Resource Utilization through Autoscaling in Cloud Environment

Gopesh R. Banker[1] Gayatri Pandi (Jain)[2]
[1]Student [2]Assistant Professor
[1,2]Department of Computer Engineering
[1,2]LJIET, Ahmedabad

Abstract— In recent years, cloud computing has evolved as a much efficient, proactive and widely accepted technology. Mostly virtualization techniques are used to serve the requests from the clients. A major issue arising for all cloud users is to handle the dynamic number of requests from time to time and using efficiently all the existing resources. This problem is overcome by the autoscaling mechanism of Virtual Machines. Basic autoscaling mechanisms are naïve as they only check for individual VM healths. The outcome of this research comes up with an efficient autoscaling mechanism that performs an efficient scaling in a cloud environment. The results of the research can be used in real time cloud setup so as to achieve high efficiency and serve more number of requests than the other pre-existing schemes. The results are simulated in an AWS environment which shows how better this scheme is then the currently existing ones.

Key words: Cloud computing, cloud resource scaling, dynamic scaling, virtual machine utilization

I. INTRODUCTION

Cloud Computing is one of the blooming technologies in the current IT trend. It is based on a model which enables IT resources on a request basis and on a pay as you go pricing. These IT resources are being shared in public. Cloud allows its users to dynamically allocate and deallocate resources at run time. As cloud providers charge their users on a pay per hour model, it is necessary that that efficient utilization of existing resources takes place. These resources include Virtual Machines (VM), Network Bandwidth and the Storage capacity allocated to the Virtual Machine. Virtualization techniques that consumes of hypervisors are used for creating these Virtual Machines. In this paper, we only consider on auto-scaling mechanisms of Virtual Machines. Ideally, the load on the VM decides whether to scale up or scale down a resource. If the CPU utilization reaches a particular upper threshold level, the cloud infrastructure scales up the VM resource i.e. a new VM is added to the cluster of VMs and the load balancer starts sending new request on this newly created VM and vice-versa. But, just considering a threshold value for VM utilization is a naïve solution and cannot be used in real world.

Another factor arising is the VM boot up time and the VM shutdown time. Once you start a VM, it doesn't start from the next minute itself, it takes some time for the booting process. Same is the case for shutting down a VM. Moreover, it also depends on the configuration of the resource. Cost is also an issue in the cloud environment because any VM created is charge on an hourly cycle and if the scaling mechanism is not proper, the cost of the user would be highly impacted. In this paper, we propose an autoscaling mechanism where scaling can be done based on the overall CPU Resources of all VMs utilized by the particular user instead of calculating it for individual VMs. The successing sections in this paper shows the related works carried out in autoscaling of cloud resources. The next section proposes a new autoscaling mechanism followed by results and analysis.

II. RELATED WORK

A. Cloud Auto-Scaling with Deadline and Budget Constraints[1]

As a computing platform, clouds own distinct characteristics compared to utility computing and grid computing. Ming Mao, Jie Li and Marty Humphrey[1] have identified some of them which can largely affect the way people use cloud platforms, especially in cloud scaling activities. They describes various metrics to be accounted for while scaling a VM resource:

1) Unlimited Resource Limited Budget:

Ideally, a cloud infrastructure offers its users unlimited power of computing tasks and storage capacity. This enables users to scale to an extremely large size, but are not free, they are charged on an hourly basis for each of the resource being utilized. Hence, a cloud auto-scaling mechanism should explicitly consider user budget constraints while acquiring a resource.[1]

2) Non – Ignorable VM Instance Acquisition Time:

Even if you can scale cloud resources anytime, it doesn't mean that cloud scales it very fast. Based on their research, they estimate the instance acquisition time to be of 10 minutes approximately before the resource is ready to be used.[1] Same is the case for VM shutting down time which takes about 2-3 minutes in Windows Azure[6]. This implies that users have to consider two issues in cloud dynamic scaling activities. First, count in the computing power of pending instances. If an instance is in pending status, it means it is going to be ready soon. Ignoring pending instances may result in booting more instances than necessary, therefore waste money. Second, count how long the pending instance has been acquired and how long further it needs to be ready to use. If the startup time delay can be well observed and predicted, application admin can acquire machines in advance and prepare early for workload surges.[1]

3) Full Hour Billing Model

The hourly billing model saves money when the users shut down the machines. But, VMs resources are always billed on an hourly basis. A 1 minute usage is charged for a whole hour and a 60 minute usage is also charged for the same whole hour and if the resource was started and shutdown two times in an hour, then it is charged for two VM resources. Hence, a better policy is to only shutdown the VM resource when it approaches its full hour operation, once an instance is started.[1]

4) Multiple Instance Types

Instead of offering single instance type, cloud providers now offer various instance types for its users. They can use

different types of instances based on their performance and usage. For example, EC2 instances are grouped into three families, standard, high-CPU and high-memory. Standard instances are suitable for all general purpose applications. High-CPU instances are well suited for computing intensive application, like image processing. High-memory instances are more suitable for I/O intensive application, like database systems and memory caching applications[1]. One important thing is that instances are charged differently and not necessarily proportional to its computing power. For example, in EC2, c1.medium costs twice as much as m1.small. But it offers 5 times more compute power than m1.small. Thus for computing heavy jobs it is cheaper to use c1.medium instead of the least expensive m1.small. Therefore, users need to choose instance type wisely. Choosing cost-effective instance types can both improve performance and save cost[1].

From all these parameters taken into account, Ming Mao, Jie Li and Marty Humphrey proposed a solution to select a VM instance type that is suitable for the current job processing i.e. either they are High CPU intensive, High I/O intensive or High Memory intensive. They implemented this architecture on the Windows Azure platform where they ran a real scientific application (MODIS)[1].

Based on the policy to shut down the VM instance once it reaches its one hour billing cycle, they find out the average processing time to be as follows.

	Mix Avg 30 jobs/hour STD 5 jobs/hour	Computing Intensive Avg 30 jobs/hour STD 5 jobs/hour	I/O Intensive Avg 30 jobs/hour STD 5 jobs/hour
General 0.085$/hour Delay 600s	Average 300s STD 50s	Average 300s STD 50s	Average 300s STD 50s
High-CPU 0.17$/hour Delay 720s	Average 210s STD 25s	Average 75s STD 15s	Average 300s STD 50s
High-IO 0.17$/hour Delay 720s	Average 210s STD 25s	Average 300s STD 50s	Average 75s STD 15s

Table 1: Average processing time.[1]

The instance acquisition for VMs is shown in the below figure 1.

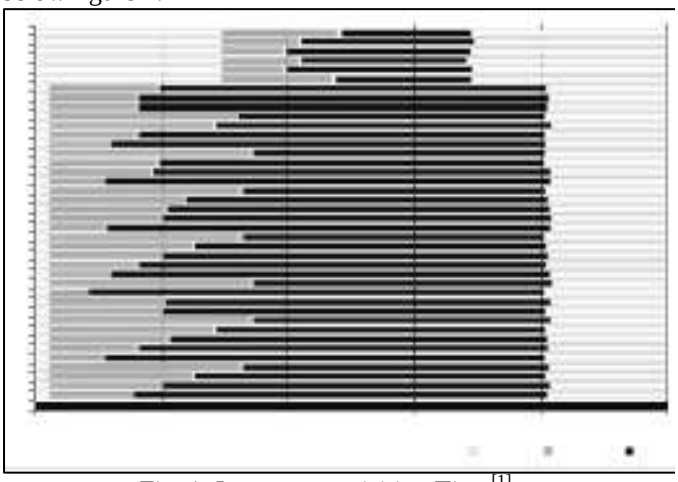

Fig. 1: Instance acquisition Time[1]

B. Integrated and Autonomic Cloud Resource Scaling[2]

Typical Cloud resource scaling solutions possess following limitations:

1) Thresholding mechanisms used to trigger scaling decision process are not Cloud-ready. That is, they are not suitable for huge scale and dynamic Cloud environment. They are naïve or rudimentary causing spurious scaling of resources[2].
2) In a Cloud, resources from three different domains, compute, storage and network, are acquired/released (scaled up or down) on demand (typically in virtualized form, including application or software resources). But in current solutions resources from these three domains are scaled in isolation rather than in combination resulting in spurious auto-scaling[2].
3) The performance metrics that trigger scaling are considered in isolation. For example, compute resources are scaled based on CPU/compute resource metric only, but not in correlation with, for example, metrics from the network domain[2].
4) While existing solutions may scale compute or storage resources automatically when specified condition or policy is satisfied, network resources are scaled statically.[2]
5) Network resources are not scaled in relation to compute and storage domains and vice versa. For example, when virtual compute resources are scaled, virtual load-balancer (LB) or firewalls (FW) may also need auto-scaling.[2]

C. IACRS Tenant Interface Input

The interfaces and relevant parameters provided in IACRSTI are as follows (we call the input IACRS Policy or IACRSPol):

- Resource R to be monitored and auto-scaled (typically this will be an existing resource URI).
- Performance metric to be monitored (PMT).
- Reference to a resource (end-point 2: EP2), if the metric is binary (such as response time, delay or jitter from one resource to another).
- Group ID of the group R belongs to and optionally an indication whether R is a parent of the group.
- Threshold Policy (TP) for R and PMT:
  - ThrU = #%, ThrbU = #%, ThrL = #%, ThroL = #%, Duration = # sec (see Section B below for definition).
- Auto-scaling policies (SP) for a resource and metric:
  - Resource acquisition (SP-RA) policy: <increment in #%> (INCR), <max resources in #> (MAX), where increment is the number of resources to be acquired, which is calculated as <original # of resources in a group> * INCR rounded to the next higher number and MAX is the maximum number of resources that can be acquired when no more resources will be acquired irrespective of load on all the resources in a group.

– Resource release (SP-RR) policy: <decrement in #%> (DECR), <min resources in #> (MIN), where decrement is the number of resources to be released, which is calculated as < # of resources in a group> * DECR rounded to the next higher number and MIN is the minimum number of resources that should be retained when no more resources will be released irrespective of load on all the resources in a group. If SP-RR is not specified, then SP-RA will be used with INCR replaced with DECR and MIN defaulting to 1.

The scaling algorithm developed is as below:

1) Metric value trending up with persistence:

– When the PMT_Value crosses above the ThrU for the first time the duration clock starts to tick and we set a variable Tick_Up_Start to 1.
– If PMT_Value remains above the ThrU or oscillates around it, but remains above the ThrbU for the specified duration (such as 5 minutes), then stop (reset) clock tick and set Tick_Up_End.
– If PMT_Value persisted for duration since crossing ThrU and remains above ThrbU sincelast time Scale value was 00 or 01 (this latter condition is to prevent repeated pattern such as the one shown in Figure 2), then set the variable Scale to 11 (which may potentially trigger scaling):

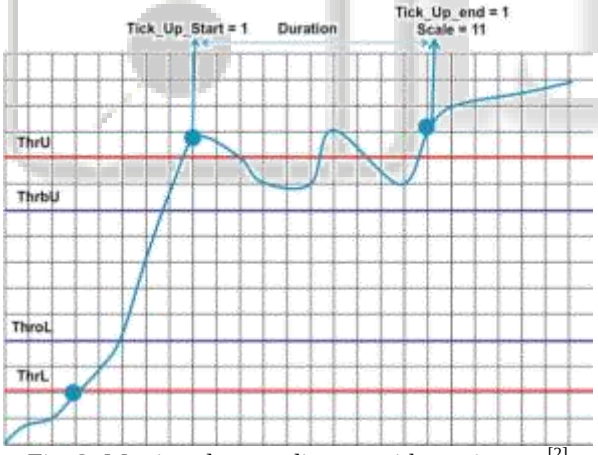

Fig. 2: Metric value trending up with persistence[2]

2) Metric Value Trending Down from above:

– If metric value remains below the ThrbU for the specified duration, then variable Scale will be set to 01 (see below group/correlation heuristics on how Scale value 01 is utilized). The lock for this duration starts to tick when the PMT_Value crosses below the ThrbU for the first time, since it was above ThrU.

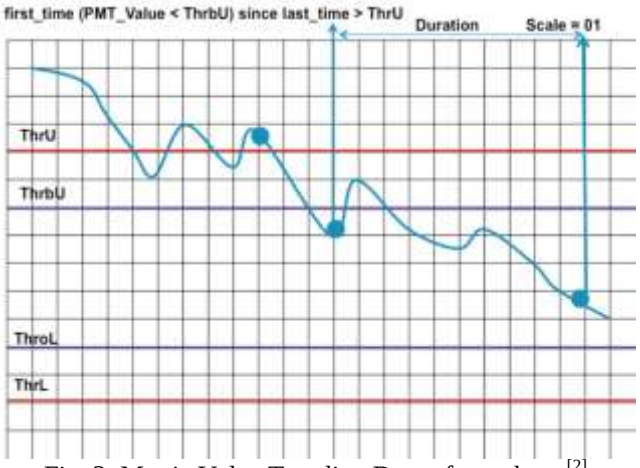

Fig. 3: Metric Value Trending Down from above[2]

3) Metric value trending down with persistence:

The heuristics algorithm is similar as PMT_value trending up with persistence except that the algorithm is applied around ThrL and ThroL and Scale value is set to 00 when PMT_Value crosses below ThrL for the first time since Scale value was 11 or 01 and remain around ThroL for the specified duration. A PMT_Value behavior is shown in Figure 4.

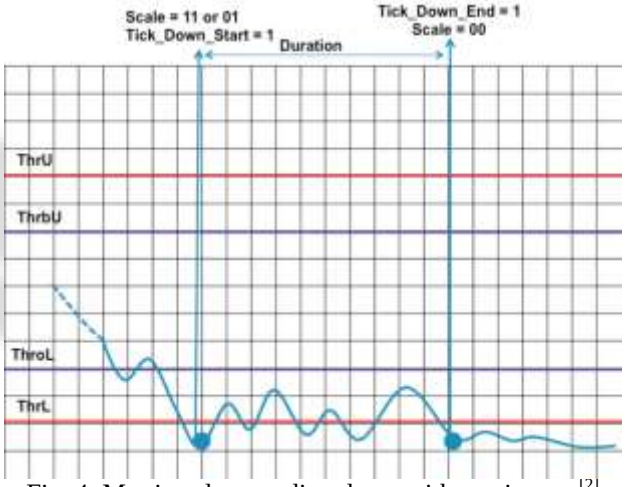

Fig. 4: Metric value trending down with persistence[2]

4) Non-scaling Metric Value behavior:

– The PMT_Value for non-scaling behavior graph is shown in Figure 5, in which the Scale value will retain its last value.

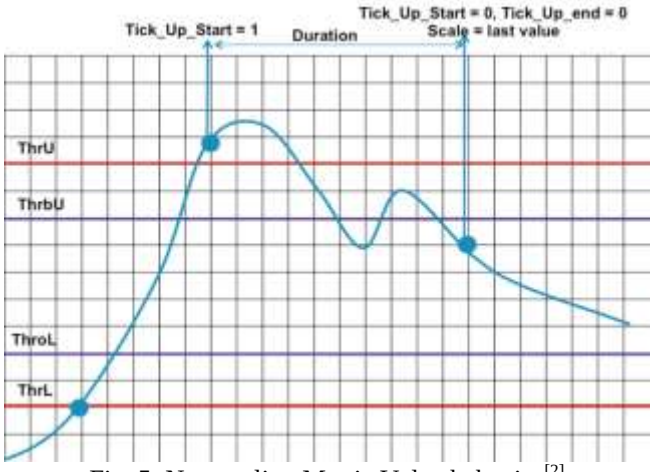

Fig. 5: Non-scaling Metric Value behavior[2]

D. An Auto-Scaling Mechanism for Virtual Resources to Support Mobile, Pervasive, Real-Time Healthcare Applications in Cloud Computing[3].

In this article, Yong Woon Ahn and Albert M. K. Cheng, Jinsuk Baek, Minho Jo, Hsiao-Hwa Chen[3], propose a novel auto-scaling mechanism in order to dynamically adjust the number of VMs to handle deadline-critical real-time data, which varies in size over time. In consequence, the resizing of the virtual resources for processing the given data is achieved on an on-demand basis. The key mechanism is to predict the volume of future data. Although it is not necessarily trivial to predict the exact moment at which a large volume of data will be delivered from sensor-based medical devices, most objects monitored by sensor-based devices typically show symptoms before transitioning to an abnormal state.[3]

Basically, it is the task of a VM manager to scale VMs up or down. Here, Real Time Application Servers (RTA servers) are configured in the VMs and they are responsible for the scaling of VMs.

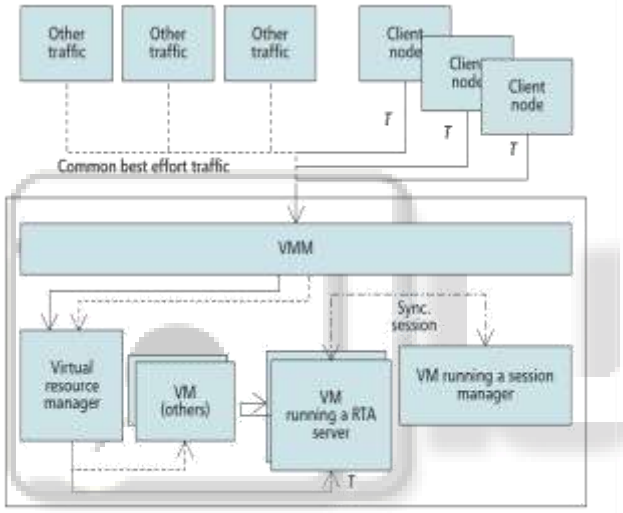

Fig. 6: Physical machine architecture with virtual RTA servers[3]

At any particular time, the VCPU load is represented by a state N. If, the VCPU load increases or decreases, the state transition is made in forward or backward direction. We can represent this transition in an angular format ranging between -90° and +90°, where -90° represents 0% load and +90° represents 100% load which is a degree representation D[i]. In order to provide more accurate predictions, the number of transitions should vary depending on the value ofD[i]. Let us assume that there are M sections between –90° and 90°. If the value of D[i] is equal to or greater than w90°/M/2, but smaller than (w+1)90°/M/2, and D[i] is bigger than 0°, the analyzer moves the state w transitions forward. If the value of D[i] is equal to or smaller than w(–90°)/M/2, but still smaller than (w+1)(–90°)/M/2, and D[i] is smaller than 0°, the analyzer moves the state w transitions backward. Figure 7 shows an example, when there are four different sections, including F1, F2, B1, and B2.[3]

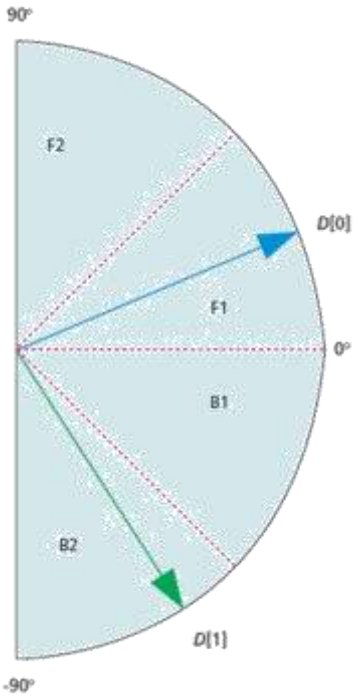

Fig. 7: State diagram for transitions of the proposed auto-scaling mechanism[3]

If the root RTA server reaches State N, it determines whether it needs more computing resources or not by checking the states of other child RTA servers. If it is needed, it launches a new child RTA server, where the overloaded pending real-time tasks will be assigned. On the other hand, if D is smaller than 0°, it now makes w backward transitions. When it reaches at State 1, it terminates one of its child VMs and intercepts the workload of the terminating child RTA.[3]

E. Efficient Auto-scaling in the Cloud using Predictive Models for Workload Forecasting[4]

Here, Nilabja Roy, Abhishek Dubey and Aniruddha Gokhale[4] have proposed an autoscaling algorithm, which is dependent on the workload prediction. The algorthim uses the receding horizon control and iterates over the number of look ahead steps and calculates the cumulative costs. For every future time step, it computes the cost of selecting each possible resource allocation by calculating the response time for that particular machine configuration. Once the response time is calculated, it is used to calculate the cost of the allocation which is a combination of how far the estimated response time is from the SLA bounds, cost of leasing additional machines and also a cost of re-configuration. The cost of reconfiguration is computed based on the number of machines that need to updated. Obviously reconfiguration will incur some costs and thus the algorithm will try to reduce the amount of reconfiguration. Each of these cost components will have weights attached to them which may be varied depending on the type of application and its requirements. Applications are required to specify which factors are more important to them, and our auto-scaling algorithm will honor these specifications in making the decisions.[4]

III. PROPOSED WORK RESULTS AND ANALYSIS

After reviewing these works, we find the problem to find an efficient autoscaling policy for a cloud user where scaling can be done based on the overall CPU resources of all Virtual Machines utilized by the particular user instead of calculating it for a single VM.

Based on the above problem, we hereby propose an efficient algorithm that checks for all the VM health's and based on the below specified parameters, it decides whether to scale the VM instance or not.

A. Parameters:

- CPU Utilization
- Virtual Machine (VM) status
- Network Response Time
- Past Heuristic Data

B. Proposed Solution Algorithm

1) The programmer assigns 3 parameters initially:
   - Current CPU load
   - A CPU threshold value
   - A threshold for Network Response Time
2) For all VM nodes, we collect current CPU load and store in a vector array.
3) Then ping each VM and get their Network Response Time. Save it in another vector array.
4) Calculate average CPU utilization and average Network Response Time.
5) Calculate average load from past heuristics information which will be given by the cloud user.
6) Now, the autoscaling function is executed which decides to scale or not. So if average CPU utilization is greater than CPU threshold and average Network Response Time is greater than threshold Network Response Time and average load is greater than current load, scale up virtual machines and if the values are lower than the specified values, scale down.
7) Repeat the same process after specific time interval.

C. Flow Chart

The flow chart for the proposed algorithm is as shown below in Figure 8.

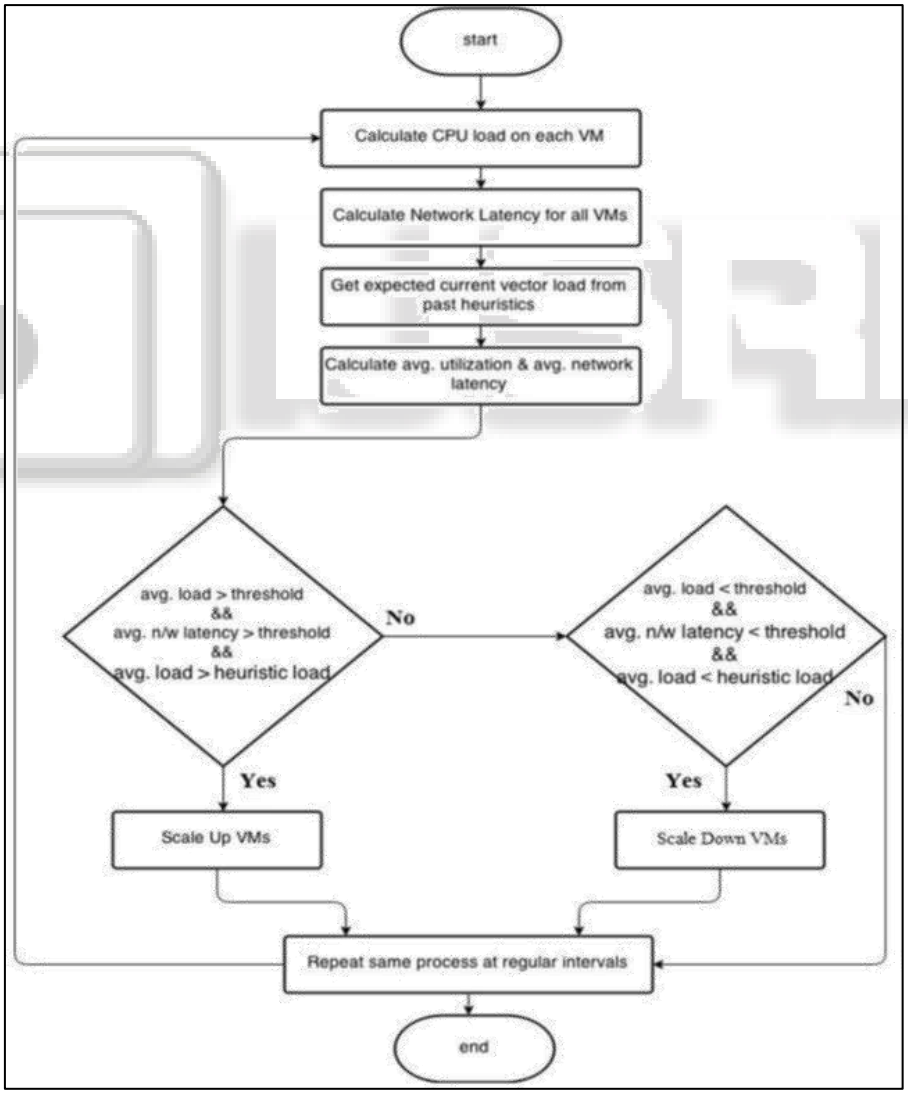

Fig. 8: Flow Chart for proposed algorithm

D. Results and Analysis

The simulations for the above proposed algorithm is carried out in the Amazon Web Services environment. The results suggest that when overall cpu load and network response time increases, the number of active instances also increases. The graph of CpuLoad vs Number of Instances is shown in the Figure 9 in the next page.

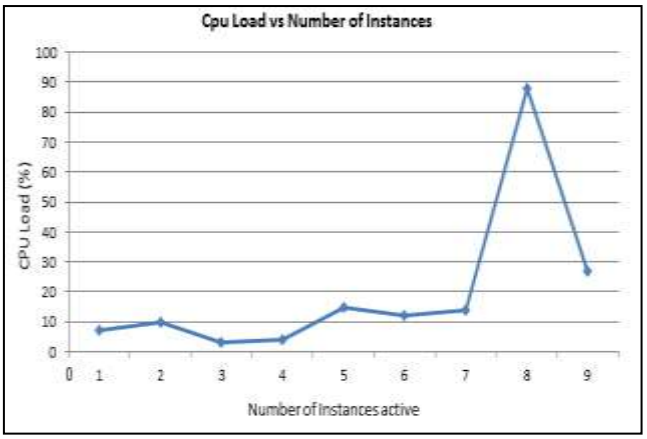

Fig. 9: Graph of CPU Load vs Number of Instances

As and when the time passes, we increase the cpu load to high extent and once the tasks are completed, the load decreases as well. During this time period, the number of instances goes on increasing because the cpu load and the network response time goes on increasing. This causes the algorithm to increase the number of active instances and once the tasks are completed, the number of active instances goes down. This graph of network response time vs Timeline is as shown in the Figure 10 below.

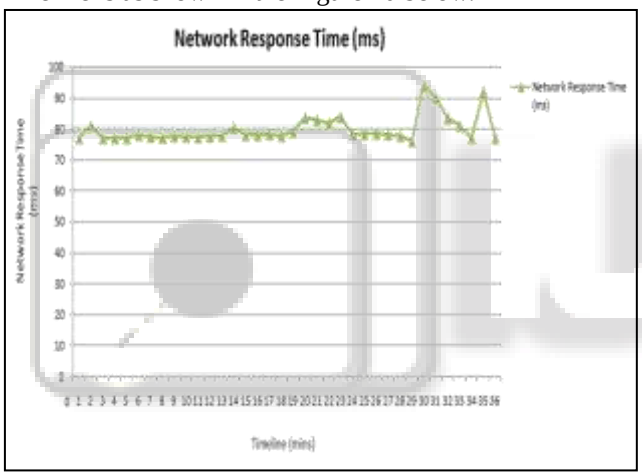

Fig. 10: Graph of Network Response Time vs Timeline

The graph of Cpu Load vs Timeline is shown below in Figure 11.

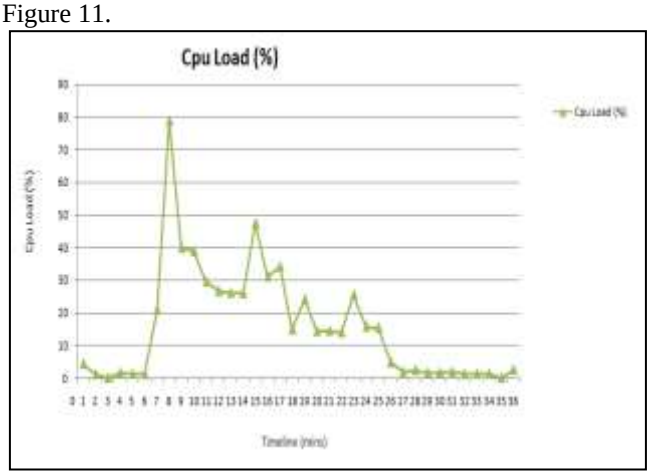

Fig 11: Graph of Cpu Load vs timeline

The graph of number of active instances vs Timeline is shown below in Figure 12.

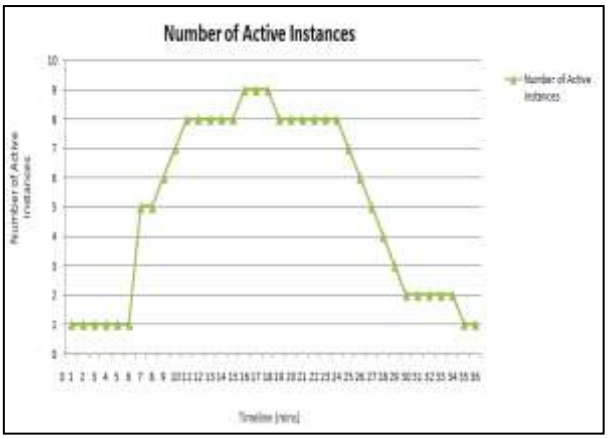

Fig. 12: Graph of Active Instances vs Timeline

From the results generated, it can be seen that the algorithm is capable enough of continuously adding newly created instances when such requirement arises and vice versa. The percentage of requests served is also increased while simulation of the algorithm is carried out. SIAA[4] has a fail count of 10.1% whereas the proposed algorithm has a fail count of 2.0%, whereas on the other hand, SIAA[4] proves better in terms of currently existing CPU usage as it uses 77.7 % of existing CPU resources and the proposed algorithm uses 61.16 % of existing CPU resources.

This comparison is shown in the chart in Fig 13 below. By this analysis, it proves that the proposed algorithm proves much better in a case where sensitive applications are being used as there is a much lower ratio in fail count of arriving requests compared to Spot Instances Aware Algorithm[4].

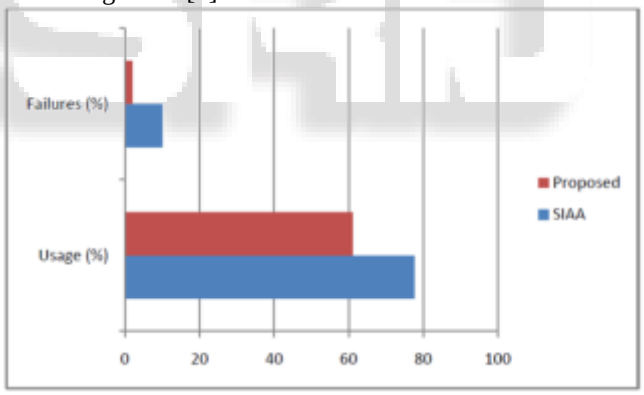

Fig. 13: Comparison of usage and Request Failures

The no. of instances are much less compared to other algorithms when a quite large amount of load is dumped to the VMs. Thus, it may prove cost effective in some instances but this research doesn't provide a trustworthy guarantee that it would work the same in every environment.

IV. CONCLUSIONS

The investigation for the limitations of the existing scaling mechanisms implemented in the publicly available cloud environment is carried out in this research. In order to overcome these shortcomings, a new autoscaling algorithm is proposed. One of the major characteristics of Cloud is on-demand acquisition or release of resources in the Cloud. The automated scaling process benefits both the Cloud service provider and the cloud user. The evaluations are performed on the Amazon EC2 Cloud Service which supports

autoscaling. The results, produced are better and efficiently performs scaling after the experiments. Certain parameters that were considered in isolation for scaling mechanisms have been used here in a combined effort to increase efficiency. The proposed algorithm proves to be much better in serving the requests arriving at the VM for sensitive applications. The current system calculates the heuristic data for cpu load only. The future work can be carried out on the system to collect heuristic data for network latency also. Moreover, the scaling process is done on a particular instance configuration only, which can be extended for different instances configurations. The cpu memory and the threshold limits for the number of VMs running at a particular moment of time can also be taken into account so that more extensions to this work can be carried out. Utilization of VMs on the basis of arriving tasks would be the future work that would be carried out by this thesis as well.

REFERENCES

[1] M. Mao, J. Li, and M. Humphrey, "Cloud Auto-Scaling with Deadline and Budget Constraints," Proc. 2010 IEEE/ACM Int'l. Conf. Grid Computing, Oct. 2010, pp. 41–48.
[2] M. Hasan, E. Magana, A. Clemm, L. Tucker, and S. Gudreddi, "Integrated and autonomic cloud resource scaling, " in iEEE Network Operations and Management Symposium (NOMS 2012), Apr. 2012, pp. 1327-1334.
[3] Y. W. Ahn, A. M. Cheng, M. Jo, and H.-H. Chen, "An Auto-Scaling Mechanism for Virtual Resources to Support Mobile, Pervasive, Real-Time Healthcare Applications in Cloud Computing," IEEE Netw., 2013.
[4] N. Roy, A. Dubey, and A. Gokhale, "Efficient autoscaling in the cloud using predictive models for workload forecasting," in Cloud Computing (CLOUD), 2011 IEEE International Conference on. IEEE, 2011, pp. 500-507.
[5] http://aws.amazon.com/autoscaling/
[6] http://azure.microsoft.com/
[7] http://en.wikipedia.org/wiki/Cloud_computing/
[8] http://aws.amazon.com/ec2/

