Derivative IaaS Layer Utilizing Low Priority Server Instances for Web Applications With State Persistence

Ljiljana Milješić*, Miroslav Zarić**
*University of Novi Sad, Faculty of Technical Sciences, Novi Sad,
** University of Novi Sad, Faculty of Technical Sciences, Novi Sad
ljialjanamiljesic@gmail.com, miroslavzaric@uns.ac.rs

Abstract— **Cloud providers offers plenty of choice to their potential customers, spanning all standard types of services IaaS, PaaS, SaaS, FaaS, BPaaS... Sometimes it is quite hard to properly determine someone's exact needs and evaluate the total cost of cloud-based solution, and even harder to compare if that cost, in a long run, is comparable to the cost of the on-premise solution. If we concentrate at the virtual machine market, all major cloud providers (Amazon WS, MS Azure, Google, IBM…) offer on demand instances of different computational characteristics, dedicated instances for demanding customers, but entry level configurations that comes with some strong limitations, are also available. Their price may be very attractive to small and medium enterprises, one that are usually struggling with internal IT support, and the one that may be best served by managed service offered by the cloud providers. Those entry level instances usually represent virtual machines created and run on spare capacity of cloud computing hardware that currently is not used for on demand services. Since the lifespan of these instances is pretty unpredictable, they are usually best utilized for applications that do the batch processing and can easily withstand loss of computing capacity. This paper discuss the possibility of using these entry level instances even for a bit more demanding applications, concentrating on Amazon Web Services spot instances.**

I. INTRODUCTION

Cloud computing has become ubiquitous feature and a success story of modern day computing. Lately, besides standard infrastructure as a service (IaaS), platform as a service (PaaS), software as a service (SaaS) [1], several other approaches are gaining traction such as: container as a service, function as a service, business process as a service, and other more specialized or tailor-made services.

In each of these offerings, the greatest advantage for potential customers is that they are relieved of operational management of the hardware and/or software layers. They can easily concentrate on utilizing whatever platform and software they need to drive their business, while cloud providers are taking care of hardware replenishment, OS upgrades, software patches and version upgrades.

But every service comes at a price, and in case of cloud computing it may be quite hard to estimate what the final cost, over prolonged period of time, may be. It gets even worse if one tries to do the comparison with total cost of ownership (TCO) of the similar system(s) on premise.

But cloud providers usually argue that such comparison is probably deficient anyway since hidden cost of ownership may easily be overlooked. And cloud approach has additional strength – it is offering pay-as-you-go payment model, while on premise solutions usually requires large upfront investment (or at least leasing contract or credit structure to meet large investment needs event before the business has started).

One of the most common usages of cloud services is leasing the computational power, usually in a form of virtualized computing environment. Such a service usually is delivered as an IaaS offering of the cloud providers. Several models exist that are common to large cloud providers (Amazon Web Services [2], Microsoft Azure [3], Google Cloud [4], IBM Cloud (ex. SoftLayer) [5]. Virtualization and usage of Virtual Machine (VM) images makes this offering extremely easy to utilize [6], since an instance of a VM may be provisioned in a fairly short time.

Although several tenants usually are running their virtualized environments alongside each other, using the same host server, dedicated server instances are also available, and give customers full control. However, they are usually the most expensive contracts since cloud provider cannot share resources with other cloud customers.

On-demand model, probably most popular, gives the customers flexibility and appropriate computational power needed in any given moment. In this model, the resources are allocated to satisfy current load. Customer pays for the resources utilized over a given period of time. This way customers can choose to run the configuration they need on a timeframe they need and pay only for actual usage. However, even this model can be quite expensive if prolonged term of running the high-end configurations are used. In such case, other models, such as contract commitment, where infrastructure is leased for the whole contract duration, may serve better.

However, on-demand model, as one of the most used, usually leaves some resources free, unutilized over a period of time. Idle resources are practically losing money for cloud providers. So, it's not a surprise that cloud providers are offering these spare resources at a fraction of the price of on-demand services.

In Amazon Web Services these instances are called *spot instances*, Google calls them *pre-emptive VMs*, and Azure has introduced them as a *Low priority VMs*. All of them comes with a limitation - in case that on-demand (more expensive) services require more resources, these low

priority instances will be taken offline, and their resources will be allocated to the on-demand computing instances.

Even with such a rigorous limitation, these instances may be well used for some purposes such as distributed calculations, batch processing tasks and similar – the processing that can easily survive the loss of some instances (computing nodes).

However, their small price tag may be very attractive for small and medium enterprises (SME), if they could design their application to utilize these instances the best possible way. The aim of this paper is to show one possible solution for utilization of these low-cost, low-priority instances, that may, in some cases, be beneficial for SME, since those types of enterprises may find the costs of IT infrastructure and IT organization to support their day-to-day businesses too much burden for financial success of their businesses.

II. CONTAINERS AS A DEPLOYMENT OPTION

Besides virtualization, that creates instances of virtual machines, the containers have become increasingly popular approach for deploying applications. Basically, they offer the form of virtualization, but at a higher level than standard virtual machines, virtualization of a user-space. In this case several instances of containers share the OS platform. A container should provide everything that is required to successfully run deployed application, i.e. application code, runtime environment, third party tools and libraries, all conveniently packed as a single deployable package.

This package than may be easily transferred from one machine to another (or VM) capable of running the container. This way virtualization overhead is lower (you don't have to install OS as a part of the deployment), but also the isolation level is currently still debated as being less than the isolation level provided between VMs.

For many usage scenarios containers provide a convenient way for application distribution. The most popular container platform is Docker [7], and on cloud services usage of Kubernetes[8] container orchestration platform is gaining momentum.

Introduction of Checkpoint and Restore in UserSpace (CRIU) capabilities have given the container capabilities to support robust application that can be stopped, their state persisted and restored on restarting the container (on same or different machine). The solution described in this paper is relying on this feature to provide expected behavior of the application in moments when a necessary switch to another host is performed. CRIU was first introduced in Virtuozzo, and later introduced as part of the Docker. The basic functionality that CRIU provides is ability to stop the running application, persist its current state (practically creating a snapshot of everything in its address space), and ability to transfer this application image on another machine and restoring it from this snapshot. It is obvious that this approach is well suited for migration of application between server instances, and therefore may provide a mean for migrating application from the spot instance that is going to be revoked to another server. Amazon WS has, in 2015, introduced Spot Fleet API, that enables bidding process for the same instance type. However, if all the instances available are of the same type, that will not shield a client's application from sudden removal of the instance. In order to support running "classic" web application on spot instances, a medium layer of derivative IaaS needs to be created between IaaS provider and client. In such solution client will receive near on-demand experience, while running cheaper spot instances.

III. LEVERAGING SPOT INSTANCES FOR CREATION OF DERIVATIVE IAAS LAYER

Spot instance provisioning on AWS (and similarly on other cloud offerings) act as an auction mechanism, where users are bidding to acquire appropriate spot instances. Therefore, in any given moment, the price of running spot instance is determined by the current demand for the given server type.

Since spot instances are usually traded for the fraction of the price of on-demand instances, it is argued here that good planning of server instance types and well-thought bidding strategy for spot instances may provide a running environment capable of running desired application at very low price.

Appropriate management of spot instances and their provisioning, with migration of application containers from one instance to another (to accommodate for price change), may create a new, derivative layer of server instances we can choose from at any given moment, keeping our overall price as low as possible, but maintaining the capacity needed to run the applications. Of course, provisioning of the spot instances and container migration must be performed at appropriate times to ensure uninterrupted running of application(s).

In order to achieve this, the creation of the mixed system containing several types of instances (spot and on-demand) is proposed [9]. The system contains master management server, several instances of worker servers, backup servers and elastic load balancer.

IV. PROPOSED MANAGEMENT SYSTEM FOR SPOT INSTANCES

A. System Architecture

Management Server is deployed as an on-demand server instance and is running Python application acting as a management node. This application is the core of the system, and is always aware of currently running workers, their instance types, assigned IPs, and available backup instances to be used when next spot instance revocation signal is received.

Worker Servers is a spot instance server running desired application in a container. Management application (daemon) decides which instance type meets the minimum requirements for worker server. Worker servers are grouped, so several servers of the same type forms one pool. Obviously, it is a good strategy not to have only one spot instance type across server fleet, because this would increase the probability that all servers would receive revocation signal at the same time, or at a very short interval. With several different spot instance types, that are also acquired at different bidding price, the probability that all servers would be revoked at the same time is lower.

Backup servers are proposed to be on-demand instances, with minimum configuration needed to run the application. The number of backup servers should be the same as the number of workers, but some optimization is also possible, since the worst-case scenario, where all workers would be revoked is highly unlikely, due to the strategy of spot

instance provisioning. Although these servers are on-demand, their total price is very low since most of the time they are not active (on-demand instances are payed only for running time). The architecture of the proposed system is given in Figure 1.

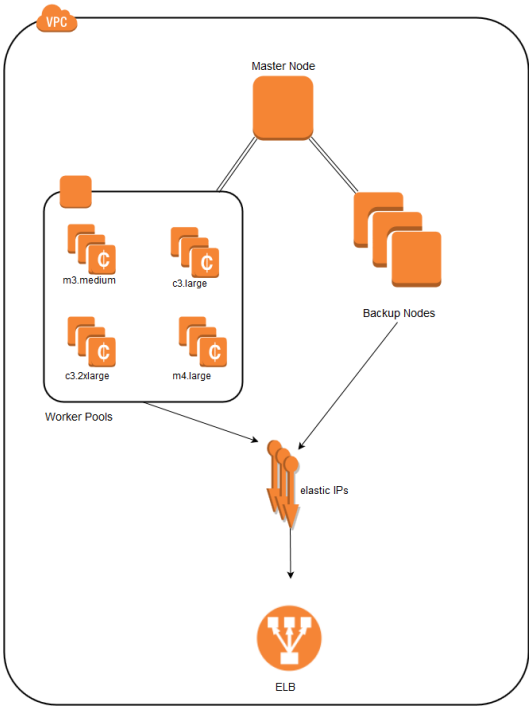

Figure 1. The proposed system architecture.

B. System Dynamics

The system configuration needs to be flexible, but some presumptions are necessary prior to the first start of master daemon, such as minimum desired number of workers (and based on that, indicative number of backup servers). On a first startup the management server will acquire necessary spot instances to instantiate appropriate number of workers. To be truly in the spirit of IaaS and cloud computing, the number of spot instances later should be adaptable, to accommodate for dynamic changes in application load.

When spot instance (*worker server)* receives withdrawal (revocation) signal, it has 2 minutes to shed the load. Since most of the instances are bided at optimum possible price, more instances of the same type will probably receive notification at the same time or in a short period. In this moment, worker takes a snapshot of the container application, while master node is preparing a backup instance to accept the container.

Master node checks if available spot instances, ready to receive the application, exist.

- Checks if there are spot instances available that were bided for a (slightly) larger price than the one being revoked (Worker B)
  - If no spot instances are available, master node wakes up backup instance and after the backup server has started, master node sends a message informing the backup server of the worker whose load should be transferred. In this case backup server becomes additional worker (Worker B)
- Master then manages and monitors message exchanges between old worker instance (Worker A) and its replacement (Worker B)
- Worker A creates snapshot image, Worker B server receives image and starts application
- Master checks if replacement server Worker B is assigned Elastic IP of original Worker A

After this Worker A is no longer active node, and may stop the application, or even be outright stopped by the cloud management system.

Additionally, to keep the cost of the operations at the minimum, the master node will immediately start the bidding procedure to acquire a new spot instance. Although at this point we may be certain that new spot instance will come at a higher price than the one that was revoked, it is still very possible that we could acquire new instance for lower price than the price of running on demand backup instance. If the price on the spot instance market is currently higher than the price of on-demand instance we are currently using, than master node will not try to provision it.

V. Proposed System Evaluation

A small web shop application was created that requires state persistence to track customers shopping cart. The application was packaged for container deployment. The requirement for the system has been set up so that only server instances with 4 or more CPU cores and with 8 or more GB of RAM are chosen during the bidding period. Tests were used to give answer to two critical questions:

1. Whether the system is capable of migrating containers successfully in a given timeframe?
2. If the overall price of the system would justify it?

Two system setups have been configured for test runs in a period of 78 days (April - Jun 2017).

1.System consisting of one master node, one worker and one backup instance. Here practically only one worker pool exists. The type of the spot server instance chosen to meet the requirements is r4.xlarge, from AWS us-east-1e zone/datacenter. Price change for this spot instance is given in Figure 2. During the test period, the price of this instance has risen over the bid spot instance price, and this instance has been revoked. In subsequent bid, the same type of spot instance has been provisioned for a slightly higher price, but in AWS us-east-1d zone/datacenter. However, later during the trial run, the price of this instance has also risen over the price the system has bid, and the instance was once more revoked. In a subsequent auction, master server obtained the new server instance, of type i3.xlarge from us-east-1a zone/datacenter, and it was not revoked till the end of test period.

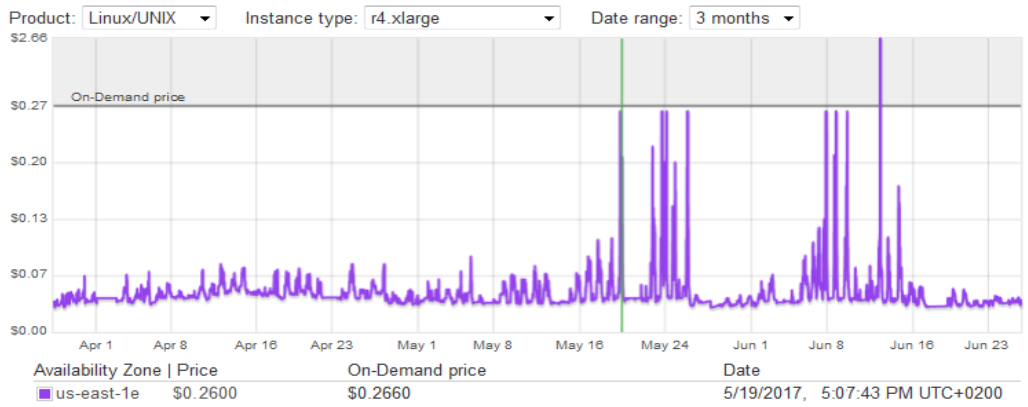

Figure 2. Price dynamics for first test system spot instance type.

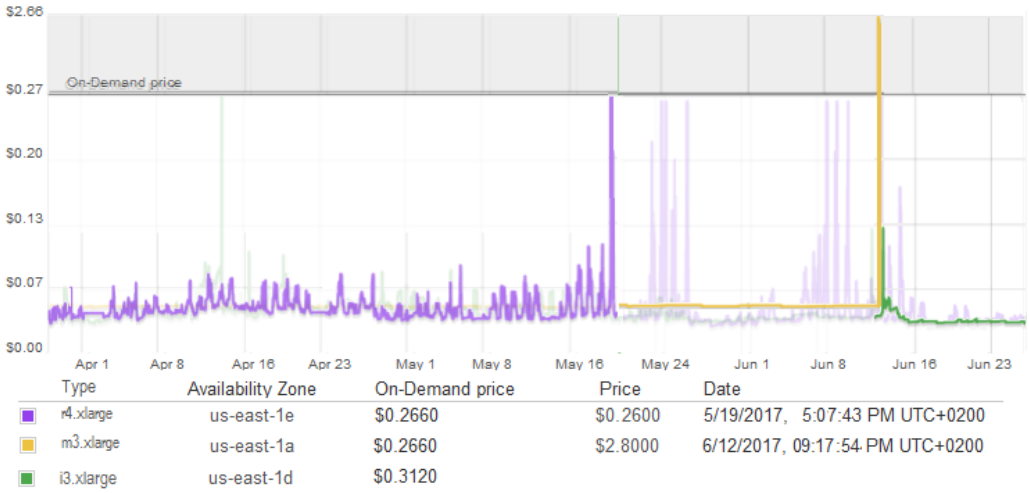

Figure 3. Price dynamics for second test system spot instance pool.

2. The second system setup consisted of one master node, two pools with two worker servers, and 4 backup servers. During the test period, just like in a first system setup, the system has gone through two revocation and auction cycles. Price change and instance type dynamic of the first of two worker server pools is given in Figure 3.

After the given period total cost were summed, and final price comparison with cheapest appropriate on-demand instance was made.

For the first system setup total cost of the cheapest appropriate on-demand instance would, for the given period, amount to 388.032 $, while costs of the system running the spot instances amounted to 120.5558 $. Therefore, the proposed system offered saving up to 68.93%.

For the second system setup total cost of running the cheapest 4 instances of on-demand servers would, for the given period, amount to 1552.128 $, while for the system running two pools of spot instances total costs were 393.612 $. Again, the proposed system offered savings, this time up to 74.64 %.

VI. Conclusions

Test systems were successfully run for 78 days. According to the then current state of spot instance prices proposed systems were able to offer significant savings. Furthermore, during trial run, the systems showed capabilities to transfer application load from one spot instance to another in a short time, and existence of backup servers made this transition transparent to the users, that may have noticed only a smaller delay if they were interacting with the site exactly in the moment when elastic IPs were transferred from one server to another. However, further testing for heavier application loads would be reasonable, to test if the servers maintain their capabilities of making state snapshots and transferring them to another server in the given timeframe. Nevertheless, these tests gave the proof of concept, and with careful planning of application containers and utilization of CRIU concepts, this approach promises possible large savings (dependent on the current state of server instance market), that could be very interesting for SME wishing to keep their IT infrastructure costs at bay.

REFERENCES

[1] B. Sosinsky, "Cloud Computing Bible", *John Wiley & Sons, 2010*

[2] Amazon Web Services, available at: https://aws.amazon.com

[3] Microsoft Azure, Cloud offerings, available at: https://azure.microsoft.com/en-us/

[4] Google Cloud (Platform), available at: https://cloud.google.com

[5] IBM Cloud, available at: https:// ibm cloud gpus.

[6] Daniel A. Menascé, "Virtualization: Concepts, Applications, and Performance Modeling", available at: https://cs.gmu.edu/~menasce/papers/menasce-cmg05-virt-slides.pdf

[7] Docker containerization platform, available at: https://www.docker.com/what-docker.

[8] Kubernetes, Open Source Container Orchestration Platform, available at: https://kubernetes.io

[9] Lj. Milješić, „AWS Spot Server Instance Allocation Management For Stateful Applications", master thesis, Library of Faculty of Technical Sciences, University of Novi Sad, 2017.