A Preliminary Review of Function as a Service platform running with AWS Spot Instances

Luciana da Costa Marques
Institue of Mathematics and Statistics
University of Sao Paulo
Sao Paulo, Brazil
lucianadacostamarques@gmail.com

Alfredo Goldman
Institute of Mathematics and Statistics
University of Sao Paulo
Sao Paulo, Brazil
gold@ime.usp.br

***Abstract*—Cloud computing enabled users to easily implement what was previously a complex data center infrastructure, reducing its maintenance related costs. However, cloud costs can be challenging to predict with all its variety of resources and price schemes. Amazon Web Services offers the spot pricing mechanism for virtual machines (VM), in which an end user buys idle VM capacity for up to 90% cost reduction. But, these machines can be terminated by AWS at any given time, so not every application may be a good fit using them. We here explore the deployment of function as a service (FaaS) platforms in spots by using *funcX*, a distributed and high performance FaaS project. We chose MASA-OpenMP, a DNA sequence comparison project, to be executed in two different scenarios: one with in-memory execution in spot VMs, orchestrated with *HADS*, a Hibernation-Aware Dynamic Scheduler for spot instances, and the other using a *funcX* client deployed in spot instances as well. Our initial results showed that costs using the *funcX* approach are potentially smaller, and it also offers more implementation flexibility. We also present possible next steps for more experiments and investigation.**

***Index Terms*—cloud computing, spot instances, preemptive virtual machines, function as a service**

I. Introduction

Cloud pricing and costs have received both academic and industry interest for many years now. With the rise of popularity of big cloud providers, such as Amazon Web Services (AWS), it has become incredibly easier to build software services, since that involved building a complex infrastructure prior to the Cloud. This has amplified services and products that can be built on top of it, and demand will only increase in coming years. This is because cloud resources are chosen based on their characteristics (e.g. CPU, Disk) rather than what would imply in their physical construction [6]. Another interesting cloud computing fact is that users do not need to upfront big financial investments, but pay-as-you-go for the resources needed and used [7].

With this increased demand, it is expected, naturally, that the relevance in cloud pricing will increase as well. With the development of the public cloud market, the resources provided by these companies were also expanded, but this resulted in part the providers' resources being underutilized. Monetizing these idle, underutilized resources became then a challenge. To address such challenges, AWS launched in November 2009 an auction mechanism to provide users with these temporarily unused virtual machines, named *spot instances* [8].

A. Spot instances

This work is concentrated in the field of preemptive virtual machines, most specifically AWS' *spot instances*, providing a preliminary review of how these machines can be used to run Function as a Service (Faas) platforms. This type of virtual machine has received a considerable amount of attention in the past decade after AWS started offering it in 2009.

What is most interesting about spot instances and their definition opposed to their on-demand equivalent is that their price can be up to 90% cheaper while AWS offers no Service Level Agreement (SLA) for them, meaning they can be terminated at any given time. An example of such price difference can be seen in Table I, extracted from [8]. When renting a virtual machine through their Elastic Cloud Computing (EC2) product, one can choose its pricing mode: on-demand and spot. In the first case, the EC2 instance has a fixed price and 99% of guaranteed availability, while in spot model there is no such guarantees but it offers the potential of high savings.

TABLE I: Comparison between on-demand instance prices and spot instance prices for Linux as the operating system, March 30, 2022, US East (Ohio), extracted from [8]

Instance Type	vCPU	Memory (GB)	Spot instances (per-hour)	On-demand (per-hour)
a.medium	1	2	\$0.0049	\$0.0255
a1.xlarge	4	8	\$0.0197	\$0.1020
m4.10xlarge	40	160	\$0.4245	\$2.0000
m5n.2xlarge	96	384	\$0.9578	\$5.7120

B. Function as a Service and funcX

Function as a Service (FaaS) is a paradigm that aims to simplify the deployment of applications. With this paradigm, the user registers a function, which is a piece of code with defined inputs and logic to be executed, into the FaaS system, and this function is executed by it returning generated outputs. The benefit of this paradigm is that users usually do not have to develop the physical aspect of the system (such as a computer or data center), neither the virtual one (virtual machines, containers) [12].

FaaS systems are offered by cloud providers usually in a pay as you go implementation that don't necessarily scale for big applications. With this challenge, *funcX* project was proposed to enable users to use FaaS in a distributed environment [12] [3].

Concerning the use of *funcX* service, it provides registration of remote systems by deploying a *funcX* agent, and users interact with it via a REST API that can be hosted virtually. It offers registration of remote systems and, with this, allows function registration and sharing that can be used by *funcX* clients.

The use case of FaaS is interesting to be explored with execution of spot instances. Since functions can be executed independently, if a large application can be broken down into smaller pieces (in this case, functions), then a net of spot instances can be used to deploy and execute such functions. This work is a preliminary investigation in running *funcX* using *spot instances*.

II. Previous Work

The papers that mainly inspired this research effort are: [7], [10], [12] and [3]. They are briefly detailed in this chapter for better context. As the writing of this paper, there is no knowledge of prior work citing deploying Function as a Service systems with preemptive virtual machines.

A. Deadline Constrained Bag-of-Tasks Dynamic Scheduler

The work in [7] presents Hibernation-Aware Dynamic Scheduler (*HADS*), a scheduler for efficiently allocating *spot instances* while taking advantage of their hibernation feature. This is an interesting aspect of *spots* because, instead of terminating the instance after the two minute termination notice, the VM's state is recorded into an EC2 Block Storage (EBS) and the user is only charged for the EBS memory usage during hibernation time. In this way, the VM can restart when it becomes available again from the point it got hibernated.

This work also explores Burstable VMs by proposing an extension to *HADS* name *Burst-HADS*, which not only considers the hibernation feature but also the burst capacity of burstable on-demand instances. This type of VM offers the same technical specs as regular instances, but with the potential 20% less costs but with controlled CPU power offered to users.

Both *HADS* and *Burst-HADS* focus on the execution of *Bag-Of-Tasks* (BOTs) applications, which the author defines as a task that can be divided in a set of independent and identical smaller tasks and thus can be run independently. This kind of application is ideal for *spot instances* because knowing the cost of a small task it is possible to infer the cost of all the tasks together, and they can be executed in a fault-tolerant manner.

The architecture of both frameworks is based on a coordinator-worker architecture standard. The coordinator defines the scheduling plan, chooses the VMs to be launched, request the resources to the provider and migrates tasks to other VMs. The workers operate as asynchronous applications working on the background of each VM, with the main responsibility to communicate to the coordinator any update in the VM's status.

The scheduler optimizes the execution to be ran in *spot instances* as much as possible, respecting a set of inputs defined by the user. In case a running spot hibernates and there is no other substitute available, then it allocates an on-demand VM so the total execution time will respect the input deadline.

The framework execution steps are defined below:

1) User provides input parameters, such as instance types that can be considered to be allocated, deadline constraint for total execution time, and others (defined in *env.json* and *job.json* files).
2) Then the initial scheduling map is initiated.
3) Finally, the VMs are allocated and workers are started.
4) In case of spot termination or hibernation prior to the deadline constraint, new VMs are scheduled.

HADS software implementation was used to run the experiments described in III.

B. Comparing SARS-CoV-2 Sequences using a Commercial Cloud with a Spot Instance Based Dynamic Scheduler

The work in [10] focused on executing a High Performance Computing (HPC) application with *spot instances* in AWS in comparison with the same execution but using on-demand instances. The application chosen was *MASA-OpenMP* [4], a parallel implementation of the Smith-Waterman algorithm for string comparison in instant time complexity. For the *spot instances* experiments, the application was also run using the *Burst-HADS* framework [9], a dynamic scheduler specialized in bag-of-tasks applications to run efficiently in the cloud minimizing costs given a deadline constraint.

This genome comparison is an application of great interest to be run in the cloud, for it mostly consists in a string comparison algorithm that be paralleled. Due to the covid-19 pandemic, comparing sars-cov DNA sequences became relevant for scientists to understand better the disease. According to the work, each DNA sequence needed to be compared to other 22,600 sequences (with 30,000 characters each), making it necessary to run 22,600 MASA-OpenMP executions to finish the task.

Apart from the number of concomitant executions, [10] also defined well one of the main problems for cloud end users: *unlike the advertising of cloud providers who advocate the ease of use of cloud environments as one of the main advantages, when considering all the necessary variables to be defined to execute a given application efficiently, a cloud can become a complex environment where any decision impacts directly in the final execution and respective monetary costs.*

For that reason, using the *Burst-HADS* framework [9] makes it easier for the end user on scheduling efficiently *spot instances* and what to do when they become unavailable. It also takes advantage of the hibernation feature of AWS [1] which made possible to handle hibernation easier in the

AWS environment. More details on the implementation of this framework is in chapter IV-A.

The execution model of this work inspired the methodology approach used in this paper, described more in depth in section III. Since there were 22,600 independent and very short tasks, running them in sequence would compromise the efficiency of the experiment. So *Burst-HADS* allows grouping tasks in a *supertask* to be executed, and since each task is very short, there is no need to use check points for this execution. The tasks in a super task are executed sequentially, and each of them is done in a few milliseconds.

The experiments in this work aimed for using the *spot instance's* hibernation feature, so the final set of VM types used were the C family, optimized for intense computing (c3, c4, c5), the M family (m3, m4 and m5) and the R family, optimized for massive data processing in memory (r3, r4 and r5).

Another important information is that everything was done in a single region (us-east-1) and using the x86_64 architecture. In each of these machine types, three executions of MASA-OpenMP were run to obtain the average time spent (in seconds) of a single genetic comparison (NC_045512.2 and MW240720.1) and its associate monetary costs in U.S. dollars.

This work presented final results for one task of DNA comparison and a bag of tasks divided in *supertasks*. For all the scenarios, using *spot instances* had a better price efficiency when comparing to on-demand only.

C. Serverless Supercomputing: High Performance Function as a Service for Science

The work in [3] is where *funcX* is presented. The main purpose of this project was driven by recent demands of different applications (Machine Learning, Data Analytics) for software to be more distributed and close to data, and also to be triggered by events such as the arrival of new data. The authors cite challenges to be overcome with a distributed FaaS platform, including: complex and expensive infrastructure, slow and unreliable network communication, security challenges, inflexible authorization and authentication models and unpredictable schedule delays. This caused monolithic applications to be decoupled into smaller and independent parts that can be here defined as functions, enabling a more efficient execution of the whole system.

FuncX is thus proposed as a high performance function-as-a-service (FaaS) open-source project. This project enables distributed, high performance function execution that can be deployed in different infrastructure types, such as cloud computing (in the case of this work, spot instances), but also supercomputers and clusters (used in different research fields). Users can register functions to be executed by using the Python programming language, which can be deployed independently from of the physical location where it will run.

The authors present a brief survey of commercial FaaS platforms, such as AWS Lambda, Google Functions and Azure Functions. While these platforms present good integration with other resources within the same platform such as file systems, monitoring and others, as applications scale in data intensiveness they bring pricing models challenges and oftentimes its deployment is tied to that same platform. Because of this, open-source platforms such as *funcX* offer more flexibility since they can be deployed on-premise and are not tied to a specific pricing model.

The survey also includes open-source projects, such as *Apache Open Whisk*, *Fn* and *Kubeless* and academic platforms, *SAND* and *ABACO*. They all are Docker or Kubernetes-based when deployed externaly, which can bring additional development overhead. When compared with these platforms, *funcX* provides more flexibility of deployment for not being tied to Docker, being able to be deployed locally and not attaches to a specific pricing model. As a disadvantage, it only allows function implementation in the Python programming language, while other platforms also support other popular languages.

Concerning the authentication and authorization part, it uses Globus Auth [11], an authentication and authorization academic platform that supports different applications from the research community. It is used to outsource the user/client authentication process for the project.

The *funcX* service consists in a registry of endpoints and registered functions. The service provides a REST API to register and manage functions, as well as setup, initiate and terminate endpoints so users can execute such functions. Interactions between users and registered functions are done between REST APIs.

In other to prove optimization, automation, fault-tolerance and scalability, the authors investigated and described use cases for scalable metadata extraction, machine learning inference as a service, synchrotron serial crystallography, neuroscience, and correlation spectroscopy. This work showed the overall benefit of using *funcX* as a FaaS platform, because it is not only has similar performance as other solutions in the market but it also offers great flexibility for different use cases.

D. Real-time HEP analysis with funcX, a high-performance platform for function as a service

The work in [12] is one of the references listed in *funcX*'s website [5]. It presents a detailed use case of the project for High-Energy Physics (HEP), which requires the processing of massive volume of data.

The use of *funcX* in this case was favourable due the the following advantages:

- *funcX* accelerated software development by facilitating the break down of large applications into smaller modules (functions). This was even further optimized when scheduling compute resources, for some functions were appropriate to run with CPU while others with GPU.
- The *funcX* endpoints provided a common interface to registered functions and resource allocation was scaled according to demand.
- The possibility of using containers offer uniform environments, reducing reproducibility errors of scientific results.

- Globus Auth, used by *funcX* to outsource its Authorization and Authentication flow, facilitated authentication for the experiments.
- As demonstrated in [3], function execution is low-latency, which optimizes the analysis execution performed in Jupyter Notebooks.

As part of this work, the authors provided some code listings, which helped the development of the function definition code in python programming language as illustrated in Code Listing 1, which was used in all the experiments in subsection II-C.

Apart from the function code definition, a backend project is also described for Coffea, a HEP analysis framework, which provides tools for data visualization, analysis and correction. The project makes processing of event data in parallel due to *funcX*, enabling real-time data analysis. Since all the processes use the same registered function, this process can be easily scaled.

III. Methodology

This chapter will describe how the work was developed and the preliminary results obtained for deploying a FaaS system using AWS spot instances. Overall, the methodology followed up to this point is similar to in [10]: the experiments are executed with *spot instances*, and their final cost is compared to what it would have been if executed with on-demand prices.

It is important to emphasize that this is still a work in progress and that more experiments and investigations will be conducted for this work.

A. Tests with HADS to execute MASA-OpenMP

Here are defined the replicated experiments as described in [10]. It consists in executing the application of *MASA-OpenMP* using the *Burst-HADS* framework, but we limited the number of DNA sequence comparison tasks to only one. The executed code can be found in [2].

B. Definition of input sets

The input sets for executing *HADS* are the following:

1) Accepted machine types (M)
2) Deadline constraint (maximum amount of time to complete the application). (D)

For defining the instance type input set M, the choice of the instance types was based on the experiments conducted in [7], which focused on the C instance family due to its better availability in the execution region (*us-east-1*) and reasonable pricing. It was also not included the possibility of hibernation in this set of experiments since it was not relevant for this particular initial investigation.

The definition of D was arbitrary, as the main focus in this step was to understand the framework's behavior and financial results. The value used was the default provided in the implementation (5 minutes), which was more than enough for a simple execution of MASA-OpenMP.

So, summarizing the experiments performed: *HADS* was used to schedule five different executions, each of them

Instance Type	vCPU	Memory (GiB)	On-demand hourly rate
c4.large	2	3.75	$0.10
c4.xlarge	4	7.5	$0.199
c4.2xlarge	8	15	$0.398
c4.4xlarge	16	30	$0.796
c4.8xlarge	36	60	$1.591

TABLE II: VM attributes used in the MASA-OpenMP experiments

consisting in a single DNA comparison task and in a different AWS instance type (*c4.large*, *c4.xlarge*, *c4.2xlarge*, *c4.4xlarge* and *c4.8xlarge*).

C. Testing with funcX endpoints

This set of experiments with *funcX* is to execute the same BoT application of DNA comparison as performed in [10], but using the FaaS computing paradigm. To perform this, a function is defined to be executed.

To deploy *funcX* using spot instances, the methodology followed was this:

1) A remote virtual machine in *spot* billing mode is set up to start and host a new *funcX* endpoint.
2) The defined DNA comparison function is then executed by client calling the endpoint.
3) Total time to execute the function is counted, and an extra 45s are added to the total execution time to account for the starting and terminating period of the machines (empirically calculated).

An example of a function to be executed with *funcX* can be defined as in Code Listing 1.

Code Listing 1: Python file to execute a function in a funcX endpoint

```
from globus_compute_sdk import Client
from globus_compute_sdk import Executor

end_id=<ENDPOINT ID>

file1=<DNA SEQUENCE 1>.fasta
file2=<DNA SEQUENCE 2>.fasta

directory=<DIRECTORY TO EXECUTE MASA>

Client()

def masa_function(command):
    import subprocess
    import os
    os.chdir(directory)
    r = subprocess.run(command.split())
    return r.returncode

with Executor(endpoint_id=end_id) as gce:
    cmd="./masa-openmp file1 file2"
    future=gce.submit(masa_function, cmd)
    print(future.result())
```

IV. Results and discussion

A. Executing MASA-OpenMP with HADS

The results for the execution of MASA-OpenMP with *HADS*, referred in subsection III-A, are shown in Table III. All results are for a single execution only. For each instance type, their technical specification are presented, as well price breakdown by scheme (on-demand and *spot*), as well as the total experiment time (including waiting for scheduling) and the estimated cost given by *HADS*.

TABLE III: MASA-OpenMP execution with HADS results

Instance Type	On-demand hour rate	Total time	Estimated cost	Estimated spot hour rate
c4.large	$0.100	2min43s	$0.0018	$0.0398
c4.xlarge	$0.199	2min30s	$0.0040	$0.0960
c4.2xlarge	$0.398	2min37s	$0.0087	$0.1995
c4.4xlarge	$0.796	3min07s	$0.0217	$0.4178
c4.8xlarge	$1.591	3min18s	$0.0227	$0.4127

Time elapsed and estimated cost were data provided by *HADS* software. The time elapsed for all five results are close, the minimum being 2 minutes and 30 seconds, while the maximum being 3 minutes and 18 seconds. The estimated cost increases with the machine size, as expected, since the smaller machines tend to cost less. Since the spot hour rate is variable, the estimated spot hour rate column was calculated by $spot\ hour\ rate = (3600/total\ time\ in\ seconds) * estimated\ cost$. These same values were used in subsection IV-B as reference for comparison between HADS and *funcX*.

One interesting aspect in Table III is that the estimated spot instance hour rate for c4.8xlarge instance is slightly smaller than for c4.4xlarge instance. Since the price of spot instances can vary throughout the day and only one experiment was performed for each instance type, it is not possible to state if this pattern would always be observed, but this may have happened due to a lower demand in that particular time for c4.8xlarge in comparison to c4.4xlarge in the *us-east-1* region.

Regardless of the machine size, the *spot* cost was always smaller than the equivalent on-demand price, indicating that for this particular task *spot instances* have a cost advantage.

B. MASA-OpenMP execution with funcX endpoint

This experiment's goal was to do a simple test to analyze the execution of MASA-OpenMP with a *funcX* endpoint. This consisted in deploying a *spot instance*, varying the instance type using the same set from subsection IV-A, and executing this function by calling this endpoint from a personal computer.

The total execution time is the sum of the function execution, and an additional 45s to account for instance initialization time (around 15s) and instance termination (around 30s). Both the instance initialization and termination time were measured manually, and as an improvement for future experiments this measurement will be automatized.

Another limitation to this experiment was that both the spot instance and the personal computer had to authenticate with *Globus Compute* as mentioned in subsection II-C. The authentication work in both machines was also done manually, and the time required by it was not added to the total executed time. Ideally this part should also be automatized, which could change the total execution time but not significantly. Still, for more precise results this will be automatized in future experiments.

TABLE IV: MASA-OpenMP execution with funcX

Instance Type	On-demand hour rate	Spot hour rate	Total time	Estimated cost
c4.large	$0.100	$0.040	48.02s	$0.0005
c4.xlarge	$0.199	$0.096	47.66s	$0.0013
c4.2xlarge	$0.398	$0.199	47.42s	$0.0026
c4.4xlarge	$0.796	$0.418	47.43s	$0.0055
c4.8xlarge	$1.591	$0.413	47.48s	$0.0054

V. Conclusion and Future Work

This analysis is a work in progress, but at this moment there is already some insightful data. The time to implement *funcX*'s setup was considerably easier than the one with HADS, and scheduling a spot instance with a *funcX* endpoint is a relatively more straightforward task. There were some limitations to the experiments with *funcX* as mentioned in section IV: the time to initiate and terminate an instance were manually measured and added to the total execution time, while the authentication and endpoint setup work were also done manually but not accounted for. These measurements will be automatized in setup scripts so data can be more precise in future results, however the authentication and endpoint setup time would not have increased the total time significantly.

Based on the current data, we here conclude that *funcX* has a potential to offer more flexibility, more straightforward implementation and potentially less monetary costs. For a more thorough analysis, though, we want to explore other applications and scenarios.

A. Next steps

Given the presented conclusions, we will follow these next steps in this research work:

- Automation for the following measurements: initialization and termination of instance in AWS in the *funcX* tests;
- Perform another BoT test with MASA-OpenMP but this time with more tasks in parallel, such as in [10]. We can explore the scability feature of *funcX* by varying the number of clients executing a similar function and the number of endpoints, and comparing the results with executing multi-task with HADS;
- Contribute with HADs to schedule uninterruptible *funcX* endpoints. HADs has the potential of improving cost efficiency of using *funcX* since user can input a set of

accepted instances and HADs schedules and terminates instances in the most efficient manner, however now only for a deadline constrained tasks;

- Although the focus of this work was in investigating the spot instance usage, it would be interesting to compare the costs of *funcX* experiments with similar applications running with AWS Lambda, AWS's commercial FaaS platform product.

Acknowledgment

The authors would like to thank Luan Teylo Gouveia Lima, for your immense support with the experiments with HADS and for always being available for questions, and the São Paulo Research Foundation (Fundação de Amparo à Pesquisa do Estado de São Paulo - FAPESP) for funding project 19/26702-8.

References

[1] A AWS. *Amazon EC2 Spot Lets you Pause and Resume Your Workloads*. 2017.

[2] *Burst-HADS source code*. URL: https://github.com/luanteylo/hads_.

[3] Ryan Chard et al. *Serverless Supercomputing: High Performance Function as a Service for Science*. 2019. arXiv: 1908.04907 [cs.DC].

[4] Edans F. De O. Sandes et al. "MASA: A Multiplatform Architecture for Sequence Aligners with Block Pruning". In: *ACM Trans. Parallel Comput.* 2.4 (Feb. 2016). ISSN: 2329-4949. DOI: 10.1145/2858656. URL: https://doi.org/10.1145/2858656.

[5] *Federated function as a service*. Accessed: 2023-06-09. URL: https://https://funcx.org/.

[6] Gareth George et al. "Analyzing AWS Spot Instance Pricing". In: *2019 IEEE International Conference on Cloud Engineering (IC2E)*. 2019, pp. 222–228. DOI: 10.1109/IC2E.2019.00036.

[7] Luan Teylo Gouveia Lima. "Scheduling Deadline Constrained Bag-of-Tasks in Cloud Environments using Hibernation prone Spot Instances". PhD thesis. Fluminense Federal University, 2021. URL: https://site.ic.uff.br/teses-e-dissertacoes/#tab-102701.

[8] Liduo Lin, Li Pan, and Shijun Liu. "Methods for improving the availability of spot instances: A survey". In: *Computers in Industry* 141 (2022), p. 103718. ISSN: 0166-3615. DOI: https://doi.org/10.1016/j.compind.2022.103718. URL: https://www.sciencedirect.com/science/article/pii/S0166361522001154.

[9] Luan Teylo et al. "A dynamic task scheduler tolerant to multiple hibernations in cloud environments". In: *Cluster Computing* 24.2 (2021), pp. 1051–1073.

[10] Luan Teylo et al. "Comparing SARS-CoV-2 Sequences using a Commercial Cloud with a Spot Instance Based Dynamic Scheduler". In: *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 2021, pp. 247–256. DOI: 10.1109/CCGrid51090.2021.00034.

[11] Steven Tuecke et al. "Globus Auth: A research identity and access management platform". In: *2016 IEEE 12th International Conference on e-Science (e-Science)*. IEEE. 2016, pp. 203–212.

[12] Anna Elizabeth Woodard et al. "Real-time HEP analysis with funcX, a high-performance platform for function as a service". In: *European Physical Journal Web of Conferences*. Vol. 245. European Physical Journal Web of Conferences. Nov. 2020, 07046, p. 07046. DOI: 10.1051/epjconf/202024507046.