Fast analysis and prediction in large scale virtual machines resource utilisation

Abubakar, A., Barbhuiya, S., Kilpatrick, P., Ngo, V., & Nikolopoulos, D. S. (2021). Fast analysis and prediction in large scale virtual machines resource utilisation. In *Proceedings of the 10th International Conference on Cloud Computing and Services Science, CLOSER 2020* SciTePress.

Fast Analysis and Prediction in Large Scale Virtual Machines Resource Utilisation

Abdullahi Abubakar*[1], Sakil Barbhuiya[1] , Peter Kilpatrick[1], Ngo Anh Vien[1] and Dimitrios S. Nikolopoulos[2]

*_Department of Computer Science. Waziri Umaru Federal Polytechnic, Birnin Kebbi. Nigeria._

[1]_School of Electronics, Electrical Engineering and Computer Science. Queen's University, Belfast, UK_

[2]_John W. Hancock Professor of Engineering and Professor of Computer Science. Virginia Tech, USA_

{_aabubakar02, sbarbhuiya03, p.kilpatrick, v.ngo_}_@qub.ac.uk, dsn@vt.edu_

Keywords: Virtual Machine, Cloud/Data Centre, Prediction, Scalability, Performance, Partitioning, Parallelism, Big Data.

Abstract: Most Cloud providers running Virtual Machines (VMs) have a constant goal of preventing downtime, increasing performance and power management among others. The most effective way to achieve these goals is to be proactive by predicting the behaviours of the VMs. Analysing VMs is important, as it can help cloud providers gain insights to understand the needs of their customers, predict their demands, and optimise the use of resources. To manage the resources in the cloud efficiently, and to ensure the performance of cloud services, it is crucial to predict the behaviour of VMs accurately. This will also help the cloud provider improve VM placement, scheduling, consolidation, power management, etc. In this paper, we proposed a framework for fast analysis and prediction in large scale VM CPU utilisation. We use a novel approach both in terms of the algorithms employed for prediction and in terms of the tools used to run these algorithms with a large dataset to deliver a solid VM CPU utilisation predictor. We processed over two million VMs from Microsoft Azure VM traces and filter out the VMs with complete one month of data which amount to 28,858VMs. The filtered VMs were subsequently used for prediction. Our Statistical analysis reveals that 94% of these VMs are predictable. Furthermore, we investigate the patterns and behaviours of those VMs and realised that most VMs have one or several spikes of which majority of those spikes are not seasonal. For all the 28,858VMs analysed and forecasted, we accurately predicted 17,523 (61%) VMs based on their CPU. We use Apache Spark for parallel and distributed processing to achieve fast processing. In term of fast processing (execution time), on average, each VM is analysed and predicted within three seconds.

1 INTRODUCTION

Organisations across various fields are producing and storing vast amounts of data. These large volumes of data are generated at an unprecedented rate from heterogeneous sources such as; sensors, servers, social media, IoT, mobile devices, etc. (Assunção et al., 2015). These heterogeneities gave birth to the term "big data". According to Rajaraman (Rajaraman, 2016), about 8 Zettabytes of digital data were generated in 2015. Kune *et al* (Kune et al., 2016), estimated that by 2020, the volume of data will reach 40 Zettabytes. This implies that the volume of data being generated is doubling every two years.

As cloud/data-centres continue to grow in scale and complexity, effective monitoring and managing cloud services becomes critical. It is essential for cloud providers to efficiently manage their services to facilitate the extraction of reliable insight and to optimise cost (Oussous et al., 2018). The two major challenges of handling large volumes of data (big data) are management and processing. Managing the complexity of big data (velocity, volume and variety) and processing it in a distributed environment with a mix of applications remain open challenge (Khan et al., 2014). Analysing Virtual Machines (VMs) is important, as this can help cloud service providers such as Amazon, Google, IBM and Microsoft gain insights to understand the needs of their customers, predict their demands, and optimise the use of resources. To manage the resources in the cloud efficiently, and to ensure the performance of cloud services, it is crucial to predict the behaviour of VMs accurately. Based on the above motivation we set two major objectives;

- fast data analysis
- achieve accurate prediction

Most Cloud providers running VMs have a constant goal of preventing downtime, increase performance and power management among others. Therefore, one of the most effective way to achieve these goals is to be proactive by predicting the behaviours of the VMs. A good predictor can help improve VM placement, scheduling, consolidation, power management, e.t.c. Furthermore, to ensure the quality of services is maintained, a good predictor of VM resource utilisation can serve as a powerful tool for anomaly detection. In this paper, a prediction model for forecasting CPU utilization is proposed.

The novelty of this work is using an adaptive predictive model which is both accurate and efficient for prediction. Additionally, our model is designed to quickly handle realistic large scale dataset such as Azure VMs traces (Cortez et al., 2017). We use a novel approach both in terms of the algorithm employed for prediction and the tools used to manage our large scale dataset. We use Apache Spark for parallel and distributed processing to achieve fast processing, and ARIMA, a statistical model for scalable time-series analysis for prediction.

In terms of contribution, we address fast, scalable prediction of VM behaviour. In addition, we communicate some useful information extracted (discovered) from the datasets. These pieces of information are useful for other researchers who want to carry out further investigations, such as Anomaly detection among others. We also highlight methods and tools employed to achieve scalability, fast data analytics, and accurate prediction. Additionally, a sanitised dataset for 28,858 VMs has been made available for other researchers (see the appendix section).

We processed over two million VMs from Microsoft Azure VM traces and filter out the VMs with complete one month of data which amount to 28,858VMs. We refer to this as long-running VMs. The filtered VMs were subsequently used for prediction. To the best of our knowledge, we are the first to predicts all the long-running 28,858VMs of Azure VMs traces and accurately predicts 17,562VMs. Work done by Comden et al (Comden et al., 2019), is the closest work to us in terms of number of VMs analysed from the same dataset. However, they only analysed 1003 Azure VMs. In their work, they proposed an automatic algorithm selection scheme, which can help cloud operators to pick the right algorithm to manage cloud computing resources. They use four prediction methods namely; Random Forest, Naive, Seasonal Exponential Smoothing and SARMA. They empirically study the prediction errors from Azure VM traces and propose a simple prediction error model. The model creates a simple online meta-algorithm that chooses the best algorithm (Comden et al., 2019).

The rest of this paper is organised as follows. Section 2 presents our methodology and approach. Section 4 presents the experimental processes pursued. Section 5 presents results demonstrating the capability of our method. We discuss related work in Section 3 and conclude in Section 6.

2 METHODOLOGY

This section describes the approach (experimental workflow) implemented to achieve the set objectives. In this section, we present our framework which aimed at predicting VMs resource utilisation as depicted in Figure 1. The experimental workflow is separated into four phases, namely; Data source, Data management, Modelling and Result analysis. These phases are briefly discussed below;

- ***Data Source:*** A Microsoft Azure VMs dataset is analysed. The dataset has 125 files which when concatenated contain over two million VMs with 117GB file size. The VMs ran during 30 days interval between 16th November, 2016 and 16th February,2017(Cortez et al., 2017). The information includes; identification numbers for each VM and resource utilisations (minimum, average, and maximum CPU utilisation) reported at the time interval of every 5 minutes.
- ***Data Management:*** Performing analytics on large volumes of data requires efficient methods to store, filter, transform, and retrieve the data (Assunção et al., 2015). Therefore it is very important when chosen the appropriate framework for big data analytics to consider several critical aspects, such as data size, computing capacity, scalability, fault tolerance and framework functionality (Singh and Reddy, 2015). Based on these critical aspects, we choose Apache spark as our data processing framework. This is because a number of studies (Merrouchi et al., 2018; Alkatheri et al., 2019; Marcu et al., 2016), show that Spark is the best in terms of measuring processing time, task performance, low latency, good fault-tolerance and scalable computing capability among others. The dataset analysed is divided into 125 files which when concatenated amount to 117GB. In other to have a complete one month data, this large volume of data demands fast and accurate preprocessing tasks, including integration, filtering and transformation. These tasks were easily and accurately handle with Spark. The

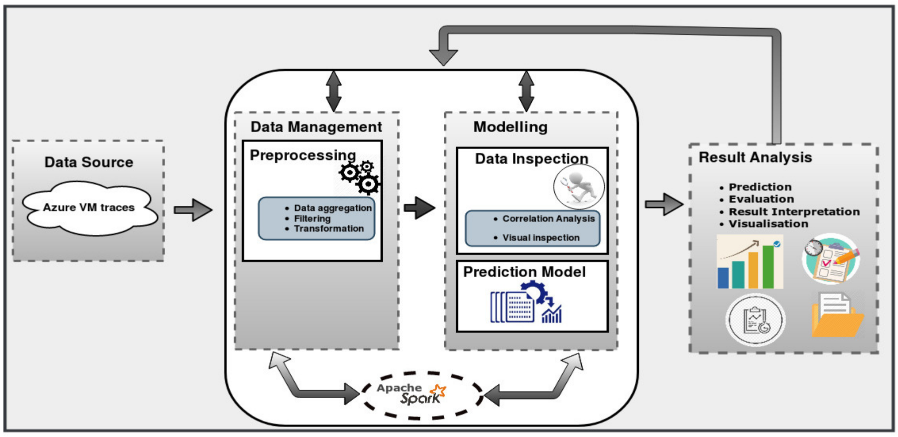


Figure 1: Model Architecture (Analytic workflow)

result of the preprocessed data is then used for prediction, evaluation, visualisation and interpretation.

- ***Modelling:*** Studies in (Assunção et al., 2015), show that analytics solutions can be classified as descriptive and/or predictive. Descriptive analytics is concerned with modelling past behaviour by using historical data to identify patterns and create management reports, whereas, Predictive analytics attempts to forecast the future by analysing current and historical data (Assunção et al., 2015). We adopted both solutions while analyzing and predicting Azure VMs. We first implemented descriptive analytics which we referred to as data inspection, by using historical data to identify patterns on the prepared data (the result of the preprocessed data). The main reasons for implementing this phase is to help us select an appropriate predictive model. We carried our data inspection by implementing autocorrelation analysis and decomposition. Autocorrelation is a characteristic of data which shows the degree of similarity between the values of the same variables over successive time intervals. It summarises the strength of a relationship between an observation in a time series with observation(s) at prior time steps (Andrews, 1991; Brownlee, 2017). Whereas, Decomposition is a mechanism to split a time series into several components, each representing an underlying pattern category (Hyndman and Athanasopoulos, 2018). Decomposition provides a structured way of thinking about how to best capture each of these components in a given model (Brownlee, 2017) and in terms of modelling complexity (Shmueli and Lichtendahl Jr, 2016).
After descriptive analytics, we proceeded to predictive analytics. Accurate prediction of VMs is vital for cloud resources allocation. Therefore it is important to choose a good predictive model. Based on the outcomes of descriptive analytics which comprises autocorrelation analysis and decomposition, we implemented ARIMA, a classical time series prediction algorithm to forecast the behaviour of each VM.

- ***Result analysis & Visualisation:*** In this phase, the experimental results are analysed and interpreted in accordance with the guidance and recommendations of (Assunção et al., 2015). They suggest that initial result generated by a first model is not enough to draw a final conclusion. There is a need to re-evaluate results and might sometimes lead to possible modification to generate new models or adjust existing ones. In our experiment, the re-evaluation was done by regulating the percentage of both training/testing data, and running the predictive model for any changes effected.

3 RELATED WORK

Many researchers such as (Wang et al., 2016; Comden et al., 2019; Calheiros et al., 2015) are working in the area of cloud computing to predict the behaviours of VMs. The two most popular VM prediction approaches are homeostatic prediction (Lingyun Yang et al., 2003) and history-based prediction. In homeostatic prediction, the incoming data point at the next time instance is predicted by adding or subtracting a value such as the mean of previous data point from the current data point. Whereas, history-based methods analyze previous data points instances and extract patterns to forecast the future (Kumar and Singh, 2018).

To use historical information, VM CPU utilisation can be presented in the form of time series. A time series data is a set of data points measured at successive points in time spaced at uniform time interval (Hyndman and Athanasopoulos, 2018). Studies (Kumar and Singh, 2018) show that data centre workloads tend to present behaviour that can be effectively captured by time series-based models.

Classical models have been widely used for time series prediction. These models including Autoregression (AR) (Gersch and Brotherton, 1980), Holt-Winter (Yan-ming Yang et al., 2017), Exponentially Weighted Moving Average (EWMA) (Fehlmann and Kranich, 2014), Moving Average (MA), and Autoregressive Integrated Moving Average (ARIMA) (Ahmar et al., 2018) among others, can be used for VMs prediction.

Li (Li, 2005), proposed an Autoregression (AR) based method to predict the webserver workload. The author used a linear combination of past values of the variable to forecast the value for upcoming time instances. The model was strictly linear in nature, therefore, does not take into account non-linear behaviour that might occur in a cloud setting. Wang (Wang et al., 2016) used a combination of ARIMA and BP neural network to predict CPU utilisation of a single Virtual Machine. Furthermore, (Calheiros et al., 2015), develop a workload prediction module using the ARIMA model. The predicted load is used to dynamically provision VMs in an elastic cloud environment for serving the predicted requests taking into consideration the quality of service (QoS) parameters, such as response time and rejection rate. The major drawback of their work is, they manually tweak the parameters of the ARIMA(p,d,q) which is time-wasting and could not fit for analysing multiple VMs concurrently. Therefore, to increase the applicability of our proposed work, we adopted ARIMA-based prediction. We developed a method to grid search ARIMA hyperparameters to determine the optimal values for our model. We automate the process of training and evaluating ARIMA models on different combinations of model hyperparameters. We specify a grid of (p, d, q) ARIMA parameters to iterate on the training data which avoids manual parameter tweaking.

We are analysing a very large scale dataset with the goal of achieving fast processing and accurate prediction. Therefore, it is crucial when chosen the appropriate framework for big data analytics to consider several critical aspects, such as data size, computing capacity, scalability, fault tolerance and framework functionality (Singh and Reddy, 2015). Based on the above critical aspects, we choose Apache spark as our data processing framework. This is because, all of the studies reviewed here (Alkatheri et al., 2019; Merrouchi et al., 2018; Marcu et al., 2016; Singh and Reddy, 2015) indicate that, Spark is the best in terms of measuring processing time, task performance, low latency, good fault-tolerance and scalable computing capability, compared to Hadoop, Storm and Flink. Work done by Comden et al (Comden et al., 2019), is the closest work to us in terms of number of VMs analysed from the same dataset. However, they only analysed 1003 Azure VMs, whereas we analysed 28,858VM.

4 EXPERIMENT

This section describes the details of the experiments conducted. These include tool selection, experimental steps, dataset description, prediction algorithm selection and implementation.

4.1 Experimental tool

Data analytics is a complex process that demands expertise in data understanding, cleaning up, proper method selection, and analysing & interpreting the results. Tools are fundamental to help perform these tasks. Therefore, it is important to understand the requirements in order to choose appropriate tools (Assunção et al., 2015).

4.1.1 Data analytic tool

Apache Spark is one of the most widely used open-source big data processing engines (Armbrust et al., 2015). It is a hybrid processing framework that provides high-speed batch as well as real-time processing on large scale datasets. It has wide support, integrated libraries and flexible integrations (Zaharia et al., 2012). Apache Spark claimed to be 100 times faster than Hadoop (Shvachko et al., 2010) due to in-memory computation (Vernik et al., 2018).

Spark's generality has several important benefits. First, we can easily develop an application because spark uses a unified API. Secondly, we can develop a parallel and distributed application to achieve faster processing speed of about 100x faster in memory and 10x faster on disk (Shvachko et al., 2010).

Going by the above benefits, we propose a scalable data analytic framework and predictive model based on Apache Spark. Additionally, we chose Spark to achieve the efficiency of quickly writing code that runs in a distributed system. Furthermore, Spark handles the parallelisation and organisation of the data processing tasks neatly. In our experiment, we use spark version 2.3.1 with scala version 2.11.8.

4.1.2 Prediction tool

Classic time series models such as Autoregressive Integrated Moving Average (ARIMA) (Ahmar et al., 2018; Calheiros et al., 2015), Holt-Winter (Yan-ming Yang et al., 2017) and Exponentially Weighted Moving Average (EWMA) (Fehlmann and Kranich, 2014) can be used for VMs prediction. In our experiment, prediction models are developed base on ARIMA.

ARIMA models are the most general class of models for forecasting a time series which can be stationarised by transformations such as differencing and lagging (Hyndman and Athanasopoulos, 2018). Differencing is useful, it helps stabilise the mean of a time series by removing changes in the level of a time series (Hyndman and Athanasopoulos, 2018) (Brockwell and Davis, 2016). ARIMA is an extension of the ARMA model which is a combination of Auto-Regressive(AR) and Moving Average (MA) models. The AR and MA models represent time series that are generated by passing the input through a linear filter which produces the output Y(t) at any time t. The major difference between the two is, with AR models time series are generated using the previous *output values Y(t - τ)*. Whereas, MA models use only the *input values X(t -τ)* to generate the time series (Yu et al., 2016), where τ= 0,1,2,3....n. ARIMA apply the ARMA model not immediately to the given time series, but after its preliminary differencing, which is the time series obtained by computing the differences between consecutive values of the original time series (Cheboli et al., 2010). The model is generally specified as (p, d, q), where p is the number of autoregressive order, d is the differencing order, and q is the moving average (Brockwell and Davis, 2016). The prediction equation is given by;

$$\hat{\mathbf{Y}}_t = \alpha_0 + \sum_{i=1}^{p} \phi_i * L^i * Y_t + \sum_{i=1}^{q} \theta_i * L^i * \varepsilon_t + \varepsilon_t \qquad (1)$$

Where α_0 is constant, ϕ is autoregressive coefficients, θ is moving average and ε_t is white noise at time $_t$, and L is the lag operator which when applied to Y returns the prior value.

The choice of ARIMA was attributable for it scalability in related work (Yu et al., 2016; Wang et al., 2016; Ahmar et al., 2018; Calheiros et al., 2015; Schmidt et al., 2018). Moreover, it is specifically built and works well for time series data (Hyndman and Athanasopoulos, 2018).

4.1.3 Environment

The experiments are set up and run on a High Performance Computing (HPC) cluster called “Kelvin2”. The operating system is Centos7 (64 bit). The compute nodes are HP Apollo 200 servers, with high performance 40Gbps Infiniband fabric (32Gbps effective bandwidth per link). Our experiment is limited to a cluster of four nodes.
Table 1 summarizes the cluster resources used in our experiment.

Table 1: Cluster Resources

Node Name	Memory Capacity (TB)	No of Core	Speed per Core (GHz)
Smp01	1.0	40	2.60
Smp02	1.0	40	2.60
Smp03	1.5	24	2.60
Smp04	1.5	24	2.60
Total	5.0	128	

4.2 Experiment Dataset

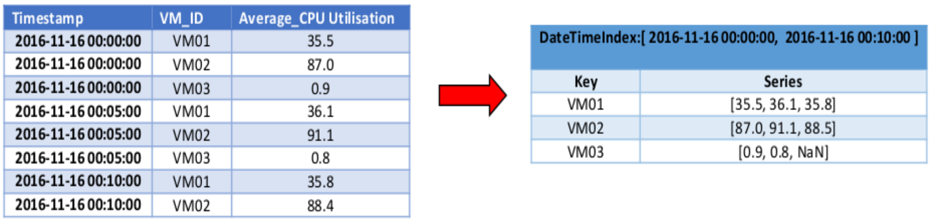
This study was conducted on a realistic dataset of Microsoft Azure Virtual Machines (Cortez et al., 2017). The dataset contains over two million VMs that ran during 30 days interval between 16th November, 2016 and 16th February, 2017 (Cortez et al., 2017). The information includes; identification numbers for each VM and resource utilisations (minimum, average, and maximum CPU utilisation) reported at the time interval of every 5 minutes. The total number of VMs in the dataset is 2,013,767.

4.3 Experimental Steps

4.3.1 Data Preprocessing

Average CPU utilisation from Microsoft Azure VMs is used in our experiment. The dataset has 125 files which when concatenated amount to 117GB. In other to have a complete one month data, this large volume of data demands preprocessing tasks including integration, filtering and transformation.

We loaded and concatenated these files in Apache Spark as depicted in Figure 1. We first create a DataFrame with three columns namely; VM ID, Time and Average CPU. Then filter and grouped each record by VM ID. Finally, we convert our data to the array of dense vector for model consumption. Figure 2 depicted how our transformed data look. It is important to note that from over two million VMs analysed, we realized that only twenty-eight thousand eight hundred and fifty-eight (28,858) VMs has a complete one month data. Therefore, predictions were carried out on the VMs with complete one month of data.



Timestamp	VM_ID	Average_CPU Utilisation
2016-11-16 00:00:00	VM01	35.5
2016-11-16 00:00:00	VM02	87.0
2016-11-16 00:00:00	VM03	0.9
2016-11-16 00:05:00	VM01	36.1
2016-11-16 00:05:00	VM02	91.1
2016-11-16 00:05:00	VM03	0.8
2016-11-16 00:10:00	VM01	35.8
2016-11-16 00:10:00	VM02	88.4

DateTimeIndex:[2016-11-16 00:00:00, 2016-11-16 00:10:00]

Key	Series
VM01	[35.5, 36.1, 35.8]
VM02	[87.0, 91.1, 88.5]
VM03	[0.9, 0.8, NaN]

Figure 2: Preprocessed result indicating both original and transformed data

4.3.2 Data Inspection

As discussed in section 2, and depicted in figure 1, the prediction process started by modelling past behaviour of the filtered long-running VMs to identify patterns. The main reason for data inspection is to help us find out whether our VMs are predictable or not. Furthermore, it will also help us select an appropriate prediction model, since our goal is to develop a model that makes accurate predictions. We carry our data inspection by implementing autocorrelation analysis and decomposition.

We started with the autocorrelation analysis. We applied the Durbin-Watson (DW) statistical test on our filtered VMs. The Durbin-Watson statistic is a test for autocorrelation in the residuals from a statistical regression analysis. The test always produces a test statistic that ranges between 0.0 to 4.0. Value of 2.0 (the middle of the range) suggests no autocorrelation detected. Conversely, values closer to 0.0 indicates positive autocorrelation, while values above 2.0 indicate negative autocorrelation. In our experiment, the DW test suggests that about 94 percent of the VMs indicates autocorrelation detected, and hence are predictable.

Time series data can exhibit a variety of patterns, and it is often helpful to split and decompose a time series into several components, each representing an underlying pattern category (Hyndman and Athanasopoulos, 2018). This is because decomposition provides a structured way of thinking about how to best capture each of these components in a given model (Brownlee, 2017) and in terms of modelling complexity (Shmueli and Lichtendahl Jr, 2016). We decompose our time series into four components. Namely;

- ***Observed***: The original observed data in series
- ***Trend***: The long term movement in a time series.
- ***Seasonal***: The repeating short-term cycle in the series
- ***Residuals***: Time series after the trend and seasonal components are removed

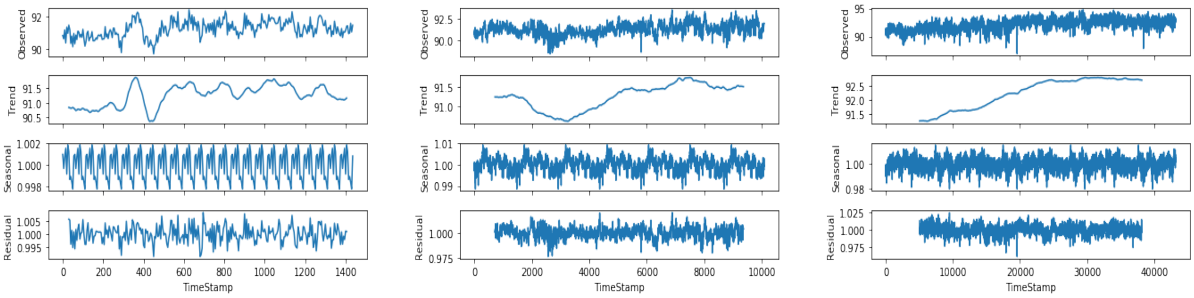
To get more insight and have more understanding of our data, we implemented decomposition. Figure 3 is a decomposition result of a randomly selected VM. In Figure 3(a), we used first-day data to check the properties of the observed data. The seasonality information extracted from the series indicates a pattern is repeating itself every hour. The residuals are also interesting, showing periods of high variability towards the middle of the series. We extend our data to one week to check for daily properties as shown in Figure 3(b). Here we can see that the pattern is also repeating itself every day. The trend decrease toward the second day but gradually increases on the third day and increment is maintained throughout the week. Finally, Figure 3(c) represents the decomposition of a complete/one month of data (8640 data points) to check for trend, seasonality, and residue. We can see that the trend from the series does seem reasonable, indicating gradual upward increment in the series. The seasonality information extracted is also interesting, showing repeating pattern every week.

Furthermore, we carried out a visual inspection and realised that most series contain one or several big spikes as shown in figure 4. As most spikes are not seasonal, it could interest other researchers to investigate more and find out whether those spikes could potentially be anomalies.

4.3.3 Prediction Method

Having realised and confirmed that our VMs are predictable based on the DW test and decomposition (see Figure 3), we proceeded to predictive analytics.

The filtered VMs with complete (one month) data, which amount to twenty-eight thousand, eight hundred and fifty-eight (28,858) are used for our experiment. Each of the 28,858 VM is predicted. All the 28, 858 VMs predicted has 8,640 data points. We used ARIMA on Apache Spark to develop our model for prediction, which was implemented in the Scala programming language. We modelled the relationship between CPU utilisation (dependent variable) and timestamp (independent variable). As recommended by (Brownlee, 2017), we split the data into a training set and a test set. The training set was



(a) Decomposing first day data (b) Decomposing first week data (c) Decomposing one month data

Figure 3: Decomposition result of a sample VM (checking for hourly, daily and weekly behaviours)

used for model preparation and the test set was used to evaluate it. We respect the temporal ordering in which values were observed when splitting our data, since the time dimension of observations does not allow random split into groups. The dataset is split in two: 90% for the training and 10% for the testing.

After the train-test split, the process of fitting the ARIMA model based on the Box-Jenkins method (Box et al., 2015) is done on the training data set. According to Box-Jenkins method (Box et al., 2015), the time series must be transformed into a stationary time series, that is, for each $(X_t, X_{t+ø})$, the mean and variance of the process must be constant and independent of t, where ø is the time difference (lag) between the data points. This transformation is achieved by differencing the original time series.

We developed a method to grid search ARIMA hyperparameters to determine the optimal values for our model. We automate the process of training and evaluating ARIMA models on different combinations of model hyperparameters. We specify a grid of (p, d, q) ARIMA parameters to iterate on the training data which avoids manual parameter tweaking. A model is created for every parameter combination and its performance evaluated based on the scale-dependent error as suggested by (Hyndman and Athanasopoulos, 2018), in our case, Mean Square Error (MSE). Then we selected the best model based on the computed residuals. The best model is the one with the least MSE/residuals, which is then applied to the data to generate predictions (forecast).

After generating predictions, all predicted data points are compared with the expected values on our test dataset with a forecast error score calculated. It is important to note that forecast errors are different from residuals. Forecast errors are calculated on the testing data while residuals are calculated on the training data (Hyndman and Athanasopoulos, 2018). We again use MSE to measure the forecast accuracy. This is because it is more sensitive than other measures and it penalises large errors. MSE is calculated as the average of the squared forecast error values as shown in equation (2). The forecast error function is defined as;

$$\textbf{MSE} = \frac{1}{n}\sum_{t=1}^{n}(\hat{y}_t - y_t)^2 \tag{2}$$

Where $\hat{y}_t$ and y_t are predicted and expected data point at time t, respectively.

An MSE of zero, or a very small number close to zero indicates accurate prediction (Hyndman and Athanasopoulos, 2018; Aggarwal, 2018) and (Brownlee, 2017).

5 RESULTS & DISCUSSION

In this section, we present experimental results demonstrating the capability of our method both in terms of accurate prediction and in terms of faster processing as seen in figure 5 and figure 8 respectively. We also describe the technique employed in our experiment to achieve faster analysis followed by presentation of the result obtained.

5.1 Results Analysis

The performance of prediction is evaluated by computing the prediction error. On a scale of 0 to 100 (Unit of CPU utilisation), we classified prediction performance for each VM in three categories;

- Low Mean Square Error (LMSE): VM with error score (MSE) ranging between 0.0 to 0.9.
- Medium mean Squaure Error(MMSE): VM with error score (MSE) ranging between 1.0 to 5.0.
- High mean square Error(HMSE): VM with error score (MSE) above 5.0

The VMs that fall in the first category are considered as VMs with an accurate prediction. This is because an MSE of zero, or a very small number near zero indicates accurate prediction (Hyndman and

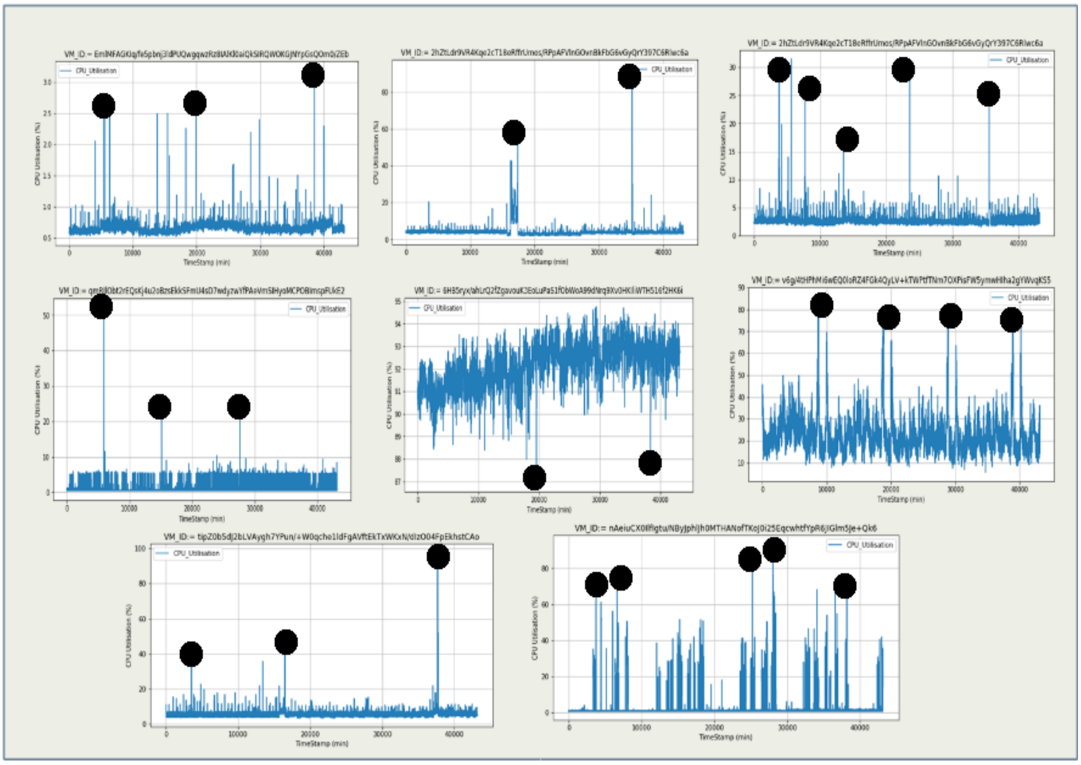
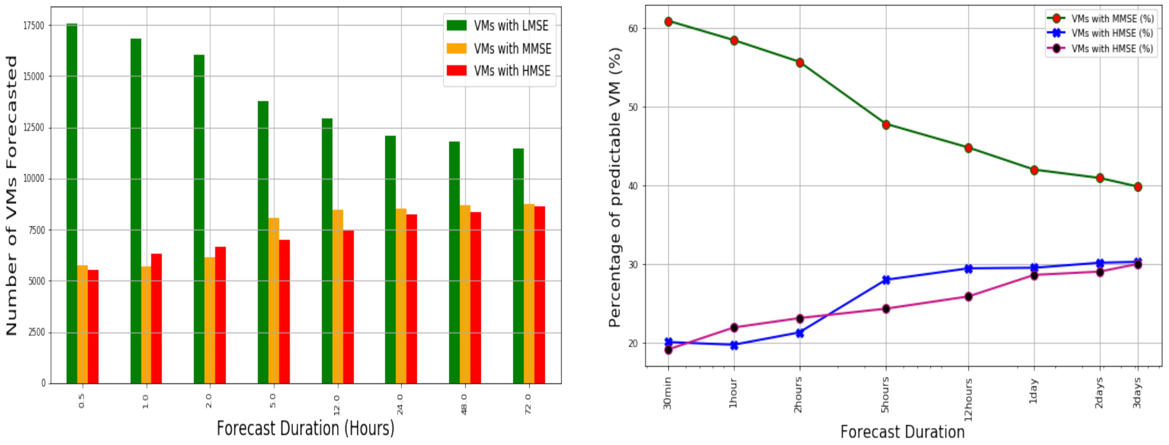


Figure 4: Time series CPU utilization of nine randomly selected VMs



(a) MSE for VMs over different forecast period

(b) Percentage of MSE for VMs over different forecast period

Figure 5: Forecast result

Athanasopoulos, 2018; Aggarwal, 2018; Brownlee, 2017).

Finally, we forecast the future of our time series after automating the (p,d,q) parameters and obtaining the best fit. We varied the forecast period and compute the forecast error for every VM for each period. The eight different forecast periods implemented are 30 minutes, 1 hour, 2 hours, 5 hours, 12 hours, 1 day, 2 days & 3 days. We started with short-term forecast and forecasted 30 minutes. With 30 minutes forecast, our model displays good prediction performance by accurately predicting the behaviour of more than seventeen thousand VMs as shown in Figure 5(a). With this short-term forecast, for all the 28,858 Vms analysed, we accurately predicted 17,523 (61%) VMs. We gradually increase the length of the forecast period through a number of hours, to days as shown in Figure 5. We notice that, the shorter the forecast period,

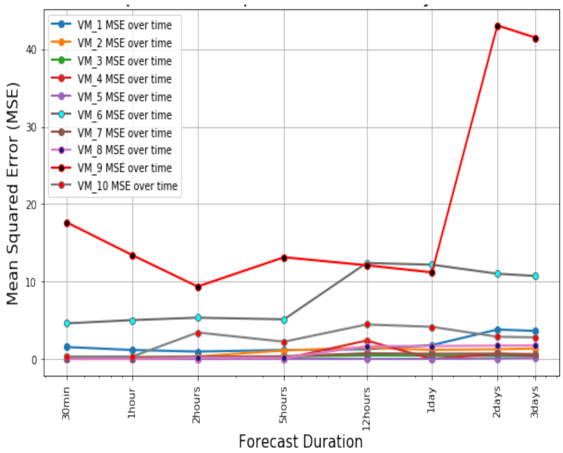
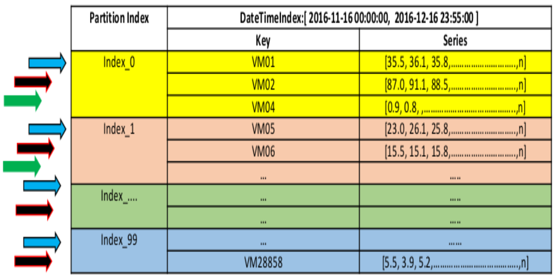


Figure 6: MSE for ten randomly selected VMs on different forecast periods

the better the prediction and vice versa as expected. Figure 6 is an MSE plot for ten randomly selected VMs. The figure clearly shows that for every VM, the shorter the forecast periods, the lower the MSE. This is because we are analysing realistic dataset and in the real world, data comes from a complex environment, and it is evolving over time. Therefore, predictions become less accurate over time (Zambon et al., 2018).

However, with a five-hour forecast, which is ideal for cloud providers, more than thirteen thousand VMs were accurately predicted (about 49% of the total VMs analysed). Experiment by (Zheng et al., 2013) reveals that in cloud/data centres, a five-hour forecast is optimal for cloud providers to take decisions such as VM migration. the Authors design a migration progress management system called ”Pacer”. Pacer manages migration by controlling the migration time of each migration process and coordinating them to finish at the desired time. With bandwidth speed of 32MBps, Pacer estimated approximately 2 hours (precisely 6500seconds) for migrating a VM of size 160GB (Zheng et al., 2013).



Partition Index	DateTimeIndex:[2016-11-16 00:00:00, 2016-12-16 23:55:00]	
	Key	Series
Index_0	VM01	[35.5, 36.1, 35.8,.........................,n]
	VM02	[87.0, 91.1, 88.5,.........................,n]
	VM04	[0.9, 0.8, ,.................................,n]
Index_1	VM05	[23.0, 26.1, 25.8,.........................,n]
	VM06	[15.5, 15.1, 15.8,.........................,n]
	...	.....
Index_....	...	.....
	...	.....
Index_99	...	......
	VM28858	[5.5, 3.9, 5.2,...............................,n]

Figure 7: Parallel and distributed processing

5.2 Fast Processing and Scalability

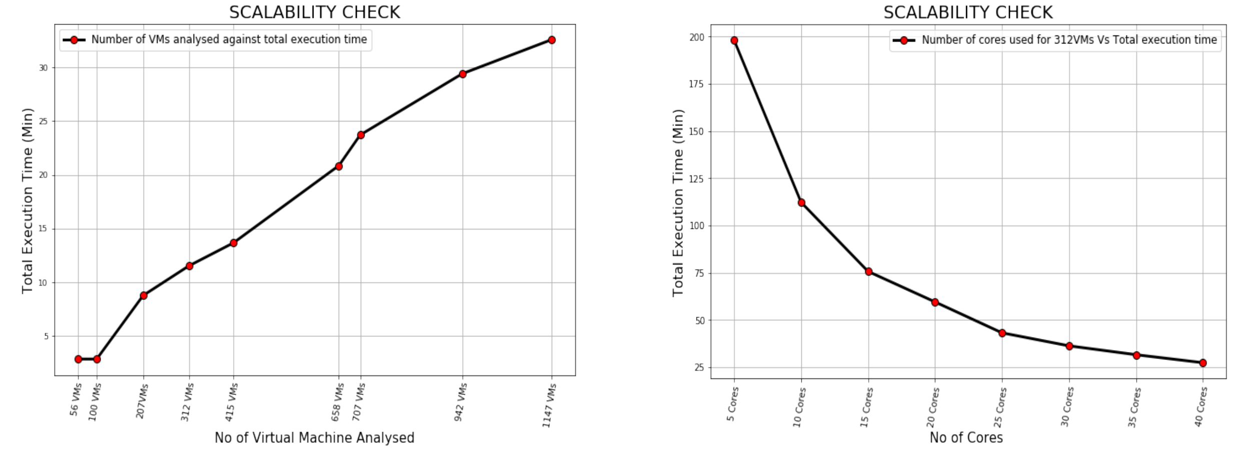
In this section, we discuss how we achieved fast processing on our framework. Furthermore, we also prove that our system is a scalable parallel and highly efficient system. As discussed in section 4.1.1, our dataset is bulky and cannot be processed by means of conventional techniques. We need to use prevailing technology to perform massive-scale and complex computing. Parallel computing models that provide the parallel and distributed data processing for the cloud providers, can accelerate the processing of large amounts of data.

5.2.1 Fast Processing

Spark uses a dispatcher which is called ”driver node” to manage jobs assigned to different distributed workers. Each job consists of several independent tasks which can be executed in parallel. If the execution fails, the driver node relaunches the task. Moreover, Spark has a speculation feature which can identify if a task is too slow and in this case, the task will be stopped and executed again (Vernik et al., 2018).

Data Partitioning in Spark helps achieve more parallelism (Zaharia et al., 2015). Although spark fit in our problem domain, it is important to note that to achieve faster processing and scalability, there is a need to induce expertise on data partitioning with parallel and distributed processing. In our experiment, we split our data (VMs) into partitions and executes computations on each partition in parallel as shown in Figure 7. The partitions are represented by yellow, brown, green and blue colours. The blue, black and green arrows indicate the first, second and third set VMs to be processed for each partition. It is important to note that, the processing is done in parallel. Insufficient partitions might lead to improper resource utilisation such as less concurrency. In Spark cluster mode, with insufficient partition, data might be skewed on a single partition and a worker node might be doing more than other worker nodes. Conversely, too many partitions may result in excessive overhead in managing many small tasks. The task scheduling may take more time than the actual execution time (Gounaris et al., 2017; Karau et al., 2015). Therefore there is a need for proper partitioning to keep our Spark computations running efficiently. Reasonable partitions can lead to utilisation of available cores in the cluster and avoid excessive overhead in managing small tasks (Petridis et al., 2016).

To achieve efficiency and fast processing, Spark documentation recommended that each partition



(a) Scalability check by varying the number of VMs and keeping the number of cores constant

(b) Scalability check by varying the number of Cores and keeping the number of VMs constant

Figure 8: Scalability Check

should hold a maximum of 128Mb of data per partition (Karau et al., 2015). Based on the teachings of (Mandal, 2019) and (Karau et al., 2015), we keep our Spark computations running efficiently by choosing the number of partitions based on the following equation.

$$\theta = \frac{\gamma}{\lambda} \tag{3}$$

Where θ represents number of partition, γ is the total input dataset size, and λ is the partition size (λ has a constant value of 128Mb).

We used Dynamic configuration mechanism (Gounaris et al., 2017) to partition our data. The number of partitions used for preprocessing was 915 partitions, this was as a result implementing equation (3) on our original data, whose size was 117GB. After filtering the long-running VMs, we dynamically re-partition our data to 140 partitions to run our predictive model on the filtered long-running VMs. With data partitioning mechanism, we achieved efficient use of resources and faster processing. In our experiment, we estimated approximately three (3)seconds for predicting each VM (from preprocessing to prediction and evaluation).

It is also important to note that, we also implemented sequential processing on a sample of 312 VMs to test the performance of our parallel processing. Irrespective of the number of cores used, it took 206.46 minutes to run (analysed and forecast) 312 VMs when the data wasn't partitioned. Conversely, It only took us 11.51 minutes with partitioning and parallelism mechanism. That is about 17.8x speed improvement, 93.28% faster than sequential processing and 1660% increase in performance.

5.2.2 Scalability

Scalability is a measure of a parallel system's capacity to increase speedup in proportion to the number of processors (Kumar et al., 1994). In our experiment, we check for scalability in two ways. At first, we keep the number of cores constant (100 cores) and varies the problem size by gradually increasing the data size. For each data size chosen, we compute the execution time to determine how long it takes to finish the job. Keeping the number of processors constant and increasing the problem size leads to a positive linear relationship (scalability) as seen in Figure 8(a). In the second method, we keep the problem size constant(sample of 312 VMs) and vary the number of cores as shown in Figure 8(b). For every chosen number of core, we run the experiment and compute the execution time to determine how long it takes to finish the job. Our experimental result shows, the execution time decreases in proportion to the increase in cores. This shows that our system is a scalable parallel and highly efficient system. Because a scalable parallel system can keep efficiency by increasing the number of processors and the problem size simultaneously. A scalable parallel system can be made cost-optimal by adjusting the number of processors and the problem size (García and Carriegos, 2019).

6 CONCLUSION

In this paper, we proposed a framework for fast analysis and prediction in large scale VM CPU utilisation. Our model is designed to quickly handle realistic large scale dataset such as Microsoft Azure

VMs traces. We processed over two million VMs from Microsoft Azure VM traces and filter out the VMs with complete one month of data which amount to 28,858VMs. The filtered VMs were subsequently used for prediction. For fast processing, we implemented our framework using Apache Spark. We partition our data and run our models in parallel to achieve high scalability. Our framework provides an efficient data processing method for large scale Virtual Machines in a cloud Settings. We use ARIMA, a statistical model for time series to predict our VMs. With short-term prediction, we accurately predicted 61% of the total 28,858VMs analysed. In term of execution time, on average, each VM is analysed and predicted in three (3) seconds.

To the best of our knowledge, we are the first to predict the CPU Utilisation for all the long-running 28,858VMs of Microsoft Azure VMs traces and accurately predicted 17,562VMs.

Since we realised that most VMs have one or several spikes of which majority of those spikes are not seasonal. Future work requires further investigation on those spikes and to find out whether they are potential anomalies.

ACKNOWLEDGEMENTS

This project was funded by Petroleum Technology Development Fund (PTDF) of Nigeria (PTDF/ED/PHD/AA/1133/17).

We thank Michael Davies and Mohseen Koohi Esfahani for comments that greatly improved the manuscript, and we thank 5 “anonymous” reviewers for their so-called insights.

REFERENCES

Aggarwal, C. C. (2018). A survey of stream clustering algorithms. In *Data Clustering*, pages 231–258. Chapman and Hall/CRC.

Ahmar, A. S., Guritno, S., Rahman, A., Minggi, I., Tiro, M. A., Aidid, M. K., Annas, S., Sutiksno, D. U., Ahmar, D. S., Ahmar, K. H., et al. (2018). Modeling data containing outliers using arima additive outlier (arima-ao). In *Journal of Physics: Conference Series*, volume 954, page 012010. IOP Publishing.

Alkatheri, S., Abbas, S., and Siddiqui, M. (2019). A comparative study of big data frameworks. *International Journal of Computer Science and Information Security,*, page 8.

Andrews, D. W. (1991). Heteroskedasticity and autocorrelation consistent covariance matrix estimation. *Econometrica: Journal of the Econometric Society*, pages 817–858.

Armbrust, M., Das, T., Davidson, A., Ghodsi, A., Or, A., Rosen, J., Stoica, I., Wendell, P., Xin, R., and Zaharia, M. (2015). Scaling spark in the real world: performance and usability. *Proceedings of the VLDB Endowment*, 8(12):1840–1843.

Assunção, M. D., Calheiros, R. N., Bianchi, S., Netto, M. A., and Buyya, R. (2015). Big data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*, 79-80:3 – 15.

Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

Brockwell, P. J. and Davis, R. A. (2016). *Introduction to time series and forecasting*. springer.

Brownlee, J. (2017). *Introduction to time series forecasting with python: how to prepare data and develop models to predict the future*. Machine Learning Mastery.

Calheiros, R. N., Masoumi, E., Ranjan, R., and Buyya, R. (2015). Workload prediction using arima model and its impact on cloud applications’ qos. *IEEE Transactions on Cloud Computing*, 3(4):449–458.

Cheboli, D., Chandola, V., and Kumar, V. (2010). Anomaly detection for time series: A survey. Technical report, Technical Report in progress, University of Minnesota, Department of ....

Comden, J., Yao, S., Chen, N., Xing, H., and Liu, Z. (2019). Online optimization in cloud resource provisioning: Predictions, regrets, and algorithms. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(1):16.

Cortez, E., Bonde, A., Muzio, A., Russinovich, M., Fontoura, M., and Bianchini, R. (2017). Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 153–167. ACM.

Fehlmann, T. and Kranich, E. (2014). Exponentially weighted moving average (ewma) prediction in the software development process. In *2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement*, pages 263–270.

García, J. F. and Carriegos, M. V. (2019). On parallel computation of centrality measures of graphs. *The Journal of Supercomputing*, 75(3):1410–1428.

Gersch, W. and Brotherton, T. (1980). Ar model prediction of time series with trends and seasonalities: A contrast with box-jenkins modeling. In *1980 19th IEEE Conference on Decision and Control including the Symposium on Adaptive Processes*, pages 988–990.

Gounaris, A., Kougka, G., Tous, R., Montes, C. T., and Torres, J. (2017). Dynamic configuration of partitioning in spark applications. *IEEE Transactions on Parallel and Distributed Systems*, 28(7):1891–1904.

Hyndman, R. J. and Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts.

Karau, H., Konwinski, A., Wendell, P., and Zaharia, M. (2015). *Learning spark: lightning-fast big data analysis*. " O’Reilly Media, Inc.".

Khan, N., Yaqoob, I., Hashem, I. A. T., Inayat, Z., Ali, M., Kamaleldin, W., Alam, M., Shiraz, M., and Gani, A. (2014). Big data: survey, technologies, opportunities, and challenges. *The Scientific World Journal*, 2014.

Kumar, J. and Singh, A. K. (2018). Workload prediction in cloud using artificial neural network and adaptive differential evolution. *Future Generation Computer Systems*, 81:41 – 52.

Kumar, V., Grama, A., Anshul, G., and Karypis, G. (1994). Introduction to parallel computing: Design and analysis of algorithms. *Benjamin/Cummings Publishing Company, Redwood City, CA*, 18:82–109.

Kune, R., Konugurthi, P. K., Agarwal, A., Chillarige, R. R., and Buyya, R. (2016). The anatomy of big data computing. *Software: Practice and Experience*, 46(1):79–105.

Li, T.-H. (2005). A hierarchical framework for modeling and forecasting web server workload. *Journal of the American Statistical Association*, 100(471):748–763.

Lingyun Yang, Foster, I., and Schopf, J. M. (2003). Homeostatic and tendency-based cpu load predictions. In *Proceedings International Parallel and Distributed Processing Symposium*, pages 9 pp.–.

Mandal, K. (2019). Comprehensive guide for tuning spark big data applications and infrastructure.

Marcu, O.-C., Costan, A., Antoniu, G., and Pérez-Hernández, M. S. (2016). Spark versus flink: Understanding performance in big data analytics frameworks. In *2016 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 433–442. IEEE.

Merrouchi, M., Skittou, M., and Gadi, T. (2018). Popular platforms for big data analytics: A survey. In *2018 International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS)*, pages 1–6.

Oussous, A., Benjelloun, F.-Z., Lahcen, A. A., and Belfkih, S. (2018). Big data technologies: A survey. *Journal of King Saud University - Computer and Information Sciences*, 30(4):431 – 448.

Petridis, P., Gounaris, A., and Torres, J. (2016). Spark parameter tuning via trial-and-error. In *INNS Conference on Big Data*, pages 226–237. Springer.

Rajaraman, V. (2016). Big data analytics. *Resonance*, 21(8):695–716.

Schmidt, F., Suri-Payer, F., Gulenko, A., Wallschläger, M., Acker, A., and Kao, O. (2018). Unsupervised anomaly event detection for vnf service monitoring using multivariate online arima. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 278–283.

Shmueli, G. and Lichtendahl Jr, K. C. (2016). *Practical Time Series Forecasting with R: A Hands-On Guide*. Axelrod Schnall Publishers.

Shvachko, K., Kuang, H., Radia, S., Chansler, R., et al. (2010). The hadoop distributed file system. In *MSST*, volume 10, pages 1–10.

Singh, D. and Reddy, C. K. (2015). A survey on platforms for big data analytics. *Journal of big data*, 2(1):8.

Vernik, G., Factor, M., Kolodner, E. K., Ofer, E., Michiardi, P., and Pace, F. (2018). Stocator: Providing high performance and fault tolerance for Apache Spark over object storage. In *CCGRID 2018, 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, May 1-4, 2018, Washington DC, USA*.

Wang, J., Yan, Y., and Guo, J. (2016). Research on the prediction model of cpu utilization based on arima-bp neural network. In *MATEC Web of Conferences*, volume 65, page 03009. EDP Sciences.

Yan-ming Yang, Hui Yu, and Zhi Sun (2017). Aircraft failure rate forecasting method based on holt-winters seasonal model. In *2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, pages 520–524.

Yu, Q., Jibin, L., and Jiang, L. (2016). An improved arima-based traffic anomaly detection algorithm for wireless sensor networks. *International Journal of Distributed Sensor Networks*, 12(1):9653230.

Zaharia, M., Das, T., Li, H., Hunter, T., Shenker, S., and Stoica, I. (2012). Discretized streams: A fault-tolerant model for scalable stream processing. Technical report.

Zaharia, M., Wendell, P., Konwinski, A., and Karau, H. (2015). Learning spark. *O'Reilly Media*.

Zambon, D., Alippi, C., and Livi, L. (2018). Concept drift and anomaly detection in graph streams. *IEEE transactions on neural networks and learning systems*, 29(11):5592–5605.

Zheng, J., Ng, T. S. E., Sripanidkulchai, K., and Liu, Z. (2013). Pacer: A progress management system for live virtual machine migration in cloud computing. *IEEE Transactions on Network and Service Management*, 10(4):369–382.

APPENDIX

The sanitised dataset for the filtered Long-running VMs can be downloaded from github repository via `https://github.com/abafo22/Filtered-long-running-Azure-VM-traces`