2013 International Conference on Cloud & Ubiquitous Computing & Emerging Technologies

Portable Autoscaler for Managing Multi-Cloud Elasticity

Jyoti Namjoshi, Yohan Wadia, Rohini Gaonkar
Cloud Services, Research & Innovation Group, iGATE
Mumbai, India
jyoti.namjoshi@igate.com, yohan.wadia@igate.com, rohini.gaonkar@igate.com

***Abstract*—Ability to scale resources up or down dynamically as per changes in workload conditions is one of the key features of clouds. We present here a framework for elastic scaling of cloud resources that is portable across clouds from a wide range of private and public cloud providers and that can be easily integrated with other frameworks.**

Keywords—Auto-scaling; Autoscaling, Autoscaler, Dynamic Scaling; Multi-cloud; Cloud Scalability Management; Unified Cloud Management; Automated Cloud Management; Private Cloud Management; Hybrid Cloud Management; Cloud Management and Monitoring; Cloud Computing

I. INTRODUCTION

Ability to horizontally scale IT resources up or down dynamically as per changes in workload conditions, often termed as elasticity of cloud, is one of the key features offered by clouds. It allows organizations to meet their peak demands in fluctuating workload conditions and helps to save cost by keeping machines in a shutdown state when load decreases. There are many cloud providers in the market today providing clouds based on either public or private cloud model. Clouds based on both these models as well as hybrids of them are increasingly in use today.

II. MOTIVATION

The motivation for the iAutoScaler framework described in this paper came from creating a common management and monitoring framework that could be used with private clouds based on various private cloud platforms. We were implementing private clouds based on a variety of platforms for multiple requirements and use cases, and while managing and monitoring these clouds, we felt a need to build a common framework that will perform functionality of automatic scaling based on workload changes in any of these clouds. We also wanted the framework to extend to public clouds so that it could be used for managing cloudburst kind of scenarios using a hybrid cloud. While cloud management tools such as RightScale, Scalr, enStratus, and public Infrastructure as a Services (IaaS) cloud providers such as Amazon Web Services (AWS) and Microsoft Windows Azure [1] offer autoscaling functionality for specific clouds, we wanted the functionality to work with a wide variety of clouds while also integrating well with, or embedding into, the Unified Cloud Management and Monitoring framework created by us to support our cloud offering that provides many additional cloud management functions. We wanted control on the framework to provide us the flexibility required to build around it.

III. ARCHITECTURE

A. Goals

The goal was to create a framework that is based on open standards, is portable across a wide range of private and public cloud providers, and is extensible and easy to integrate with. We wanted the framework to provide elastic scalability based on policies or rules on cloud resource health parameters as well as provide ability to schedule increasing and decreasing scalability for user specified periods.

B. Components

The key components of the framework we call as iAutoScaler are (a) User Interfaces, (b) Monitoring Engine, (c) Decision Engine, (d) Provisioning Manager, and (e) Database.

The framework provides Web based User Interfaces (UI) using which cloud administrators and other framework stakeholders can configure, launch, monitor, and manage autoscaling of cloud resources of their interest on various clouds.

The Monitoring Engine comprises of a monitoring manager and monitoring agents. The agents continuously monitor cloud health parameters and provide visibility into the state of cloud machine instances.

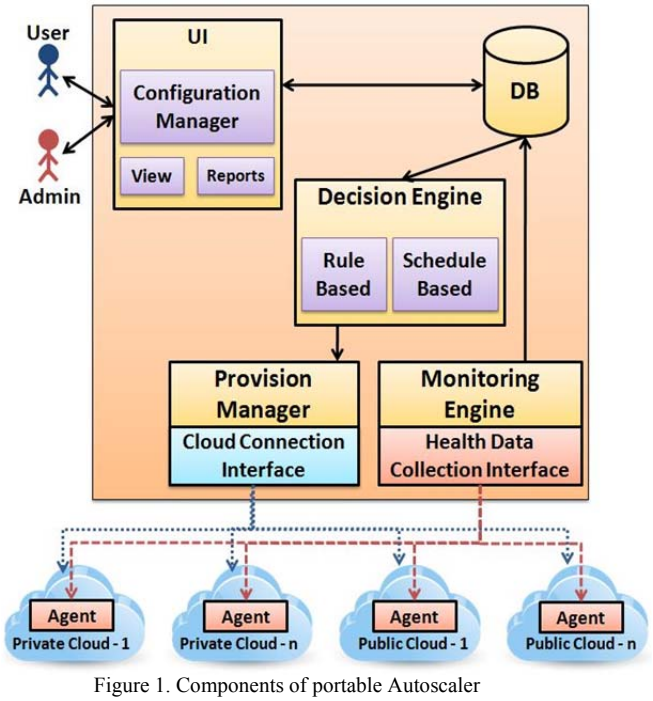

Figure 1. Components of portable Autoscaler

978-0-4799-2235-2/13 $31.00 © 2013 IEEE
DOI 10.1109/CUBE.2013.19

Vital information about cloud machines' CPU activity, memory consumption and network activities are captured by these agents and stored in the Database for processing by the Monitoring manager.

The Decision Engine is responsible for deciding on whether and when to scale resources, based on the configuration inputs that the user provides to the framework and to direct the Provisioning Manager to initiate autoscaling action.

The prime responsibility of the Provisioning Manager is to provision and de-provision specified number of resources on specified regions of clouds. This is made possible with the help of cloud-agnostic application programming interface (API) that acts as an abstraction layer across clouds from multiple providers hiding their complexity and providing a uniform interface to connect with them.

The Reporting Engine provides required reports to the users about the performance, capacity and detailed autoscaling activity so that they can make informed decisions with regards to improving quality of service.

IV. DESIGN

A. Autoscaling Techniques

Provisioning resources for an application that has highly fluctuating resource requirements can be challenging. Under-provisioning of resources can affect the application performance, while over-provisioning of resources can result in provisioned but idle resources incurring unnecessary costs [2]. Planning the capacity for an average workload will save cost, but performance will get impacted by sudden increase in load beyond the average. This can discourage customers and in turn affect organization's revenue. To overcome these issues, various types of techniques and models are developed. These autoscaling techniques can be majorly classified as schedule-based and rule-based techniques [3].

In schedule-based autoscaling techniques, the system is configured by setting parameters to increase the resources at a particular time of a day, or on any date in the future depending on some pre-planned events or predictions, and then decrease them after a specified interval. As the scaling actions are configured to perform based on time of the day, these techniques cannot handle unexpected variations in workload on a continuous basis. However, the advantage is that at a given schedule, resources are pre-configured a little in advance so that the application does not suffer time lag in resource allocation. The schedule-based techniques are useful when workload is anticipated well in advance such as in case of Railway Booking Website that is active for few hours of the day, and expects number of users till a certain limit, or for an application that deals with promotional, seasonal offers, or surveys that is time bound and planned in advance.

Rule-based techniques typically continuously monitor the resources in a system and based on rules or policies defined by users, carry out scaling actions. They work on principles of monitor, analyze and act. They react to changes that occur in the system. E.g., when defined threshold on a given parameter is breached, they can add more resources.

B. Tools

The choice of tools for the iAutoScaler was guided primarily by our goals. We chose a set of open source tools in terms of which the detailed design was carried out.

Ganglia was selected to implement Monitoring Engine. Ganglia is a scalable, agent-based open source software for monitoring high performance computing systems. It uses open standards based technologies for data representation, transfer and visualization, and provides comprehensive view on real-time and historic performance data of clustered machines. It collects metrics using daemons designed for data collection from geographically distributed machine clusters.

sFlow was chosen to complement Ganglia to monitor health of virtual machines, applications running on them, and network infrastructure.

MySQL was the choice for database for storing configuration data and cloud resources' health parameters data collected by Monitoring Engine.

We decided to use Apache Deltacloud APIs for the abstraction layer or façade for interfacing across clouds. Deltacloud connects to different clouds using cloud provider specific drivers. Scripts that are portable across clouds from different providers and that contain resource provisioning and de-provisioning instructions for automatic elastic scaling are created using Deltacloud APIs. These scripts call the Representational State Transfer (REST) APIs exposed by Deltacloud. Deltacloud supports a wide variety of clouds for compute or for storage or for both. Using its APIs, drivers can be created for additional clouds as well.

C. Logical and Operational Flow

For the initial development of the framework, we have focused on implementations of the reactive rule-based and predictive schedule-based techniques of autoscaling. The typical operational flow and few features are discussed in this section.

Before providing configuration for automatic scaling, the clouds must be prepared for certain pre-requisites such as acquiring access credentials, setting up cloud connection interface with cloud specific driver, injecting monitoring agent into machine images that will be used for scaling.

During configuration of the iAutoScaler, users provide their cloud access credentials for secure connection to clouds to be managed for automatic scaling. Users specify logical groups of different classes of virtual resources to be monitored. Different classes typically indicate different configurations such as those with high end configuration required for large compute-intensive tasks or those for I/O intensive tasks or for average online transaction processing loads. Logical group typically maps to a cluster within a cloud. The framework accesses machine image identifier for each of the configuration and provisions and de-provisions instances based on it to up-scale or down-scale. Users further provide configuration details such as upper and lower threshold values on various resource health parameters, the duration for which to tolerate the crossing of a threshold on a parameter without taking a scaling action, additional buffer times required between de-provisioning and provisioning of resources, the number of resources by which to scale up or down as a unit, and the maximum and minimum

total allowable resources in scaled up and down conditions respectively.

In a typical flow of iAutoScaler, a monitored metric value is aggregated across the provisioned logical group of resources by the framework at a regular interval of time. If the monitored metric value crosses its upper threshold for duration beyond the specified tolerance time, and if the current number of resources plus the scale increment is within the limit of maximum resources set for the group, then resources are scaled up by the increment value. If a monitored metric value crosses its lower threshold for duration beyond the specified tolerance time, and if the current number of resources minus the scale decrement is within the limit of minimum resources set for the group, then resources are scaled down by the decrement number.

It also supports the following features:

- Suspend and Resume scaling operations: The framework is capable of temporarily suspending the state of an autoscaling process. It can maintain the current state of the autoscaling environment but will not react to any workload changes. This feature is typically useful for maintenance activities such as debugging or tweaking certain configuration parameters of the application without triggering an unwanted scale out activity. User can then resume the autoscaling process once the maintenance work is done.
- Stop and start scaling instances: With a single click, the user can manually de-provision all the instances running in the logical group together without modifying or deleting any autoscaling configuration. This kills all the running resources and suspends the autoscaling process. The user can again reinitiate the autoscaling group when required.
- This is useful when a user wants to shutdown resources and save costs without needing to disturb the configuration.
- Schedule-based Scaling: Users can specify the time durations (schedules) for which to scale resources within a group and the number by which to scale them during the scheduled period
- Multiple triggers per autoscaler task: Users can set multiple triggers for each autoscaling group. Each trigger defines different configuration for threshold limit, scale increment or decrement, and metric name, thus providing users with more control and flexibility on the scaling actions.
- For example, a trigger can be set to increment resources by one if average load crosses the threshold of 50%. Another trigger can be set for incrementing resources by double the size if the average load keeps on increasing and crosses 70%. This gives more flexibility on the number of increments on each level of threshold breach for various metrics.
- Maintaining healthy number of resources: This framework can also be used to set fixed number of resources working at any given time. To configure this, the minimum, maximum and desired size of resources must be set to a same value. If the instance becomes unresponsive, it is replaced by another healthy instance.
- E-mail notification: Administrators can configure email notification to registered users on status of accounts and autoscaling activities.
- Report generation: Using the iAutoScaler, periodic and on-demand reports can be generated based on autoscaling activities for different accounts to track resource utilization for optimization of the system. Reports can be automatically sent to assigned account holders and administrators.

D. Technical Qualities

The framework is portable as it works with multiple cloud environments. It is light on footprint. Being based on open standards, the framework is interoperable and extensible, and can well integrate with our more comprehensive cloud management and monitoring framework. The Monitoring Engine is scalable as Ganglia software is based on a scalable architecture. The framework handles security through access control for users, provides secure connection to cloud APIs through user access keys, and provides SSH based secure communication to machine instances on clouds. To improve the availability of the framework itself, it can be used in clustered mode. Its user interfaces are simple and intuitive.

V. IMPLEMENTATION AND TESTING

A. Test-bed

We performed testing of the iAutoScaler Framework in two phases. In the first phase, we tested it against deployment of instances on Amazon Web Services Elastic Compute Cloud (EC2) infrastructure [4]. The environment consisted of an m1.large instance containing four elastic compute units and 7.5 GB RAM. On this instance, we configured Monitoring Engine manager, Provisioning Manager along with the User Interface, the Decision Engine and the Database.

We also created a custom Amazon Machine Image (AMI) that consisted of a Monitoring Engine agent pre-configured to communicate with the Monitoring Engine manager. We launched a set of m1.large instances out of this AMI and continuously monitored them. Using the User Interface, we supplied the autoscaling configuration details such as minimum number of instances to be maintained at all times, maximum number of instances to span in case the average CPU load increases beyond a pre-set threshold, the duration for which to tolerate the crossing of the threshold , etc. We then increased the CPU load on the set of instances artificially using Stress open source tool. We set the Stress parameters such that the CPU load on each of the test instances was different. Based on the configuration parameters provided by the user and the CPU load metric readings from the Monitoring Engine, the Decision Engine compares and decides the next appropriate set of actions. In the test runs, the Decision Engine successfully scaled instances in and out depending on the average CPU load generated by the test instances. Similar tests were conducted against other parameters such as Network in, Network out, Disk reads, Disk writes, etc. and the results were conclusive.

In the second phase of the testing, we tested the iAutoScaler against a Eucalyptus [5] cluster. We installed and configured Eucalyptus Enterprise version 3.2 on a cluster test bed of 4 HP Compaq 8200 Elite small form factor PCs each outfitted with 8GB RAM and quad core processors. Alongside

the Eucalyptus cluster we also had a VMware vCloud [6] based Private Cloud environment on two HP BL460c G7 Blades. We had integrated both the environments using VMware Broker component from Eucalyptus that allowed us to launch Eucalyptus Machine Images (EMI) in both the Eucalyptus standard cluster as well as the VMware cluster on high-end servers. We executed the iAutoScaler on similar sets of instances as done for AWS using the same Stress tool. Results were as per expectations.

We further tested iAutoScaler on a hybrid cloud comprising of AWS, Eucalyptus and VMware Clouds each having similar configuration details as above.

We are continuing to test it on additional private clouds such as OpenStack and public clouds and integrating it with our Unified Management & Monitoring Framework that performs management and monitoring of service incidents and problems, cloud performance, availability, security, patch management etc. in a hybrid cloud scenario.

VI. BENEFITS

iAutoScaler provides ability to automatically scale across a large number of clouds using rule based and schedule based techniques to enable users to reduce cost and energy consumption. Automated management of scalability reduces IT management complexity, improves management staff productivity.

It improves organizations' agility and business focus. The framework enables us to productively deliver our cloud management and monitoring service offering and lowers risk for customers.

VII. CONCLUSION

Multiple clouds could be effectively managed for elastic scalability using the techniques and tools selected for the iAutoScaler framework as described above, and the framework could be well-integrated into a more comprehensive multi-cloud management and monitoring framework.

REFERENCES

[1] Rajkumar Buyya, James Broberg, Andrzej M. Goscinski. Cloud Computing: Principles and Paradigms, pages 1-49, John Wiley & Sons, 17-Dec-2010.

[2] Ming Mao. Auto-scaling to minimize cost and meet application deadlines in cloud workflows. In High Performance Computing, Networking, Storage and Analysis (SC), 2011, pages 1-12. IEEE, 2011.

[3] Auto-scaling Techniques for Elastic Applications in Cloud Environments by Department of Computer Architecture and Technology University of the Basque Country, pages 11-14.

[4] Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/, 2013.

[5] Nurmi, D, Wolski, R., Grzegorczyk, C., Obertelli, G., Soman, S., Youseff, L., Zagorodnov, D. The Eucalyptus Open-Source Cloud-Computing System. In Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on, pages 124-131. IEEE, 2009.

[6] VMware vCloud Director (VMware vCloud). http://www.vmware.com/products/vcloud-director/, 2013.