HPC@Cloud: A Provider-Agnostic Software Framework for Enabling HPC in Public Cloud Platforms

Vanderlei Munhoz[1], **Márcio Castro**[1]

[1] Universidade Federal de Santa Catarina (UFSC)
Centro Tecnológico – Departamento de Informática e Estatística (INE)
R. Delfino Conti, Trindade, Florianópolis – SC, 88040-900 – Brazil

munhoz@proton.me, marcio.castro@ufsc.br

Abstract. *The cloud computing paradigm democratized compute infrastructure access to millions of resource-strained organizations, applying economics of scale to massively reduce infrastructure costs. In the High Performance Computing (HPC) context, the benefits of using public cloud resources make it an attractive alternative to expensive on-premises clusters, however there are several challenges and limitations. In this paper, we present HPC@Cloud: a provider-agnostic software framework that comprises a set of key software tools to assist in the migration, test and execution of HPC applications in public clouds. HPC@Cloud allows the HPC community to benefit from readily available public cloud resources with minimum efforts and features an empirical approach for estimating cloud infrastructure costs for HPC workloads. We also provide an experimental analysis of HPC@Cloud on two public clouds: Amazon AWS and Vultr Cloud.*

1. Introduction

The cloud computing paradigm is defined as a *pay-per-use model* that allows the convenient on-demand access to a configurable group of computing resources in a rapid manner with minimum effort and contact with the provider [Mell and Grance 2011]. Public cloud providers conveniently offer their services to *any* person through the Internet, without requiring long-term contracts or any interaction with the provider whatsoever. They democratized compute infrastructure access to millions of organizations with few capital resources, applying economics of scale to massively reduce IT costs [Buyya et al. 2019]. The major public cloud providers today are Amazon Web Services (AWS), Microsoft Azure, Alibaba Cloud, Google Cloud Platform (GCP), and Huawei, with a combined market share of 80% [Gartner 2022].

To even further optimize infrastructure usage, cloud providers can rent their spare capacity at substantial discounts, reclaiming it whenever there is a more profitable request to be met — a business model pioneered by AWS in 2009 to sell idle infrastructure and increase revenue. Machines commercialized through this business model are typically called *spot machines*, which are ephemeral instances with a considerably lower cost per hour than standard machines, running on top of the same type of infrastructure with identical characteristics and performance.

In the High Performance Computing (HPC) context, the benefits of using public cloud resources make it an attractive alternative to expensive on-premise clusters. Moreover, the spot market opens new possibilities for small research groups to run HPC applications on highly-parallel infrastructures with much lower financial costs. However,

the software ecosystem necessary to make possible a sustainable HPC cloud platform is not yet mature [Netto et al. 2018]. Cost advisors, large contract handlers, DevOps solutions, automation APIs and HPC-aware resource managers are current software gaps in this regard.

In this paper, we propose *HPC@Cloud*: a provider-agnostic software framework for enabling HPC in public cloud platforms with minimum efforts. Overall, *HPC@Cloud* brings the following contributions to the state of the art on Cloud HPC:

1. New much needed open-source tools for assisting researchers and consumers on how to migrate legacy HPC code efficiently to the public cloud, using documented open-source tools well established in the enterprise world; and
2. An empirical approach for estimating cloud infrastructure costs for HPC workloads, which helps consumers gain a quick impression of the pros and cons of using public cloud resources for HPC.

The remainder of this work is organized as it follows. In Section 2, we present the background and motivation of this work. In Section 3, we discuss related works. In Section 4, we present our proposed software framework (*HPC@Cloud*), with a discussion regarding the major trade-offs and the reasoning behind our decisions. In Section 5, we present our applied evaluation method and gathered experimental data. In Section 6, we analyze and discuss the obtained results. Finally, we draw our conclusions and discuss future work in Section 7.

2. Background and Motivation

2.1. Cloud Computing

The cloud computing paradigm allows organizations to reduce information technology costs by offering compute infrastructure and software services through a *pay-as-you-go* pricing model, similar to public utilities. This pricing model reduces the capital and technological entry barrier for small organizations, who can create and destroy resources on-demand, paying only for what they use. Users can then dynamically scale resources in real-time as their application requirements change, reducing resource over-provisioning and under-provisioning.

Overall, cloud computing platforms can be divided into three types: *public*, *private*, and *hybrid*. *Public cloud* providers conveniently offer their services to *any* person through the Internet, without requiring long-term contracts or any interaction with the provider whatsoever. In contrast, *private cloud* platforms offer their services only to a specific organization or limited group of users with special access. Finally, *hybrid clouds* provide a connection between public and private clouds, so organizations can avoid relying on a single public cloud provider or only on its own infrastructure.

The cloud computing service model is traditionally described by three types of services. *Infrastructure as a Service* (IaaS) refers to online services that provide APIs for users to spawn and manage compute infrastructure, including low-level details such as network, storage and backups. The user often can choose the computing capacity of the infrastructure to be rented — typically in terms of virtual CPUs (vCPUs) — and can also configure other details such as hypervisor types, pre-installed OS, accelerators, and more. *Platform as a Service* (PaaS) refers to services that the user can use to create and deploy custom software applications using a configurable environment hosted by the

cloud provider. Runtime, middleware and software features are abstracted from the user, and managed by the provider. Finally, *Software as a Service* (SaaS) are ready-to-use software applications with specific purposes that the provider typically offers through APIs, which users can use directly into their applications.

2.2. Spot Markets

Cloud providers can further optimize resource usage by renting spare infrastructure under considerable discounts, in what is traditionally known as *spot market*. Spot machines are ephemeral instances with a considerably lower cost per hour than standard (persistent) machines, running on top of the same type of infrastructure with identical characteristics and performance. Since this market has attracted a good amount of attention from enterprises and organizations looking to reduce infrastructure costs, most major cloud providers offer some kind of spot market for IaaS nowadays. The downside comes from the fact that spot machines can be revoked on short notice at any time by the cloud provider (usually whenever capacity needs to be reclaimed for standard persistent instances), being responsibility of the user to preemptively save data and recover applications in case of instance repossessions.

When requesting a spot machine, users bid the maximum price they are willing to pay per hour for the requested resources. The provider only meets requests if the specified spare capacity is available at a price lower or equal to the bid value. This model is completely different from on-demand pricing, where an instance request is met immediately. This characteristic is an important detail in contexts where workloads have a defined deadline for completion, as spot requests have no guarantee by the provider ever to be met. Moreover, spot prices for each machine type often fluctuate dynamically based on market demand in each availability zone, although this depends on the provider (Oracle, for instance, offers a fixed discount for their spot machines).

2.3. Challenges for Cloud and HPC Convergence

In the HPC context, the public-utility pricing model makes IaaS offerings an attractive alternative for budget-constrained researchers and organizations, as the reliance on expensive on-premise specialized hardware can be eliminated, and the capital and technological entry barrier for HPC can be substantially reduced. Public cloud platforms evolved to support rapid allocation and deallocation of virtualized resources, allowing users to dynamically scale infrastructure according to their needs. As such, most companies use cloud resources to guarantee service availability and disaster recovery capabilities for their enterprise systems. In most common cases, these companies neither require low-level knowledge about the underlying hardware nor need high-speed communication between nodes, which are typical requirements for HPC workloads [Santos and Cavalheiro 2020].

Although spot markets and alternative low-cost cloud providers may be attractive options for small organizations, traditional HPC still has a long way to go before being able to take advantage of cheap cloud resources efficiently. As the end-user will likely suffer spot instance repossessions during job execution, fault tolerance strategies are imperative for executing stateful applications — which is the case for most HPC workloads. Furthermore, as the available capacity also varies between different instance types, careful consideration is required when choosing machine flavors, given that the number of instance repossessions can be high and slow down execution considerably, even when employing low-overhead fault tolerance mechanisms. Additionally, estimating optimal

bids for minimum costs is a hard problem, given the large optimization space and lack of available information from most cloud providers, which often lack transparency regarding pricing dynamics and historical data.

Unfortunately, the software ecosystem necessary to make possible a sustainable HPC cloud platform is not yet mature [Netto et al. 2018]. There is a lack of cost advisors, large contract handlers, DevOps solutions, automation APIs and HPC-aware resource managers. Cost advisors that are popular in standard cloud environments for example, not only must be able to predict the duration of the HPC jobs of a user, but also need to infer for how long an experiment with an unknown number of jobs will take to be executed completely. Further research also indicates that the current public cloud pricing models do not take into account the peculiarities of HPC workloads [Al-Roomi et al. 2013]. Because of the shared nature of public cloud resources, the performance of HPC workloads under development can be significantly different each time they are tested by the users, making DevOps for HPC extremely complex.

3. Related Work

Most of the research and industry efforts regarding HPC and cloud computing is focused on understanding the cost-benefit of moving legacy applications from on-premise clusters to a public cloud platform. Research in the field is also focused on how to extract the best performance possible from unknown underlying hardware, connected through a relatively much slower and unreliable network [Netto et al. 2018].

Somasundaram et al. followed a similar approach that of our study, proposing and testing a resource broker for minimizing execution time and costs of jobs in HPC Clouds [Somasundaram and Govindarajan 2014]. Their solution is based on a Particle Swarm Optimization (PSO) allocation mechanism. *Peña-Monferrer et al.* proposed a cloud native framework for executing Computational Fluid Dynamics (CFD) workloads based on an hybrid cloud architecture [Peña-Monferrer et al. 2021]. The majority of the computational load is executed in a traditional on-premises HPC cluster, while the data processing of the results is done using public cloud infrastructure. Our research has a different scope and infrastructure, since we focus on bringing HPC workloads entirely to the public cloud. Another framework was proposed by *Wong et al.*, which focused on the deployment and exposure of HPC applications as SaaS offerings in public clouds [Wong and Goscinski 2013]. Our research differs as it does not tackle the issue of servicing HPC applications but the feasibility of executing them using public cloud infrastructure. Although the researchers addressed the issues of configuring cloud resources for HPC, they do not analyzed the cost-effectiveness of actually executing these workloads in the cloud.

Recent works have also been studying the cost-effectiveness and reliability of spot machines. *Qu et al.* discussed both aspects when using AWS spot instances [Qu et al. 2016]. However, they focused on enterprise web applications and not HPC workloads, being a different context than the one considered in our work. *Teylo et al.* presented an extensive evaluation of AWS spot machines for scheduling bag-of-tasks applications [Teylo et al. 2021], which we consider complimentary to our research. However, our work goes further, bringing a generalized toolset for migrating not only bag-of-tasks applications but tightly-coupled HPC workloads as well. *Li et al.* presented a comparative bibliographical investigation on the pros and cons of the spot and fixed-pricing schemes for cloud computing [Li et al. 2015], whereas our study presents a soft-

ware framework proposal and *practical analysis* of how to actually migrate MPI applications to the cloud.

Finally, there have been some efforts on providing fault tolerance for applications running on cloud infrastructures. *Gong et al.* and *Voorsluys et al.* investigated the efficiency of fault tolerance techniques based on BLCR using AWS infrastructure, analyzing how replicated execution can help on reducing monetary costs [Gong et al. 2015, Voorsluys and Buyya 2012]. We consider our study complimentary, as our proposed framework can be extended to support not only BLCR, but other fault tolerance mechanisms as well. Our research also has a larger scope, focusing on building tools to test and predict job execution costs. *Egwutuoha et. al* proposed a fault tolerance framework based on Process Level Redundancy (PLR) for executing HPC applications in the Cloud. Our research differs in scope and also considers different fault tolerance mechanisms, such as system-level checkpoint restart. *Yi et al.* presented an analysis of spot AWS resources cost-reductions granted by different checkpointing policies [Yi et al. 2010]. Differently, in this paper, we neither focused on evaluating checkpointing policies nor limited our contributions to a specific cloud provider and pricing model.

4. The *HPC@Cloud* Framework

In this section, we present our proposed software framework for HPC using public cloud resources (*HPC@Cloud*). Although we implemented a first version with support for two cloud providers — Amazon AWS and Vultr Cloud — our framework can easily be extended to support other providers. *HPC@Cloud* is an open-source software and it is available at `https://github.com/vnderlev/hpccloud`.

4.1. Architectural Overview

To assist researchers on migrating legacy HPC applications to public cloud platforms, *HPC@Cloud* features a set of tools (executed on the user's machine) to configure cloud infrastructure, execute jobs, monitor, predict costs and interface with the cloud in an automated and provider-agnostic manner. Our framework architecture is comprised of four principal modules with the following purposes:

1. IaaS resource management (provider integration, infrastructure creation, monitoring and destruction);
2. MPI job management and fault tolerance mechanisms;
3. Application testing and costs prediction; and
4. Experiment metadata gathering and storage.

Figure 1 depicts a high-level overview of *HPC@Cloud*, presenting the existing integrations between modules and cloud providers. Our *Workload Manager* is built on top of `mpiexec`, SLURM and Berkeley Labs Checkpoint/Restart (BLCR) package [Hargrove and Duell 2006] (further detailed in Section 4.3) for system-level fault tolerance. This component interacts with the *Infrastructure Manager* built on top of Terraform[1] — Terraform plans are generated by the *Workload Manager* based on input from the user. Workloads are executed through SSH. Metadata regarding experiments, be them tests or full experiments, are collected and stored for further use by the *Costs Prediction* module. *Costs Prediction* and *Application Testing* are *Workload Manager* extensions, which in practice are a collection of methods.

[1] `https://www.terraform.io/`

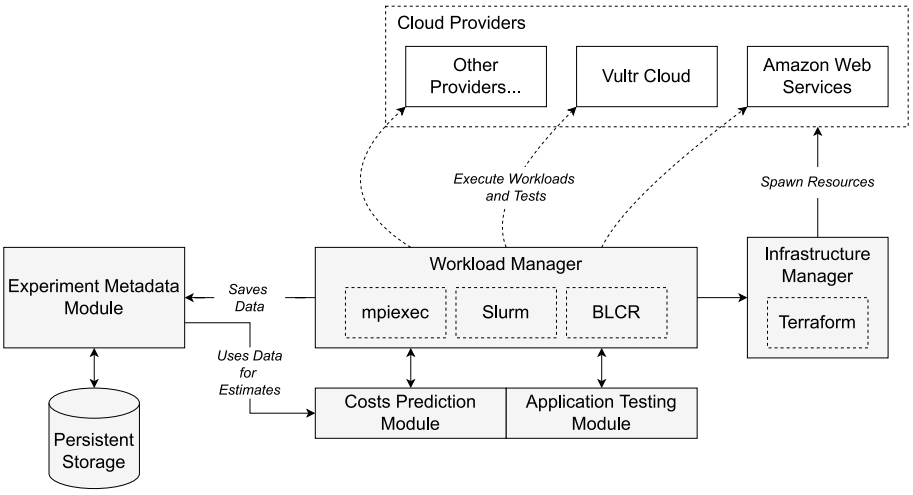

Figure 1. Overview of *HPC@Cloud* main modules and components.

4.2. Managing IaaS Resources

Despite efforts in standardization, configuring and managing cloud resources often requires provider-specific knowledge about the platform and its APIs, which are also subject to arbitrary changes by the providers. Managing infrastructure in a repeatable manner is a desired feature not only for HPC but in the enterprise world as well. The state of the art regarding infrastructure management and operations lies in the concept of Infrastructure as Code (IaC), which refers to the practice of configuring computing resources through machine-readable definition files, rather than relying on interactive configuration tools [Rahman et al. 2019].

We applied the concept of IaC into the *HPC@Cloud* framework for creating, monitoring and destroying IaaS resources. Aiming at the convergence of cloud computing and HPC, we seek to employ the same set of pre-existing open-source tools whenever possible in our proposed framework. For infrastructure configuration, we opted for the Terraform tool, which follows a declarative approach for infrastructure definition. With Terraform, we are able to define our virtual clusters on top of hundreds of cloud providers, including AWS and Vultr, two cloud platforms tested in this paper.

Figure 2 presents the *Infrastructure Manager* module workflow, highlighting integrations with cloud providers and the *Workload Manager* module. The end-user must define the required infrastructure for his/her workloads in a declarative manner, while the framework is responsible for managing the requested resources with minimum costs. The IaaS management module is able to read the infrastructure specifications by the user and converting them into Terraform plans. When using spot infrastructure, these plans are then applied and monitored periodically to make sure all machines are working as expected. In case of revoked spot instances, the IaaS module is able to re-apply the original terraform plans, or create new ones to request new instances. Workload execution can then be resumed from a previous checkpoint.

After a workload finishes execution, the cluster worker nodes are terminated.

When requesting spot instances, the *Infrastructure Manager* needs to be configured to use a bidding strategy that can be customized for specific providers. In our implementation, we devised a strategy for periodically increasing bid values for AWS machines

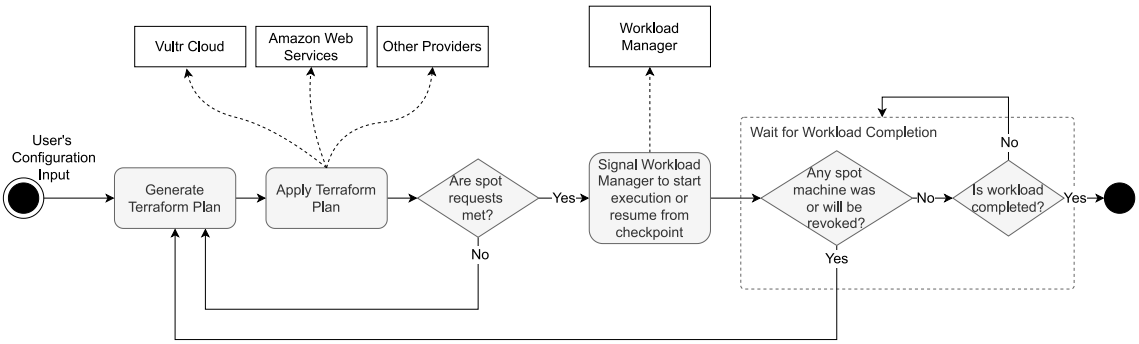

Figure 2. Infrastructure manager module workflow.

until the requests are met. This approach favors quick infrastructure creation for a relatively small cost.

4.3. Managing Workloads

Workloads are managed using ubiquitous tools in the HPC world, such as `mpiexec` (in case of MPI applications) and SLURM. This allow users that are already familiarized with these tools to deploy their applications in the cloud without hassle, easing the migration process. In short, after spawning a cluster, the *Workload Manager* copies application files (defined by the user) to the cluster file system and execute them using `mpiexec`.

To support spot infrastructure, the *Workload Manager* must be aware of failures (spot instance repossessions) and provide a set of fault tolerance strategies to resume execution. We currently added support for a system-level rollback restart mechanism based on the BLCR package [Hargrove and Duell 2006]. In this approach, no changes to the MPI application code are required. To take advantage of spot repossession notifications from AWS, we improved the basic BLCR strategy to be able to orchestrate checkpoints, launch new spot requests and listen to spot eviction alarms sent by the cloud provider. These new functionalities are handled by a job manager running externally to the spot cluster, which can be configured to trigger checkpoints periodically or only upon notification by AWS. Periodical checkpoints are required when spot eviction alarms are unavailable or unreliable, which may be the case for most providers.

4.4. Testing and Costs Prediction

HPC@Cloud also features a module for aiding HPC developers on testing their applications with minimal costs. As public cloud environments are unreliable, we implemented a set of tools to gather network and I/O performance data from clusters, and also execute short-lived sample workloads that must be prepared by the user.

With the data gathered from short experiments, it is possible to estimate the total execution costs at least for uniform workloads that do not have large variations in time, such as grid-based physics simulations — which comprises a large chunk of traditional HPC. Bag-of-tasks workloads can also be predicted by running a single task using one cluster node. This mechanism allows users to both test their applications for bugs and performance using cloud resources, allowing the quick destruction of resources after testing.

4.5. Metadata Gathering

Finally, *HPC@Cloud* has an optional feature that can be enabled with the user's consent that allows the gathering of data related to experiments execution, both tests and actual workloads. This data can help in the decision making rules for the other framework modules, such as the *Infrastructure Manager* and *Costs Prediction* module. For example, when choosing spot instance types, the *Infrastructure Manager* can base its decision on historical pricing and instance revocation frequency data.

5. Experimental Evaluation Method

To evaluate *HPC@Cloud*, we implemented a beta version using Python, Terraform and the MVAPICH[2] distribution of MPI with BLCR support enabled. Support for AWS and Vultr Cloud is currently implemented, with functions able to generate Terraform plans for creating clusters of homogeneous virtual machines. The user can define the number of nodes and their characteristics, such as number of vCPUs and memory.

The Terraform plan generator is set to create the minimum required resources to emulate a real cluster environment, which includes a Virtual Private Cloud (VPC) resource, network security rules and a shared block storage resource for the cluster. Instances are then created and placed inside the VPC. To configure machines, the user must specify the OS and provide a shell script for automating dependency installation. The *Infrastructure Manager* module will then spawn a single machine and prepare a virtual machine snapshot for future use. With the snapshot ready, copies of the machine can be created much faster, without requiring the execution of the initialization shell script. In the case of spot machines, the user can define the maximum value per hour he/she is willing to pay for the cluster. If there are no available spot resources, the *Infrastructure Manager* module has a bidding policy of incrementing gradually the spot bidding until all requests are met.

To run workloads, we considered a parallel solver implementation for Laplace's heat diffusion equations as a study case, which is based on a popular alternative FDM technique called Forward-Time Central Space (FTCS) method [Kafle et al. 2020]. For the experiments, we consider a square region of n^2 mesh cells with non-periodic borders as the physical domain to be simulated. This domain is then broken into subdivisions and distributed to each MPI rank for iterative computation, resulting in a Cartesian topology of MPI processes. Subdivisions are made following a one-dimensional decomposition strategy and global boundaries have a constant temperature value. Each subdivision needs

[2] `https://mvapich.cse.ohio-state.edu/`

Table 1. Tested Providers and Instance Types.

Cloud Provider	Instance Type	vCPUs	Base Cost (USD/h)	Spot Cost* (USD/h)
AWS	`t3.2xlarge`	8	0.3328	0.1561
	`c5.2xlarge`	8	0.3400	0.1986
	`c6i.2xlarge`	8	0.3400	0.2265
Vultr	`vhp-4c-8gb`	4	0.0710	-
	`vhp-8c-16gb`	8	0.1430	-
	`vhp-12c-24gb`	12	0.2140	-

*Week-average costs at time of publication.

information from its neighbors to compute each iteration at the internal borders. To reduce messaging, we employ a ghost-cell pattern updated at the end of each iteration.

We executed experiments defining three types of persistent machines in Vultr Cloud and three types of spot machines from AWS (Table 1). Machine types are CPU-only and marketed as for general computing purposes. As AWS limits the total amount of cores to 128 for standard accounts, we focus on smaller flavors such as the 8 vCPUs machines. Machines were created at AWS in the `us-east-1` availability zone, and in the `dfw` availability zone in Vultr Cloud. For each machine type described in Table 1, we executed 10 test runs and 10 full runs of the MPI application. Spot machines from AWS are requested following the bidding strategy described in Section 4, while Vultr machines are persistent and promptly created. We induced spot revocations artificially by destroying instances randomly with a helper script. We gather the MPI workload execution times as well as the time spent waiting for spot requests to be met and time for infrastructure to be ready. We ignored storage and network costs as they are insignificant compared to the compute infrastructure costs.

We evaluate *HPC@Cloud* using the following metrics:

1. Time spent setting up infrastructure for running experiments, taking into consideration the spot bidding strategy cost-effectiveness;
2. Performance impact of the fault tolerance strategy in terms of idle time during spot revocations; and
3. Accuracy of cost predictions.

6. Experimental Results

Figure 3 shows the obtained results from spot AWS and Vultr machines, highlighting average idle and execution times. Results indicate that spot machines are quickly spawned without a substantial performance degradation using our bidding strategy (from 2% up to 8% of the application total execution time). The *Infrastructure Manager* module was able to provide the required machines quickly, both when using Vultr and AWS services. It is noted that machine spawn times are consistent between machine types, with larger machines taking more time to be ready (from 3% for the smaller machines up to 27% of the application total execution time for clusters comprised of large instances).

During our experiments, not a single spot request was refused because of unavailable capacity. Assuming that the provider always has resources available, our *HPC@Cloud* framework will have no problems preparing spot clusters quickly. In cases where spot instances of a specific flavor may not be available, our spot bidding strategy can be customized to change machines to a different type (thus, creating heterogeneous clusters) or to use persistent infrastructure temporarily. Allowing heterogeneous clusters to be created increases flexibility and reduces allocation time when spot instances are scarce but this will probably result in load imbalance issues. In theses cases, it is up to the target application to balance its workload whenever needed.

Figure 4 presents a comparison between estimated and actual total execution times for the experiments conducted with Vultr and spot AWS resources. Predictions for Vultr (left-hand side) costs are exact, as machines are billed by hour and costs can easily be estimated in hours timescales at our context. AWS machines that are billed by seconds (right-hand side) have inaccuracies, as our estimation mechanism is not accurate enough for predicting execution time at seconds timescales. Error bars represent the range of predictions calculated during our series of experiments.

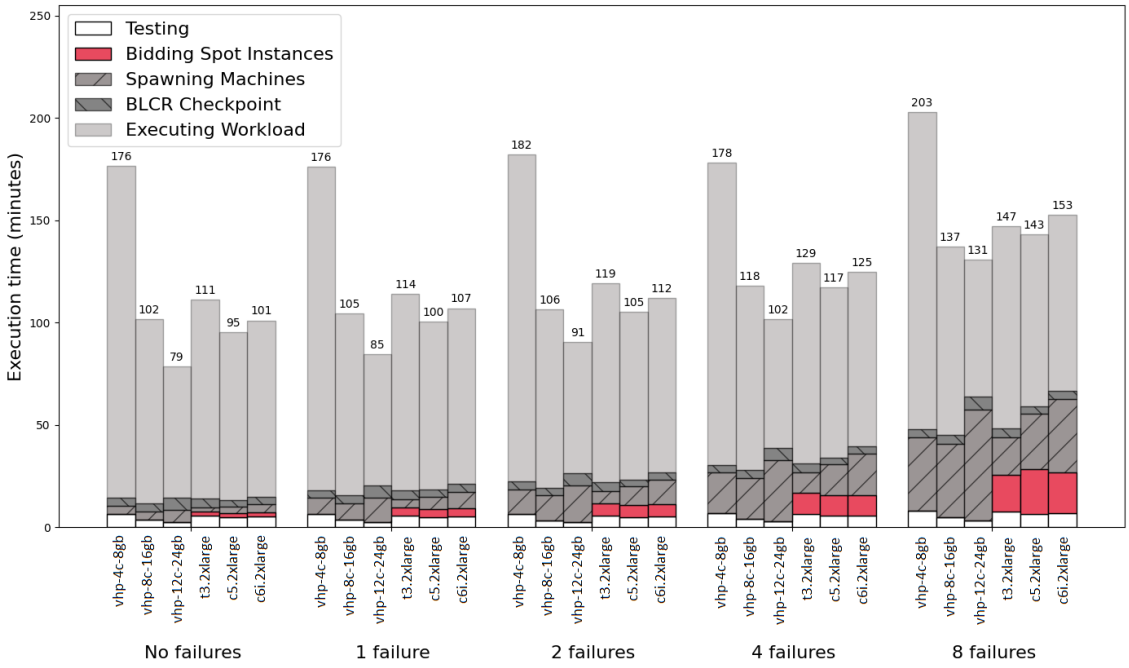

Figure 3. Workload execution experiments.

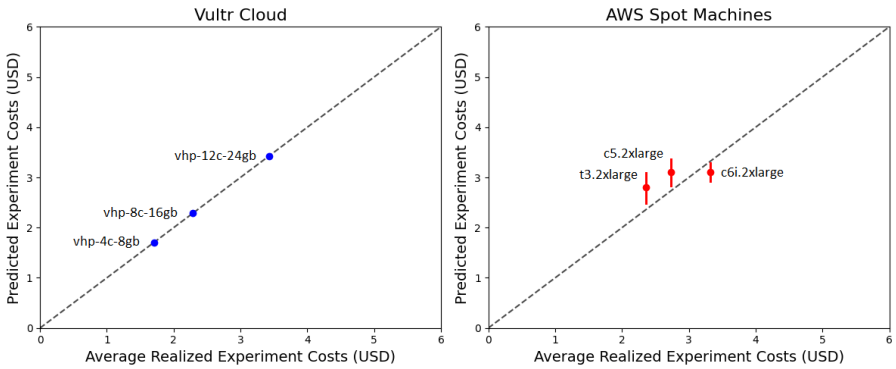

Figure 4. Predicted and actual execution costs comparison.

Estimating problem running times at transient spot clusters is unreliable, as there is no current real-time prediction strategy embedded in our proposed solution for estimating how many spot evictions will happen during execution. Other unknown variables play a role on total execution time, such as time between placing bids and allocation of spare infrastructure, which has no guarantees of ever being met by the provider. Aggressive bid strategies can help reduce allocation time, however they also come with increased cost. In cases where there is no spare infrastructure for spot placements, there is nothing that the user can do besides waiting for an unknown amount of time or changing machine types.

7. Conclusion

To assist the migration of legacy HPC applications to the public cloud, we proposed the *HPC@Cloud* framework, comprised of software modules for automated virtual environment configuration, experimental application testing and costs prediction. Our *Costs Prediction* module estimated the execution time of the MPI application with up to 94% accuracy, accruing minimal extra costs for large workloads and a relatively small execution

time overhead (up to 4% increase in total execution time, both for Vultr and AWS). Finally, despite our focus on an MPI case study application, the proposed fault tolerance strategy and the job *Cost Prediction* module can be generalized to a wide range of tightly-coupled MPI applications and other cloud providers not considered in this study.

We proposed a fault tolerance strategy for enabling the use of cheaper transient spot resources. Our simple strategy proved to be effective, taking a negligible amount of time to realize checkpoints when comparing to the total workload execution time. Careful consideration must be taken for small workloads, as the use of persistent machines for a short time span will likely be more cost-effective than using spot machines. Our cost estimations also show that using spot resources do not guarantee cost savings when considering multiple cloud providers. Persistent Vultr machines demonstrated similar performance than AWS spot machines for a slightly smaller cost.

As it can be noted from Figure 3, a growing number of failures — or spot instance revocations — can degrade application performance considerably, specially given that most of the time the application needs to stay idle waiting for spot requests to be met and the machines be prepared by the provider. To mitigate this, *HPC@Cloud* makes use of provider-specific notification systems, such as the one provided by AWS CloudWatch, to preemptively prepare machines before failures occur. It is also possible to employ analytical models to estimate when a spot failure may occur, as indicated by related research. As future works, we intend to explore optimization strategies to reduce the fault tolerance overheads.

Regarding cloud providers, we intend to add support for Microsoft Azure and IBM Cloud spot machines into our *Infrastructure Manager* module. We also plan to improve our *Workload Manager* to support different system-level checkpoint strategies besides BLCR, such as container migration technologies. Containers can also make workload migration easier and faster, as container images are more portable than virtual machines. Finally, we intend to add support for dynamic fault tolerance approaches that do not require the full MPI world restoration, permitting workload execution with a reduced number of nodes instead of waiting for the entire original cluster to be restored. This may reduce total execution times up to 20% based on gathered data from our experiments.

References

Al-Roomi, M., Al-Ebrahim, S., Buqrais, S., and Ahmad, I. (2013). Cloud Computing Pricing Models: A Survey. *International Journal of Grid and Distributed Computing*, 6:93–106.

Buyya, R., Srirama, S. N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., Gelenbe, E., Javadi, B., Vaquero, L. M., Netto, M. A. S., Toosi, A. N., Rodriguez, M. A., Llorente, I. M., Vimercati, S. D. C. D., Samarati, P., Milojicic, D., Varela, C., Bahsoon, R., Assuncao, M. D. D., Rana, O., Zhou, W., Jin, H., Gentzsch, W., Zomaya, A. Y., and Shen, H. (2019). A Manifesto for Future Generation Cloud Computing: Research Directions for the Next Decade. *ACM Computing Surveys*, 51(5).

Gartner (2022). Available at: https://www.gartner.com/en/newsroom/press-releases/2022-06-02-gartner-says-worldwide-iaas-public-cloud-services-market-grew-41-percent-in-2021.

Gong, Y., He, B., and Zhou, A. C. (2015). Monetary cost optimizations for MPI-based HPC applications on Amazon clouds: checkpoints and replicated execution. In *SC*

'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–12.

Hargrove, P. and Duell, J. (2006). Berkeley lab checkpoint/restart (BLCR) for Linux clusters. *Journal of Physics: Conference Series*, 46:494.

Kafle, J., Bagale, L. P., and K. C., D. J. (2020). Numerical Solution of Parabolic Partial Differential Equation by Using Finite Difference Method. *Journal of Nepal Physical Society*, 6(2):57–65.

Li, Z. E., Zhang, H., O'Brien, L., Jiang, S., Zhou, Y., Kihl, M., and Ranjan, R. (2015). Spot Pricing in the Cloud Ecosystem: A Comparative Investigation. *Journal of Systems and Software*, 114.

Mell, P. M. and Grance, T. (2011). SP 800-145. The NIST Definition of Cloud Computing. Technical report, National Institute of Standards & Technology, Gaithersburg, MD, USA.

Netto, M., Calheiros, R., Rodrigues, E., Cunha, R., and Buyya, R. (2018). HPC Cloud for Scientific and Business Applications: Taxonomy, Vision, and Research Challenges. *ACM Computing Surveys*, 51.

Peña-Monferrer, C., Manson-Sawko, R., and Elisseev, V. (2021). HPC-cloud native framework for concurrent simulation, analysis and visualization of CFD workflows. *Future Generation Computer Systems*, 123:14–23.

Qu, C., Calheiros, R. N., and Buyya, R. (2016). A Reliable and Cost-Efficient Auto-Scaling System for Web Applications Using Heterogeneous Spot Instances. *J. Netw. Comput. Appl.*, 65(C):167–180.

Rahman, A., Mahdavi-Hezaveh, R., and Williams, L. (2019). A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108:65–77.

Santos, M. A. d. and Cavalheiro, G. G. H. (2020). Cloud infrastructure for HPC investment analysis. *Revista de Informática Teórica e Aplicada*, 27(4):45–62.

Somasundaram, T. S. and Govindarajan, K. (2014). CLOUDRB: A framework for scheduling and managing High-Performance Computing (HPC) applications in science cloud. *Future Generation Computer Systems*, 34:47–65. Special Section: Distributed Solutions for Ubiquitous Computing and Ambient Intelligence.

Teylo, L., Arantes, L., Sens, P., and Drummond, L. M. d. A. (2021). Scheduling Bag-of-Tasks in Clouds using Spot and Burstable Virtual Machines. *IEEE Transactions on Cloud Computing*, pages 1–1.

Voorsluys, W. and Buyya, R. (2012). Reliable Provisioning of Spot Instances for Compute-intensive Applications. In *2012 IEEE 26th International Conference on Advanced Information Networking and Applications*, pages 542–549.

Wong, A. K. and Goscinski, A. M. (2013). A unified framework for the deployment, exposure and access of HPC applications as services in clouds. *Future Generation Computer Systems*, 29(6):1333–1344. Including Special sections: High Performance Computing in the Cloud & Resource Discovery Mechanisms for P2P Systems.

Yi, S., Kondo, D., and Andrzejak, A. (2010). Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 236–243.