fleischauer, I. Foster, M. Papka, and J. Winans, "Scalable pCT image reconstruction delivered as a cloud service," *IEEE Transactions on Cloud Computing*, vol. Preprint, no. 99, pp. 1–1, 2015.

[11] E. Skidmore, S.-j. Kim, S. Kuchimanchi, S. Singaram, N. Merchant, and D. Stanzione, "iPlant atmosphere: A gateway to cloud infrastructure for the plant sciences," in *Proceedings of the 2011 ACM Workshop on Gateway Computing Environments*, ser. GCE '11. New York, NY, USA: ACM, 2011, pp. 59–64.

[12] L. K. Zentner, S. M. Clark, P. M. Smith, S. Shivarajapura, V. Farnsworth, K. P. Madhavan, and G. Klimeck, "Practical considerations in cloud utilization for the science gateway nanoHUB.org," in *Procceedings of the 4th IEEE International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2011, pp. 287–292.

[13] W. Voorsluys and R. Buyya, "Reliable provisioning of spot instances for compute-intensive applications," in *Proceedings of the 26th International Conference on Advanced Information Networking and Applications (AINA)*. IEEE, 2012, pp. 542–549.

[14] A. Andrzejak, D. Kondo, and S. Yi, "Decision model for cloud computing under SLA constraints," in *Proceedings of the International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 2010, pp. 257–266.

[15] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 49.

[16] M. Malawski, G. Juve, E. Deelman, and J. Nabrzyski, "Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 22:1–22:11.

[17] J. Goecks, A. Nekrutenko, J. Taylor *et al.*, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biol*, vol. 11, no. 8, p. R86, 2010.

[18] I. Foster, "Globus Online: Accelerating and democratizing science through cloud-based services," *Internet Computing, IEEE*, vol. 15, no. 3, pp. 70–73, May 2011.

[19] R. Ananthakrishnan, J. Bryan, K. Chard, I. Foster, T. Howe, M. Lidman, and S. Tuecke, "Globus Nexus: An identity, profile, and group management platform for science gateways and other collaborative science applications," in *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 2013, pp. 1–3.

[20] M. Wilde, I. Foster, K. Iskra, P. Beckman, Z. Zhang, A. Espinosa, M. Hategan, B. Clifford, and I. Raicu, "Parallel scripting for applications at the petascale and beyond," *Computer*, vol. 42, no. 11, pp. 50–60, 2009.

[21] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor-a hunter of idle workstations," in *Proceedings of the 8th International Conference on Distributed Computing Systems*. IEEE, 1988, pp. 104–111.

[22] M. Livny, J. Basney, R. Raman, and T. Tannenbaum, "Mechanisms for high throughput computing," *SPEEDUP journal*, vol. 11, no. 1, pp. 36–40, 1997.

[23] R. Chard, K. Bubendorfer, and B. Ng, "Network health and e-Science in commercial clouds," *Future Generation Computer Systems*, vol. Preprint, pp. –, 2015.