Cost-Aware Elastic Cloud Provisioning for Scientific Workloads

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 provides an efficient model to host and scale scientific applications. While cloud-based approaches can reduce costs as users pay only for the resources used, it is often challenging to scale execution both efficiently and cost-effectively. We describe here a cost-aware elastic cloud provisioner designed to elastically provision cloud infrastructure to execute analyses cost-effectively. The provisioner considers real-time spot instance prices across availability zones, leverages application profiles to optimize instance type selection, over-provisions resources to alleviate bottlenecks caused by oversubscribed instance types, and is capable of reverting to on-demand instances when spot prices exceed thresholds. We evaluate the usage of our cost-aware provisioner using four production scientific gateways and show that it can produce cost savings of up to 97.2% when compared to naïve provisioning approaches.**

I. Introduction

Scientific researchers routinely leverage commercial cloud computing providers as an alternative to using in-house resources [1]. The motivation for so doing is typically the potential for reduced infrastructure costs and support for elastic scalability, therefore improving both the scale and speed of scientific discovery. However, taking advantage of these features is non-trivial as users must provision resources while taking into account aspects such as instance types, availability zones, and pricing models. Naïve provisioning approaches can result in unnecessarily high costs [2].

We present here a multi-faceted cost-aware elastic resource provisioner that elastically provisions cloud computing resources. The provisioner monitors jobs queued for execution. When queue conditions meet predefined conditions, it requests and configures new cloud instances to execute waiting jobs. The provisioner is equipped to balance static and dynamic pricing models, such as on-demand and spot pricing supported by Amazon Web Services (AWS). On-demand instances are offered at a static price, whereas spot instances are offered at a fluctuating price that varies based on real-time demand. To increase the number of instance types considered the provisioner matches job requirements with suitable instance types. Under some situations the provisioner leverages on-demand instances to increase reliability and to avoid abnormally high spot prices. It also over-provisions instances to avoid bottlenecks associated with particular instance types; as this approach can sometimes result in unused instances, it also dynamically assigns unused instances to waiting jobs.

We have evaluated the provisioner with workloads from four production Globus Galaxies-based [3] gateways. Globus Galaxies is a cloud-based platform for creating Software-as-a-Service gateways. It combines the Galaxy [4] workflow engine, HTCondor [5] job submission system, Globus data management [6] capabilities, and AWS Elastic Compute Cloud (EC2) to provide an integrated analysis platform to researchers across domains. We evaluate the provisioner by comparing the total cost of workload execution when different instance and availability zones are considered. Our results show that cost-aware provisioning approaches can provide monetary savings of up to 97.2% when compared to a baseline naïve approach.

The remainder of this paper is organized as follows. We discuss related work in Section II. In Section III we present the cost-aware provisioning algorithm. In Section IV we evaluate the provisioner on real-world scientific analysis workloads. Finally, we summarize our contributions in Section V.

II. Related Work

Many scientific computing applications leverage cloud resources to host computing applications [7], [8] and serve science gateways [9] to researchers. The ability of the cloud to scale elastically to meet the requirements of scientific research has frequently been demonstrated. However, the need to provision cost-effectively the required cloud resources is often overlooked.

Cost-minimization can be crucial to the effective use of cloud resources. Others have proposed provisioning algorithms that mix static, dynamic, and priority-based instance selection and attempt to schedule workloads efficiently over dynamically provisioned resource pools [10]. These investigations show that significant benefits can be obtained with respect to utilization, performance, and failure reduction. However, in contrast to the work presented here, these investigations focused on a single instance type and constant price model.

Cost-aware approaches have been applied to auto-scaling schedulers designed to meet dynamic deadlines [11]. Such approaches have been shown to provide both cost savings and improved resource utilization. Similarly, cost-aware scheduling approaches can provide improved reliability when deploying applications across clusters composed of spot instances [12].

III. Cost-Aware Provisioning

Our cost-aware provisioner monitors a job queue and creates new EC2 instances to satisfy workload requirements. Where possible the provisioner uses application profiles that describe compute, disk, and memory requirements of jobs to restrict

2159-6190/15 $31.00 © 2015 IEEE
DOI 10.1109/CLOUD.2015.130

the set of possible instance types. It uses the EC2 APIs to monitor real-time spot instance prices, and makes selection and allocation decisions based on predefined policies. Algorithm 1 outlines the algorithm used to provision resources.

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 suitable instance types

Cloud providers offer a variety of different instance types, each designed to meet the computational requirements of different applications. For example, Amazon EC2 offers 28 different instance types, each at a different price. Scientific applications also vary with respect to their compute, memory, disk, and I/O requirements. For example, analysis of applications included in four Globus Galaxies gateways shows that different applications can require between 4 to 32 CPUs, and from small to large memory and disk. Thus, not all applications can execute on all instance types; furthermore, each individual application makes the most cost-effective use of a specific instance when it requirements are aligned with that specific instance's capabilities.

In order to match applications with appropriate instance types we first define application profiles. We use HTCondor as our scheduler, and thus we can use ClassAds to express application requirements [13]. To do so, we extracted job execution information from workflow logs and manually constructed basic application profiles for almost 50 of the most frequently used applications. The provisioner uses these profiles to filter the list of possible instance types to only those that are suitable for the given application (Line 12). The provisioner is currently able to evaluate 14 different EC2 instance types based on application CPU, disk, and memory requirements. If no profiles are available for an application, we use instead a default instance type that is capable of satisfying the requirements of any Globus Galaxies application.

B. Comparing across instance types and availability zones

AWS, like many cloud providers, offers several acquisition and pricing models with different service levels. *On-demand* instances are offered at a static advertized hourly rate and are guaranteed to be available for the duration of the lease. *Spot* instances are offered at a fluctuating price based on real-time demand. A user bids the maximum price that they are prepared to pay for an instance. An instance is supplied immediately, at an initial price equal to the spot price, if the spot price is less than the bid price. The user is then charged for the instance, while it is running, at a rate per unit time determined by the fluctuating spot price, except that if that price exceeds the bid price, the instance is terminated. Within a particular Amazon EC2 region instances are offered in several availability zones—distinct locations engineered to be isolated from one another. Instance prices may differ significantly between availability zones based on supply and demand.

Our cost-aware provisioner uses AWS APIs to retrieve real-time spot and on-demand instance prices across availability zones. After filtering the list for suitable instance types (Line 12), the provisioner determines the current price for each instance type in each availability zone (Line 15). It orders instance types by cost and requests the cheapest instance (Line 22-24). It may therefore select a faster instance than is necessary if the cost is lower than the best match. As some applications are more sensitive to termination (e.g., long-running applications) 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.

C. Over-provisioning instance requests

Spot instances are inherently unreliable as the market price varies with demand and provisioned instances are terminated if the spot price exceeds the bid price. High degrees of contention for a particular instance type can rapidly raise the spot price of that instance type, in one or more availability zones. Often, such contention also results in significant delays in provisioning new instances. To reduce the affect of high contention the provisioner is equipped to over-provision instance requests by attempting to request more instances than are required (Line 7). The provisioner monitors outstanding instance requests and when one is satisfied unneeded requests are either terminated or migrated to another waiting job (Lines 10 and 27). Over-provisioning improves the overall throughput of the gateway by reducing the risk of having to wait extended periods of time for requests to be satisfied.

D. Reverting to to on-demand instances

As spot markets are based on demand the price of commonly used instance types can sometimes greatly exceed the

on-demand price. This situation arises in the case of the m2.4xlarge and r3.8xlarge instances, which frequently exceed $6 and $14, respectively, compared to their on-demand prices of $0.98 and $2.80. In order to mitigate the risks associated with spot market volatility (i.e., increased costs and delayed requests), the provisioner is designed to revert to on-demand instances based on a customizable timeout and threshold (Lines 8-10). If the request timeout is reached, the provisioner requests the cheapest on-demand instance that can satisfy the job's requirements. Similarly, if spot prices exceed on-demand prices by the threshold, it will revert to on-demand instances (Lines 16-17). Thus, the threshold is a proxy for maximum bid price irrespective of the instance type selected. Importantly, the provisioner will only revert to on-demand instances after other instance types and availability zones are considered.

IV. Evaluation

We used four workflow traces, each from a production Globus Galaxies gateway, to evaluate the provisioner. To isolate the effect of individual provisioning techniques, we further evaluated the role of varying the scope of the provisioner search process to either a single instance types or any one of 14 different instance types, and to either a single availability zone (us-east-1c), or any availability zone that supports the desired instance type. This gave us four search scopes, as follows:

- **Single instance, single availability zone (SI-SAZ)**: A single instance type (m2.4xlarge) in a single availability zone (us-east-1c).
- **Multi-instance, single availability zone (MI-SAZ)**: Any suitable instance type, in a single availability zone (us-east-1c).
- **Single instance, multi-availability zone (SI-MAZ)**: A single instance type (m2.4xlarge), across all availability zones.
- **Multi-instance, multi-availability zone (MI-MAZ)**: Any suitable instance type, across all availability zones.

To avoid spot instance termination and establish a worst-case scenario, we set the spot instance bid price to $6.56[1] for each scope. The cost of executing a workload is computed by calculating the price (using advertized on-demand prices and historical spot prices) over the period of execution.

A. Workload

To evaluate the provisioner under realistic conditions, we gathered execution traces from four Globus Galaxies gateways. These traces include over 12,000 job executions over a 60-day period. Fig. 1 shows the number of jobs executed each day for each gateway. The figure highlights the varying and volatile usage patterns of these gateways, which we view as providing a good basis for evaluation under various load conditions.

B. Cost

We used the gateway execution traces to compute the cumulative cost of the four search scopes for each of the four gateways. Fig. 2 shows the cost of using the SI-SAZ approach (solid line) compared with the MI-MAZ, approach (dotted line) for each gateway. The figure highlights the significant cost savings that can be achieved by the provisioner. TABLE I presents the breakdown of the four provisioning strategies when applied to each of the gateways. The savings between the SI-SAZ and MI-MAZ approaches range between 30.9% and 97.2%, with an average cost reduction of 77.8%.

[1]$6.56 is the maximum spot price during the period of evaluation.

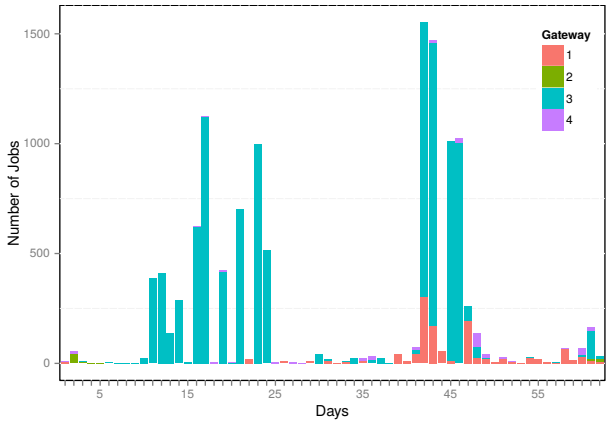

Fig. 1. The number of jobs executed each day over a 60 day period.

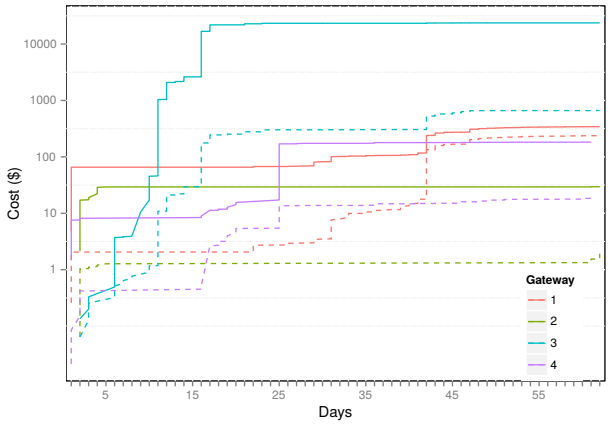

Fig. 2. The cost of the naïve and cost-aware provisioner for each of the monitored gateways over a 60 day period on a logarithmic scale.

TABLE I
TOTAL 60 DAY COST COMPARISON BETWEEN GATEWAYS

Gateway	SI-SAZ	SI-MAZ	MI-SAZ	MI-MAZ
1	$343.08	$250.87	$237.26	$237.22
2	$29.93	$3.83	$1.92	$1.92
3	$23,720.82	$21,532.29	$670.32	$665.03
4	$183.04	$23.69	$15.30	$18.77
Total	**$24,276.86**	**$21,810.67**	**$924.78**	**$922.93**

Fig. 3 shows the cumulative cost across all four gateways. Note the significant savings achieved by the MI-SAZ and MI-MAZ approaches when compared with their single-instance

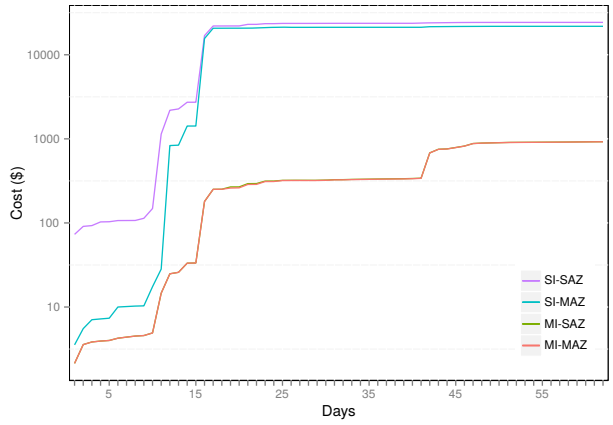

Fig. 3. The total cumulative cost of operating four Globus Galaxies gateways with different provisioner search scopes over a 60 day period on a logarithmic scale.

alternatives. An overall cost reduction of $23,353, or 96.1%, is achieved between the SI-SAZ and MI-MAZ scopes. Unexpectedly, the advantage of considering multiple availability zones in addition to multiple instances types is relatively insignificant, with less than $2.00 or 0.01% difference between the MI-SAZ and MI-MAZ scopes.

C. Reverting to On-demand Instances

To avoid prolonged job queuing times when spot resources are not immediately available, the provisioner reverts to using on-demand instances if a configurable timeout is exceeded. To assess these capabilities, we projected the cost of using various timeouts across the four gateways, based on historical spot instance prices. The results are shown in TABLE II. We only consider jobs where a new instance is launched.With a five minute threshold, the total cost increases by over 650%, or $6959.77. Timeout values of 10, 15, and 20, minutes result in approximately double the cost.

TABLE II
THE COST OF REVERTING TO ON-DEMAND INSTANCES WITH VARIOUS TIMEOUT VALUES

Timeout	∞	5	10	15	20
Cost	922.93	6959.77	2239.87	2001.98	1960.04

V. Summary

Here we presented a cost-aware elastic provisioner that is capable of elastically provisioning cloud resources on-demand. The provisioner is designed to monitor a job submission queue and provision cloud instances based on pre-defined policies. It is able to select suitable instance types to host a given application based on application profiles, select the most cost-effective instance type across availability zones using on-demand and spot prices, over-provision resources when instances are highly contested, and revert to stable on-demand instances when spot instance prices are volatile or requests are delayed.

We demonstrated, via analysis of four production Globus Genomics gateways, that our cost-aware provisioner is able to reduce costs when compared with the existing naïve provisioning approach. Our results show that increasing the search scope to consider additional instance types and availability zones results in a 97.2% reduction in costs.

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] 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.

[3] R. K. 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," submitted to Concurrency and Computation: Practice and Experience. 2015.

[4] 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.

[5] 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.

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

[7] 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 International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2008)*. IEEE Press, 2008, p. 50.

[8] R. Chard, R. K. Madduri, N. T. Karonis, K. Chard, K. L. Duffin, C. E. Ordoñez, T. D. Uram, J. Fleischauer, I. T. Foster, M. E. Papka, and J. Winans, "Scalable pCT image reconstruction delivered as a cloud service," submitted to IEEE Transactions on Cloud Computing. 2014.

[9] 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 *Proceedings of the 4th IEEE International Conference on Utility and Cloud Computing (UCC)*. IEEE, 2011, pp. 287–292.

[10] 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*. IEEE Computer Society Press, 2012, p. 22.

[11] 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.

[12] 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.

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