Spring 5-16-2019

Predictive Analysis for Cloud Infrastructure Metrics

Paridhi Agrawal
San Jose State University

Predictive Analysis for Cloud Infrastructure Metrics

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Paridhi Agrawal

May 2019

The Designated Project Committee Approves the Project Titled

Predictive Analysis for Cloud Infrastructure Metrics

by

Paridhi Agrawal

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2019

Dr. Thomas Austin	Department of Computer Science
Dr. Katrina Potika	Department of Computer Science
Dr. Benjamin Reed	Department of Computer Science

ABSTRACT

Predictive Analysis for Cloud Infrastructure Metrics

by Paridhi Agrawal

In a cloud computing environment, enterprises have the flexibility to request resources according to their application demands. This elastic feature of cloud computing makes it an attractive option for enterprises to host their applications on the cloud. Cloud providers usually exploit this elasticity by auto-scaling the application resources for quality assurance. However, there is a setup-time delay that may take minutes between the demand for a new resource and it being prepared for utilization. This causes the static resource provisioning techniques, which request allocation of a new resource only when the application breaches a specific threshold, to be slow and inefficient for the resource allocation task. To overcome this limitation, it is important to foresee the upcoming resource demand for an application before it becomes overloaded and trigger resource allocation in advance to allow setup time for the newly allocated resource. Machine learning techniques like time-series forecasting can be leveraged to provide promising results for dynamic resource allocation.

In this research project, I developed a predictive analysis model for dynamic resource provisioning for cloud infrastructure. The researched solution demonstrates that it can predict the upcoming workload for various cloud infrastructure metrics upto 4 hours in future to allow allocation of virtual machines in advance.

Keyword - Cloud computing, time-series analytics, resource allocation.

ACKNOWLEDGMENTS

I would like to sincerely offer my gratitude to my advisor Prof. Thomas Austin for his immeasurable support and guidance throughout the course of this project. His valuable advice allowed me to focus in the right direction and constantly improve my experiments. I am also thankful to my committee members, Prof. Benjamin Reed and Prof. Katerina Potika for reviewing my work and providing their constructive feedback. I would also like to thank my team and co-workers at Autodesk Inc., for their encouragement and for granting me permission to use their dataset. Finally, I would like to appreciate my partner for being tolerant and patient, and for continuously motivating me to keep working hard.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

Legacy applications are increasingly migrating to the cloud for higher reachability and availability to their clients. Cloud computing is a form of virtualization that allows provisioning of on-demand resources to the client by allocating a network of remote servers hosted on the internet. In order to host an application on the cloud, services like elastic load balancers, storage, and servers are made available on a pay-as-you-go manner by cloud-providers such as Amazon through Amazon Web Services (AWS) [1].

As these applications are usually tied with a service level agreement (SLA) between the enterprise user and cloud provider, in order to assure the quality of service (QoS), the cloud provider has to perform resource provisioning, which is the process of allocating resources on the internet to the cloud hosted application [2] [3] . Resources can be automatically allocated during periods of high demand, and de-allocated during periods of low demand. Resource provisioning based only on SLA defined thresholds tends to be slow and inefficient. Hence, a need for pro-active/dynamic resource provisioning has arisen in the cloud industry.

Over the years, the resource provisioning systems have been developed in a reactive manner, i.e., they fulfill the threshold as per the SLA(s) defined between the enterprise and the cloud-provider. Such an approach is reactive, i.e., the resource allocation is triggered once the defined threshold is breached and often the time to react is insufficient. The setup time caused by the cloud-provider can be hazardous for the enterprise's business. These drawbacks motivate to research in alternative methodologies like dynamic/pro-active resource provisioning which can take advantage of time-series forecasting machine learning techniques to predict resource allocation

well in advance and avoid shortages.

Pro-active resource allocation can help prevent SLA breaches and enable cloud service providers to forecast their customer needs in advance. This technique enables them to trigger resource allocation ahead of time. It can also ensure higher uptime for the service.

The project aims at experimenting with time-series forecasting techniques to develop a machine learning model to make prediction about when to up-scale/down-scale the resources; in turn, these improved predictions enable the cloud provider to comply with the defined SLAs and avoid QoS breaches by triggering resource provisioning in a predictive manner. This project also aims to compare studies on resource provisioning methods implemented in the cloud industry. This project tries to answer the following questions:

- *Can machine learning technique be used to predict future resource needs?*
- *What machine learning technique* can best solve the problem of *when to provision?*
- *What are the key features of the cloud metrics that can contribute to a predictive model?*

This paper is organized as follows: Chapter 2 provides background about resource provisioning in the cloud, related challenges, and the motivation for this project. It also highlights attempts that have been made to enable resource provisioning. Chapter 3 discusses some of the machine learning techniques used to develop predictive analytic models for dynamic resource provisioning. Chapter 4 explains the implementation overview for this research. Chapter 5 covers experimental results on various cloud metrics and multiple time series. Chapter 6 contains the conclusions and future scope of the project. The articles selected for this project include thesis projects, published papers, and articles in the field of cloud resource provisioning.

CHAPTER 2

Background and Related Work

2.1 Resource provisioning in cloud computing

Infrastructure as a Service (IaaS) is a trending field in the cloud computing industry. One of the biggest challenges that IaaS has to overcome is resource management. A survey by Guruprasad et al. [4] describes resource provisioning in cloud computing as a process of selection, deployment and run-time administration of software(e.g. databases, loadbalancers and so forth) and hardware resources(e.g. CPU, storage, network, and servers) for guaranteeing ensured performance of the application. Inefficient resource provisioning can starve the application of resources and lead to degraded QoS and violation of the defined SLAs [3].

2.2 Challenges in resource provisioning

In a cloud environment, various applications run with varying workload patterns. Many challenges can be faced when dealing with an application that experiences fluctuating resource demands. A few of these challenges are highlighted below:

1. **Setup-time for resource allocation**: Cloud environment scaling leads to acquiring or releasing resources as the workload for the application changes. If resource provisioning is done every time the system load changes, it will cause an overhead to boot or turn-off a acquired resources with every workload change. This attaches a setup-time for resource allocation, i.e, the time taken by a resource to be ready. Latency can also be added when the resources are shut-off. On an average, scaling latency time is between 5 and 10 minutes. Due to this wasteful latency time, it is necessary that we predict the future workload and effective provisioning decisions are made as fast as possible to allocate the resources *a priori* [5] [6].

2. **Cost optimization**: It is difficult for cloud providers to fulfill SLAs while minimizing costs. This is mainly due to the fluctuating resource demands. Cloud providers incur less resource cost when resources are under provisioned as less resources are allocated. But, this may lead to SLA violations due to poor performance. On the other hand, if resources are over-provisioned, the cloud providers and the enterprise users both experience high resource costs. Thus optimizing cost is challenging in cloud environment. [5]

3. **Resource optimization**: A cloud application can be used by many users and for a varying amount of time. The workload on an applications depends on the number of users requesting the application and the types of calls they make to the application. Thus, it is essential to understand the incoming load in order to make effective decisions about the type of resources that can best fulfill the incoming requests. This is challenging as it requires comprehending the nature of customer requests.

2.3 Resource provisioning techniques

Resource provisioning can be classified into two categories:

- Reactive resource provisioning
- Pro-active resource provisioning

Details on resource provisioning by the above methods are expanded in subsection 2.3.1 and 2.3.2.

2.3.1 Reactive resource provisioning

Currently, most of the cloud providers use reactive resource provisioning mechanisms for resource allocation to their users [7]. However, such an approach is useful only for applications that have unchanging or predictable workload/demands. In this environment, the cloud user decides the required resource demands/threshold as

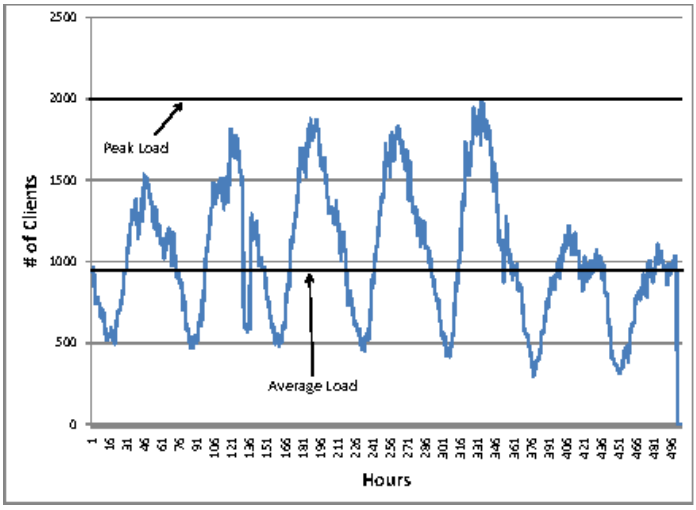


Figure 1: Customer load on FIFA website during world cup 1998.

per the SLA requirements [8]. For instance, Figure 1 depicts the workload on FIFA website during the soccer world cup of 1998 [9]. It can be observed that the workload is fluctuating depending on the number of users requesting the website. This trend is observed in various commercial websites and is typical in a cloud environment. If thresholds are selected based on average load as seen in Figure 1, the cost incurred for the acquired resources would be less(under-provision). But as the load increases, this will result in bad performance of the application. This can cause degraded QoS and lead to SLA violations. On the other hand, if thresholds are selected on the basis of peak load, then the performance is not affected during peak loads but most of the resources would sit idle and will be underused for few time intervals. This makes it ineffective, as it leads to excess resource allocation during low demand periods as well. Both the cloud provider and the enterprise user bear the cost for the excess resources provisioned. [10] [5]. The cloud providers try to provide the maximum/best-effort resources, thereby preventing SLA violations.

Several researchers have proposed various methods to achieve reactive resource provisioning.

Vecchiola et al. [11] proposed a deadline-driven provisioning system for scientific applications with large computational requirements. This method was designed to reduce the application execution time as the static variable and efficiently solved the resource allocation problem. On the contrary, the scope of this approach did not scale well and it was not suitable for data-intensive applications [4].

Enhancing the above method, a paper by Li et al. [12] proposed an optimal solution to resource allocation. This paper introduced a third-party Software as a Service (SaaS) provider that interacted with SaaS user and cloud provider. Their solution was targeted to accomplish two goals:

1. Maximize the throughput for the user with a limited budget and strict deadline and ensure QoS.

2. Maximize the profit for the cloud provider without exceeding the higher power consumption bounds for provisioning virtual machines (VM).

Both of the above approaches are reactive and did not scrutinize the setup-time for the newly allocated resources by the cloud provider to integrate with the existing infrastructure. Thus, the reactive resource provisioning mechanism could not complement the speed of workload changes in a cloud infrastructure environment.

2.3.2 Pro-active resource provisioning

Due to the dynamic nature of cloud applications, the resources must be dynamically allocated in correspondence to the rapid change in the workload. The pro-active resource provisioning method is useful where the workload of the application is less predictable. This method overcomes the limitations of reactive resource provisioning by allocating resources when needed and de-allocating them when they are not needed. According to the prediction results of this method, cloud providers can allocate suffi-

cient resources to the application at the appropriate time with guaranteed QoS and no SLA violations.

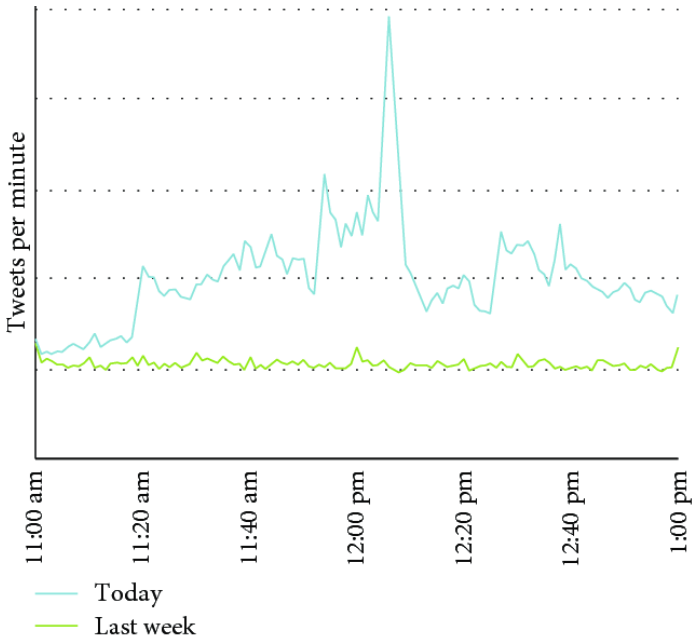


Figure 2: Twitter workload on Obama's inaugural day.

This approach is more challenging as there is a need to determine the future demand/load forecasting for an application. For instance, Figure 2 shows workload on Twitter's web-servers on Obama's Inaugural Day [13]. The dramatic variations in the workloads leads to complicated resource usage patterns that are harder to predict. In the cloud, applications can also be co-hosted. In this situation, they will complete for shared resources and are likely to encounter load fluctuation. Also, sometimes due to the security benchmarks set by the enterprise customer, cloud providers cannot leverage internal characteristics of applications like recurring resource usage to be prepared for periodic load demands [14]. It is essential that resource utilization predictions are accurate.

Balaji et al. [15] performed a comparative study on predictive models for cloud infrastructure management. In their comparison, various dynamic resource provision-

ing methods were compared and some of the limitations were exposed. Most of the recent research in dynamic resource provisioning is done with load balancing and scheduling mechanism. Hu et al. [7] found that these solutions are not very adaptive and heterogeneous towards the workload.

2.3.2.1 Dynamic provisioning using load balancing & scheduling techniques

Fernandez et al. [2] proposed a system that consists of a scheduler, profiler, and predictor module. The profiler acts like an analyzer of the application workload and the scheduler helps in allocating resources to high priority tasks. The scope for the predictor was limited in this paper. This approach proved to efficiently answers the question ‘‘when to provision?’’ but assumed similar behavior for all resources in the cloud infrastructure.

In contrast to the above approach, Bunch et al. [16] proposed a system that allocates VMs to the user based on the characteristics of the job to be executed. This approach helped to balance the load on the resources and improve system performance. The decision to allocate a VM is made on the fly according to the incoming job and the requires QoS. This approach is not practical for medium to large applications.

On a broader scope, these methods outperform reactive methods but are limited by the type of resource that is being allocated. These approaches fail to take advantage of the information that can be attained by taking a closer look at the historical behavior of the application. Thus, investigating machine learning techniques for pro-active provisioning can lead to a more adaptive and generic solution to resource provisioning.

2.3.2.2 Dynamic resource provisioning with machine learning techniques

The previous research done in this area is limited and has not been explored completely. Early contributions to the research support the key goal of developing

an adaptive resource provisioning system that regards the diversity of the cloud application. In order to form accurate predictions about the application workload, it is useful to consider machine learning techniques as they can help in discovering the hidden pattern in the resource demands for a given application [17].

Keung et al. [10] apply machine learning techniques like artificial neural network (ANN) and linear regression for prediction of required resources by the applications. Their proposed technique was able to react to future demand variations prior to their occurrence. The combination of the machine learning techniques was focused on identifying the connection between application QoS target and current cloud resources. It was able to adapt to the changes in workload pattern to alter resource provision dynamically.

CHAPTER 3

Technical Approach

For an AWS cloud hosted application to run reliably, it needs to be monitored constantly to check various metrics like CPU utilization, memory utilization, traffic and other key application program interface (API) metrics. CloudWatch is an AWS supported tool that monitors such metrics for an AWS hosted service on an hourly basis and retains these logs up-to 15 days . With the help of such metrics, a pro-active resource provisioning system can be developed to predict the upcoming workload on the application or to predict the upcoming resource demands.

In this project, this prediction task is considered as a time-series prediction problem and machine learning techniques like Moving Average (MA), Auto-Regression (AR), and Auto-Regressive Integrated Moving Average (ARIMA) [18], which are forecasting techniques to predict future values in a time series susceptible to seasonality, are applied on application metrics to make resource workload predictions. This information can be used to make provisioning decisions.

3.1 Dataset

In order to develop the predictive model, real-time workload data is required. The dataset used in this literature was developed by collecting monitored metrics for Autodesk Inc., which is a software corporation that develops software used in construction, media, education and entertainment industries, cloud hosted applications [19].

The following two applications were part of the dataset for this project:

- **AutoCAD mechanical (ACM)**: ACM toolset is an add on for AutoCAD software, which is mechanical engineering design software. It is used to speedup the mechanical computer aided design (CAD) process within AutoCAD [20]. For this application, CPU utilization metrics was monitored daily and collected

every hour over a period of 60 days. This dataset is the collected for the cluster and is the average % CPU utilization over 1 hour for the cluster. The type of requests to the cluster are same over 60 days. The peaks observed in the dataset is across all the nodes in the cluster and hence was probably caused by a background dependency for the cluster.

- **Object Storage Service (OSS)**: OSS is cloud storage service useful in storing, retrieving application data for a construction app. It allows various file formats like AutoCAD file format, images and media, etc [21]. This application has three AWS accounts, namely, OSS-web, OSS-upload, and OSS-download. To develop the dataset, all three accounts were monitored for average % cpu-utilization, network-in, network-out for the cluster. The latency in milliseconds(sum over 1 hour) and traffic (request per minute, sum value every hour) metrics for the instance were collected daily for 60 days.

These collected metrics are essentially time-series data. The next section discusses the machine learning techniques applied on this dataset to find pattern and anomalies within the series.

3.2 Machine Learning techniques for workload prediction

In order to predict incoming workload, it is essential to analyze historical data and inspect patterns within. Since the collected metrics is a temporal dataset, time-series analysis and forecasting can be performed on them.

3.2.1 Time series

A simple time series can be represented as a series of observations over uniform time intervals, such as x_{t-2}, x_{t-1}, x_t, where x is the observation at time interval t. The problem statement for this research is to predict x_{t+1} value for the given resource metric. The prediction of x_{t+1} is based on the measured series x_{t-2}, x_{t-1}, x_t up-to time

t. For example, in case of CPU utilization metrics, the aim is to forecast the CPU utilization in the next time interval.

There are three main characteristics of in a time series:

- **Trend**: Trend refers to an upward or downward movement in a series over a period of time. When the time-series analysis 3.2.2 displays an upward movement, it is called an upward trend. Downward trend is the opposite of upward trend. When the is no trend in the series, it is said to have a stationary or horizontal trend. Figure 3 shows that the workload trace on Wikipedia is periodical and follows a stationary trend after every 24 hours. This trace is picked randomly from traces available at WikiBench [22], which is a web hosting benchmark [23].

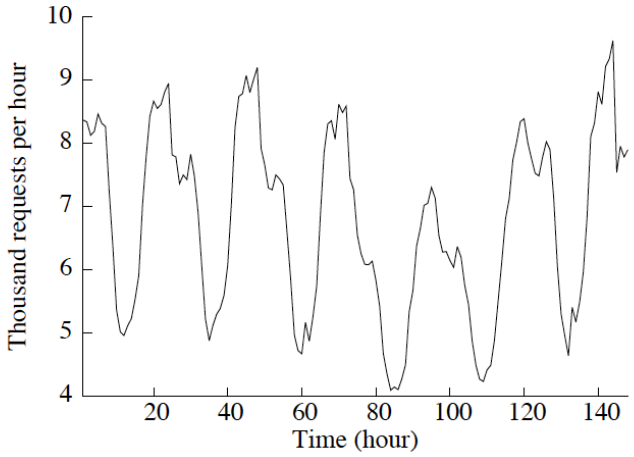


Figure 3: Wikipedia workload during a week

- **Seasonality**: Seasonality is repeating pattern within a fixed time period. Figure 1 depicts the a seasonal trend of users on FIFA website, depending on the time of the day.
- **Noise**: This contributes to irregular patterns observed in the time series that last for a short duration. Figure 2 is an example of irregular time series, where

Twitter web-servers experiences a sudden outburst of requests. Such patterns are hard to predict as they happen erratically.

3.2.2 Time-series analysis

Time-series analysis (TSA) includes strategies for investigating time series data, so as to extricate significant statistics and other features of the data. In time-series forecasting, it is necessary to examine a given data prior to applying any forecasting techniques on it.

The purpose of this analysis is to identify trends and seasonality in the time series data. In this section, a few of these techniques are discussed that are critical to the experiments.

- **Stationarity Test:** For any time series forecasting technique to work well, it needs a time series that is stationary over time [6]. A time series is said to be stationary if its statistical properties like mean and variance remain constant over time [24]. This can be analyzed by using the techniques listed below:

  1. **Rolling statistics plot**- This is visual analysis on the given time-series. We plot the rolling variance and rolling average and verify if they vary over time. A visual sample for this technique is represented in Figure 4. As seen in the figure, the given series has varying mean and variance over time, thus it is a non-stationary series.

  2. **Augmented Dickey-Fuller Test**: This is a statistical analysis test to check for stationarity. A time series can be denoted as

     $$y_t = a * y_{t-1} + \varepsilon_t$$

     where y_t is the observation at time t and ε_t is the error term. Similarly y_{t-1} can be denoted as $y_{t-1} = a * y_{t-2} + \varepsilon_{t-1}$ and so on. If the series has a

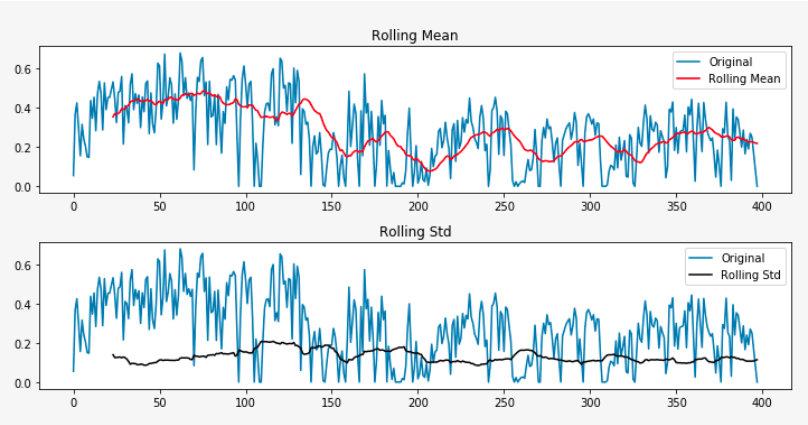


Figure 4: Rolling statistics plot against original time series.

unit root, i.e $a = 1$, then the series is non-stationary. For example, if we assume $a = 1$ and $y_0 = 0$, then

$$y_t = y_0 + \sum_{i=1}^{t} \varepsilon_i$$

y_t is a sum of all the errors up-to t and y_0 and the series becomes a non-stationary, as the variance for y_t is

$$Var(y_t) = \sum_{i=1}^{t} \sigma^2 = t\sigma^2$$

which varies over time t. This is the test for *Null Hypothesis* [25]. In ADF, we say a series is non-stationary if we fail to reject the null hypothesis. In

```
Results of Dickey-Fuller Test:
Test Statistic                  -9.874495e+00
p-value                          3.915122e-17
#Lags Used                       1.000000e+00
Number of Observations Used      9.390000e+02
Critical Value (5%)             -2.864623e+00
Critical Value (1%)             -3.437333e+00
Critical Value (10%)            -2.568412e+00
```

Figure 5: Test sample of ADF test

this research, *statsmodels.tsa.stattools* Python libraries is used to implement

ADF. A sample result is shown in Figure 5. If the 'Test statistic' signed value is lower than signed 'Critical values', then the series is said to be stationary.

3. **KPSS (Kwiatkowski-Phillips-Schmidt-Shin) Test:** KPSS is also a technique to determine stationarity. This test compliments ADF, as presence of null hypothesis is considered a stationary series. KPSS defines the null hypothesis as trend stationary, to an alternate hypothesis of a unit root series [26]. In this research, *statsmodels.tsa.stattools* Python libraries is used to implement KPSS. A sample result is shown in Figure 6. If the 'Test statistic' signed value is lower than signed 'Critical values', then the series is said to be stationary.

```
Test Statistic            0.100788
p-value                   0.100000
Lags Used                22.000000
Critical Value (5%)       0.463000
Critical Value (1%)       0.739000
Critical Value (2.5%)     0.574000
Critical Value (10%)      0.347000
dtype: float64
```

Figure 6: Test sample of KPSS test

4. **Decomposing**: This method is used to decompose a given time series to identify trends and seasonality patterns in the data. There are two decomposing techniques:

   (a) Additive model: This considers a time series y_t as:

$$y_t = S_t + T_t + E_t$$

where S_t is the seasonality component, T_t is the trend component, and E_t is the noise component, in a series at time t

(b) Multiplicative model: This model suggests a non-linear model with increasing or decreasing changes over time. Therefore, y_t series is a multiplication of individual components.

$$y_t = S_t * T_t * E_t$$

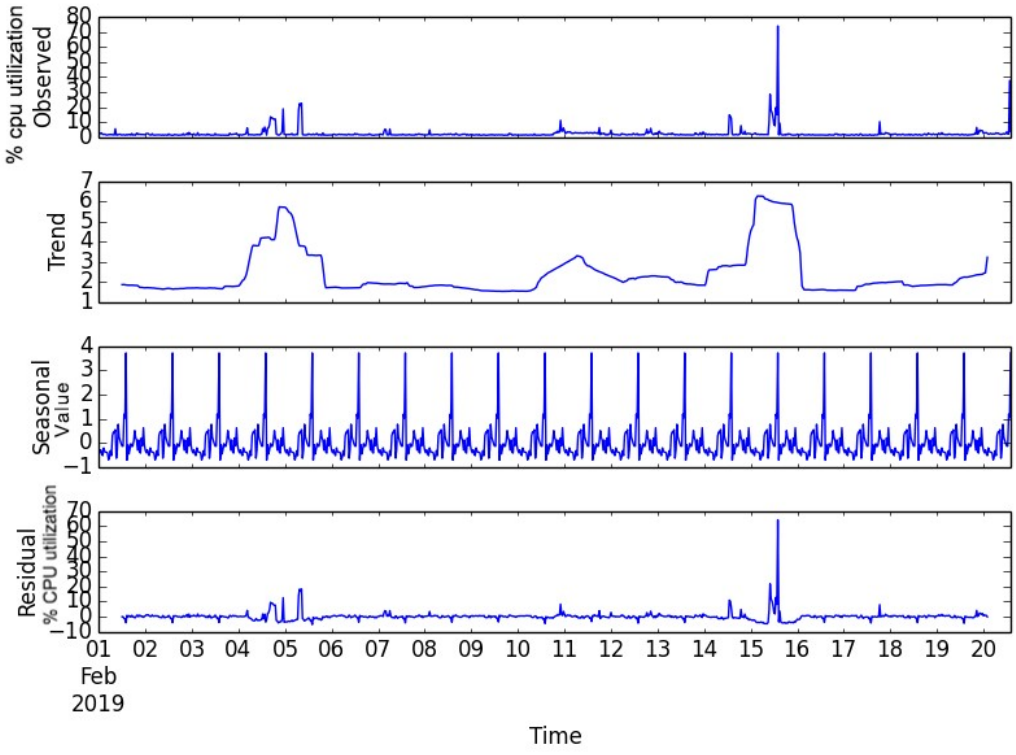


Figure 7: Sample of Time-series decomposition on ACM workload

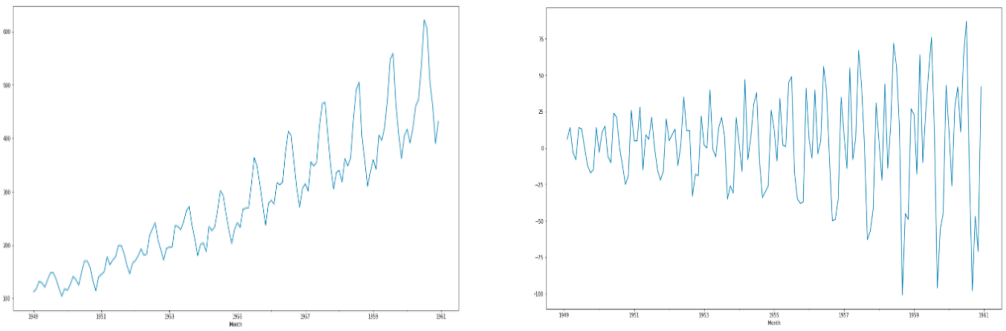
The *statsmodels* library in Python, provides implementation of decomposition. A sample output from this method is shown in Figure 7.

- **Techniques to stationarize time series:** The following techniques cans be used to make a time series stationary:

    1. **Differencing**: Given a non-stationary series, we will ‘‘differentiate’’ the data until the series becomes stationary. Differencing is done by replacing each value in time series y_t as a difference of it's previous term, i.e,

    $$y_{d_t} = y_t - y_{t-1}$$

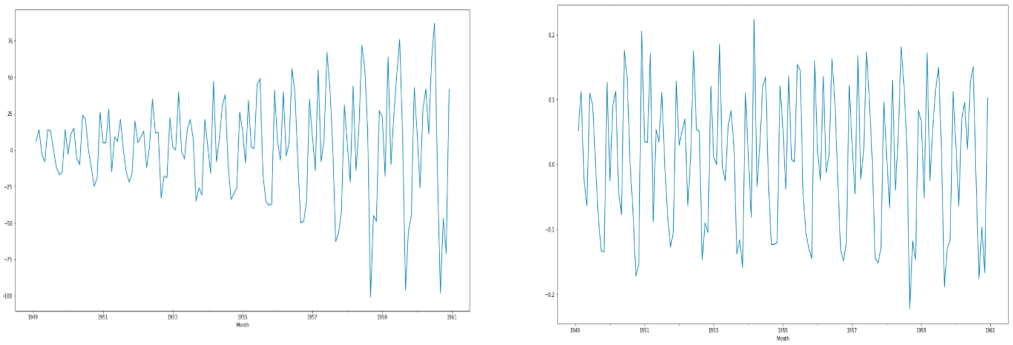
    The number of times differencing is needed is called the ‘‘order of differencing, d’’. This technique helps in removing both trend and seasonality patterns in the series. A sample of differencing effect can be seen in Figure 8.



(a) Non stationary time series (b) 1-order Differenced time series

Figure 8: Example of Differencing.

    2. **Transform**: The simplest way to overcome trend patterns in a series is to apply transformation techniques like taking log, cube root, square root, etc [24]. Other transformation techniques involve smoothing by taking rolling averages in the series. The affect of log transform can be seen in Figure 9. The log transform is applied on a differenced time-series seen in Figure 8(b).



(a) Differenced time series (b) Log transformed time series

Figure 9: Example of log transformation.

3. **Decomposition**: As discussed above, we can decompose a given time-series into trend and seasonality components. The residual time series as shown in Figure 7 is the time series that remains after removing the trend and seasonality component from the observed time series. This can be done for both additive and multiplicative model.

If a series is non-stationary according to ADF test and stationary as per KPSS test, then the series is trend stationary and removing trend component can make it stationary. For the vice-versa condition, series has to be differenced. After applying any of the above techniques, the resulting series must be tested for stationarity again, until a stationary series is obtained.

3.2.3 Time-series forecasting techniques

Time series forecasting implies the use of statistical machine learning models to make predictions about future occurrence based on historic/observed data. In this research following machine learning models were implemented to predict the future workload for a given service:

1. **Auto-Regressive time series model (AR(p)):** In any regression model, the output value is dependant on some combination of the input value. For example,

in a liner regression model, output y can be written as equation (1), where y is the forecasted result, c_0 and c_1 are optimized coefficients and x is the input value.

$$y = c_0 + c_1 * x_1 \tag{1}$$

AR is a regression technique, in which, the predicted output value is based on the input values at the previous time interval [18]. The input variables in AR are called lag variables. The order, p, for AR model is the number of lags used to model.

An AR model can be denoted as:

$$X_t = c + \sum_{i=1}^{p} \varphi_i * X_{t-i} + \varepsilon_t$$

Where $\varphi_1, .., \varphi_p$ are model parameters, ε_t is noise and c is a constant. Partial Auto-Correlation Function (PACF) is a method to infer the lag parameter, p, for AR model.

2. **Moving Average time series model(MA(q)):** The MA method is based on white noise in the series [18]. It uses a weighted average of white noise values over q previous time intervals. A MA model can be denoted as:

$$X_t = \mu + \epsilon_t + \sum_{i=1}^{q} \theta_i * \epsilon_{t-i}$$

where μ is the mean of the time series, $\theta_1, .., \theta_q$ are model parameters, and $\epsilon_t, \epsilon_{t-1}, .., \epsilon_{t-q}$ are noise error terms. Order q denotes how many time intervals are to be included to calculate the weighted average. Auto-Correlation Function (ACF) is a method to infer the model order, q, for MA model.

3. **Auto-Regressive Integrated Moving Average model (ARIMA(p,d,q):** The ARIMA model predicts the next interval value in the series as a linear function of differenced input observations and the noise errors at previous time intervals [18].

   This method is a combination of AR and MA models along with the differencing time series analysis step to stationarize the time series, defined as *Intergration (I)*. Thus, the three parameters of the model can be inferred as

   p : is associated with the order for AR model.

   d : is associated with the order of differencing to be applied to the time series.

   It forms the integrated part of the model.

   q : is associated with the order for MA model.

   ARIMA is considered a generic model as it can be applied on non-stationary series as well. This is due to the I parameter of the model that does the differencing on the series. Both MA and AR model can not be used for non-stationary series.

4. **Hyper-parameter settings for forecasting techniques:** The hyper-parameters p for AR and q for MA model can be inferred by plotting the ACF and PACF plots.

   (a) **Auto-Correlation Function(ACF):** ACF is a correlation function which provides correlations values of observations in a series with its lagged values. It highlights how are the present observation of a series correlated to its past observations. ACF function considers all the components of a series as described in section 3.2.1 while finding correlations. Hence, it is referred as 'complete auto-correlation' function [27].

An ACF plot for a series can be visualized to identify the order for MA model. For an MA model, ACF will have positive ACF values at lags involved in the model. Figure 10 shows that ACF value for MA series is

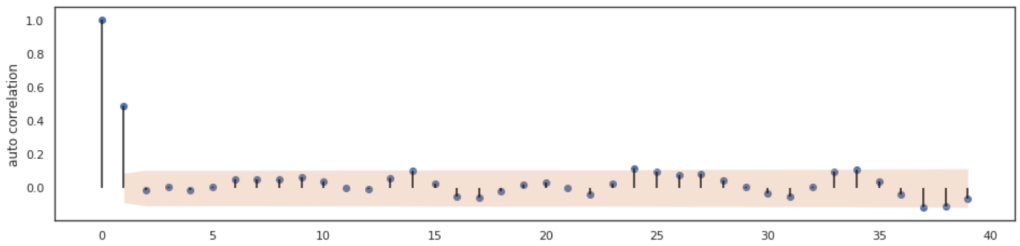


Figure 10: Auto-correlation plot for MA(1) series

non-zero up to lag=1 and shuts off after this. Thus, the order, $q = 1$ and this represents a MA(1) series.

For an AR(p) model, the ACF plot does not shut-off clearly after p lags. Thus, it can't be used to determine the order for the model [28]. A sample of this is represented in Figure 11.

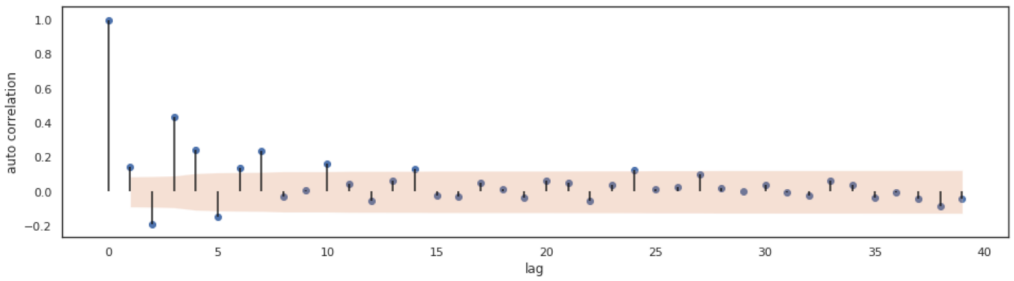


Figure 11: Auto-correlation plot for an AR series

(b) **Partial Auto-Correlation Function (PACF)** PACF functions provides a partial correlation of a time series with its lagged values, after removing the values from the time series that are at all shorter lags. It contrasts with ACF function in finding correlation, as it focuses on finding correlation of the residual values (after removing earlier lag values) with

the next lag value [27]. The PACF plot is useful in identifying order for

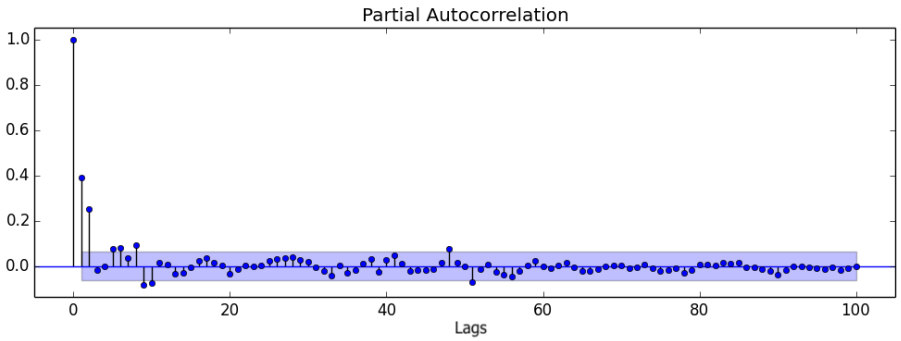


Figure 12: Partial Auto-correlation plot for AR(2) series

AR model. The PACF of an AR(p) model becomes zero at lag p+1 and greater. This means the PACF for AR shuts off after lag p. This can be seen in Figure 12.

The PACF for a MA series will not shut off, rather decays to zero(0) slowly. Therefore, it can not be used to determine the order of MA model. This can be seen in Figure 13.

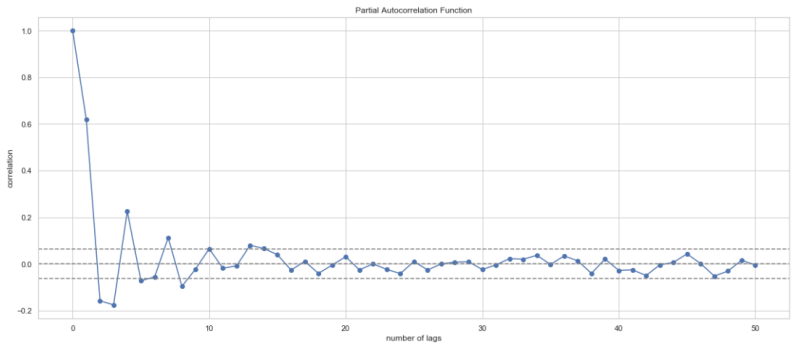


Figure 13: Partial Auto-correlation plot for MA(1) series

3.3 Evaluation metrics:

1. **Mean Absolute Error(MAE)**: MAE measures the average value of errors in the prediction set. The absolute values is considered for the prediction errors. Prediction error is the difference between the observed value and the predicted value [14]. It is evaluated as the equation (2), where n is the number of observations in the prediction set, y_{obs} and y_{pred} are the real and predicted values respectively.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_{pred_j} - y_{obs_j}| \tag{2}$$

2. **Root Mean Squared Error(RMSE):** RMSE is a quadratic metric that measures the average error in prediction results. It is the square root of mean of squared difference between original and predicted value. observation [29]. It is evaluated as the equation (3), where n is the number of observations in the prediction set, y_{obs} and y_{pred} are the real and predicted values respectively.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_{pred_j} - y_{obs_j})^2} \tag{3}$$

   If RMSE is zero, then there are no errors in the predictions.

In this research, Python libraries under *sklearn.metrics* were used to examine error metrics for the experiments.

In the next chapter, details of implementation methodology is discussed with an overview of the workflow diagram.

CHAPTER 4

Overview of the Implementation Design

This chapter provides an overview of the developed experimentation design. The objective of this research is to perform predictive analysis on cloud infrastructure metrics, in order to forecast the upcoming application workload or resource demands. The result of this analysis, can be used by resource provisioning systems to make in-time provisioning decisions. An overall system level design can be seen in Figure 14.

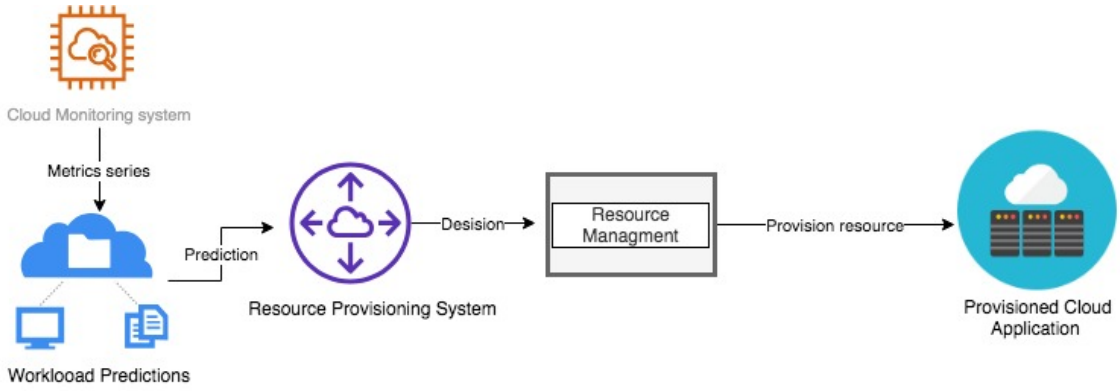


Figure 14: System level design

4.1 Implementation Methodology

In this section, a detailed overview is presented for implementing the prediction model for resource provisioning system.

The general steps to implement a forecasting model are:

1. **Loading the data** : Load the collected dataset into the system. In this research, real-world workload time-series data for Autodesk's ACM and OSS service is used.

2. **Data Pre-processing**: Data cleaning and pre-processing of the time series must be done. Some of the pre-processing to be done are: making a uni-variate

series, getting the timestamps, and converting them to to correct datatype needed by the model, etc.

3. **Time series analysis**: In this step, we analyze the given time series to identify and rectify trends, seasonality, and noise from the series and form a stationary time series.

   (a) Stationarity test: It is easier to making accurate predictions for a stationary time-series. Therefore, it is essential to perform stationarity test on the given time series. If the series fails the stationarity test, then we proceed to stationarize.

   (b) Stationarize the series: In this step we stationarize the series using the techniques discussed in Chapter 3. The added benefit of this step is that we can determine the d value (Intergration order), if differencing is used to stationarize the series.

4. **Generate ACF and PACF plots**: Plotting ACF and PACF graphs can help in determining orders for the forecasting model. This step also provides insights about the time-series.

5. **Identify the values for model order, p and q** : Using the ACF and PACF graph, identify if the series is AR or MA and determine the p and q values for modelling the forecast model.

6. **Build the forecast model**: Build a time-series forecasting model based on the parameters calculated in the previous step. This step also involves splitting the processed data into training and validation set. The model is fitted onto the training set.

7. **Forecast using validation set**: In this step,we test the forecast model on the validation set and predict the future values for the time series.

8. **Evaluate the results**: To verify the model performance, evaluated the prediction results using MAE and RMSE metrics. If the results are bad, then repeat the process from step 4-8 by hyper-parameter tuning.

The discussed methodology is depicted by a work-flow diagram in Figure 15.

In this research, the forecasting model are implemented using Python *statsmodels.tsa* library, which provides a rich set of modules for performing time-series analysis.

Figure 15: Implementation workflow

CHAPTER 5

Experiments and Results

This chapter covers the experimental setup, details of the experiments performed, and their results. The purpose of this research was to perform analysis of cloud infrastructure metrics to understand the following:

- *Can we accurately predict future workload/resource demands for a cloud hosted service?*
- *Can predictive analysis on infrastructure metrics help in preventing SLA violations?*

These questions are answered by the experimental results. The techniques used to perform predictive analysis on the cloud metrics are - ARIMA, MA, and AR.

5.1 Experimental Design

Each cloud infrastructure metric time-series is analyzed and forecasted using the above mentioned techniques. We limit the analysis to two sets of experiments per series and display the results. In the first set of experiments, we analyze the time series for stationarity and forecast the series using forecasting models. In the next set of experiments, we apply stationarity techniques to the given time series and get the forecasting results. In each set of experiments, we use RMSE and MAE metrics to evaluate the prediction results.

Finally, we compare the results of each forecasting technique to understand which time series model best applies to the given time series.

5.2 Experimental requirements

The requirements to implement the dynamic resource provisioning model are as follows:

- An AWS cloud hosted application.

- A virtual machine, deployed using Unix operating system.
- Load balancing test scripts, to run load test on application's cloud infrastructure.
- Cloudwatch to gather metrics data.
- Installed Python version 2.7
- *statsmodels.tsa and sklearn.metrics* libraries to implement time-series forecasting techniques on the collected data.

5.3 Experiments

In this section, we will discuss the predictive analysis performed on the following infrastructure metrics for ACM and OSS service.

5.3.1 Analysis on ACM CPU-utilization metrics:

The ACM CPU utilization dataset is an hourly time series. Figure 16 is a visualization of this time series. The ACM series was tested for stationarity using

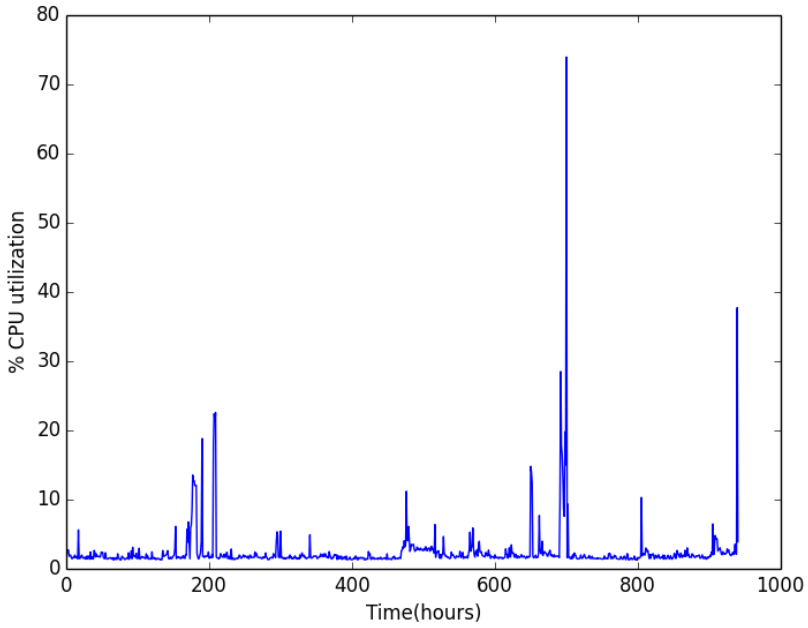


Figure 16: ACM CPU utilization workload

Rolling statistics plot, ADF and, KPSS techniques. Results for these test are given in

Figure 17, Figure 18, and Figure 19.

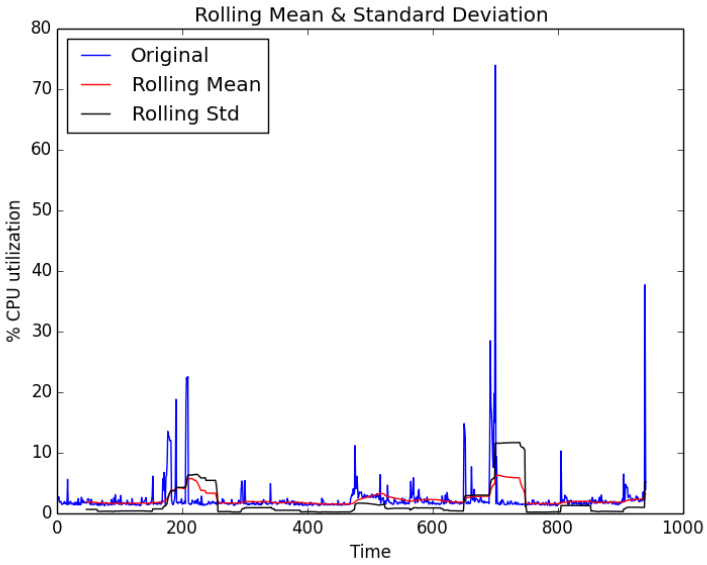


Figure 17: ACM CPU utilization Rolling statistics plot

```
Results of Dickey-Fuller Test:
Test Statistic                -7.504293e+00
p-value                        4.174036e-11
#Lags Used                     9.000000e+00
Number of Observations Used    9.310000e+02
Critical Value (5%)           -2.864649e+00
Critical Value (1%)           -3.437393e+00
Critical Value (10%)          -2.568426e+00
dtype: float64
```

Figure 18: ACM CPU utilization ADF test

Both ADF (Fig. 18) and KPSS (Fig. 19) show that the series is stationary as the test statistic value is smaller than 1 % critical value. Thus, we can say with 99 % confidence that the time series is stationary. As per the Rolling mean test in Figure 17, it can be seen that there is some trend in the series as the variance is changing over time. Following experiments were performed to stationarize the series and forecast the future values.

```
Test Statistic           0.100788
p-value                  0.100000
Lags Used               22.000000
Critical Value (5%)      0.463000
Critical Value (1%)      0.739000
Critical Value (2.5%)    0.574000
Critical Value (10%)     0.347000
dtype: float64
```

Figure 19: ACM CPU utilization KPSS test

Details of the experiments are given below.

1. **Without applying any transformation:** Since the series is stationary, it can be used for forecasting directly. This experiment serves as the baseline for the predictive analysis.

   - **Analyzing ACF and PACF Plots**: The ACF and PACF plot seen in Figure 20 shows that this series is a pure AR series as it has a slow decaying

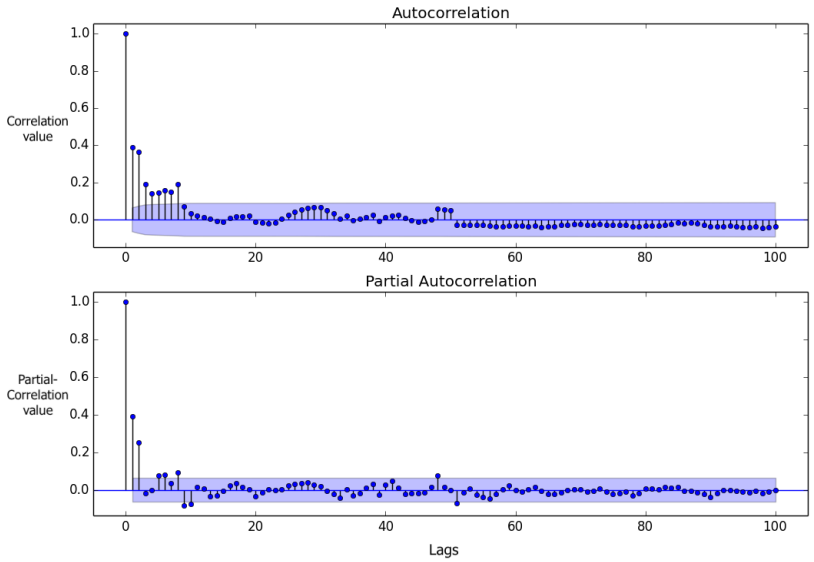


Figure 20: ACF and PACF for ACM CPU utilization

ACF and the PACF shuts-off after lag = 2. The shaded part of the ACF

and PACF denotes the upper and lower bounds of 95% confidence interval. Thus, $p = 2$ and $q = 0$. Using these model parameters, the forecasting models were built. The series is split into training and test sets in a 70:30 ratio. Table 1 shows the training and testing data split.

Table 1: Data Split

Data	Number of intervals used
Training Set	672
Testing Set	269

- **Forecasting results:**

  (a) **AR model:** The AR model was developed using *statsmodels.tsa.armodel* Python 2.7 library. The model fitted on training data is validated by making predictions on test data. Lag window, or p = 2. Figure 21 shows the results from this model. Table 2. highlights the model performance against the evaluation

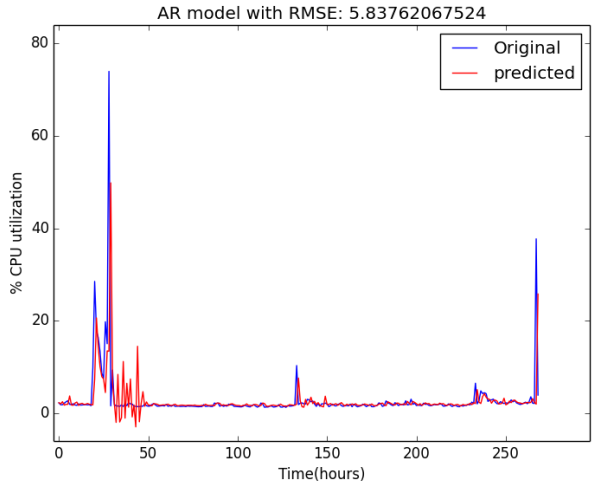


Figure 21: AR model predictions results

metrics.

Table 2: Prediction results for AR

Evaluation Metric	Value
RMSE	5.838
MAE	1.458

(b) **MA model**: The MA model was developed using *statsmodels.tsa* Python 2.7 library. The model fitted on training data is validated by making predictions on test data. Lag window, or q = 3 was chosen for the experiment. Figure 22 shows the results from this model.

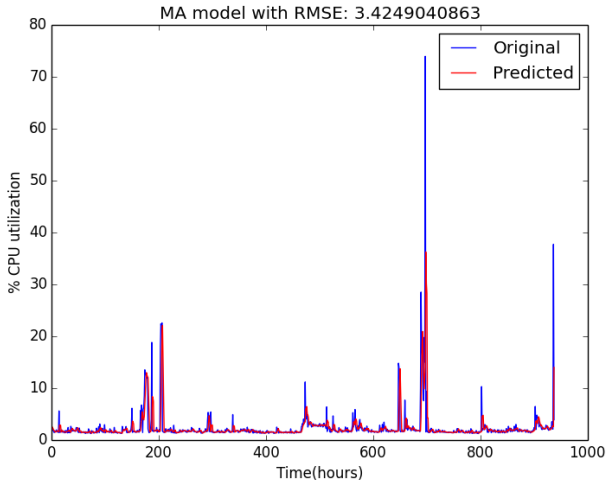


Figure 22: MA model predictions results

Table 3 highlights the model performance against the evaluation metrics.

Table 3: Prediction results for MA

Evaluation Metric	Value
RMSE	3.425
MAE	0.866

(c) **ARIMA model:** The ARIMA model was developed using $statsmodels.tsa.arima - model$ Python 2.7 library. The model fitted on training data is validated by making predictions on test data. Model parameters used were, p = 2, d = 1, q = 0. Figure 23 shows the results from this model. Table 4 highlights the model performance

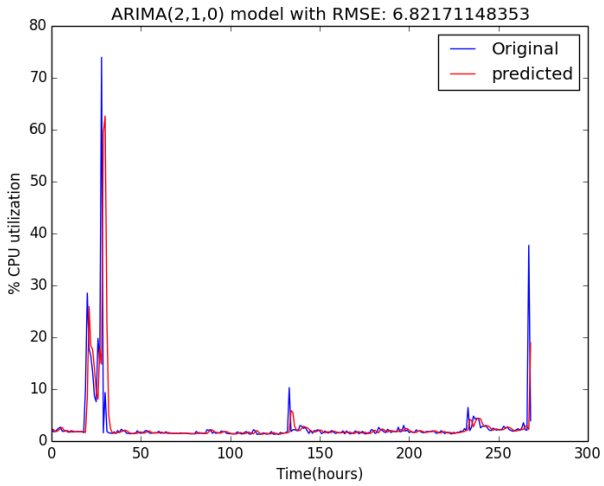


Figure 23: ARIMA model predictions results

against the evaluation metrics.

Table 4: Prediction results for ARIMA

Evaluation Metric	Value
RMSE	6.822
MAE	1.473

All the models were able to forecast the future workload and show satisfactory results as RMSE, MAE values closer to 0 are considered good prediction results. De-trending and removing seasonality from that data will help to improve the results.

2. **Using Log transform**: To reduce the affect of trend and seasonality, log transform is applied to the time series. The startioanrity test for the resulting time series are shown in Figure 24 and Figure 25. The data split remains the same as the previous experiment.

```
Results of Dickey-Fuller Test:
Test Statistic                -9.874495e+00
p-value                        3.915122e-17
#Lags Used                     1.000000e+00
Number of Observations Used    9.390000e+02
Critical Value (5%)           -2.864623e+00
Critical Value (1%)           -3.437333e+00
Critical Value (10%)          -2.568412e+00
```

Figure 24: ADF test on log transformed CPU metrics

```
Test Statistic            0.145853
p-value                   0.100000
Lags Used                22.000000
Critical Value (5%)       0.463000
Critical Value (1%)       0.739000
Critical Value (2.5%)     0.574000
Critical Value (10%)      0.347000
```

Figure 25: KPSS test on log transformed CPU metrics

- **Analyzing ACF and PACF Plots**: The ACF and PACF plot seen in Figure 26. show that this series is a pure AR series as it has a slow decaying ACF and the PACF shuts-off after lag = 2. Thus, $p = 2$ and $q = 0$. Using these model parameters, the forecasting models were built. The series is split into training and test sets in a 70:30 ratio. Table 5 shows the training and testing data split.

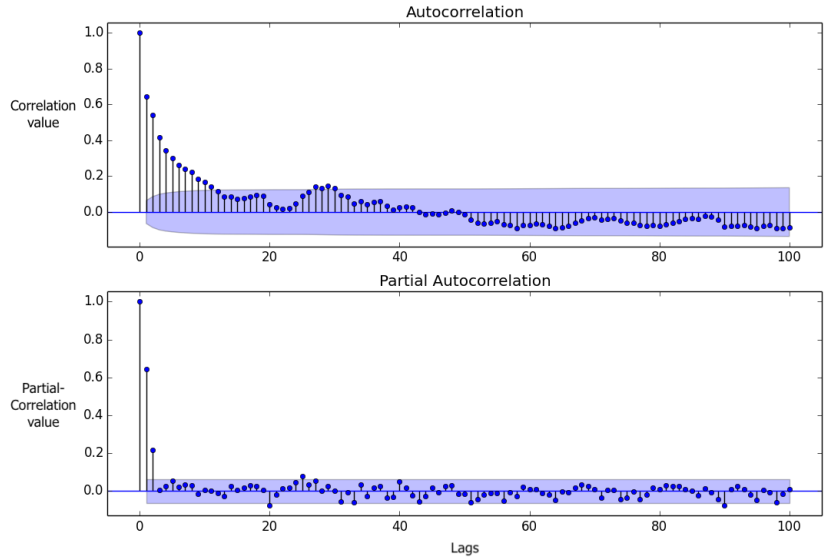


Figure 26: ACF and PACF for log transformed CPU utilization

Table 5: Data Split

Data	Number of intervals used
Training Set	672
Testing Set	269

- **Forecasting results:**

  (a) **AR model:** The AR model was developed using *statsmodels.tsa.armodel* Python 2.7 library. The model fitted on training data is validated by making predictions on test data. Lag window, or p = 2. Figure 27 shows the results from this model. Table 6 highlights the model performance against the evaluation metrics.

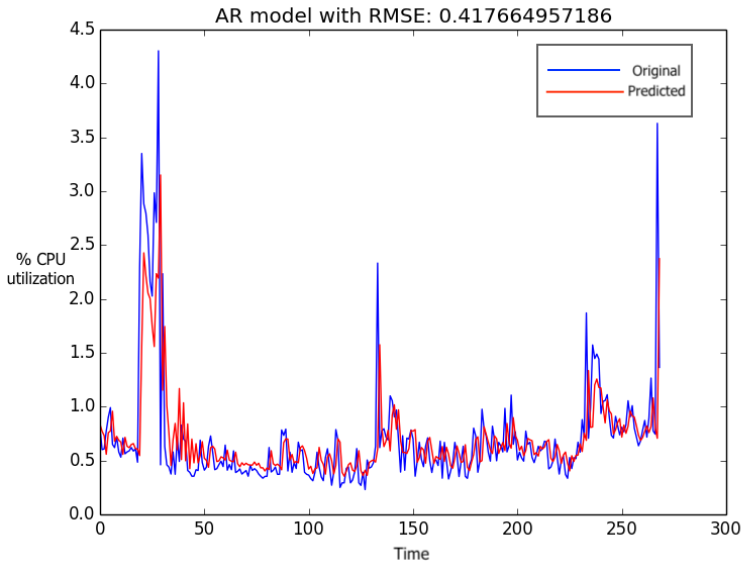


Figure 27: AR model predictions results

Table 6: Prediction results for AR

Evaluation Metric	Value
RMSE	0.418
MAE	0.211

(b) **MA model**: The MA model was developed using *statsmodels.tsa* Python 2.7 library. The model fitted on training data is validated by making predictions on test data. Lag window, or q = 5 was chosen for the experiment. Figure 28 shows the results from this model.

Table 7 highlights the model performance against the evaluation metrics.

Table 7: Prediction results for MA

Evaluation Metric	Value
RMSE	0.425
MAE	0.235

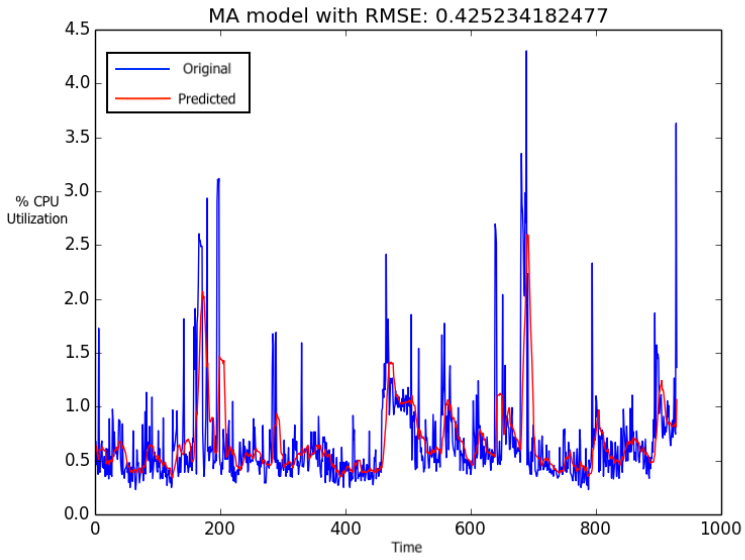


Figure 28: MA model predictions results

(c) **ARIMA model:** The ARIMA model was developed using $statsmodels.tsa.arima - model$ Python 2.7 library. The model fitted on training data is validated by making predictions on test data. Model parameters used were, p=2, d=1, q=0. Figure 23 shows the results from this model. Table 8 highlights the model performance against the evaluation metrics.

Table 8: Prediction results for ARIMA

Evaluation Metric	Value
RMSE	0.415
MAE	0.197

The evaluation results for all the models show a tremendous improvement. This is because log transform stabilizes the time series. This helps the forecasting models to make more accurate predictions.

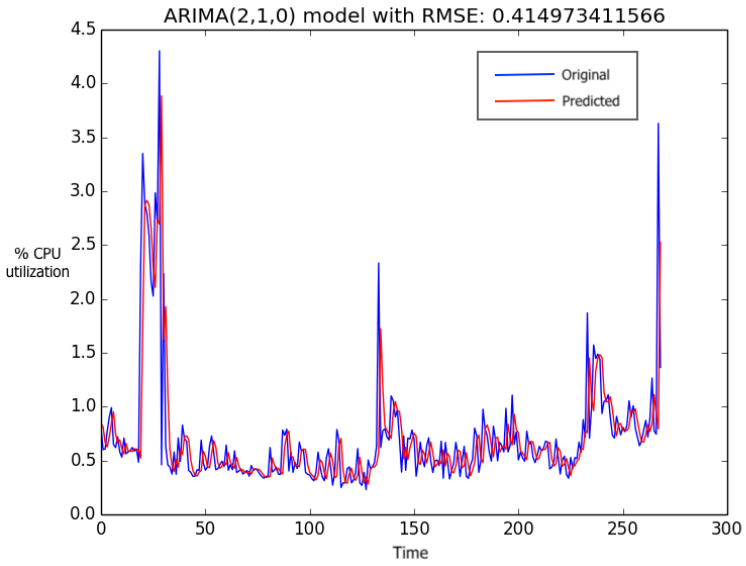


Figure 29: ARIMA model predictions results

The results from this experiment successfully answer the project goals. It can be seen from the prediction plots that we can forecast the future resource workload for any given infrastructure metrics by performing predictive analysis on them by using the above machine learning techniques.

5.4 Comparison of Forecast models

The results from the model can be compared to see overall, which model performs the best. Figure 30 shows the RMSE results of ARIMA vs. MA vs. AR for both the experiments. It can be seen that AR and ARIMA have slight differences but overall have accurate prediction results. Each of the model performed better after log transformation. MA model performed better in the baseline experiment as well compared to the other models. Figure 31 compares the MAE results of these models. Results remain the same. Overall, ARIMA had the least errors in experiment 2 and can be used as the machine learning technique to perform predictive analysis on cloud infrastructure metrics.

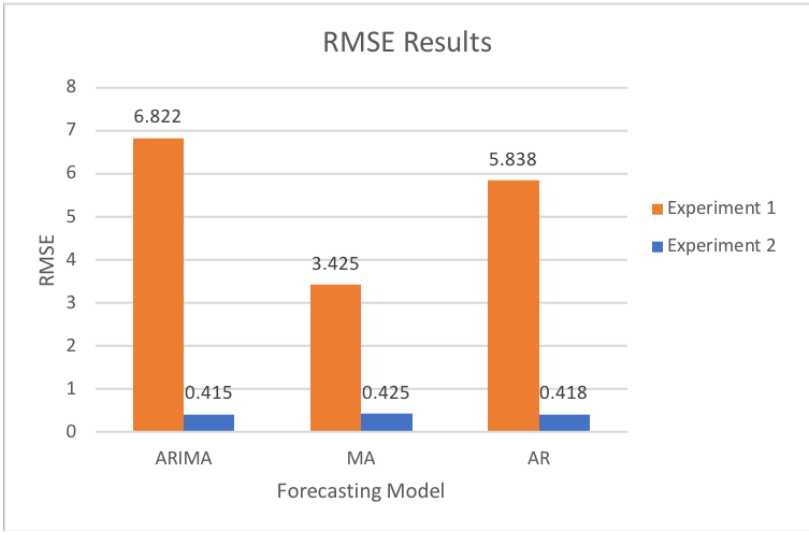


Figure 30: Comparison of model RMSE results

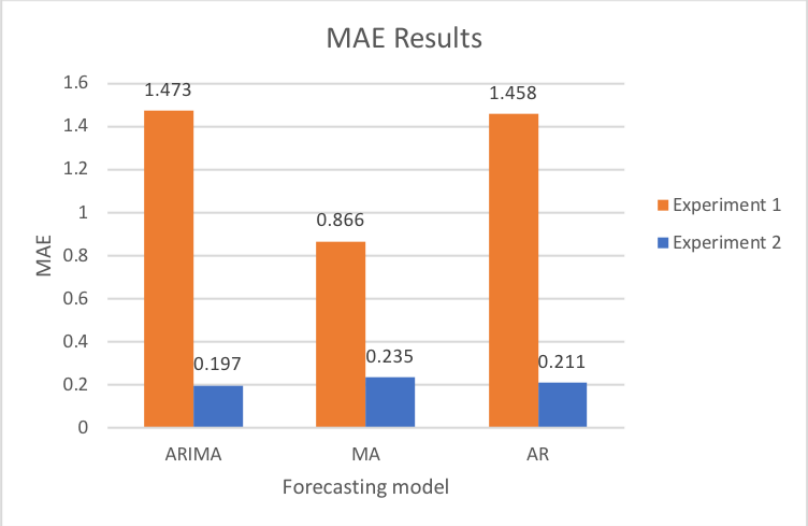


Figure 31: MAE model predictions results

Since, these models were able to accurately predict the future workload forecast, this prediction value can be provided to a resource provisioning system to make timely decision about when to trigger the resource provisioning task. Hence, this can lead to reduced SLA violations.

Additional experiments are available in Appendix A.

CHAPTER 6

Conclusion

The dynamic nature of the cloud poses the biggest challenge in resource management in cloud computing. Historically, both static and dynamic approaches have been successful in the cloud industry. Though reactive approaches tend to be used commonly throughout the industry, they fail to capture the heterogeneity of the cloud infrastructure [10]. The literature agrees that pro-active resource provisioning overcomes the problem of over-utilization and guarantees QoS with minimal SLA violations as they are dynamic. The experiments in this literature extract real world resource utilization traces, tests for stationarity, stationarize the series, and apply forecasting techniques to predict resource utilization. These experiments showcase that machine learning techniques, like time-series forecasting can be leveraged to develop a pro-active resource provisioning system as they allow us to accurately make resource demand predictions. Finally, machine learning techniques for time-series prediction needs to be explored further and adopted industry-wide. Their impact can be meaningful in this research area as they help to understand the inter-connections between applications past workload balance and current QoS requirements. Such a solution will help in balancing SLA violations by the cloud provider and QoS requirements of a cloud user.

Future work on this research can be to use Supervised learning and Deep-learning techniques to predict future workload. The Long Short-Term Memory neural network is designed to interpolate hidden patterns in a long sequence of observations [30]. LSTMs can be used to model time-series data and help in uncovering hidden patterns in the series. Time-series forecasting methods are unsupervised learning techniques. In future, supervised learning techniques like SVMs can also be leveraged to model temporal data and make meaningful predictions.

LIST OF REFERENCES

[1] F. P. Miller, A. F. Vandome, and J. McBrewster, *Amazon Web Services*. Alpha Press, 2010.

[2] H. Fernandez, G. Pierre, and T. Kielmann, ''Autoscaling web applications in heterogeneous cloud infrastructures,'' in *2014 IEEE International Conference on Cloud Engineering*, March 2014, pp. 195--204.

[3] C. Wu, Y. Lee, K. Huang, and K. Lai, ''A framework for proactive resource allocation in iaas clouds,'' in *2017 International Conference on Applied System Innovation (ICASI)*, May 2017, pp. 496--499.

[4] H. S. Guruprasad and b. B H, ''Resource provisioning techniques in cloud computing environment: A survey,'' *International Journal of Research in Computer and Communication Technology*, vol. 3, pp. 395--401, 03 2014.

[5] N. Roy, A. Dubey, and A. Gokhale, ''Efficient autoscaling in the cloud using predictive models for workload forecasting,'' in *2011 IEEE 4th International Conference on Cloud Computing*, July 2011, pp. 500--507.

[6] F. J. Baldán, S. Ramírez-Gallego, C. Bergmeir, F. Herrera, and J. M. Benítez, ''A forecasting methodology for workload forecasting in cloud systems,'' *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 929--941, Oct 2018.

[7] Y. Hu, B. Deng, and F. Pengand, ''Autoscaling prediction models for cloud resource provisioning,'' in *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, Oct 2016, pp. 1364--1369.

[8] W. Iqbal, M. Dailey, and D. Carrera, ''Sla-driven adaptive resource management for web applications on a heterogeneous compute cloud,'' in *Proceedings of the 1st International Conference on Cloud Computing*, ser. CloudCom '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 243--253. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-10665-1_22

[9] ''1998 world cup web site access logs.'' http://ita.ee.lbl.gov/html/contrib/WorldCup.html.

[10] S. Islam, J. Keung, K. Lee, and A. Liu, ''Empirical prediction models for adaptive resource provisioning in the cloud,'' *Future Gener. Comput. Syst.*, vol. 28, no. 1, pp. 155--162, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.future.2011.05.027

[11] C. Vecchiola, R. N. Calheiros, D. Karunamoorthy, and R. Buyya, ''Deadline-driven provisioning of resources for scientific applications in hybrid clouds with aneka,'' *Future Gener. Comput. Syst.*, vol. 28, no. 1, pp. 58--65, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.future.2011.05.008

[12] C. Li and L. Y. Li, ''Optimal resource provisioning for cloud computing environment,'' *J. Supercomput.*, vol. 62, no. 2, pp. 989--1022, Nov. 2012. [Online]. Available: http://dx.doi.org/10.1007/s11227-012-0775-9

[13] B. Stone, ''Inauguration day on twitter,'' https://blog.twitter.com/official/en_us/a/2009/inauguration-day-on-twitter.html, 2009.

[14] R. Hu, J. Jiang, G. Liu, and L. Wang, ''Efficient resources provisioning based on load forecasting in cloud,'' *The Scientific World Journal*, vol. 2014, p. 321231, 02 2014.

[15] M. Mahjoub, A. Mdhaffar, R. B. Halima, and M. Jmaiel, ''A comparative study of the current cloud computing technologies and offers,'' in *Proceedings of the 2011 First International Symposium on Network Cloud Computing and Applications*, ser. NCCA '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 131--134. [Online]. Available: https://doi.org/10.1109/NCCA.2011.28

[16] C. Bunch, V. Arora, N. Chohan, C. Krintz, S. Hegde, and A. Srivastava, ''A pluggable autoscaling service for open cloud paas systems,'' in *Proceedings of the 2012 IEEE/ACM Fifth International Conference on Utility and Cloud Computing*, ser. UCC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 191--194. [Online]. Available: http://dx.doi.org/10.1109/UCC.2012.12

[17] G. E. P. Box and G. Jenkins, *Time Series Analysis, Forecasting and Control*. San Francisco, CA, USA: Holden-Day, Inc., 1990.

[18] P. A. Dinda and D. R. O'Hallaron, ''An evaluation of linear models for host load prediction,'' in *Proceedings. The Eighth International Symposium on High Performance Distributed Computing (Cat. No.99TH8469)*, Aug 1999, pp. 87--96.

[19] ''Autodesk Inc.'' www.autodesk.com, (Accessed on 04/26/2019).

[20] ''Autocad mechanical,'' https://knowledge.autodesk.com/support/autocad-mechanical/learn/caas/CloudHelp/cloudhelp/2019/ENU/AutoCAD-Mechanical/files/GUID-1354621C-1BE8-4C79-882D-2FC14F27108A-htm.html, Jun 19 2018.

[21] ''Object storage service,'' https://forge.autodesk.com/developer/learn/viewer-app/overview, 05 2018, (Accessed on 04/20/2019).

[22] ''Wikibench,'' http://www.wikibench.eu/, (Accessed on 04/21/2019).

[23] Y. Cui, Q. Chen, and J. Yang, ‘‘Automatic in-vivo evolution of kernel policies for better performance,’’ *CoRR*, vol. abs/1508.06356, 2015.

[24] A. Jain, ‘‘A comprehensive beginner‘s guide to create a time series forecast,’’ https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/, 02 2016, (Accessed on 04/06/2019).

[25] ‘‘Unit root test,’’ https://en.wikipedia.org/wiki/Unit_root, 2005, (Accessed on 04/22/2019).

[26] D. Kwiatkowski, P. Phillips, P. Schmidt, and Y. Shin, ‘‘Testing the null hypothesis of stationarity against the alternative of a unit root. how sure are we that economic time series have unit root?’’ *Journal of Econometrics*, vol. 54, pp. 159--178, 10 1992.

[27] J. Salvi, ‘‘Significance of ACF and PACF Plots in Time Series analysis,’’ https://towardsdatascience.com/significance-of-acf-and-pacf-plots-in-time-series-analysis-2fa11a5d10a8.

[28] R. Vink, ‘‘Algorithm breakdown: AR, MA and ARIMA models,’’ https://www.ritchievink.com/blog/2018/09/26/algorithm-breakdown-ar-ma-and-arima-models/, 09 2018, (Accessed on 02/28/2019).

[29] V. Rogério Messias, J. Estrella, R. Ehlers, M. Santana, R. Santana, and S. Reiff-Marganiec, ‘‘Combining time series prediction models using genetic algorithm to autoscaling web applications hosted in the cloud infrastructure,’’ *Neural Computing and Applications*, 12 2015.

[30] N. Sinha, ‘‘Understanding-LSTM,’’ https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47, (Accessed on 04/19/2019).

APPENDIX

Additional Results

A.1 Analysis on OSS Web Latency workload

The OSS service had three accounts: Web, Upload, and Download. In this section, results for predictive analysis on OSS web latency metrics are presented. The OSS Web latency dataset is an hourly time series. Figure A.32 is a visualization of this time series. The ACM series was tested for stationarity using Rolling statistics plot, ADF and KPSS techniques. Results for these test can be given in Figure A.33, Figure A.34, and Figure A.35.

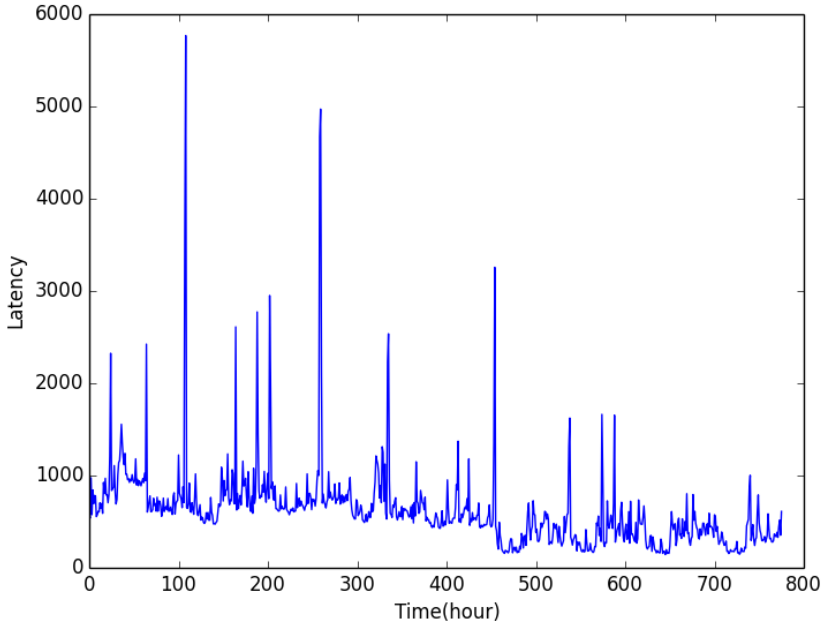


Figure A.32: OSS web latency workload

It's interesting to see that the series is stationary as per ADF test but KPSS show that the series is non-stationary as the test statistic value is greater than critical values. This means that the series has to be differenced to add stationarity to it. As per the Rolling mean test, it can be seen that there is downward trend in the

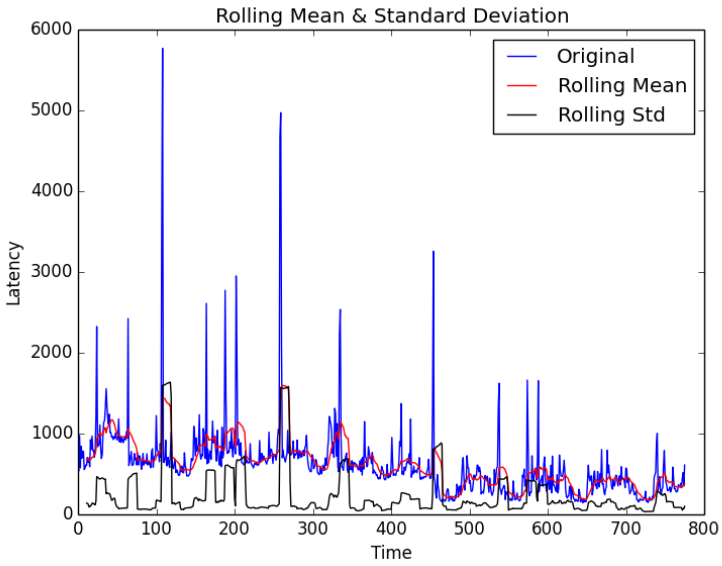


Figure A.33: OSS latency Rolling statistics plot

```
Results of Dickey-Fuller Test:
Test Statistic                  -3.734954
p-value                          0.003648
#Lags Used                      13.000000
Number of Observations Used    762.000000
Critical Value (1%)             -3.438961
Critical Value (5%)             -2.865340
Critical Value (10%)            -2.568794
dtype: float64
```

Figure A.34: OSS latency series ADF test

series as the variance is changing over time. Following experiments were performed to stationarize the series and forecast the future values.

Details of the experiments are given below.

```
Test Statistic           2.441039
p-value                  0.010000
Lags Used               21.000000
Critical Value (10%)     0.347000
Critical Value (5%)      0.463000
Critical Value (2.5%)    0.574000
Critical Value (1%)      0.739000
```

Figure A.35: OSS latency KPSS test

1. **Applying log transformation and 1-degree differencing:** Since the series is non-stationary, it cannot be used for forecasting directly. In this experiment we transformed the original series by applying log transformation to reduce noise in the data. Next to stationarize it further, the series is differenced by the order of 1. The startioanrity test for the resulting time series are shown in Figure A.36 and Figure A.37.

```
Results of Dickey-Fuller Test:
Test Statistic                -1.342957e+01
p-value                        4.038606e-25
#Lags Used                     1.000000e+01
Number of Observations Used    7.640000e+02
Critical Value (1%)           -3.438938e+00
Critical Value (5%)           -2.865330e+00
Critical Value (10%)          -2.568788e+00
dtype: float64
```

Figure A.36: ADF test on log-diff transformed latency metrics

```
Test Statistic            0.02533
p-value                   0.10000
Lags Used                21.00000
Critical Value (10%)      0.34700
Critical Value (5%)       0.46300
Critical Value (2.5%)     0.57400
Critical Value (1%)       0.73900
dtype: float64
```

Figure A.37: KPSS test on log-diff transformed latency metrics

We can see the series is now stationary.

- **Analyzing ACF and PACF Plots**: The ACF and PACF plot seen in Figure A.38 shows that this series is a combination of AR and MA series as it has a slow decaying PACF and the ACF shuts-off after lag=2, 8. Using these model parameters, the forecasting models were built. The series is split into training and test sets in a 70:30 ratio. Table A.9 shows the training and testing data split.

Table A.9: Data Split

Data	Number of intervals used
Training Set	641
Testing Set	143

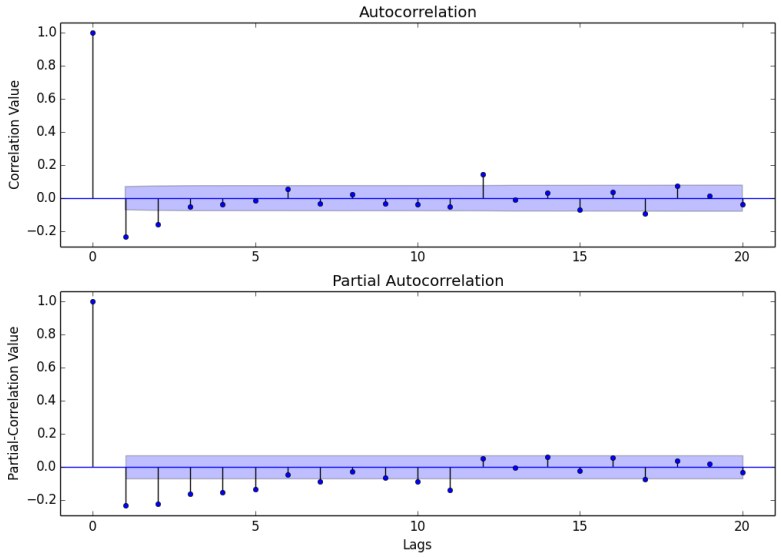


Figure A.38: ACF and PACF for OSS latency log-diff series

- **Forecasting results:**

  (a) **AR model:** The model fitted on training data is validated by making predictions on test data. Lag window, or p = 2. Figure A.39 shows the results from this model. Table A.10. highlights the model performance against the evaluation metrics.

Table A.10: Prediction results for AR

Evaluation Metric	Value
RMSE	0.301
MAE	0.232

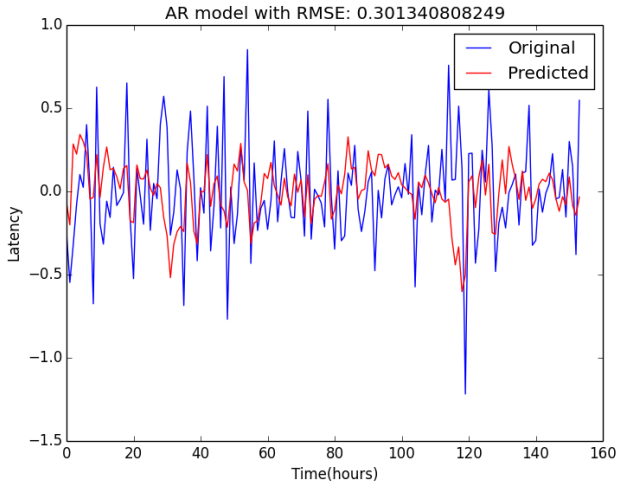


Figure A.39: AR model predictions results

(b) **MA model**: The model fitted on training data is validated by making predictions on test data. Lag window, or q = 2 was chosen for the experiment. Figure A.40 shows the results from this model.

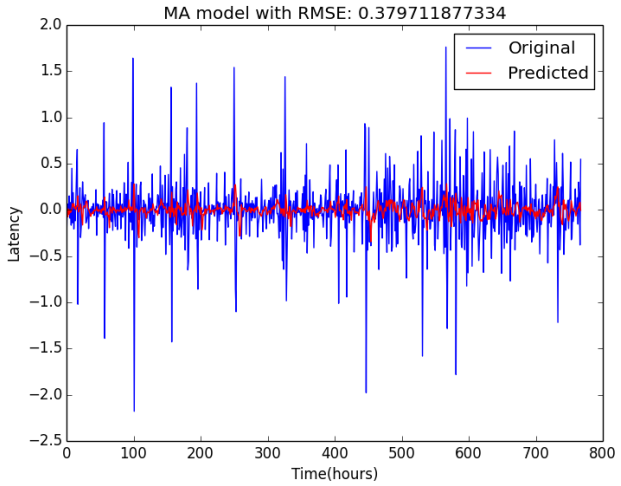


Figure A.40: MA model predictions results

Table A.11 highlights the model performance against the evaluation metrics.

Table A.11: Prediction results for MA

Evaluation Metric	Value
RMSE	0.380
MAE	0.241

(c) **ARIMA model:** The model fitted on training data is validated by making predictions on test data. Model parameters used were, p=2, d=1, q=0. Figure A.41 shows the results from this model. Table A.12

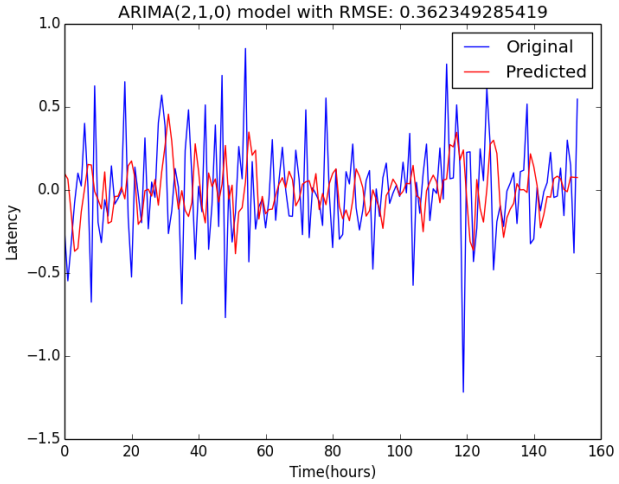


Figure A.41: ARIMA model predictions results

highlights the model performance against the evaluation metrics.

Table A.12: Prediction results for ARIMA

Evaluation Metric	Value
RMSE	0.362
MAE	0.272

All the models were able to forecast the future workload and show highly accurate prediction results as RMSE, MAE values are closer to 0.

2. **Using Log transform and mean differencing**: This is an add on to experiment 1. The process is same, hence only the results are provided. The startioanrity test for the resulting time series are shown in Figure A.42 and Figure A.43. The data split remains the same as the previous experiment.

```
Results of Dickey-Fuller Test:
Test Statistic                -7.393839e+00
p-value                        7.872365e-11
#Lags Used                     1.900000e+01
Number of Observations Used    7.450000e+02
Critical Value (1%)           -3.439158e+00
Critical Value (5%)           -2.865427e+00
Critical Value (10%)          -2.568840e+00
dtype: float64
```

Figure A.42: ADF test on log transformed mean differenced latency metrics

```
Test Statistic            0.021691
p-value                   0.100000
Lags Used                20.000000
Critical Value (10%)      0.347000
Critical Value (5%)       0.463000
Critical Value (2.5%)     0.574000
Critical Value (1%)       0.739000
dtype: float64
```

Figure A.43: KPSS test on log transformed mean differenced latency metrics

- **Analyzing ACF and PACF Plots**: The ACF and PACF plot seen in Figure A.44.
- **Forecasting results:**
  (a) **AR model:** Lag window, or p = 8. Figure A.45 shows the results from this model. Table A.13. highlights the model performance against the evaluation metrics.

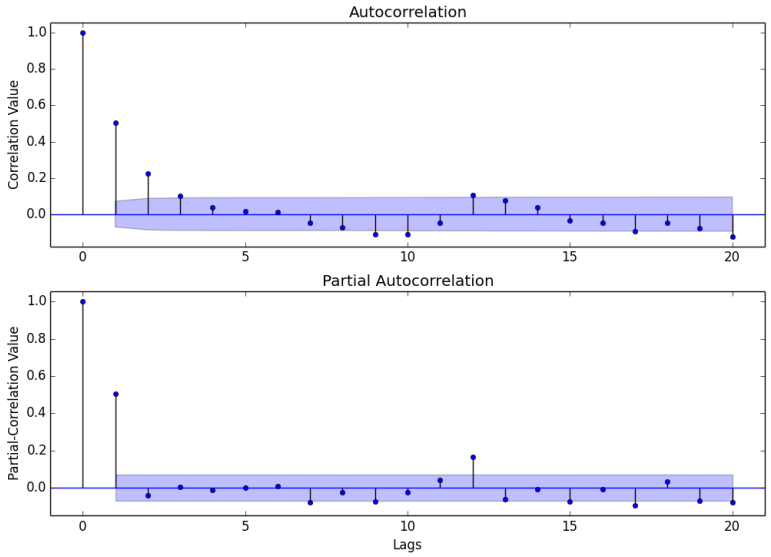


Figure A.44: ACF and PACF for log transformed mean differenced latency metrics

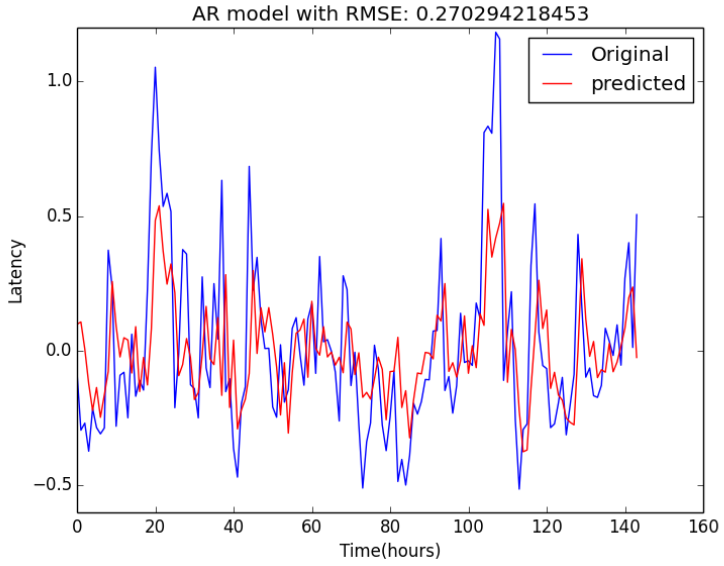


Figure A.45: AR model predictions results

(b) **MA model**: Lag window, or q = 1 was chosen for the experiment. Figure A.46 shows the results from this model.

Table A.14 highlights the model performance against the evaluation

Table A.13: Prediction results for AR

Evaluation Metric	Value
RMSE	0.27
MAE	0.207

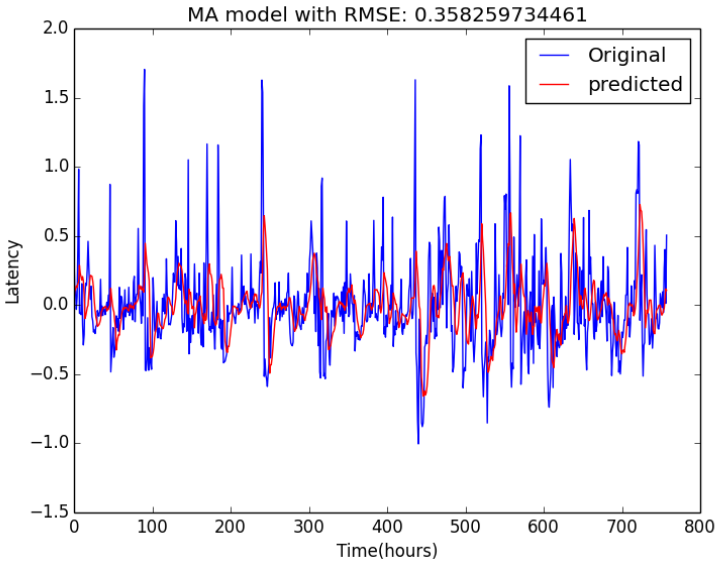


Figure A.46: MA model predictions results

metrics.

Table A.14: Prediction results for MA

Evaluation Metric	Value
RMSE	0.358
MAE	0.252

(c) **ARIMA model:** Model parameters used were, p=8, d=0, q=1. Figure A.47 shows the results from this model. Table A.15 highlights the

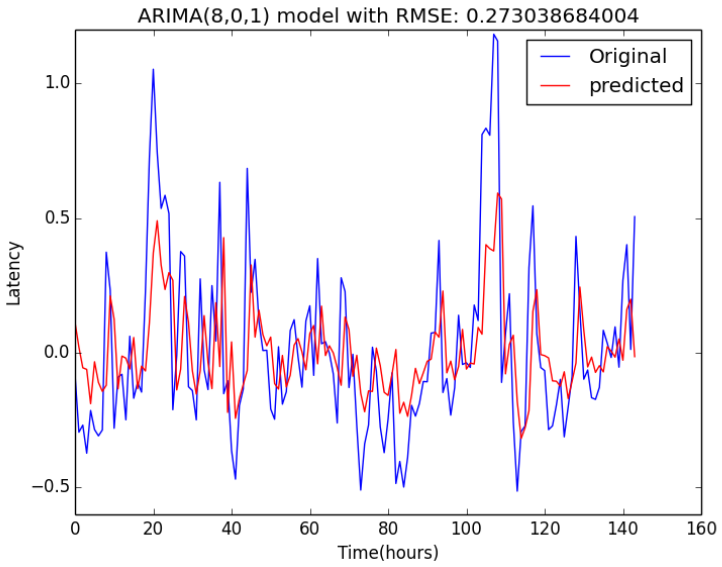


Figure A.47: ARIMA model predictions results

model performance against the evaluation metrics.

Table A.15: Prediction results for ARIMA

Evaluation Metric	Value
RMSE	0.273
MAE	0.207

A.2 Comparison of Forecast models

The results from the model can be compared to see overall, which model performs the best. Figure A.48 shows the RMSE results of ARIMA vs. MA vs. AR for both the experiments. It can be seen that AR and ARIMA have slight differences but overall have acurate prediction results. Each of the models performed better after log mean differencing transformation. Figure A.49 compares the MAE results of these models.

Results remain the same. Overall, AR had the least errors in experiment 2 and

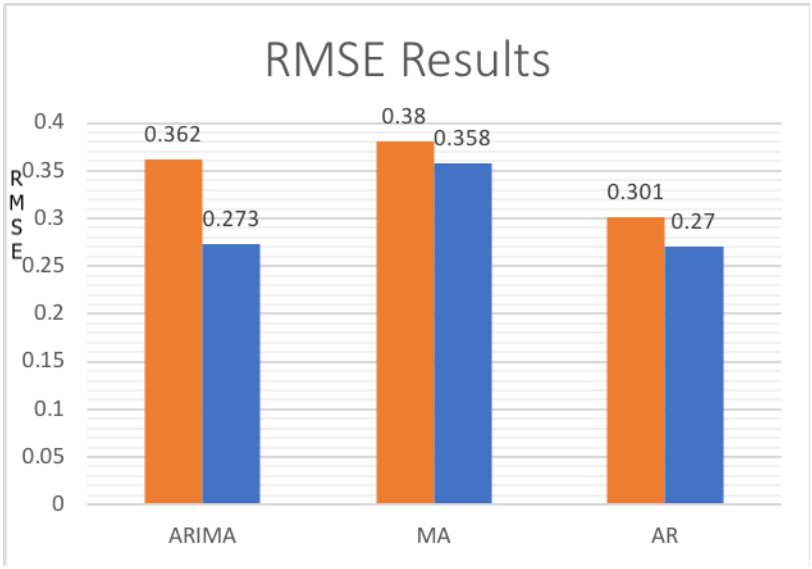


Figure A.48: Comparison of model RMSE results

can be used as the machine learning technique to perform predictive analysis on cloud infrastructure metrics.

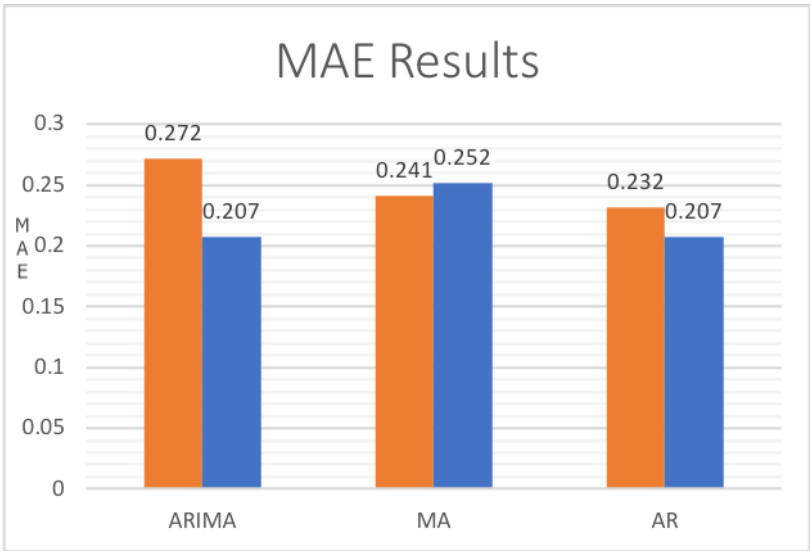


Figure A.49: MAE model predictions results