PreRative Autoscaling: A Hybrid Approach Between Predictive and Reactive Autoscaling

Anshul Roonwal, Dhanendra Jain, Muhammad Ali Ejaz, Neeraj Dixit, Shahzeb Patel
CSE 591 Spring'16 Final Project Report
Computer Science Department, Stony Brook University

Abstract

Scaling is indispensable at modern data centers due to varying workloads. While lack of scaling may cause poor performance in application or lead to power wastage, manual scaling is infeasible as it requires constant supervision in modern data center. The reactive approach of autoscaling takes action after the request has come up, while the predictive approach estimates the incoming request prior to actual request arrival and turns the servers on and off accordingly. These approaches however have their own disadvantages. The reactive autoscaling alone may cause SLA violations due to unavoidable setup time while the predictive autoscaling, though intelligent in resource allocation, may not be able to handle load spikes. To overcome the individual disadvantages of both, we introduce an optimal hybrid algorithm called PreRative auto scaling - a balanced blend of both Predictive and Reactive auto scaling.

1 Introduction

Power consumption is a major concern at data centers consisting of hundreds and thousands of servers. Once a request comes to a data center, it is fanned out to multiple servers for processing. To serve such request loads, there are multiple servers up and running at all times. But do we need all the servers up and running at all the time? Well the answer depends on the amount of workload. In a typical modern data center, lack of scaling may cause poor performance in application or lead to power wastage. But at the same time, manual scaling is infeasible as it requires constant supervision.

A classic approach to conserve power in such cases is to perform autoscaling. Autoscaling is an approach of dynamically scaling the servers up and down based on the amount of incoming requests. The two common autoscaling strategies used are reactive and predictive autoscaling. Reactive approach of autoscaling takes action after the request has come up while the predictive approach estimates the incoming requests prior to their actual arrival and accordingly turns the servers on and off. However, the reactive auto scaling alone may cause SLA violations during setup time while predictive autoscaling is though intelligent in resource allocation, may not be able to handle load spikes alone. To overcome these individual disadvantages of both, we introduce an algorithm called PreRative autoscaling. Our proposed method is a balanced approach of both predictive and reactive autoscaling. The algorithm starts off with predictive autoscaling based on provided historical data and switches to reactive autoscaling in case of unpredicted load spikes. We switch to reactive autoscaling in case of allocation disobedience or violation and back to predictive, once allocation obedience is achieved (Figure 1).

We used 10 Amazon AWS t2.micro virtual machines as our servers, 1 t2.micro instance as the apache load balancer, and 1 instance to pump load through httperf. After running our proposed algorithm for different kinds and amounts of workloads, we can assert that our hybrid approach is optimal in com-

parison to using predictive and reactive approaches individually.

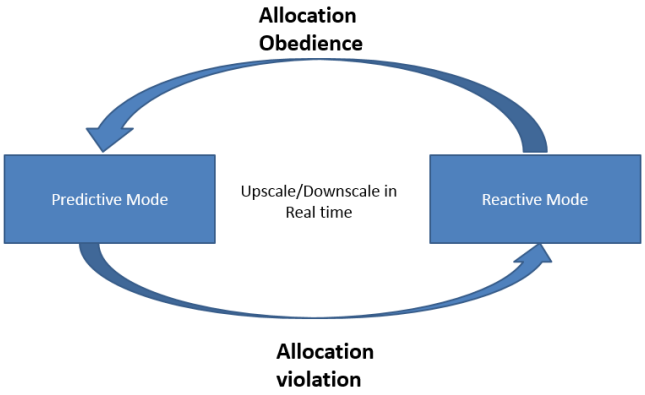

Figure 1: Allocation Obedience/Violation

1.1 Motivation

Scaling is indispensable at modern data centers due to varying workloads. Lack of scaling may cause poor performance in application or lead to power wastage. Why should all the servers at a Data Center should always remain on when not all of them are in use at all times. When the workload goes down, the resources not in use should be scaled down. Now, Manual scaling is infeasible as it requires constant supervision in modern data center. We want to reduce power consumption at Data Centers with use of better techniques. Prior knowledge of the incoming workload can help solve various problems. Solutions to these problem can help manage the resources in a way that could result in significant reduction in power consumption. Both predictive and reactive approaches are used for Auto Scaling and have their own advantages and disadvantages. It would be a good idea to combine both the approaches and make use of the combined advantages while overcoming some of their disadvantages reducing the cost further than the individual approaches.

1.2 Problem statement

Propose a new hybrid approach called 'PreRative Auto Scaling' to up-scale and down-scale resources to an optimal level with good response time comparable to reactive but with reduced power cost than that of always-on strategy. Predictive and reactive algorithms individually prove helpful but can be optimized further. We see a need to find a sweet spot between these two algorithms that can intelligently predicts load and leverages reactive scaling benefits while maintaining SLA.

1.3 Our approach

We are using an intelligent hybrid Auto Scaling algorithm that we call '*PreRative Auto Scaling*' that uses a combination of Predictive and Reactive Auto Scaling by taking the best of both the approaches.

1.4 Contributions

Our major contribution is to develop a PreRative algorithm that is a combination of both Predictive and Reactive Autoscaling algorithms. The predictive algorithm predicts future load and PreRative algorithm switches to Reactive mode in case of an allocation violation. It switches back to Predictive mode in case of allocation obedience. This way, the PreRative algorithm falls back and forth to/from reactive when results from predictive approach violates the threshold value. This threshold value is the serving capacity of one Server. Thus, our approach focuses on exact optimal number of resources to be maintained for the next given time interval. We are predicting load in current interval based on 10 days of historical data. Scaling the servers up and down is being done according to load by enabling/disabling servers on load balancer.

2 Prior Work

In this section, we briefly highlight the various approaches that have been chosen by various researchers who have worked on reducing the power consumption through auto scaling.

Anshul at el. [1] used predictive algorithms for coarse time scale and reactive algorithms for finer time scale through their paper. Jing at el. [2] predicted number of requests based on history data by

exploiting machine learning techniques and time series analysis. Yadavar at el. [3] used prediction algorithms like SVM for periodic and growing workloads and use of Neural Networks for unpredicted workload. Anshuman el at. [4] used predictive algorithm (Linear Regression) to increase intermediate enterprise profit and reduce user cost by handling on Demand requests. Roy at el. [5] describes a look-ahead resource allocation algorithm based on model predictive control. Rodrigo at el. [6] used prediction based on the ARIMA model to evaluate the impact of the achieved accuracy in terms of efficiency in resource. Laura at el. [7]came up with a new elasticity management framework that combines reactive and predictive controllers.

While all the researchers have done a commendable job, our focus significantly differs in combining the two approaches in a way that reduces the number of SLA violations along with conserving power by switching between predictive and reactive approach.

3 Experimental Setup

To perform this experiment, we are using 12 servers in total. We need to simulate the workload and need to have an intelligent load balancer where our PreRative algorithm is also running. The description of the machines used in our experiment and their functions followed by the metrics used in our project is discussed below.

3.1 Amazon AWS VMs

All the machines used in this project were Ubuntu Server 14.04 LTS (HVM), EBS General Purpose (SSD) Volume Type Virtual Machines from Amazon AWS available as t2.micro. We used 12 such virtual machines in total.

3.1.1 Web Servers

We used 10 AWS virtual machines as servers for processing the web requests. These virtual machines were switched on and off based on our algorithm. We used curl command to enable/disable the servers on the load balancer denoting on and off state of the machines. Our experiments were repeated for three types of web server loads:

1. CPU bound workload - For each request, the server did a CPU intensive operation before serving the request
2. Cache Bound Workload - For each request, the server allocated some memory within the L2 cache and accessed it before serving the request
3. I/O Bound Workload - For each request, the server wrote one line to a file (every time overwriting the previous file).

We also tried memory bound workload, but due to highly varied erratic response times (Round Trip Time: 1000, 2000, 1000, 3000, 5000, 3.2, 3.6, ...), we decided to consider this experiment for future work.

3.1.2 httperf

One of the 12 virtual machines we had was specifically used to run httperf for generating load. It would read a specified trace file and accordingly generate amount and pattern of load. We used three types of load for our experiments:

1. smooth trace - a pointed sinusoidal pattern
2. noise trace - 15% added noise in smooth trace
3. mix trace - dual of slowly varying and spikes

3.1.3 Load Balancer

We used the last available virtual machine as our apache load balancer. Our PreRative algorithm also runs on this machine. All our httperf requests are sent to this machine.

3.2 Evaluation Metrics

We used response time, average number of VMs and percentage successful predictions for the evaluations of our proposed PreRative algorithm:

3.2.1 Response Time

Response time is the time required for completion of request measured on the client side. We have measured the SLA response time on our setup by the highest number of requests that all machines can serve at a consistent response time. The response time on our setup was measured by httperf.

3.2.2 Average Number of VM's

This metric is the average number of VM's that are in ON state at any instance. Power consumption is directly proportional to the number of VM's that are powered on by PreRative algorithm. We have measured average number of VM's through a python script which runs on the same machine as that of httperf.

3.2.3 Percentage Successful Predictions

The number of successful predictions done via our PreRative algorithm was considered as our third metric. Ideally PreRative algorithm should be completely on the basis of prediction as reactive autoscaling is used as a fallback approach (in case of obedience violation). The prediction part of our algorithm is directly dependent on the merit of the prediction algorithm used.

3.2.4 SLA Violations

Along with the response time we also need to consider the number of SLA violations occurred. We should have the minimum number of SLA violations (ideally 0) at any given instance. These readings are recorded and plotted on the graph for every workload.

4 Our Solution

As mentioned above, our approach is to have a predictive analysis of the load to scale the number of VMs to serve the load. In case when the prediction goes haphazard, the scaling falls down to reactive mode and scale accordingly. This goes on till the prediction falls into tolerable limits. Our solution implies that both the predictive and reactive autoscaling algorithms will co-exist on the same setup. Another point to note here is that our algorithm lies outside of the load balancer and is implemented via a python script. The outline of our proposed PreRative algorithm is listed below:

Algorithm 1 PreRative Algorithm

```
1:  Read Slave Configuration
2:  Initialize Dataset
3:  while true do
4:      Predict the load for next time interval
5:      Scale according to this predicted value
6:      Wait in time Interval  Then read actual load
        value
7:      Calculate delta between actual and predicted
        value
8:      if delta > threshold then
9:          scale according to actual load (reactive
            scaling)
10:         reactive next time
11:     else
12:         predictive next time
13:     end if
14:     Repeats
15: end while
```

The prediction of future load is done through the following prediction techniques:

1. Weighted Average - Not presented in this report
2. Exponential Average technique - Not presented in this report
3. Exponential weighted Average technique (EW)
4. ARIMA technique (ARIMA)

From our experiments we observed that combining weighted average with a standard algorithm resulted in better predictions and as such we used only exponential weighted average and ARIMA (also weighted) techniques in this report. In these weighted algorithms the recent predictions are given higher weights than the older predictions. This approach helps in re-entering prediction obedience phase back from prediction violation.

We have used a dataset that contains a pattern database of load values for last ten days. This is valid for all types of load with their corresponding learning dataset. The noise in this dataset being approximately 15%. For a given time interval, we use the above two techniques to determine what would be the load on 11th day and so on.

5 Results

5.1 Profiling the load type

Before evaluating the performance of our approaches, we have profiled the machines for each of our approach. We have recorded the consistent response time as observed by httperf at various request rates. SLA is the instance at which the response time shoots up drastically. This is the limit of the maximum requests the complete setup of 10 machines can serve. Below are the images of the profiled data for various types of load - (Figure 2 to 4)

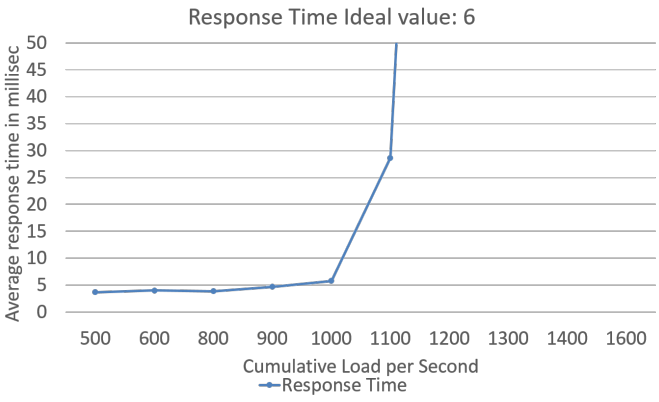

Figure 2: CPU Load Profiling

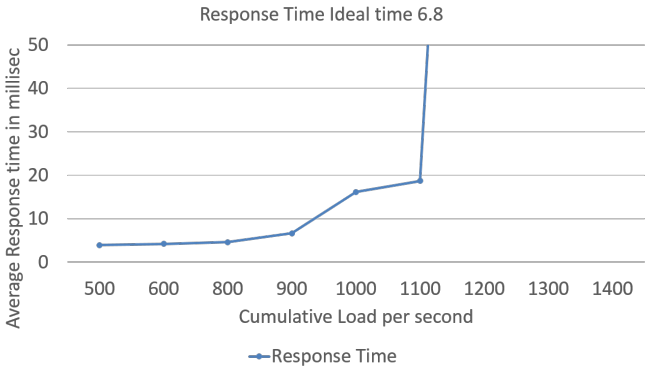

Figure 3: Cache Load Profiling

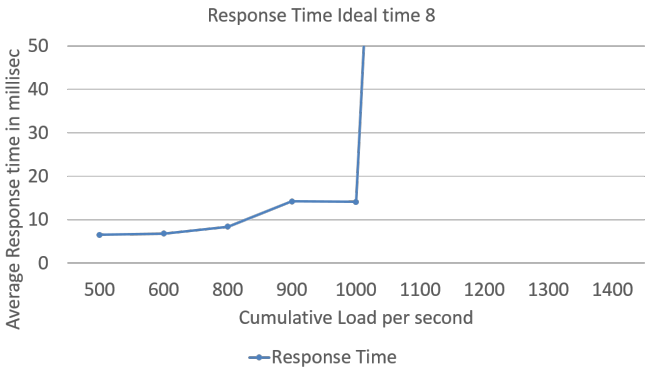

Figure 4: IO Load Profiling

5.2 Resource savings

We now present the evaluation of our approach on different types of loads with different load traces and prediction strategies. We have used a metric of number of VMs running on a given instance to quantify the energy savings. The time duration between successive autoscaling decisions in all the graphs is 10 seconds. We had initially tried with the autoscaling interval of 30 seconds but the SLA was breached for numerous requests when the load was increasing consistently. It is important to note here that the aggressiveness of scaling up is same as that of scaling down.

Below are the evaluations for CPU bound load (Figure 5 to 9) for different traces (Smooth, Noise and Mix trace) and predication algorithms used are:- Exponential Weighted (EW) and Arima. For each graph we have represented number of VMs that are ON at any load instance according to our algorithm. Average number of ON VMs are also mentioned. We can

observe that in comparison to Exponential Weighted (EW), number of on VMs for Arima at any load instance are more.

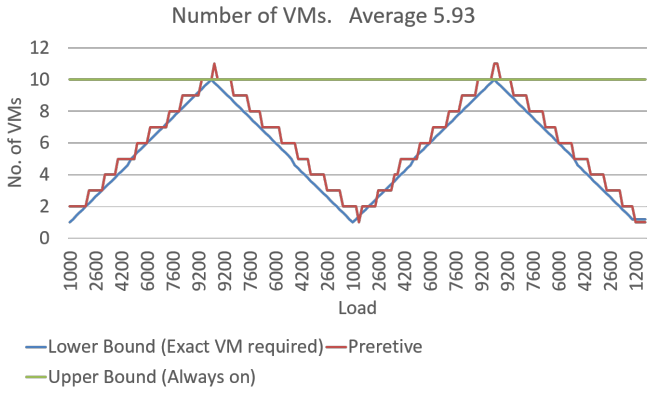

Figure 5: CPU, EW prediction and Smooth trace

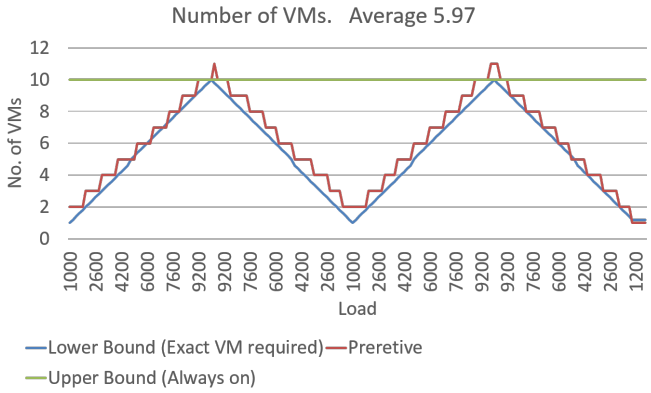

Figure 6: CPU, Arima prediction and Smooth trace

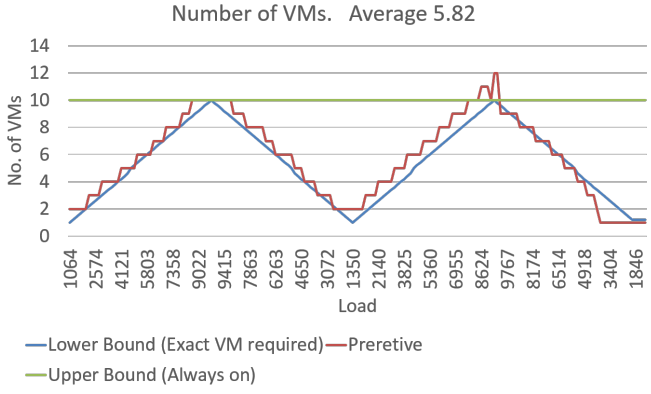

Figure 7: CPU, EW prediction and Noisy trace

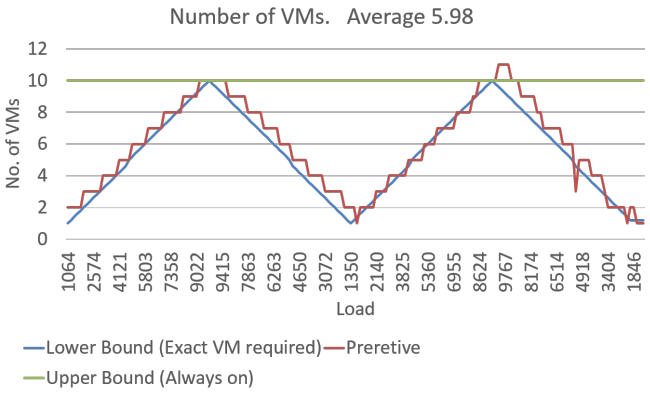

Figure 8: CPU, Arima prediction and Noisy trace

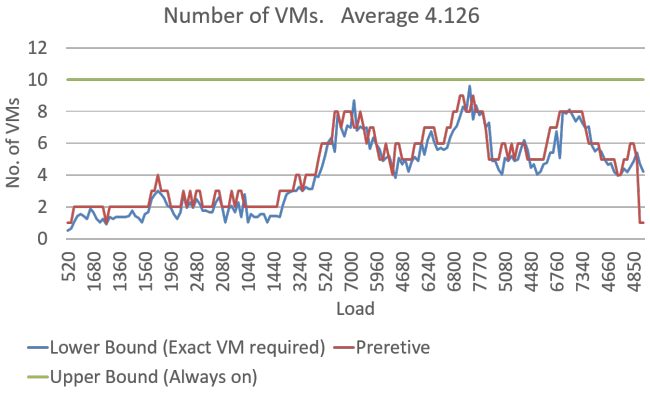

Figure 9: CPU, EW prediction and Mix trace

We evaluated the cache bound workload with different load type and prediction strategies. The results of these experiments is shown in (Figure 10 to 14). It can be observed that in comparison to CPU bound load, average number of ON VMs for Cache bound at any load instance are more, except for mix trace.

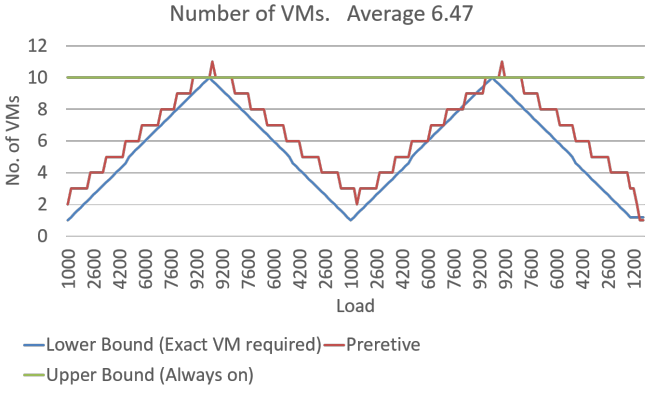

Figure 10: Cache, EW prediction and Smooth trace

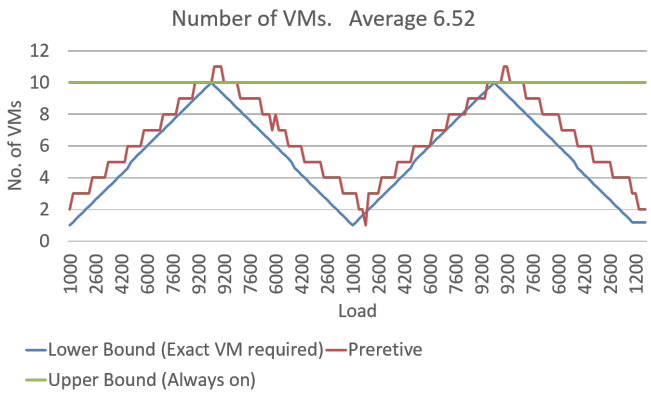

Figure 11: Cache, Arima prediction and Smooth trace

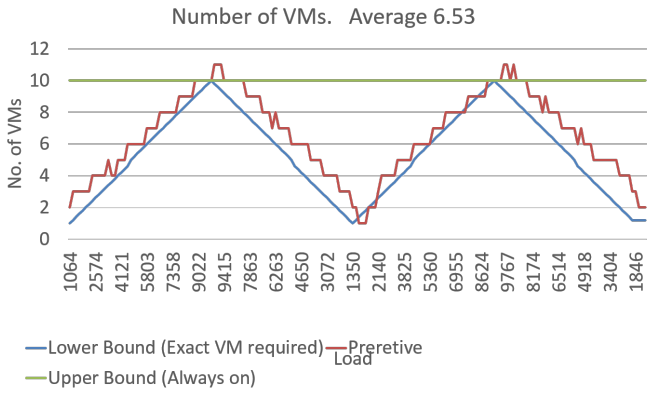

Figure 12: Cache, EW prediction and Noisy trace

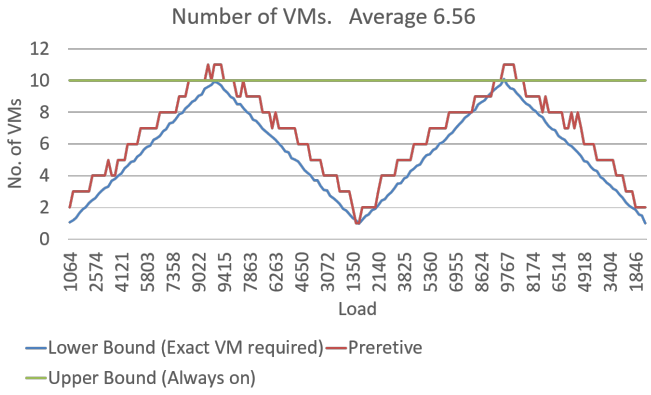

Figure 13: Cache, Arima prediction and Noisy trace

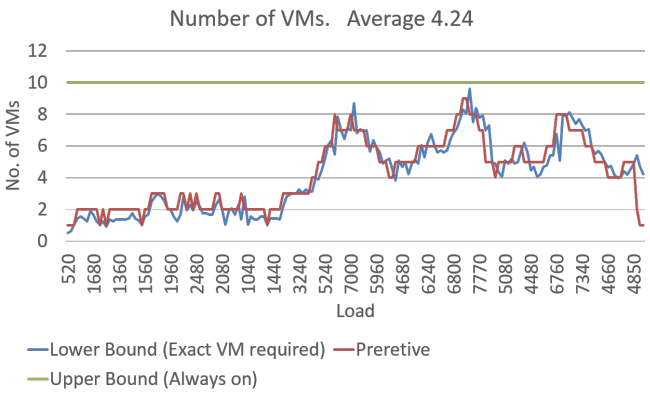

Figure 14: Cache, EW prediction and Mix trace

We have repeated the same workloads for IO bound server (Figure 15 to 17). The IO bound server is built by reading and writing a file on the disk. This type of workload also checks whether we can use prediction autoscaling for IO bound servers.

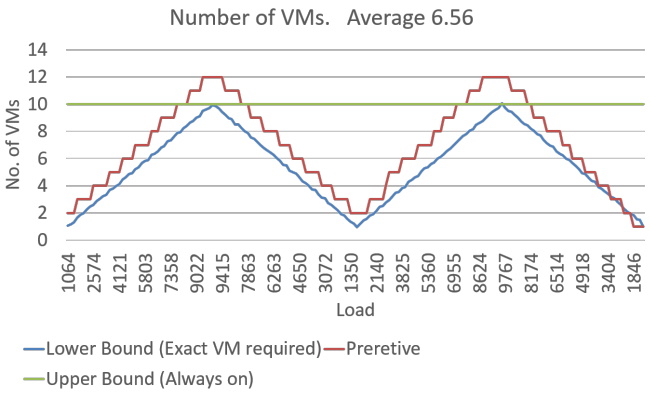

Figure 15: IO, EW prediction and Noisy trace

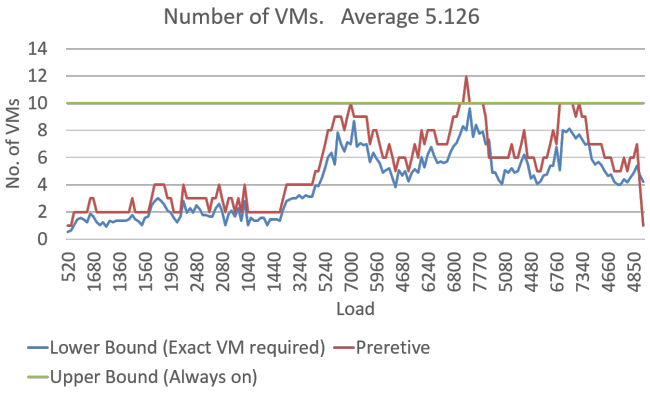

Figure 16: IO, EW prediction and Mix trace

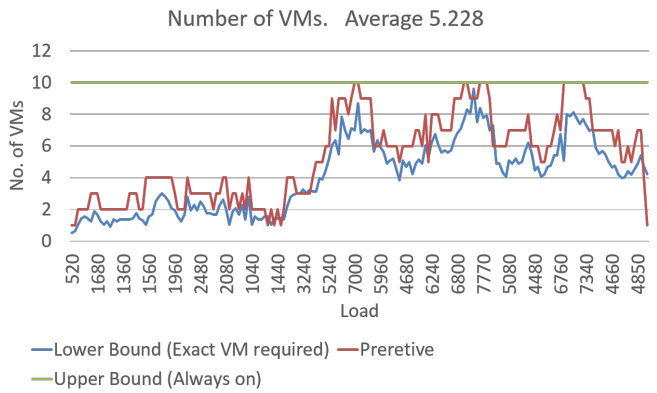

Figure 17: IO, Arima prediction and Mix trace

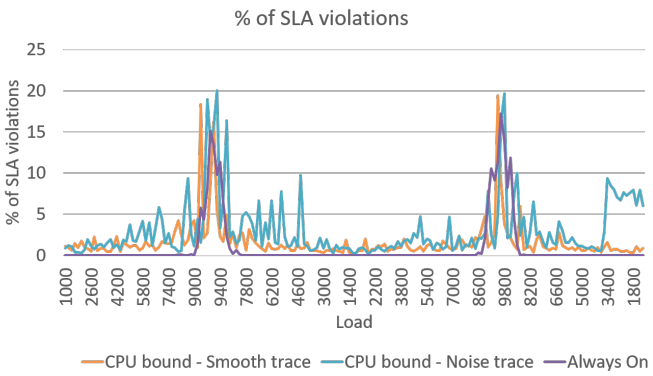

Figure 18: SLA violations on CPU load

5.3 SLA Violations Evaluation

SLA violation is determined on the basis of number of requests that have response time which exceeds the SLA determined during profiling in (Figure 2 to 4). Now we try to quantify the SLA violations and highlight it through graphs ,various scenarios where we saw major difference in SLA violation readings. For each graph, we calculated percentage of SLA violations corresponding to load at that instance. SlA violations for all three types of loads (CPU, Cache and IO) are represented for Smooth and Noise traces and compared with Always On condition (Figure 18 to 20). SLA violation are also represented for mix trace and comparison is done among CPU, Cache and IO load (Figure 21). It can be seen that, for mix trace SLA violations for CPU workload are highest and that of IO workload is lowest. We also compared SLA violations in terms of different predication algorithms - Exponential weighted (EW) and Arima (Figure 22), we can see that SLA violations for Arima are more in comparison to Exponential Weighted prediction algorithm.

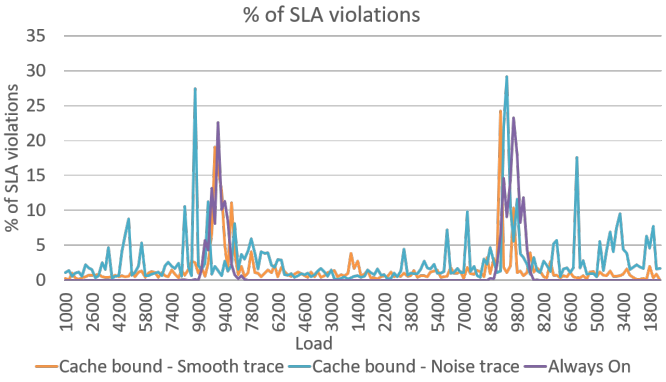

Figure 19: SLA Violations on Cache load

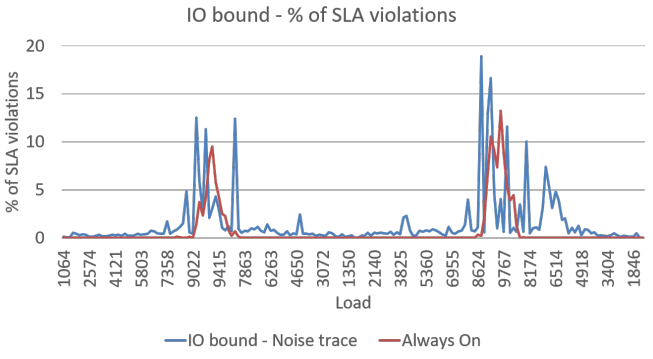

Figure 20: SLA violations on IO load compared with always on environment

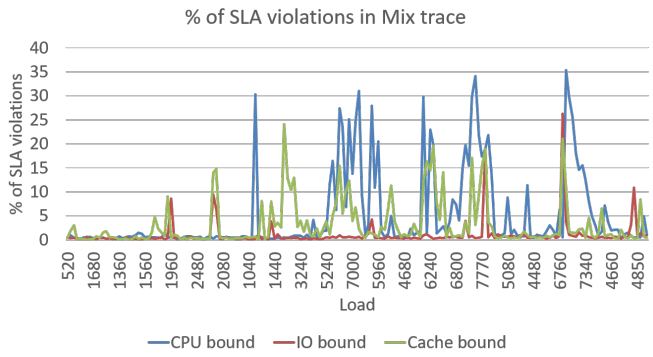

Figure 21: SLA violation on all load types

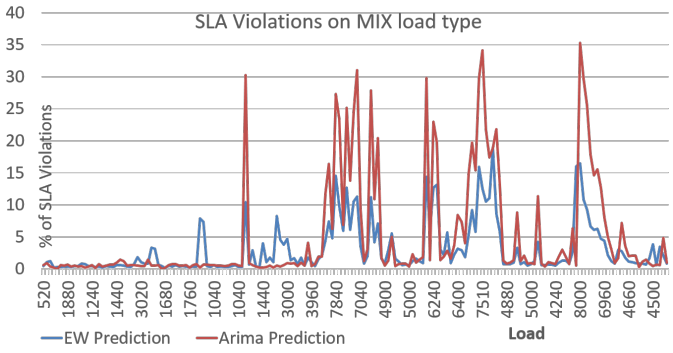

Figure 22: SLA violation with different prediction methods

6 Conclusion and Future Work

From the readings, we concluded that exponential weighted average is better than ARIMA for our PreRative autoscaling. This is concluded from the SLA violation graph, we can see more SLA violations with Arima algorithm as compared to that of exponential weighted algorithm. We feel that this is due to the error prone ARIMA method as compared to EW. We have done tight profiling for CPU and cache bound Load. The point when we reach highest CPU utilizations, the response time were drastically hit. That was the reason we saw more SLA violations when load pump rate was close to VM capacity. Hence we can say that prediction makes sense only for the pattern based load where you have more than half the chances of load incoming based on some pattern (eg. e-commerce sites). There is a direct relationship between spikes and SLA violations. Violations lead to response time latency and such errors slack and their repercussions are seen in next requests. Smaller time intervals between scaling leads to better and accurate scaling decisions Downscaling of VM's is also done as aggressively as up scaling. From the SLA violation we have seen the violation slack propagates during downscaling. Also, on an average, we observed that autoscaling decisions for CPU workload are: 73% Predictive, 27% Reactive. For cache bound load we observed 64% Predictive, 36% Reactive autoscaling decisions. We observed the best predictions on I/O bound workload: 81% Predictive, 19% Reactive.

From the graphs readings, average number of VMs for I/O bound workload is higher than CPU bound workload as well as Cache bound workload. Also the minimum number of SLA violations were seen on I/O bound workload. We have concluded that our PreRative algorithm works better if request serving time is higher (or higher service time)

One of the future work that we can think about is to adopt prudent and sane downscaling. Our setup was based on simulated data (sinusoidal). Another area of improvement is the precision of prediction by using sophisticated machine learning algorithms. But we have to ensure that this new algorithm should not slow down the load balancer in anyway. It would be better if we can use actual data for training the prediction algorithms used in PreRative algorithm. Although we have written the implementation code for using last 10 days as well as last 10 interval load values to predict the next load, we couldn't evaluate that due to the given time constraint.

References

[1] Anshul Gandhi, Yuan Chen, Daniel Gmach, Martin Arlitt, Manish Marwah IGCC 2011: hourly predictive, per minute reactive.

[2] Jing Jiang, Jie Lu, Guangquan Zhang, Guodong Long - 2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing.

[3] Ali Yadavar Nikravesh, Samuel A. Ajila, Chung-Horng Lung - 2015 10th Interna-

tional Symposium on Software Engineering for Adaptive and Self-Managing Systems.

[4] Anshuman Biswas, Shikharesh Majumdar, Biswajit Nandy, Ali El-Haraki. - 2015 IEEE.

[5] Roy, N., A. Dubey, and A. Gokhale [4th IEEE International Conference on Cloud Computing (Cloud 2011)].

[6] Rodrigo N. Calheiros, Enayat Masoumi, Rajiv Ranjan, and Rajkumar Buyya [EEE TRANSACTIONS ON CLOUD COMPUTING, VOL. 3, NO. 4, OCTOBER-DECEMBER 2015].

[7] 3)Laura R. Moore, Kathryn Bean and Tariq Ellahi [CLOUD COMPUTING 2013 : The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization].

[8] https://weblab.ing.unimore.it/people/sara/papers/casolari-valuetools2006.pdf

[9] https://en.wikipedia.org/wiki/Exponential_function