Journal of Network and Computer Applications 65 (2016) 167–180

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

A reliable and cost-efficient auto-scaling system for web applications using heterogeneous spot instances

Chenhao Qu*, Rodrigo N. Calheiros, Rajkumar Buyya

Cloud Computing and Distributed Systems (CLOUDS) Laboratory, Department of Computing and Information Systems, The University of Melbourne, Australia

ARTICLE INFO

Article history:
Received 17 September 2015
Received in revised form
2 December 2015
Accepted 1 March 2016
Available online 5 March 2016

Keywords:
Cloud computing
Auto-scaling
Web application
Fault tolerant
Cost
QoS
Spot instance

ABSTRACT

Cloud providers sell their idle capacity on markets through an auction-like mechanism to increase their return on investment. The instances sold in this way are called spot instances. In spite that spot instances are usually 90% cheaper than on-demand instances, they can be terminated by provider when their bidding prices are lower than market prices. Thus, they are largely used to provision fault-tolerant applications only. In this paper, we explore how to utilize spot instances to provision web applications, which are usually considered as availability-critical. The idea is to take advantage of differences in price among various types of spot instances to reach both high availability and significant cost saving. We first propose a fault-tolerant model for web applications provisioned by spot instances. Based on that, we devise novel cost-efficient auto-scaling polices that comply with the defined fault-tolerant semantics for hourly billed cloud markets. We implemented the proposed model and policies both on a simulation testbed for repeatable validation and Amazon EC2. The experiments on the simulation testbed and EC2 show that the proposed approach can greatly reduce resource cost and still achieve satisfactory Quality of Service (QoS) in terms of response time and availability.

1. Introduction

There are three common pricing models in current Infrastructure-as-a-service (IaaS) cloud providers, namely *on-demand*, in which acquired virtual machines (VMs) are charged periodically with fixed rates, *reservation*, where users pay an amount of up-front fee for each VM to secure availability of usage and cheaper price within a certain contract period, and the *spot*.

The spot pricing model was introduced by Amazon to sell their spare capacity in open market through an auction-like mechanism. The provider dynamically sets the market price of each VM type according to real-time demand and supply. To participate in the market, a cloud user needs to give a bid specifying number of instances for the type of VM he wants to acquire and the maximum unit price he is willing to pay. If the bidding price exceeds the current market price, the bid is fulfilled. After getting the required spot VMs, the user only pays the current market prices no matter how much he actually bids, which results in significant cost saving compared to VMs billed in on-demand prices (usually only 10% to 20% of the latter) (http://aws.amazon.com/ec2/spot-instances/). However, obtained spot VMs will be terminated by cloud provider whenever their market prices rise beyond the bidding prices.

Such model is ideal for fault-tolerant and non-time-critical applications such as scientific computing, big data analytics, and media processing applications. On the other hand, it is generally believed that availability- and time-critical applications, like web applications, are not suitable to be deployed on spot instances.

Adversely in this paper, we illustrate that, with effective fault-tolerant mechanism and carefully designed policies that comply with the fault-tolerant semantics, it is also possible to reliably scale web applications using spot instances to reach both high QoS and significant cost saving.

Spot market is similar to a stock market that, though possibly following the general trends, each listed item has its distinctive market behavior according to its own supply and demand. In this kind of market, often price differences appear with some types of instances sold in expensive prices due to high demand, while some remaining unfavored leading to attractive deals. Fig. 1 depicts a period of Amazon EC2's spot market history. Within this time frame, there were always some spot types sold in discounted prices. By exploiting the diversity in this market, cloud users can utilize spot instances as long as possible to further reduce their cost. Recently, Amazon introduced the Spot Fleet API (https://aws.amazon.com/blogs/aws/ new-resource-oriented-bidding-for-ec2-spot-instances/), which allows users to bid for a pool of resources at once. The provision of resources is automatically managed by Amazon using combination of spot instances with lowest cost. However, it still lacks fault-tolerant capability to avoid availability

* Corresponding author.

http://dx.doi.org/10.1016/j.jnca.2016.03.001
1084-8045/

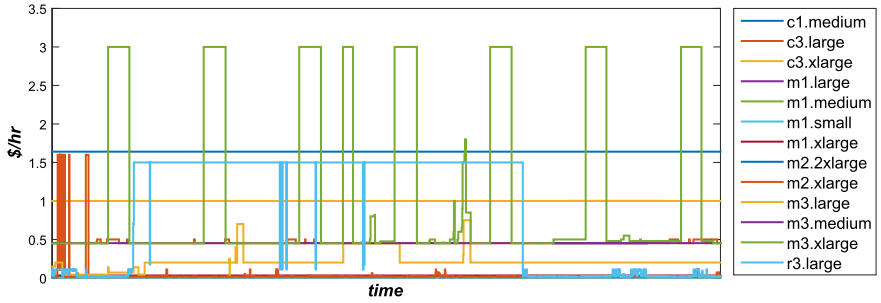

Fig. 1. One week spot price history from March 2nd 2015 18:00:00 GMT in Amazon EC2's ***us*−*east*−*1d*** Availability Zone.

and performance impact caused by sudden termination of spot instances, and thus, is not suitable to provision web applications.

To fill in this gap, we aim to build a solution to cater this need. We proposed a reliable auto-scaling system for web applications using heterogeneous spot instances along with on-demand instances. Our approach not only greatly reduces financial cost of using cloud resources, but also ensures high availability and low response time, even when some types of spot VMs are terminated unexpectedly by cloud provider simultaneously or consecutively within a short period of time.

The *main contributions* of this paper are

- a fault-tolerant model for web applications provisioned by spot instances;
- cost-efficient auto-scaling policies that comply with the defined fault-tolerant semantics using heterogeneous spot instances;
- event-driven prototype implementations of the proposed auto-scaling system on CloudSim (Calheiros et al., 2011) and Amazon EC2 platform;
- performance evaluations through both repeatable simulation studies based on historical data and real experiments on Amazon EC2;

The remainder of the paper is organized as follows. We first model our problem in Section 2. In Section 3, we propose the base auto-scaling policies using heterogeneous spot instances under hourly billed context. Section 4 explains the optimizations we proposed on the initial polices. Section 5 briefly introduces our prototype implementations. We present and analyze the results of the performance evaluations in Section 6 and discuss the related works in Section 7. Finally, we conclude the paper and vision our future work.

2. System model

For reader's convenience, the symbols used in this paper are listed in Table 1.

2.1. Auto-scaling system architecture

As illustrated in Fig. 2, our auto-scaling system provisions a single-tier (usually the application server tier) of an application using a mixture of on-demand instances and spot instances. The provisioned on-demand instances are homogeneous instances that are most cost-efficient regarding the application, while spot instances are heterogeneous.

Like other auto-scaling systems, our system is composed of the *monitoring* module, the *decision-making* module, and the *load balancer*. The monitoring module consists of multiple independent monitors that are responsible for fetching newest corresponding system information such as resource utilizations, request rates, spot market prices, and VMs' statuses into the system. The decision-making module then makes scaling decisions according to the obtained information based on the predefined strategies and policies when necessary. Since in our proposed system provisioned virtual cluster is heterogeneous, the load balancer should be able to distribute requests according to the capability of each attached VM. The algorithm we use in this case is *weighted round robin*.

The application hosted by the system should be stateless. This restriction does not reduce the applicability of the system as modern cloud applications are meant to de developed in a stateless way in order to realize high scalability and availability (Wilder, 2012). In addition, stateful applications can be easily transformed into stateless services using various means, e.g., storing the session data in a separated memcache cluster.

Table 1
List of symbols.

Symbol	Meaning
$\mathbf{T}$	The set of spot types
M_{min}	The minimum allowed resource margin of an instance
M_{def}	The default resource margin of an instance
Q	The quota for each spot group
R	The required resource capacity for the current load
F_{max}	The maximum allowed fault-tolerant level
f	The specified fault-tolerant level
O	The minimum percentage of on-demand resources in the provision
S	The maximum number of selected spot groups in the provision
r_o	The resource capacity provisioned by on-demand instances
s	The number of chosen spot groups
vm	The VM type
vm_o	The on-demand VM type
c_{vm}	The hourly on-demand cost of the vm type instance
$num(c, vm)$	The function returns the number of vm type instances required to satisfy resource capacity c
C_o	The hourly cost of provision in on-demand mode
tb_{vm}	The truthful bidding price of vm spot group
m	The dynamic resource margin of an instance

2.2. Fault-tolerant mechanism

Suppose there are sufficient temporal gaps between price variation events of various types of spot VMs, increasing spot heterogeneity in provision can improve robustness. As illustrated in Fig. 3(a), the application is fully provisioned using 40 *m3.medium* spot VMs only, which may lead it to losing 100% of its capacity when *m3.medium*'s market price go beyond the