International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 3 | ISSN : 2456-3307

A RNN-LSTM based Predictive Autoscaling Approach on Private Cloud

E. G. Radhika*[1], G. Sudha Sadasivam[2], J. Fenila Naomi[3]
[1]Department of Information Technology, PSG College of Technology, Coimbatore, Tamilnadu, India
[2]Department of Computer Science and Engineering, PSG College of Technology, Coimbatore, Tamilnadu, India
[3]Department of Information Technology, PSG College of Technology, Coimbatore, Tamilnadu, India

ABSTRACT

Web applications are the most prevalent applications of today's technology. They are typically characterized by IT resource requisites that fluctuate with usage, predictably or unpredictably. Failure to respond will impact customer satisfaction. Autoscaling is a feature of cloud computing that has the ability to scale up the cloud resources according to demand. It provides better availability, cost and fault tolerance. In the existing scenario, reactive autoscaling is used where the system reacts to changes and scale up the resources when there is a demand. The proposed system uses predictive autoscaling approach to predict future resource requisites in order to ascertain adequate resource are available ahead of time. The system uses a deep learning technique termed Recurrent Neural Network with Long Short Term Memory (RNN-LSTM) to predict the future demand based on the historical data. The predicted result is integrated with an OpenStack open source cloud platform to perform predictive autoscaling.

Keywords: Web applications, Autoscaling, Recurrent Neural Network, Long Short Term Memory, OpenStack

I. INTRODUCTION

Web applications play a major role in today's technology [1]. They may encounter different workload at different time, automatic resource provisioning needs to work efficiently and effectively. In fact, many commercial cloud services need to respond user requests quickly without interruption. It becomes an important issue for cloud services to process a large number of user requests in time. For example, a cloud service may use one VM to handle user requests. At some point of time, the number of requests could increase so quickly that the VM becomes overloaded. If the cloud platform can automatically detect such an issue, it can then dynamically add another VM to share the workload. This approach usually refers to autoscaling.

The most common scheme for auto-scaling is the reactive autoscaling approach. That is, when a cloud service is overloaded, the cloud platform that hosts the cloud service will trigger an instance only when it hits the threshold value. The other scheme for auto-scaling is to employ a predictive approach that predicts future workload and spins the instance accordingly. With this type of approach, the key for efficient autoscaling relies on how near the prediction mechanism can predict the future workload. In club to reach a good prediction result, the autoscaling mechanism can utilize a deep learning technique, which looks at the history of workload and then foresee the future workload. Then an autoscaling mechanism increases the number of virtual resources according to the prediction result.

Auto-scaling provides very good benefits, but implementation is not a facile and straightforward task. It needs the capacity to accurately foresee future workload and also should figure the right number of resources required for the expected increase in workload. A rapid spike in demand, Outages and Variable traffic pattern where different times of the day have different workload are issues in web applications [2]. However, if the thresholds are not defined opportunely, the system may not work smoothly enough. For example, a threshold is defined to be 60% of CPU usage for adding VMs in a scheme. Postulate that actual CPU utilization keeps changing from 55% to 65%. In this case, the system may keep adding VMs, and could hurt the quality of service. This implies that, the reactive approach may not be enough, it is indispensable to employ workload prediction in an autoscaling mechanism of a cloud platform. A reactive method is also used to reduce the impact of wrong workload prediction in order to obviate the case that the predicted result is not accurate enough.

At present days there are many web applications can make benefit from automatic scaling property of the cloud where the number of resource usage can be scaled up dynamically by the cloud service platform [3]. So here, present system provides a predictive autoscaling approach for web application by using a deep learning technique called Recurrent Neural Network- Long Short Term Memory. The predicted result is integrated with an OpenStack open source cloud platform. The rest of the paper is structured as follows. In Section 2 overviews related work in the area of automatic resource scaling. In Section 3 describes the proposed method for autoscaling. Section 4 shows the result analysis. Conclusively, Section 5 concludes.

II. RELATED WORK

Many cloud vendors like Amazon [4], Azure [5] offers auto-scaling as a commonly desired feature in cloud IaaS and PaaS but provides pay-per-use services. In recent years, many studies began to use different methods for workload prediction on cloud platforms.

Jianyu Sun et al [6] apply ARIMA based prediction time series prediction method for predictive strategy and threshold-based metric for checking reactive strategy. ARIMA works well for short-term prediction and gives more importance to immediate data points. Similarly, Raouia Bouabdallah et al propose an automatic resource provision based on a workload prediction by Simple Exponential Smoothing method in [7]. It was implemented on the Open Nebula Cloud Platform.

Jingqi Yang et al [8] apply Linear Regression Model (LRM) and Auto Regressive Moving Average (ARMA) based prediction methods for automatic scaling in service clouds. For autoscaling web applications in heterogeneous cloud infrastructure Hector Fernandez et al [9] used Linear Regression for linear trends, Auto Regression Moving Average (ARMA) for linear with small oscillations, Exponential Smoothing Holt-Winters for daily and seasonal and Autoregression and Vector Autoregression for correlated trends.

Nilabja Roy et al [10] used ARMA, Mean Value Analysis (MVA) and Look-ahead resource allocation algorithms for efficient autoscaling in the cloud for workload forecasting. Similarly, for modelling the autoscaling operations in cloud with time-series data Mehran N. A. H. Khan et al [11] used Exponential Smoothing and Markov chain model. Time series data are dynamic in nature; it fails to meet when there is the non-linear co-relation between the past and current data points.

Martin Duggan et al [12] applied Simple RNN method for predicting the CPU utilization accurately for short time periods. It fails to meet if there is a sudden change in CPU utilization. The above methods for workload prediction is used only for short-term prediction incase if there is large variation in the data points it fails to predict accurately. When compared to these time-series techniques, Recurrent Neural Network- Long Short Term Memory deep learning technique is used to predict the future workload accurately in case of any variance because it can remember the historical values.

III. THE PROPOSED METHOD FOR AUTOSCALING

The Figure 1 shows the overall framework of the proposed method for Autoscaling using Recurrent Neural Network- Long Short Term Memory on OpenStack Cloud Platform. The framework of the proposed method is depicted in three phases. The first phase describes the real-time data collection from OpenStack. The second phase describes the workload predictor using a deep learning technique and the final phase is about Autoscaling on OpenStack.

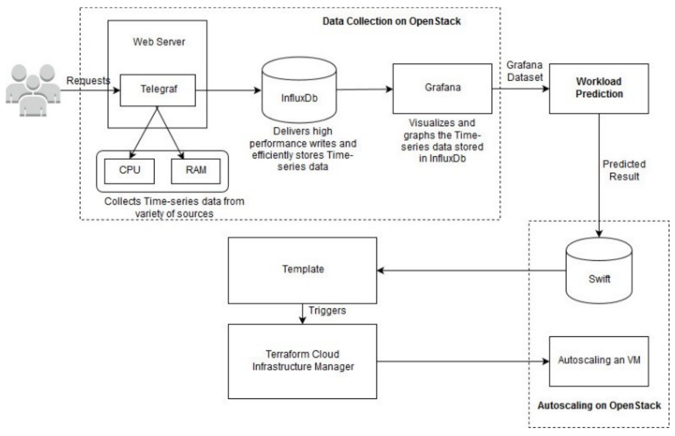
Data Collection on OpenStack
Web Server
Requests
Telegraf
CPU
RAM
Collects Time-series data from variety of sources
InfluxDb
Delivers high performance writes and efficiently stores Time-series data
Grafana
Visualizes and graphs the Time-series data stored in InfluxDb
Grafana Dataset
Workload Prediction
Predicted Result
Swift
Template
Triggers
Terraform Cloud Infrastructure Manager
Autoscaling an VM
Autoscaling on OpenStack

Figure 1. Overall Framework of Proposed Method

A. Data Collection from OpenStack

The number of user requests hits the same web server, let's consider the cloud service use only one VM (web server) to handle user requests. At some point of time, the number of users gradually increases the server becomes overloaded and the CPU and RAM utilization gets increased. To collect those metrics Telegraf, InfluxDb and Grafana [13] components are setup on OpenStack.

Telegraf is a plugin-driven server agent for collecting and reporting metrics. This can collect data from a wide variety of sources. Telegraf is maintained by InfluxData. It has **a** good support for indicting data to InfluxDB. InfluxDB is a data store for any utilization case involving large amounts of time-stamped data. Grafana is the open platform for resplendent analytics and monitoring. It makes it facile to create dashboards for displaying data from many sources, particularly time-series data.

OpenStack uses InfluxData's InfluxDB to monitor the OpenStack infrastructure. InfluxDB is utilized to store and query the OpenStack infrastructure metrics time series that are collected at the OpenStack infrastructure level. The data is visualized in graphical format in Grafana Dashboard. The data can be exported in CSV, XSL or JSON format from Grafana Dashboard.

B. Workload Prediction

The data exported from the Grafana dashboard are given as an input to workload prediction. Auto-scaling actions are performed beforehand by predicting the future resource demand. Predictive approach address the rapid spike demand, outages and variable traffic pattern issues. Compared to time-series techniques, the Neural Network method gives better results for anticipating the future workload. The drawback is, it cannot remember events if there is a very long and variant time lags between events. To amend memorization of the standard feedforward neural network, Recurrent neural network (RNN) is utilized.

RNN is a densely packed Neural Network connecting multiple hidden layers with recurrent connections. A hidden layer nodes are connected in a loop and it serves to maintain the memory between the states. The hidden layer in RNN known as Long Short Term Memory (LSTM), which is utilized to remember past values (events). It is utilized to predict the CPU/RAM utilization of the VMs with greater accuracy when compared to traditional approaches. Based on the prediction, predictive autoscaling is done by provisioning the instances ahead before the server exceeds a given workload threshold. Effective for long-term prediction and provides better results when there is a non-linear correlation between past and current values. The predicted result of Long Short Term Memory Recurrent Neural Network method is to be integrated on OpenStack.

C. Autoscaling on OpenStack

The predicted result is stored in the Swift component. Swift is an Object Storage component in OpenStack which stores data at large scale with high availability. The data from the Swift is pulled to create a template to trigger an instance by using Terraform [14]. A Terraform is a tool developed by HashiCorp that can be utilized to deploy and manage cloud infrastructure facilely by defining configuration files. It is akin to OpenStack Heat. Heat which is specific to OpenStack, Terraform is provider agnostic and can work with multiple cloud platforms such as OpenStack, AWS and VMware.

Users define configuration files that Terraform processes to create and manage infrastructure resources across multiple cloud providers. It generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, Terraform is able to determine what transmuted and create incremental execution plans which can be applied. Terraform configuration files are saved as *.tf* files. Resources represent infrastructure components, e.g. server, virtual machine, container, network port.

The configuration file contains the details about the provider that is OpenStack and the resource to be created. Once the configuration file is created, the three main terraform commands are executed to create an instance based on configuration file [15]. The terraform commands are *terraform init to in*itialize a Terraform working directory, *terraform plan* to generate and shows an execution plan and *terraform apply* for building or changing the infrastructure, i.e. creates a resource based on execution plan.

IV. IMPLEMENTATION

The Figure 2 shows the flow of proposed work and the corresponding implementation results. The proposed RNN-LSTM method has been implemented in python using Keras libraries for predicting the future workload. Based on the predicted result a predictive autoscaling is done on an OpenStack Mirantis version. OpenStack is setup by using five physical machines. Each machine has Intel® Core™ i5-6500, 8 GB RAM and 64 bit OS. A virtual cluster that is OpenStack cluster is created consisting of several VMs. An Apache Web Server and Telegraf are deployed in one VM. Similarly, InfluxDb and Grafana is deployed in another VM. When the user requests hits the Apache web server, the Telegraf monitors the metrics. InfluxDb continuously stores the data monitored by Telegraf and Grafana visualizes the metrics. The .csv file collected from Grafana contains 24 hours with two months of real-time data that is from 00:00:00 January 1, 2018 to 23:59:00 January 31, 2018 and from 00:00:00 February 1, 2018 to 23:59:00 February 28, 2018. The Figure 3 shows the dataset collected from Grafana. The dataset is given as input to Long Short Term Memory Recurrent Neural Network. This deep learning technique is used to predict the future workload.

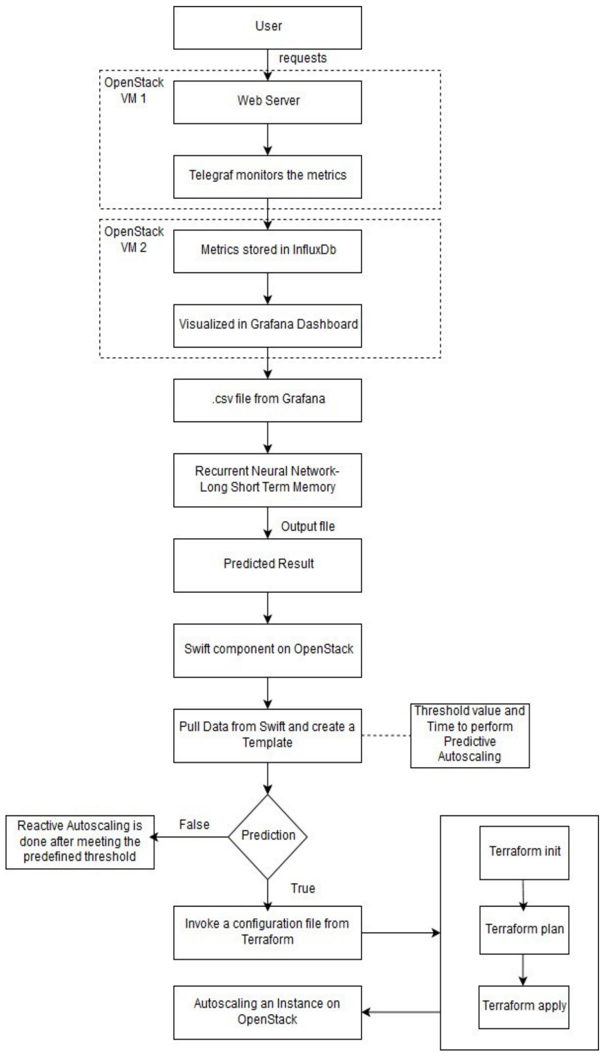
User
requests
OpenStack VM 1
Web Server
Telegraf monitors the metrics
OpenStack VM 2
Metrics stored in InfluxDb
Visualized in Grafana Dashboard
.csv file from Grafana
Recurrent Neural Network-Long Short Term Memory
Output file
Predicted Result
Swift component on OpenStack
Pull Data from Swift and create a Template
Threshold value and Time to perform Predictive Autoscaling
Reactive Autoscaling is done after meeting the predefined threshold
False
Prediction
True
Invoke a configuration file from Terraform
Terraform init
Terraform plan
Terraform apply
Autoscaling an Instance on OpenStack

Figure 2. Flow Diagram of Proposed Method.

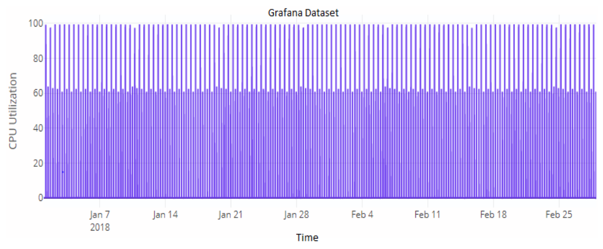

Figure 3. Dataset collected from Grafana dashboard

The steps to implement Recurrent Neural Network is shown in figure 4.

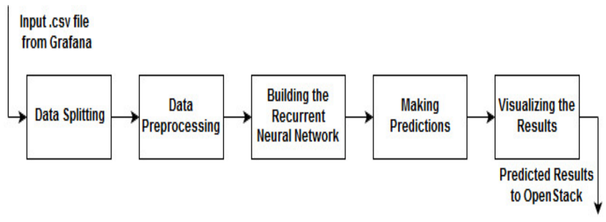
Input .csv file from Grafana
Data Splitting
Data Preprocessing
Building the Recurrent Neural Network
Making Predictions
Visualizing the Results
Predicted Results to OpenStack

Figure 4. Steps to Implement Recurrent Neural Network

First the input dataset is split into 60% for training and 40% for testing. The training dataset is preprocessed by using MinMaxScaler. Keras libraries are imported to build the Recurrent Neural Network (RNN). RNN can be called as a regressor because prediction is done by regressor, not by classification. The input and Long Short Term Memory (LSTM) layer which is a hidden layer are added using sigmoid activation. The output layer will return the predicted values. After adding the layers, the network is compiled using Adam optimizer. The Adam Optimizer gives better result compared to other optimizer. The loss is calculated using Mean Squares Error (MSE) and Mean Absolute Error (MAE) where MSE and MAE is 0.003. Finally, the network is fit to the normalized training dataset. To test the network that is built, the testing dataset is compared with the build model and the future prediction is done using predict() function. The figure 5 shows the future predicted result. The predicted result is stored in the Swift component on OpenStack. The result is pulled from Swift and a template is created as shown below:

Template to trigger an Instance

```
Input: Predicted Results (Time and CPU_Utilization)
Output: Aw32n Instance is deployed in OpenStack
Begin:
    if Time = = Current_Time
      if CPU_Utilization >= 60%
            Invoke Configuration file
            from Terraform to deploy an
            Instance.
      endif
    endif
```

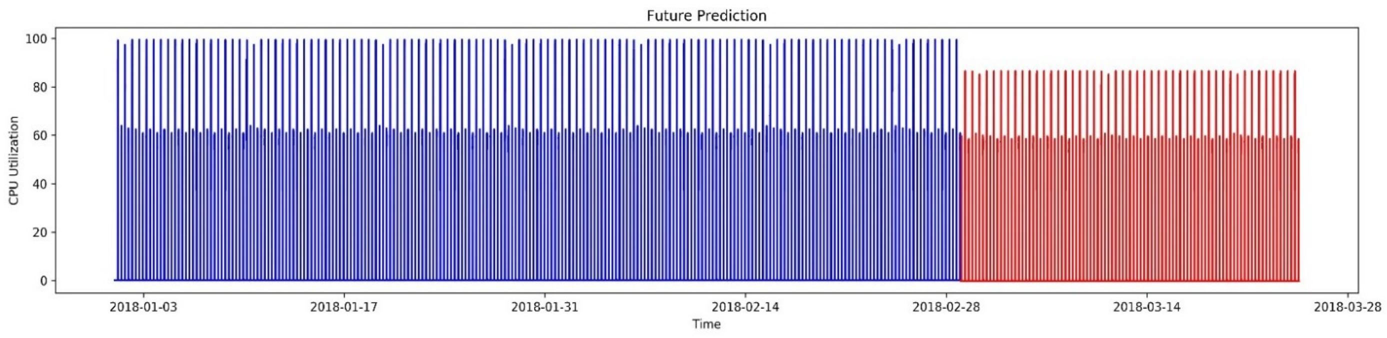

Figure 5. Future Prediction using Recurrent Neural Network- Long Short Term Memory

To perform predictive autoscaling, the threshold value is set at 60%. Based on the predicted result, the Time is compared with current time and the CPU Utilization is checked whether it is greater than or equal to 60%. If the condition is satisfied, then the configuration file named Stack.tf for deploying an Instance is invoked from Terraform as shown below:

Stack.tf configuration file

```
# Configure the OpenStack provider
 provider "openstack" {
        user_ name = "admin"
        password = "pwd"
        auth_url = "http://10.1.1.60.13:8000/v2.0"
        region = "RegionOne"
 }
 # Create an Instance
 resource = "openstack_compute_instance_v2"
"instance1"{
        name = "instance1"
        image_id = "2e0754d7-41j0-7665-4343ade"
        flavor_id = "3"
        key_pair = "mykey"
        security_groups = ["default"]
        metadata {
                this = "that"
        }
        network {
                name = "my_Network"
        }
 }
```

In Stack.tf configuration file, the provider is openstack which includes the following arguments. The user_name and password is to login with. auth_url is used as identity authentication url. RegionOne is a region in which OpenStack uses to create an instance.

To create an instance, a unique name for the resource is given as "instance1". Image_id, flavor_id and key_pair are given in the OpenStack to create desired instance. Metadata refers to metadata key/value pair to make available from within the instance. The network used is 'my_ Network' that provides the network connectivity to the instance.

Using Terraform init, plan and apply commands an instance is deployed on OpenStack. The template is run for every five minutes and the average of next 10 data points is taken. If it is above 60%, for example the CPU Utilization is above 60% from 09:49:00 to 10:49:00. The VM is kept active for 1 hour otherwise the Terraform may keep adding the VMs for every time period having high utilization which will hurt the quality of service. In case if the prediction is false that is a wrong workload prediction then reactive autoscaling is done after meeting the threshold value.

V. CONCLUSION

In this paper, Recurrent Neural Network- Long Short Term Memory (RNN-LSTM) is used to predict the future workload. Based on the result, the predictive autoscaling is done on OpenStack by triggering an instance using Terraform. The proposed system also adopts reactive autoscaling approach in order to prevent negative impact of wrong prediction. The results shows that, when the massive workload arrives, the proposed system predicts the future demand and spins the instance accordingly.

VI. REFERENCES

[1]. Ching-Chi Lin, Jan-Jan Wu, Pangfeng Liu, Jeng-An Lin and Li-Chung Song, "Automatic Resource Scaling for Web Applications in the Cloud", in Grid and Pervasive Computing, pp. 81–90, 2013.

[2]. Craig Sterrett, Yih Leong Sun, Shamail Tahir, "OpenStack Workload Reference Architecture: Web Applications",URL https://www.openstack. org/assets/software/mitaka/ OpenStackWorkload RefArchWebApps-v7.pdf.

[3]. Sushil Deshmukh , Sweta Kale, "Automatic Scaling Of Web Applications For Cloud Computing Services: A Review", in International Journal of Research in Engineering and Technology, Vol. 03, pp. 2321-7308, January 01, 2014.

[4]. AWS Auto Scaling Pricing, URL https://aws. amazon.com/autoscaling/pricing/.

[5]. Kirk Evans, "Autoscaling Azure–Virtual Machines",URLhttps://blogs.msdn.microsoft.com/ kaevans/2015/02/20/ autoscaling-azurevirtual-machines/

[6]. Jianyu Sun, Haopeng Chen and Zhida Yin,"AERS: An Autonomic and Elastic Resource Scheduling Framework for Cloud Applications" in 2016 IEEE International Conference on Services Computing (SCC), pp: 66-73, September 2016.

[7]. Raouia Bouabdallah, Soufiene Lajmi, Khaled Ghedira," Use of Reactive and Proactive Elasticity to Adjust Resources Provisioning in the Cloud Provider", in 2016 IEEE 18th International Conference on High Performance Computing and Communications, 12-14 December, 2016.

[8]. Jingqi Yang, Chuanchang Liu, Yanlei Shang, Zexiang Mao, Junliang Chen, "Workload Predicting-Based Automatic Scaling in Service Clouds", in IEEE Sixth International Conference on Cloud Computing, 2013.

[9]. Hector Fernandez, Guillaume Pierre, Thilo Kielmann, "Autoscaling Web Applications in Heterogeneous Cloud Infrastructures", 2014 IEEE International Conference on Cloud Engineering (IC2E), March 2014.

[10]. Nilabja Roy, Abhishek Dubey, Aniruddha Gokhale, "Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting", in 2011 IEEE International Conference on Cloud Computing (CLOUD), September, 2011.

[11]. Mehran N. A. H. Khan, Yan Liu, Hanieh Alipour, "Modeling the Autoscaling Operations in Cloud with Time Series Data", IEEE 34th Symposium on Reliable Distributed Systems Workshop (SRDSW), January 07,2016.

[12]. Martin Duggan, Karl Mason, Jim Duggan, Enda Howley, Enda Barrett," Predicting Host CPU Utilization in Cloud Computing using Recurrent Neural Networks", in 8th International Workshop on Cloud Applications and Security, November, 2017.

[13]. Lindsay Hill, "Using Telegraf, InfluxDB and Grafana to Monitor Network Statistics", URL https://lkhill.com/telegraf-influx-grafana-network-stats/, November 24, 2017.

[14]. How to Use Hashicorp Terraform with OpenStack,URL https://platform9.com/blog/how-to-use-terraform-with-openstack/, July 22, 2016.

[15]. Michal Medvecky, "Managing your (OpenStack) infrastructure with Hashicorp Terraform" URL https://medium.com/@michalmedvecky/managing-your-openstack-infrastructure-with-hashicorp-terraform-8c93ade214b4, May 17, 2017.