CloudScale: Predictive Elastic Resource Scaling for Cloud Systems

Sweekrut Suhas Joshi
North Carolina State University
Raleigh, NC
ssjoshi2@ncsu.edu

Suhaskrishna Gopalkrishna
North Carolina State University
Raleigh, NC
sgopalk@ncsu.edu

Bharath Banglaore Veeranna
North Carolina State University
Raleigh, NC
bbangla@ncsu.edu

ABSTRACT

One of the major challenges in resource scaling is to provision optimal amount of resources such that the applications running on the cloud systems meet their service level objectives (SLOs) with minimum incurred costs. In the proposed project, we plan to implement a resource scaling system similar to CloudScale which will employ vertical scaling techniques and migration to resolve scaling conflicts between applications. The system should be able to achieve adaptive resource allocation by performing live resource demand prediction and prediction error handling. We intend to implement this on top of Xen and evaluate by conducting a set of CPU and memory intensive experiments using RUBiS.

CCS CONCEPTS

•**Computer systems organization** →**Distributed architectures;** *Cloud computing;*

KEYWORDS

Resource scaling, Resource capping, Cloud computing

ACM Reference format:
Sweekrut Suhas Joshi, Suhaskrishna Gopalkrishna, and Bharath Banglaore Veeranna. 2017. CloudScale: Predictive Elastic Resource Scaling for Cloud Systems. In *Proceedings of CSC724: Advanced Distributed System, NCSU, May 2017 (NCSU'17),* 9 pages.
DOI: 10.475/123_4

1 INTRODUCTION

Cloud computing provide various services such as IaaS, SaaS, HaaS etc., on an on-demand basis to the customers. Optimizing the available resource in the system proves to be economically advantageous to both the service provider as well as the customer. Overallocation of resources will lead to wastage of resources in the system as well as the customer ends up paying for more than what is being used. Underallocation will impact the performance of the application and may cause a violation in the SLO. Designing a system to automatically allocate the resources to the application based on the prior history of usage and its future requirements will improve the performance of the system and also benefit economically.

The resource usage in a distributed system varies dynamically with the workload and the applications running on the system. Statically allocating resources proves to be inefficient as it does not account the peak usage and often leads to poor utilization of the available resources. Allocating the optimum quantity of resources is challenging in such an environment considering the factors of underutilization and overutilization. In this project, we implement a dynamic resource prediction and capping technique to automatically provide sufficient resources to the application. The proposed system consists mainly of three components: a resource tracking tool to measure the resource usage of each of the Virtual Machines(VM) running on the hypervisor, a prediction algorithm to calculate the amount of resource which the VM might need and a capping method to limit the resource accessed by the VMs. The monitoring process collects the resource utilization periodically and generates signatures based on Fast Fourier Transform. Using the signature obtained, the resource requirement is predicted and caps the resource allocated to each VM. During resource scaling conflict, the VMs are migrated to avoid SLO violations.

The paper makes the following contributions:

- Implemented a prediction model which handles cyclic and acyclic workloads
- Implemented Fast-Underestimation correction to minimize the effect of underestimation
- Retraining based on past prediction errors
- A load balancing mechanism to avoid SLO violation during VM migration
- Compare performance of the proposed model against several other algorithms using RUBiS workloads

The paper is organized as follows: Section 2 gives an overview of the system design and implementation of CloudScale. Section 3 describes the Experimental evaluation of the proposed system. Previous work related to our implementation are described in the Section 4. Section 5 discusses about the limitations and future work of the paper and finally Section 6 gives the conclusion.

2 SYSTEM DESIGN

This section describes the system design principles and the implementation choices deployed while designing the system. This section describes the overall system design followed

NCSU'17, NCSU
 123-4567-24-567/08/06...$15.00
DOI: 10.475/123_4

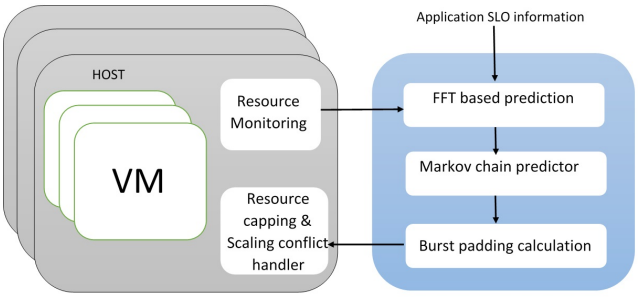

Figure 1: Architecture of CloudScale

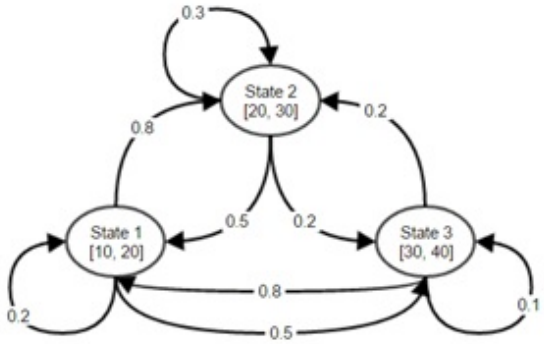

Figure 2: Markov chain state based prediction

by the detailed description of each of the subsystems of CloudScale.

2.1 Overview

The overall system of CloudScale mainly consist of three parts: a resource monitoring tool to gather the runtime resource usage statistics of each VM, a prediction algorithm to predict the future resource requirement of the VMs and a resource scaling algorithm to cap the resource usage of the VMs and handle scaling conflicts by VM migration. CloudScale handles both the CPU and memory resource of the VM for prediction and capping. The Figure 1. Illustrates the overall design of the system. The design of CloudScale is itself a distributed application. The different subsystems of the CloudScale will be running on different hosts.

The main monitoring tool will be running on each of the hosts independently. The monitoring tool is responsible for collecting the CPU and memory usage of all the VMs belonging to that host at regular intervals of time. The CPU usage of all the VMs are collected on the Domain-0 of all the hosts and the VM memory usage is collected from all the VMs individually. The collected resource usage is then given to a prediction algorithm running on a different host.

The prediction algorithm mainly consists of two parts: a state driven approach and a signature based prediction. The resource prediction is done based on the resource usage signature pattern extracted using FFT. If the system fails to detect any repeating pattern in the resource usage, it uses the state driven approach using Markov model. The prediction algorithm will be running on an entirely different dedicated host. This approach was adopted to have the least performance impact on the VMs running on the hosts. The initial deployment of the prediction algorithm was done on the Domain-0 of the host where the monitoring tool was running. It was observed that the prediction algorithm uses approximately 60 to 70 % of the CPU, which will degrade the performance of the VMs running on the host. In the current design, the prediction algorithm runs on a different host.

The prediction algorithm gives the futuristic resource usage value of each of the VMs to the scaling handling system running on the Domain-0 of all hosts. The resource allocation of the VMs are capped based on the predicted and padding values given by the predictor. If the resources available with the host cannot accommodate the resource demand of the VMs, the VMs are migrated to a different host to handle scaling conflict.

2.2 Pattern based prediction

Some repetitive computations or requests causes the resource usage to follow some pattern[10] [12]. CloudScale derives a signature for the pattern and uses the same in its prediction.

Using the last few observed resource usage values, we apply FFT (Fast Fourier Transform) [1] to find the dominating frequency f_d for the same. From the obtained result, the frequencies with the most signal power are chosen. If there are more than one frequency with the same amplitude, Cloud Scale choose the one with the lower frequency. This frequency represents the longest repeating pattern observed in the given input range.

The pattern window size Z can be calculated as $Z = (1/f_d)$. If we have the input series as $L = \{l_1, ., l_w\}$ then $\{l_1, , l_Z\}$ is one pattern and $\{l_{z+1}, , l_{2Z}\}$ is the same pattern. We divide the original series L into $Q = W/Z$ windows: $P_1 = \{l_1, , l_Z\}$, $P_2 = \{l_{Z+1}, , l_{2Z}\}$, ..., $P_Q = \{l_{(Q1)Z+1}, , l_{i,QZ}\}$. To make sure that input series contains repeating pattern, we check the similarity between all pairs of pattern windows P_i and P_j. If the Pearson correlation value of any two pattern windows is close to 1 (i.e., > 0.85) then they are considered to be similar. If all pattern windows have a Pearson value close to 1, then the pattern is considered to be repeating. Once a pattern has been detected, Cloud Scale then finds a signature S of length Z by taking the average value of samples in each position of the window Z.

To be able to predict the resource demand in the future, we need to know where the system is in the signature window. We apply dynamic time wrapping on the Z most recent resource usage values S' and S to get the offset among the two series. This allows us to determine what will be the resource demand in the immediate future using the values from signature pattern S.

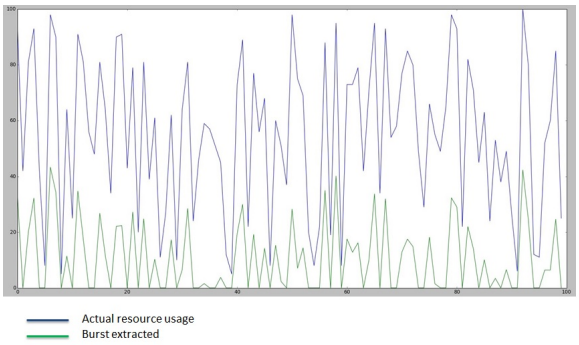

Figure 3: Burst padding using FFT

2.3 State Driven prediction

For input values which have no repeating pattern, CloudScale falls back to discrete Markov Chain (with finite states) based short term prediction [16]. In our design, we consider all resource metric values to be in the range 0 to 100. We divide these 100 symbols into M equal width bins where each bin represents a state. In our approach, we take M as 50 (See figure 2). Thus, we have two symbols per state. Using the last few resource usage values as the input, Cloud Scale learns the transmission probability matrix Px. This is a M x M matrix where an element p_{ij} (at row i and column j) denotes the probability of transitioning from state i to state j. Thus, given a current state, we can determine its next state in a similar manner. For long term prediction, we use the Chapman Kolmogorov equations: after time t, $\pi_t = \pi_{t-1}P_x = \pi_{t-2}P_x^2 = \ = \pi_0 P_x^t$, where t and 0 denote the probability distribution at time t and the initial probability distribution. Given a state i, we calculate the future state j at time t by using the i^{th} row in the matrix Pxt. The Figure 2 shows three states with values from 10 to 40. The arrows show the transition probabilities from one state to other two states and itself.

2.4 Error Correction

Due to the application, independent nature of CloudScale, the predicted resource usage might be erroneous and lead to over or under estimation of resource. To avoid underestimation of the resource, CloudScale uses both proactive and reactive approach to correct the estimation error.

(1) **Burst padding:** The overall resource allocated to a VM should be much more than the predicted value to accommodate any burst error in usage. To have a sufficient difference between the actual usage and the capped value, a padding value must be added to the resource predicted value. The burst pad is determined by using Fast Fourier transform of the last hundred resource usage pattern. The FFT coefficients of the top 80% frequency components are extracted and an inverse FFT is applied to it. This eliminates the low frequency repeating patterns in the input signal and the burst pattern is obtained. Figure 3 shows the burst pattern extracted for a sample input.

The burst padding value is determined by calculating burst density, which is the number of positive values in the burst pattern. If the burst density is greater than 50%, the pattern is considered to have large number of bursts and the padding value is calculated as the maximum value in the burst pattern. If the burst density is low, the pattern contains fewer bursts and the 80th percentile in the burst pattern is considered as the padding value.

(2) **Remedial padding:** Although the burst padding accounts for proactive error correction, the errors could still occur during runtime. Remedial padding is a reactive based error correction used to pad the values based on the occurrence of error in the previous cycle. The error value et at current time t is determined by checking the predicted value xt and the actual value x for the current instance of time t. The error et is defined as the difference of predicted value xt and the actual usage value x. The over estimation errors ($e_t > 0$) are ignored i.e., et is set to 0 if $e_t > 0$. For all underestimation errors, the last k error values are maintained and the weighted average of the error values are obtained. For a window of k, the error values will be $\{e_t, e_{t-1}, .. e_{t-k}\}$. The weighted average of the k error values denotes the remedial padding.

While applying the padding value to the VM, the maximum of the burst padding and remedial padding is considered. The overall padding value obtained is added to the resource demand prediction and the resultant is used to cap the resource allocated to the VM.

(3) **Fast Underestimation Correction:** CloudScale should correct underestimation errors as soon as possible to avoid any SLO violation. The underestimation errors are determined when the actual resource usage is more than 90% of the predicted value. During such scenario, the resource capping value is exponentially increased until underestimation error reduces. To increase the capping value exponentially, the capping value is multiplied by a scaling factor α. After each sampling period, underestimation error is checked and the capping value is multiplied by α. If the underestimation error still persists during the next sampling cycles, the value of α is increased by 0.1 until underestimation error reduces to zero.

For memory resource capping, the value of α varies between 1 to 1.5 since the memory usage growth is slow and exponential. For CPU usage, the value of α ranges between 1 to 2 as the CPU usage can rapidly increase in a short duration of time. Increasing the capping value will eliminate the underestimation error for a short duration. To have

a better prediction technique, the prediction algorithm must be retrained during frequent occurrence of underestimation errors. CloudScale retrains the Markov model when three consecutive underestimation error occurs.

2.5 Alternative prediction approaches

To compare our prediction algorithm with other techniques, we have implemented the following prediction algorithms:

- Mean: The predicted value for the successive sampling cycle will be the mean of the samples collected in the previous W cycles.
- Max: The maximum value among the previous collected input sample denotes the predicted value.
- Histogram: For a given window size W, the histogram is computed with bin size as 40. The predicted value is the average of the samples in the bin having largest histogram value [8].
- Auto-correlation: This prediction algorithm is based on the degree of correlation between the input samples. An input of window size W is shifted n times and after each shift, the degree of correlation is calculated. If the correlation coefficient is greater than 0.9, the data is said to be correlated with duration n steps [7]. The predicted value will depend on the position of the current input value and the predicted value is determined by checking the next value in the repeating pattern. If the correlation is less than 0.9, the mean value is used as predicted value.
- Auto-regression: Auto-regression uses the auto-correlation coefficients coupled with the previous input values to determine the next predicted value. For a given set of previous input values X_t, X_{t-1} etc., the auto-correlation coefficients a_0, a_1, etc. are calculated for a window size W. The next predicted value is calculated as a weighted sum of the previous input values, and the weights are the the auto-correlation coefficients [9].

$$X_{t+1} = a_0 + \sum a_i \times X_{t+i}$$

2.6 Resource monitoring and Scaling

A module in each of the hosts is responsible for collecting resource usage from each of the VMs running on the host and sending it to the prediction algorithm. After obtaining the prediction result, the resource is capped and the capped value is equal to the sum of the predicted value and the padding value. The CPU resource usage of all the VMs are collected using the *libxenstat* library provided by Xen Hypervisor. The memory usage is collected from the VMs directly by running a simple daemon in each of the VM which collects memory usage using */proc* command and sends it to the Domain-0. CloudScale uses a sampling interval of 10 seconds and the resources are collected every 10 seconds. The resource usage of all the VMs must be calculated simultaneously at the same instance of time for predicting scaling conflicts. CloudScale uses a coarse-grained time synchronization by using a trigger mechanism. The monitoring tool in the Domain-0 triggers each of the VM at the same time to send the memory usage and the CPU usage is also collected at the same time. The prediction algorithm is invoked immediately after collecting the statistics. CPU capping is done using the Xen Credit Scheduler [4] and the memory capping is performed using the xentool [6] library provided by Xen Hypervisor.

2.7 Scaling Conflict Handling

Scaling resources is an effective way of resource utilization, but there may be times when there is a conflict in scaling. The cumulative resource requirement predicted for the application VMs on a host might exceed the actual available resources. Such a scenario leads to a conflict. We are using a migration based approach to overcome this conflict. Further, for the conflict, we are considering only the CPU resource but the same technique can be extended for memory and network resources.

2.7.1 Conflict Prediction. Scaling conflict is determined if the cumulative predicted CPU usage of all the application VMs exceeds the resource pressure which is fixed. For our experiments, we considered a resource pressure of 90% i.e. if the required CPU is predicted to be 90% of the available CPU, then we declare it as a conflict. We are using a long term prediction of 100 seconds to determine if a conflict will occur [28]. However, we do not trigger migration immediately after a conflict is detected. Once conflict is predicted, we wait and check again for two more steps. If the conflict still exists, then we proceed with the migration. This is done to avoid inaccurate predictions and short term conflicts.

2.7.2 Conflict Handling. Migration is a CPU intensive task. Migrating applications after the conflict occurs inflicts serious service degradation. It causes significant SLO impact to both migrated and non-migrating applications. One way to counter this is to migrate the application by predicting the conflict in advance [CloudScale]. So, if a conflict is predicted in the future at time Tc which is tc seconds away, we start the migration now such that it is completed within tc seconds. The lead time tc depends on how far ahead we are looking into the future. Selecting how far to look ahead can be complicated. The time to look ahead should be large enough to handle migration when conflict is predicted, and short enough to get good accuracy in the prediction. In our experiments, we are looking 100s ahead as the migration time under normal load was experimentally found to take around 40s in our case.

Pre-copy live migration supported by xen is used for migrating the VMs. We consider the target host where the VM is migrated to, to be known beforehand. Furthermore, for our migration, we employed a new approach with load balancing. We are considering a scenario where a host is running multiple application servers which have identical functionality. In such a case, if conflict is predicted, we evaluate the best application server VM to migrate. The best application VM is the one which has lower CPU load and

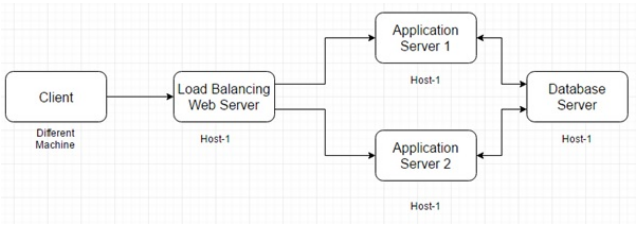

Figure 4: RUBiS setup for testing

when migrated, avoids the predicted conflict. Now, once the VM to be migrated is determined using this criterion, the load is changed such that all its requests are handled by other application VMs present on the host. Then, the VM is migrated to the new host and the load is redistributed back as it was initially. During this time, capping is disabled so that other VMs can handle the extra workload. The benefit of this approach is that the migration time reduces and also, the response time SLO violations reduce considerably.

3 EXPERIMENTAL EVALUATION

This section illustrates the evaluation and execution of the test cases to validate the performance of CloudScale. The performance of CloudScale is also compared with other existing prediction techniques.

3.1 Test Setup and Workload

We implemented the system on top of Xen Virtualization Platform [5] and performed experimental studies using PHP version of RuBIS[2] online benchmark. All of our experiments were conducted in NCSUs Virtual Computing Lab [3]. Each host was run on an x86_64 Intel dual core processor with frequency of 2 GHz and a memory of 2GB. The domain-0 was running a Ubuntu-14.4 image. The guest VMs also used Ubuntu-14.4 image. For all experiments, we pinned the guest VMs to core 1 while domain-0 could use either cores.

Resources are monitored at a sampling rate of 10s. Prediction is done for the next 10 steps, with each step being 10s. Scaling is also done once every 10s. The prediction is done at two scales: one for short term prediction, which is used to cap the resource usage for the next instance of time and another long term prediction, which is used to determine the scaling conflict. For a short term prediction, the predicted value determined is for the successive 10 seconds. For a long term prediction, we are using a look ahad window of 100 seconds.

For the test setup, we built the RuBIS system with a Web Server, two Application Servers and a Database server, all running as guest VMs on the host. Figure 4 shows the setup used for running RUBiS workload. The RuBIS client ran on a separate machine. Response time to requests were considered as the SLO in the framework. A delay of more than 3s was considered to be a violation of SLO. This was made configurable with modifications to the RuBIS client. Also, the RuBIS client would notify the notification server

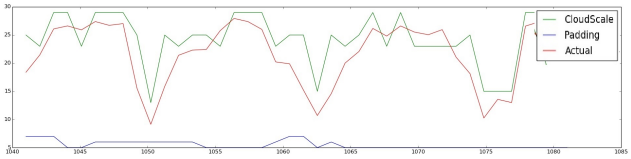

Figure 5: Prediction using CloudScale

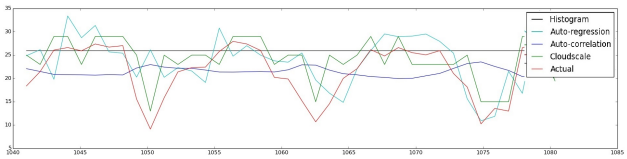

Figure 6: Prediction using other techniques

each time a violation occurred. The notification server was also made configurable. In our case, the server running the prediction algorithm was also the notification server.

The workload used for the experiments was a standard RuBIS workload consisting of three phases up-ramp, run-time, and down-ramp. The up-ramp phase is the phase when the client requests start to gradually increase. The run-time is when the request rate is maintained constant. And during the down-ramp the number of requests reduce. The up-ramp and down-ramp slowdown factor were set to 1. This factor indicates how gradually the increase or decrease happens. The workload number of database users was set to 100,000 and the number of clients was set to 300. The RuBIS standard browse_only_03.txt transition table was used for the request types, with a max number of transitions set to 20,000. For comparison, we also implemented other prediction algorithms as described in the previous sections.

3.2 Result

The evaluation results from the experiments performed show a positive indication towards distributed online predictive elastic resource scaling. Each of the results are explained in the following sections:

Figure 5 shows the plot of the predicted value over time for CPU usage prediction. As seen in the figure, the predicted value follows the actual usage pattern. The predicted value shown in the figure is the value prior to the padding. The padding value is shown the same figure below. When the padding value is added to the predicted value, the capped value will be much higher than the actual usage pattern.

Figure 6 shows the comparison of the CloudScale prediction techniques with other prediction techniques. From the figure, it can be seen that the prediction given by CloudScale is follows the actual usage pattern more accurately than other prediction techniques. Although this gives a qualitative result, to have a better comparison, we need to measure the error rate of prediction using CloudScale and other prediction techniques.

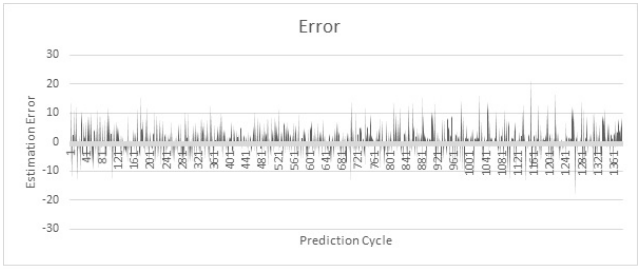

Figure 7: Under estimation and over estimation error showing on positive and negative y axis.

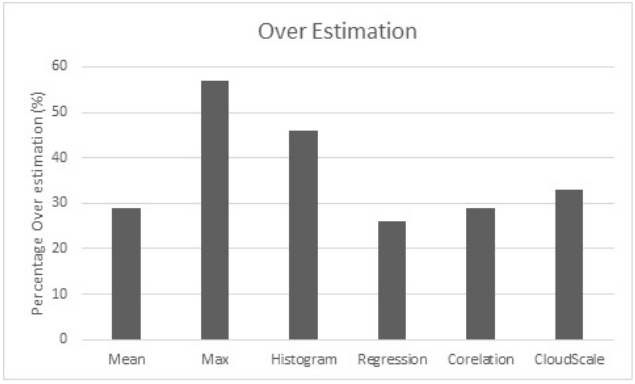

Figure 8: Percentage of over-estimated predictions for different algorithms

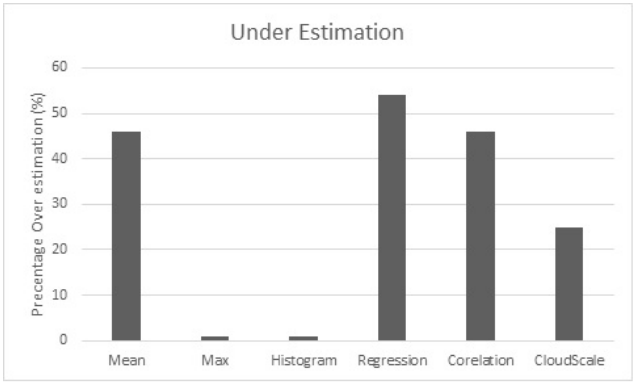

Figure 9: Percentage of under-estimated predictions for different algorithms

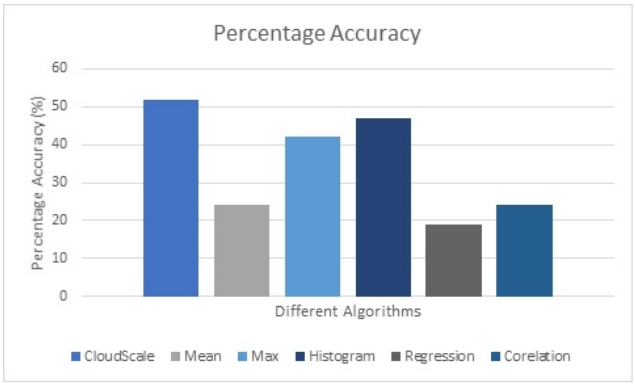

Figure 10: Plot of percentage accuracy for different algorithms keeping an error tolerance of 10%

For each of the test case mentioned in this paper, we used a training dataset of 3000 input samples. The cloudScale system was executed by disabling the capping mechanism. The CPU and memory usage of the VMs ere collected every 10s until 3000 samples were collected. During this phase, the RUBiS workload was running in all the VMs. The workload used for both the training and testing is same as described in the previous section. After obtaining 3000 samples of CPU and memory usage of all the VMs, the CloudScale was executed online. During this phase, CloudScale would have obtained sufficient input values to check for signature pattern and also train the Markov model.

To visualize the extent of under and over estimation, we plotted the error with under estimation errors showing on negative y axis and over estimation on positive y axis (Figure 7). For the sake of simplicity, we only consider values where the absolute value of error is 10% of the actual value (i.e., $|e| > 10\%$). As inferred from the graph, our prediction algorithm does quite well in terms of estimation error although some errors are relatively higher than others. To compare our algorithm to others, we plot percentage underestimation and percentage over estimation for CloudScale and several other algorithms (Figure 8 and 9). As seen, CloudScale is neither the highest nor the least in both under estimation or over estimation. As expected, max and histogram approaches always over estimate and thus have very low under estimation rates.

As mentioned earlier, these tests were conducted without applying resource cap to ensure that we capture the right results for the prediction models. Again, assuming an error window of 10% (i.e., if $|e| < 10\%$ of actual resource pressure, consider prediction as accurate), we calculated the percentage accuracy for each algorithm and plot the same in Figure 10. As we can see, CloudScale has the highest accuracy. Although it is just above 50%, we suspect it is due to the use of a small data set of 2000 values to train the prediction model initially. Another interesting observation is that histogram approach has a high accuracy rate as compared to other algorithms except CloudScale.

To evaluate our fast DTW implementation we ran a simple workload with clearly visible pattern so that fft would detect one (Figure 11). We test the fft predicted value without using DTW and while using DTW. While not using DTW, although fft gets the right pattern, it fails to predict the right value as the current resource demand (or pattern) is at some offset from the predicted pattern. Using the our fast DTW implementation as well as the python fast DTW library gives

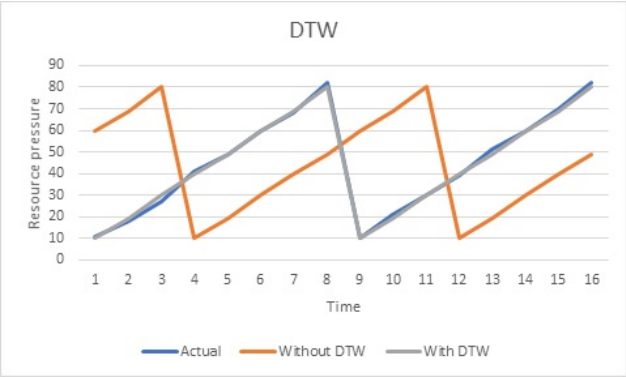

Figure 11: Real resource usage vs fft prediction with DTW and fft prediction without DTW

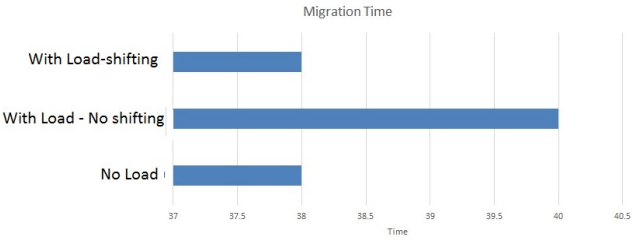

Figure 12: Migration time with and without load balancing

the correct offset and fft is able to predict with much accuracy.

Results related to VM Migration We have implemented a new algorithm to have lower SLO violation. When an Application VM is being migrated to a new host, there will be incoming requests to that VM from the clients. During the migration time, these requests will be dropped by the Application servers since the VM will be in the process of migration. This may cause SLO violation. To avoid such a scenario, we are balancing the requests from the clients to a different hosts such that the requests will be handled by a different Application server during the migration. After the VM migration is successful, the requests are forwarded to the actual Application server. To evaluate the load shifting based migration, two factors were considered:

- Migration time
- SLO Violation

To evaluate the migration time, the migration was performed

- Under no load
- With cumulative CPU load of 60% from the two application servers, and without load shifting
- With cumulative CPU load of 60% from the two application servers, and with load shifting

The VM being migrated is allocated a memory of 512 MB and a disk size of 2 GB. However, the VM uses only 6.5% of

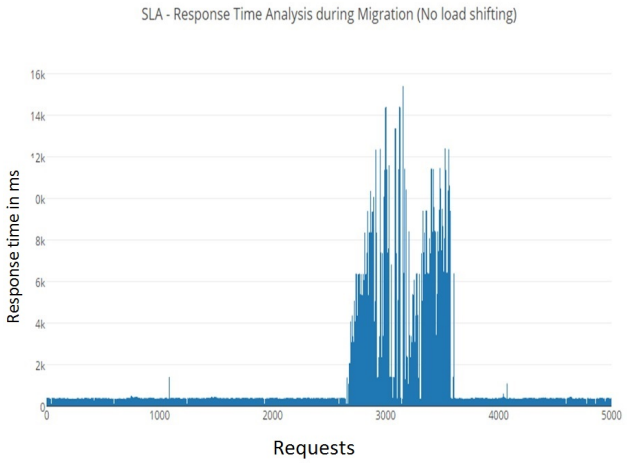

Figure 13: SLO violation without using load balancing

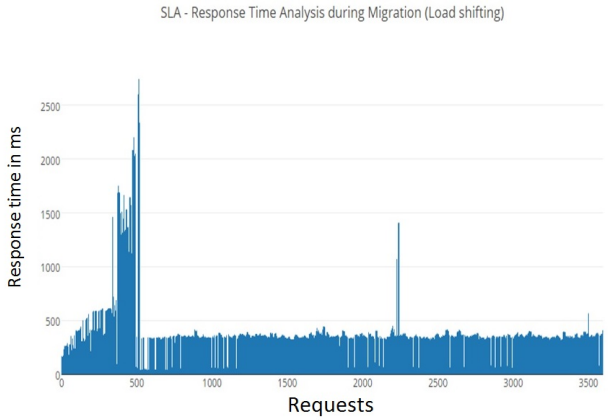

Figure 14: SLO violation using load balancing

the 2 GB total memory on the system approximately. This amounts to 130 MB of memory.

The Figure 12 shows the mean results of the time taken for migration under different scenarios. From the table, it can be seen that the migration time taken for "No Load", and "With load Load Shifting" are comparable. The migration time needed for the scenario "With load No load shifting", is slightly more. This is because of the dirty pages which need to be copied in this case. More dirty pages, more time for the migration. In our case, the VM takes 2s more to migrate the VM consuming 130 MB of memory under the With load No load shifting condition. 2s seems small but if the CPU and Memory load increases, then this value can shoot up.

A more critical aspect to consider during the migration is the amount of SLO violation. Figure 13 and 14 show the SLO Response Times under the two scenarios "With load Load Shifting and "With load No load shifting. From the

graph, it is clearly visible that the SLO Response Times is more affected in the case of "With load No load shifting with some response times touching 16000 ms where as in "With load Load shifting case, the maximum is around 2700 ms. The number of requests getting affected is also more in the case of With load No load shifting. This is because under "With load No load shifting, the VM is handling the requests while it is being migrated while in "With load Load shifting scenario, the requests are routed to another server when the VM is being migrated. The small violations seen in the "With load Load shifting, are during times when the load is being transferred, which is not much comparatively.

4 RELATED WORK

Several works have been done in the past related to a prediction driven resource scaling technique. PRESS[16] is one such system which uses a state driven prediction algorithm using Markov chain to predict the resource demand. We will be implementing a similar prediction algorithm in our system.

Our implementation is based on CloudScale[22] which uses fast four transform based signature matching combined with Markov chain predictor to analyze both recurring and non-recurring patterns in the resource demand. A predictive migration technique is used to migrate the VMs during resource scale conflict. A predetermined monitoring time is defined to check for scaling conflict to avoid erroneous migration. In our implementation, we plan on using monitoring time as a function of the predicted conflict time and the time required to complete the migration.

AGILE[19] is a system which uses a wavelet-based signature for resource demand prediction without using any advance application profiling. A proactive cloning method is employed to clone the VMs during overload to reduce the application startup time.

Some previous work is based on reinforcement learning [20] to adjust resource capping using SLO violation feedback. Such solutions require offline tuning and need more time to take more optimal decisions. In contrast, our solution needs no external parameters to be set up and prediction is done on historic usage. Unlike work in [26] ,[27], [29], [24], our prediction algorithm does not require offline profiling to be done. Offline profiling requires more time to start predicting and consumes more resource usage.

Solution presented in [15] handles cyclic workload patterns using Fourier transform. Our proposed solution handles both cyclic and non-repeating workloads. Some papers use auto regression [10] or a combination of histogram and auto regression [9] to predict for long term and short term workload. Our experiments show that CloudScale prediction algorithm is much more accurate for both long term and short term prediction.

Virtual machine migration [11] has been often used for dynamic resource provisioning. Sandpiper [28] uses a black-box approach to detect hotspots and determine resource provisioning. It automates the task of hotspot monitoring, mapping of physical to virtual resources, and initiating necessary migrations. Entropy [17] performs dynamic consolidation based on constraint programming and takes migration overhead into account. [21] leverages common data to improve live virtual cluster migration between datacenters. [18] works on improving the performance of the pre-copy live migration by optimizing the process. [22] tries to leverage long-term prediction so that migration can be triggered before conflict occurs. Our scheme is similar to CloudScale except that we use a load balanced approach to reduce the migration time and the SLO impact during migration.

Previous work done in [23] [25] [14] [13] use statistical learning methods to be able to handle and predict different resource allocations. Such models need to have prior information about the application (such as data size, processor speed etc) and involve significant overhead. Such properties make it less suitable for cloud based systems. On the other hand, our solution is application agnostic and need not be calibrated separately for each type of application

5 FUTURE WORK

Currently, the resource utilization is considered only for CPU and memory. Network related resource utilization prediction and scaling has not been analyzed. The same resource gathering, prediction and capping framework can be extended towards network usage. Apart from this, any such resource can be scaled using the same framework. Our current model manages each resource pressure separately. But, most of these resources are dependent on one another and better scaling can be achieved if we calculate the cap for each resource collectively rather than individually. Having said that, this is a difficult task to undertake as the relation among these resources is non-trivial.

In our current model, Markov Chain and FFT based prediction is performed for a relatively shorter duration of time in the future, keeping accuracy in mind. Other prediction algorithms with longer prediction step and better accuracy can be looked into [19]. Furthermore, we found that Markov Model is slow in adjusting to sudden changes in resource usage. Some more time can be invested in finding prediction methods which are more faster in adapting to changing load. As mentioned earlier, markov chain itself is used to perform long term prediction. This may not be a good idea as long term prediction using markov model is not accurate. Other long term prediction algorithms can be evaluated to replace markov model.

Currently load shifting migration approach has been performed using only the pre-copy live migration method provided by Xen. Other migration approaches such as live cloning can be evaluated. As we are already collecting resources used by each VM, we can use the same to place each VM in appropriate host (consolidate VMs into best fit hosts).

6 CONCLUSION

In this paper, an elastic resource scaling system on the lines of CloudScale[22] has been presented. An online distributed

predictive scaling system was developed. Challenges and overhead with distributed CloudScale[ref] architecture have been discussed and analyzed. A good short term prediction model and a decent long term prediction has been implemented using FFT and Markov chain algorithms. Adaptive resource capping as per CPU threshold has been incorporated in the model. Our proposed solution enhances the migration based conflict handling solution by shifting load between application servers. We show that this achieves better migration time and has a significant decrease in the response time and SLO violations. CloudScale[22] was further verified for a different workload using standard RuBIS config. CloudScale was verified for a different hardware configuration and for the more widely used Linux based Ubuntu-14.4 operating system. On the whole we see that our proposed solution has better results as compared to its other counterparts and comparable results to the original CloudScale paper[22].

REFERENCES

[1] Python FFT. https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fft.html.
[2] RUBiS Online Auction System. http://rubis.ow2.org/.
[3] Virtual Computing Lab. http://vcl.ncsu.edu/.
[4] Xen Credit Scheduler. https://wiki.xen.org/wiki/Credit_Scheduler.
[5] Xen Project. https://www.xenproject.org/.
[6] Xen Tools. https://xen-tools.org/software/xen-tools/.
[7] E. S. Buneci and D. Reed. 2008. Analysis of application heartbeats: Learning structural and temporal features in time series data for identification of performance problems. In *Supercomputing, 2008.*
[8] M. Cardosa and A. Chandra. 2008. Resource bundles: Using aggregation for statistical wide-area resource discovery and allocation. In *ICDCS, 2008.*
[9] A. Chandra, W. Gong, and P. Shenoy. 2004. Dynamic resource allocation for shared data centers using online measurements. In *IWQoS,.*
[10] G. Chen. 2008. Energy-aware server provisioning and load dispatchingfor connection-intensive internet services. In *NSDI,,.*
[11] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. 2004. Live migration of virtual machines.. In *NSDI,2004.*
[12] R. Doyle, J. Chase, O. Asad, W. Jin, and A. Vahdat. 2003. Model-based resource provisioning in a web service utility. In *USITS, 2003.*
[13] R. Doyle, J. Chase, O. Asad, W. Jin, and A. Vahdat. 2003. Model-based resource provisioning in a web service utility. In *USITS, 2003.*
[14] A. Ganapathi and H. Kuno. 2009. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *ICDE, 2009.*
[15] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. 2007. Capacity management and demand prediction for next generation data centers. In *ICWS,.*
[16] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. 2010. PRESS: PRedictive Elastic ReSource Scaling for Cloud Systems. In *IEEE International Conference on Network and Services Management (CNSM), Niagara Falls, Canada.*
[17] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall. 2007. Entropy: a consolidation manager for clusters. In *VEE, 2009.*
[18] Khaled Z. Ibrahim, Steven Hofmeyr, Costin Iancu, and Eric Roman. 2011. Optimized pre-copy live migration for memory intensive applications. In *International Conference for High Performance Computing, Networking, Storage and Analysis, 2011.*
[19] Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Sethuraman Subbiah, and John Wilkes. 2013. AGILE: elastic distributed resource scaling for Infrastructure-as-a-Service. In *Proc. of USENIX International Conference on Autonomic Computing (ICAC), San Jose, CA.*
[20] J. Rao, X. Bu, C.-Z. Xu, L. Wang, and G. Yin. 2009. VCONF: A reinforcement learning approach to virtual machines auto-configuration. In *ICAC, 2009.*
[21] Pierre Riteau, Chritine Morin, and Thierry Priol. 2010. Shrinker: efficient live migration of virtual clusters over wide area networks. In *Combined Special Issues on eScience 2010.*
[22] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. 2011. CloudScale: Elastic Resource Scaling for Multi-Tenant Cloud Systems. In *Proc. of ACM Symposium on Cloud Computing (SOCC) in conjunction with SOSP, Cascais, Portugal.*
[23] P. Shivam, S. Babu, and J. Chase. 2003. Learning application models for utility resource planning. In *USITS, 2003.*
[24] S. Govindan stat J. Choi. 2009. Statistical profiling-based techniques for effective power provisioning in data centers. In *Eurosys,.*
[25] C. Stewart, T. Kelly, A. Zhang, and K. Shen. 2008. A dollar from 15 cents: cross-platform management for internet services. In *USENIX Annual Technical Conference, 2008.*
[26] B. Urgaonkar and P. Shenoy. 2002. Resource overbooking and application profiling in shared hosting platforms. In *OSDI, 2002.*
[27] T. Wood and L. Cherkasova. 2009. Profiling and modeling resource usage of virtualized applications,. In *ICAC, 2009.*
[28] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. 2007. Black-box and gray-box strategies for virtual machine migration. In *NSDI,2007.*
[29] W. Zheng and R. Bianchini. 2009. JustRunIt: Experiment-based management of virtualized data centers,. In *USENIX Annual Technical Conference, 2009.*