Bi-LSTM approach for load prediction in distributed systems

*Note: Sub-titles are not captured in Xplore and should not be used

Hettiarachchi L.S
Faculty of Computing
Srilanka Institute of Information Technology
Malabe, Srilanka
it19132310@my.sliit.lk

Jayadeva A.S.V
Faculty of Computing
Srilanka Institute of Information Technology
Malabe, Srilanka
it19139036@my.sliit.lk

Bandara G.B.M.A.G.R.A.V
Faculty of Computing
Srilanka Institute of Information Technology
Malabe, Srilanka
it19104298@my.sliit.lk

Palliyaguruge D.N
Faculty of Computing
Srilanka Institute of Information Technology
Malabe, Srilanka
it19120980@my.sliit.lk

***Abstract*— Distributed systems are one of the most commonly used software architectures in the world. The advantages such as high fault tolerance due to eliminating single-point failures, flexibility due to ease of development and deployment, and many other factors have made this the go-to architecture for developing complex multi-dimensional applications. One of the essential features of this architecture is its ability to scale individual components within the distributed system. This process of provisioning and de-provisioning of resources. commonly referred to as elasticity of the system is traditionally done manually. But orchestration frameworks such as Kubernetes have automated this process and provided APIs to achieve this easily. Yet, this rule-based process requires the user to specify threshold values for matrices which enables the system to make inferences and scale the system when the particular conditionals are met. Since the threshold values must be specified manually, based on the inferences taken from the existing data on how the system operates, it cannot be guaranteed that the system will always work at an optimal level. Therefore, a dynamic approach to predicting the optimal state of the system based on a given set of matrices such as CPU utilization and memory was required.**

The most common approach of using data gathered continuously over a period of time to make predictions on a given matrix is commonly known as time-series predictions. Time-series predictions are a sub-branch of machine learning which uses techniques to gather data over a period of time in order to make inferences on them. Artificial neural networks (ANN), recurrent neural networks (RNN) are some of the most commonly used types of neural networks to achieve this common task. While typically Auto-Regressive Integrated Moving Average is the approach that yields the lowest error for most of the time series based prediction use cases, studies have found that more modern approaches such as Long Short Term Memory (LSTMs) and Bidirectional Long Short Term Memory(Bi-LSTM) has given a lower error comparatively.

In this review paper, the literature with respect to the given study area will be discussed and all the approaches at container level, hypervisor level and orchestrator level will be discussed. Also the multiple machine learning approaches time series predictions will be discussed and compared with one another.

Keywords—LSTM, Bi-LSTM; ARIMA; Machine Learning; Timeseries predictions; Distributed systems; Kubernetes.

I. INTRODUCTION

Due to the increase in the necessity to develop multidimensional applications with many distributed components, cloud service-based computing has not only gained popularity in the IT industry but also in academia. Some of the advancements in distributed computing with respect to cloud computing that have made the development and deployment much more convenient in a production context are the virtualization technologies such as hypervisors and VMs. These allow a logical separation between the hardware upon which the system is deployed and the system itself. But due to issues such as performance overhead and complexities with redeployments and robustness of the system, container-based technologies such as Docker and Kubernetes have become extremely popular.

One of the key features that are essential to any such framework that handles distributed workloads is its ability to dynamically scale up or down based on the exerted workloads. This ability to provision and de-provision resources based on the demand to improve performance is extremely important to cloud-based distributed systems, since overprovisioning of resources may lead to the user expending monitory assets for unwanted system resources. Also, under-provisioning of resources may cause the system to perform under sub-optimal levels which may lead to bad user experiences thus hindering the market share.

There are many approaches to incorporate auto-scaling into distributed systems. This can be done at many abstraction levels. Studies have been conducted on how to dynamically scale systems at the hypervisor level, container level, and even at an orchestrator level. But one thing that is common to all these autoscaling approaches is that all of them perform the autoscaling process reactively. Meaning that the system will scale once a certain threshold or a conditional that is specified is met. For example, the Horizontal pod auto scaler API of Kubernetes, which is an orchestrator level approach that allows the user to specify conditionals which after that is met, will provide another resource. Ex: Once 90% of CPU utilization is met, the application will provision another pod. Or if response time is more than 20ms, provision another resource.

The issue associated with an approach like this is that the system will continue to perform at a suboptimal level until the resources are allocated. Since running the resource provisioning pipeline takes time, in a commercial application context, this can be extremely potent and even might cause the applications to crash. Therefore, a proactive approach was extremely necessary in order to predict the state of a system and react accordingly. This approach should enable the system to analyze the current context of the system and predict the request workload that might be exerted up to a certain accuracy and provision resources predictively instead of reactively. This should enable the system to always perform optimally given any context.

Due to the aforementioned reasons, rule-based scaling techniques are not the optimal solution for systems where finding the threshold value for metrics such as CPU utilization and memory is difficult since the workloads to the system continuously fluctuate Many studies have been therefore conducted to incorporate deep learning and AI-based technologies to achieve dynamic scalability in distributed systems. Many of these algorithms rely on time series data analysis whilst some of them use ML algorithms to achieve autoscaling capabilities. The main objective of many of these approaches is to make time-series predictions on one key metric such as the request workload and scale accordingly. In order to achieve this, many time series-based AI approaches such as ARIMA, recurrent neural networks (RNN), artificial neural networks (ANN), long short-term memory (LSTM) are used. Whilst some of these are implemented at the orchestrator level (mostly using Kubernetes) some of these are also using container-based APIs to achieve the task.

Further studies have found that bidirectional long short-term memory (Bi-LSTM) which is an extension of the regular LSTM allows making the much more accurate than the algorithms that are mentioned above. In this review paper, the requirement for a machine learning-based algorithmic approach to forecasting the load on a distributed system will be discussed and how each time series-based prediction approach based on different studies at different abstraction phases of the system will be comparatively discussed in the following sections.

II. Background

The Time series prediction approach which is a sub-branch of machine learning is commonly used in many areas where continuous data needs to be analyzed to make inferences on a particular parameter or an array of parameters. There are many algorithms such as Artificial Neural Networks (ANN), Recurrent Neural Networks (RNN) used to make these predictions. Traditional ML-based techniques such as Auto-Regressive Integrated Moving Average (ARIMA) yield the least error for most of the time series prediction instances. Therefore, this is quite commonly used in the industry. However deep learning-based techniques such as long short-term memory (LSTM) and bi-directional long short-term memory (Bi-LSTM) have yielded less error given the context of predicting the workload on a distributed system.

In this section, the algorithms that are used are discussed in-depth along with the virtualization approaches that were used in order3 to scale the systems and make predictions.

A. Machine learning

Machine learning is the subdivision of artificial intelligence which allows computer algorithms to learn based either on data or on experiences. There are mainly 2 branches of machine learning. Namely supervised learning and unsupervised learning. The applications of both these approaches are endless. Supervised learning approaches allow algorithms to build mathematical models using data sets with both inputs and the labels associated with those features. The model is trained using these combinations of features and labels and used for predictions over time. The accuracy of the data set will improve with the size of the labeled dataset. In contrast, unsupervised learning problems mainly focus on creating models where the corresponding labels are not present in the feature set. There, the computer has to identify a structure to the given set of input data and cluster accordingly. These algorithms are mainly used for clustering problems where it has to identify patterns within the dataset to form clusters. This approach of identifying the commonalities within the data or the lack of commonalities is one of the essential features of unsupervised clustering algorithms. Other than this, there are also approaches such as semi-supervised learning, reinforcement learning, dimensionality reduction and which are also quite common.

There are mainly 3 types of problems in the machine learning domain that are being solved. All the applications can be classified under one of these or a combination of these problems. Regression problem, or more profoundly regression analysis is a combination of statistical analysis methods estimating the relationship between the dependent and the independent variable. The main purpose is to build a model identifying the dependency or the lack of one between the variables. There are many regression models to best fit the given set of data. They include linear regression (which tries to model a linear relationship between the independent variable and the dependent variable) logistic regression (which models the relationship between variables as a higher degree polynomial) etc. Classification problems mainly focus on classifying the instances into 2 or more categories based on a given set of features. The model is trained to classify it to the given categories by providing a dataset with features and labels associated with them in order to train the model. Algorithms such as logistic regression are quite popular for this. Finally, clustering is the analysis approach of identifying commonalities between data to categorize them together. The number of clusters and how they are clustered depends mainly on the algorithm that is being used. Algorithms such as k-mean are quite popular.

Although most of the machine learning problems fall under the above-mentioned categories, the requirement of load prediction in a distributed system mainly falls under time series analysis. This requires a particular parameter (in this case the workload or the number of requests on the components over time) or a combination of parameters to be predicted using a set of features. The data are gathered continuously on a timely basis. Hence the name, time series analysis. The problem mainly takes in time series data inputs such as CPU utilization, memory utilization, and the context of the current system and has to make predictions based on the inputs for the predicted workload.

B. *Deep learning*

Whilst ML leverages structured and labeled data to train the algorithms in order to make predictions, deep learning is a technique that eliminates the requirement of data to be preprocessed. A more profound definition can be given as the subset of machine learning with artificial neural networks with more than 3 layers. These neural networks mainly mimic the behavior of the human brain where it learns over time. What is essentially meant by ANN learning is that it tries to minimize the error associated with the problem. In other words, it uses techniques such as gradient descent algorithm and backpropagation to find the local minima.

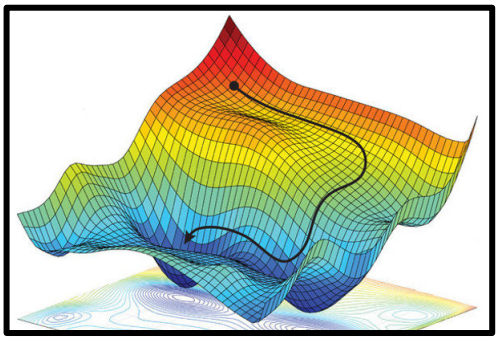


Figure II-1: Gradient decent algorithm in 3d

The gradient descent algorithm is mainly focused on finding the minima of a given cost function of a problem. The algorithm chooses a random starting point arbitrarily and decreases the value for the cost function in each step until a local minimum is reached. Finding this local minimum does not always guarantee that the global minimum is found. In mathematical terms, the gradient descent can be expressed as,

$$a_{n+1} = a_n - \alpha \frac{\partial a_n}{\partial x} \quad (1)$$

$a(x, y, z \ldots.)$ can be a function of multiple variables which is the cost function associated with the given scenario. The objective is to minimize the value of the cost function. In order to do that, a difference equation is used where a_{n+1} is the next iterative value calculated using a_n which is the current value for the iteration. $\frac{\partial a_n}{\partial x}$ is the partial differential of the function with respect to a given single variable whilst α is the step size. Although α is expressed as a constant value, in this case, the step size can also be expressed as a function of the variables such that the step size decreases as the value is reaching the local minima so that the overshooting problem is avoided. This only minimizes the cost function with respect to one single independent variable(x). This can be repeated across multiple variables to find the least cost function.

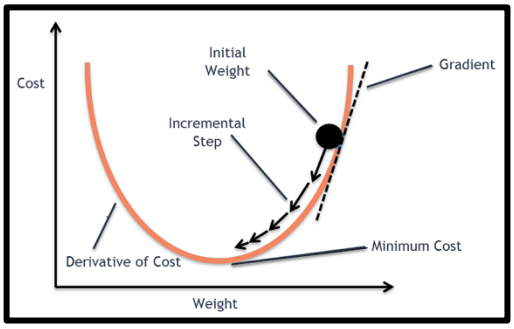


Figure II-2: Operation of gradient decent algorithm

In feed-forward artificial neural networks, the backpropagation algorithm is responsible for increasing the accuracy of the net through a method known as the chain rule. It iteratively learns the weights that should be allocated by adjusting the model parameters (weights and biases) after each forward pass.

Different types of neural networks (NN) are common for predicting workloads and for doing time series analysis. But in order to understand them firstly, lets understand a simple feed-forward NN,

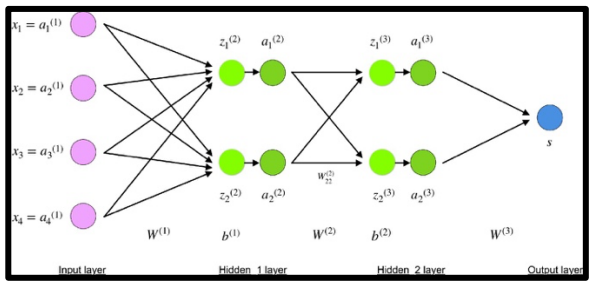


Figure II-3:Architecture of a NN

Neural networks mostly consist of 3 types of layers. The input layer with vectors or multidimensional matrices with activation levels a. More generally this can be notated as

$$x_i = a_i^{(j)} \quad i, j \in 1,2,3,4 \ldots \quad (2)$$

Simply, the activation level of the x^{th} node can be given by the element corresponding to the i^{th} row element of the j^{th} row in the multidimensional vector. These activation levels are taken as inputs to further hidden layers.

Hidden layers are located between the input layer and the output layer. This is what essentially transforms the input vector into the output vector in which a nonlinear transformation is performed by each layer. The number of hidden layers and the nodes in each layer change from problem to problem. The activation of all the hidden layers will be fed forward to the following layers. The activation of each subsequent layer directly depends on the layer before that and the weights associated with it. The activation of the hidden layers are denoted as $z_i^{(j)}$ and the weights are denoted by $w_i^{(j)}$.

These weights and the biases are tweaked in order to reduce the cost function such that the error is reduced below a threshold level. The method used to determine how to tweak the parameters of the NN such that the cost function is reduced is known as the backpropagation approach. The activation of a node in the proceeding layer can be denoted by,

$$z_i = \sigma(w_1 a_1 + w_2 a_2 + \cdots + w_n a_n) \quad (3)$$

Where w_n are the weights for all the edges connected to the node that the value is predicted while is the activations for the corresponding nodes. σ is the activation function such as the sigmoid squishification function.

Whilst traditional neural networks are vastly used in many practical applications, as discussed in further sections there are better alternatives to perform time-series predictions.

C. Time series analysis

Obtaining data periodically over a sequential period of time in order to identify trends is a quite common practice in machine learning. The analysis of data in order to create stationery and nonstationary models by identifying trends is commonly referred to as time series analysis. There are many algorithms that are commonly used to achieve time-series predictions. AR, MA, ARMA, and ARIMA are a few of them.

1) AR model (Autoregressive model)

Autoregressive model predicts the future values based on the past values for the prediction variable. If Y_t is the predicted value, It can be represented as a function of the past values of Y and the error term ϵ.

$$Y_t = f(Y_{t-1}, Y_{t-2}, Y_{t-3}, \dots, \epsilon)$$

More discretely, this can be expressed as AR(p) where p is the parameter of the number of past value terms that the current value depends on,

$$Y_t = \beta_0 + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \beta_3 Y_{t-3}, \dots + \epsilon$$

2) MA model (Moving Average model)

Moving average model is when the forecasts are made only based on the error terms.

$$Y_t = f(\epsilon_t, \epsilon_{t-1}, \epsilon_{t-1}, \dots)$$

More discretely, this can be expressed as MA(q) where q is the parameter of the number of past value terms that the current value depends on,

$$Y_t = \beta_0 + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + \phi_3 \epsilon_{t-3}, \dots$$

3) ARMA (Autoregressive Moving Average models)

Autoregressive moving average model predicts the future values based on the past values and the error terms associated with them. The model for the ARMA approach can be given as,

$$Y_t = \beta_0 + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \beta_3 Y_{t-3}, \dots + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + \phi_3 \epsilon_{t-3}, \dots$$

4) ARIMA (Autoregressive Integrated Moving Average)

Autoregressive integrated moving average is also another method to perform statistical analysis on time series data to understand the current context of the system and predict future trends. Like in regression analysis ARIMA gauges the dependency of the prediction variable when other variables are being changed. But these show proficiency in predicting when the data shows evidence of non-stationarity of the mean function. The seasonal model of the ARIMA model can be denoted as ARIMA(p,d,q) where p is the no of time periods in the auto regressive model while q is the order of the moving average model. d can be denoted as the degree of differencing (the number of substations from the previous values).

Whilst all of the above-mentioned models can be used to predict time series data, the ARIMA model is most commonly used in the industry. In the future sections, the use of the ARIMA model to make predictions and how it will hold up against models such as LSTMs will be discussed in the future sections.

D. LSTM

Long short-term memory is a recurrent neural network architecture. These are quite popular in deep learning study areas. But unlike the feed forward neural networks, long short-term memory utilizes a feedback connection to improve learning. Information persistence, which is one of the most key factors in LSTMs is what mainly differentiates them from feedforward networks. Also, this tackles one of the most potent problems in vanilla RNNs which is the vanishing gradient [36,37] (The effect of the gradient value approaching zero as it back propagates through time) problem. Gradient is the value used to update the neural networks weight.

Many classification and regression problems are solved using recurrent neural networks. RNN's excel at providing elegant solutions for dealing with sequential data.

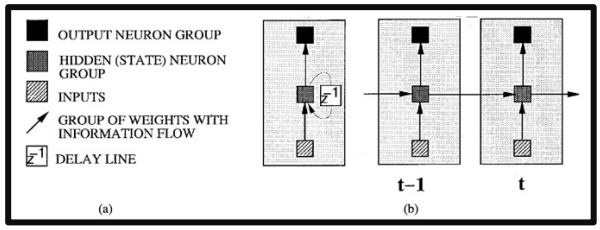


Figure II-4:Architecture of a RNN

Above Figure shows the basic architecture behind a recurrent neural network with a delay path. The input vector X_t is fed to the system sequentially. Unlike Multilayer perceptron (MLP) and Time-delay neural networks (TDNN) where the input number is fixed. This architecture can utilize all the available inputs in the vector to predict the value of Y_t.

$$\{ X_t, t = 1,2, \dots , t_c \}$$

The amount of information captured by the RNN depends on the training algorithms used and the structure of it.

E. Bidirectional Recurrent Neural networks

Bidirectional neural networks (BRNN) helps to overcome the limitations of regular RNNs such as the issues with the optimality of merging network outputs due to different networks not being independent in linear opinion pooling and logarithmic opinion pooling etc. Bidirectional neural networks encapsulate both past and future input information and have the ability to be trained using all available input information specific for a given time frame.

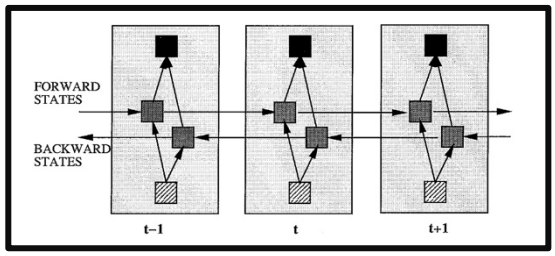


Figure II-5:Architecture of a Bi-LSTM

The architecture behind the BRNN is to split the neurons into 2 sections with divided responsibilities. The sections are responsible for the positive time direction (forward state) and negative time direction (backward state) of the RNN. The outputs of the forward state and the backward state are

independent of one another. The main advantage of BRNNs is that take care of input information from the past and the future concurrently. This is done by accounting for the information in both the forward state and backward state and directly using it to minimize the cost function. The operation of a BRNN can be expressed rotationally as

$$h_t^F = f(U_F m_t + W_F h_{t-1}^F + b_F) \quad (4)$$

$$h_t^B = f(U_B m_t + W_B h_{t-1}^B + b_B) \quad (5)$$

$$\hat{y}_t = V_F h_t^F + V_B h_t^B + b_O \quad (6)$$

Where U_F and U_B, W_F and W_B, V_F and V_B are weights of the matrices while the function f is known as the activation function , usually tanh sigmoid or logistic sigmoid. b_F, b_B, b_o are biases associated with each instance. The purpose of the BiLSTM is to generate the forward hidden sequence and the backward hidden sequence h_t^F , h_t^B respectively given the input sequence to the network as $m = (m_1, \dots, m_T)$. The output layer should also be updated by calculating the output sequence $\hat{y}$ by iterating through $t = 1..T$ in forward direction and $t = T..1$ in backward direction.

III. LITERATURE SURVEY

In the previous section, the background algorithms, and methods behind sequential data prediction (time series) that are used in predicting the network load on a distributed system were discussed extensively. In this section, the studies that are conducted in the domain in order to optimize performance in a distributed system through time-series load prediction will be discussed.

Cloud-based distributed computing is one of the most common computer architectures in the industry. Kubernetes is one of the most widespread container orchestration frameworks with almost a 51% market cap according to a study performed by data dog. [2] proposes an architecture for a custom auto scalar using deep neural networks trained for handling dynamic workloads in distributed environments, Unlike the reactive solutions proposed before the aforementioned study, this approach applies Bidirectional Long Short-term memory to predict the http workload for the future and scale accordingly. Since this approach is dynamic, it allows the system to both scales up and scale down based on the requirements.

The paper further points out that, contrasting to approaches such as RNN, ANN, LSTMs, and Bidirectional Long short-term predictions may yield better accuracy since they can preserve both future and past information for the time series data. The paper contributes majorly to solidifying that bidirectional LSTMs have better accuracy from the results of the experiments by comparing systems under real trace workload datasets. The system uses a design pattern called monitor-analyze-plan-execute (MAPE) for creating the system architecture.

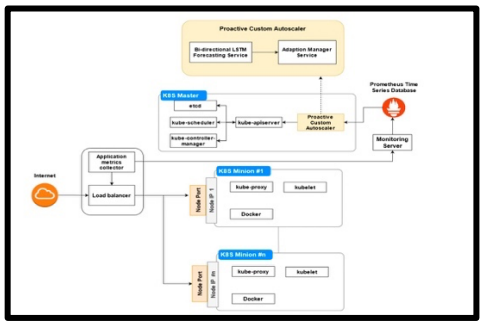


Figure III-1:Software architecture of Bi-LSTM based autoscaler

The system is comprised of 4 main components [2]. The application matric collector (responsible for monitoring the load balancing and collecting application metrics), the monitoring server (responsible for persisting the time series data gathered and exposing them via an API) [1], K8 master/minion, and the proactive custom autoscaler. The MAPE design pattern is used in this.

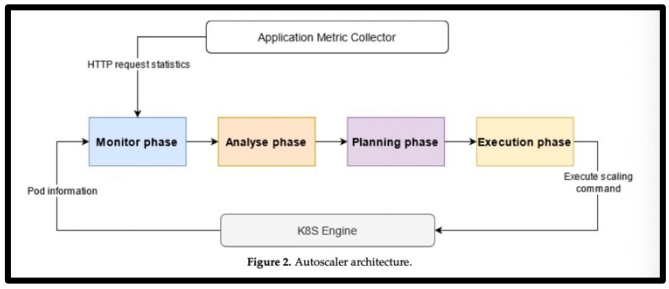


Figure 2. Autoscaler architecture.

The autoscaler consists of 4 main phases of operation. In the monitoring phase, the system gathers time-series data continuously for the BRNN. These are the time series inputs which include the number of HTTP requests per second through the application, pod information such as the number of replicas, CP, memory utilization of each replica, etc.

The analysis phase is the most important phase for the pod autoscaler. In this phase, the sequential data that is gathered in the subsequent phase is used to make time-series predictions about the workload. The Bi-LSTM obtains the continuous metrics collected using the API exposed by the monitoring server. The data $d_{t-w}, .., d_{t-2}, d_{t-1}, d_t$ is used to predict the next data in the sequence where w is the window size. In this instance, the Bi-LSTM is used to predict the workload of HTTP requests in the future.

The proposed Bi-LSTM in study [2] that is used to predict HTTP workload consists of 2 layers that move forward and backward. The inputs for the corresponding layers are given by the input layer with 10 neural cells for the last 10-time steps. The hidden layers consists of 30 hidden units. A concatenation layer that is responsible for connecting the outputs of the forward and the backward LSTMs are then used and the output of the concatenation layer is connected to the dense layer. The ReLU activation function is used in the hidden layers.

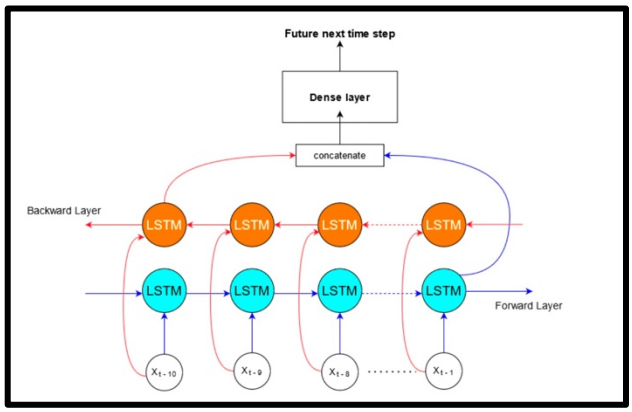


Figure III-2:Bi-LSTM model of the autoscaler

The study further discusses how the Bi-LSTM was evaluated for its accuracy and speed of prediction compared to traditional LSTM [12] and ARIMA in [6,7] by cross-referencing the network against several datasets. Some of the datasets were the web server [27] web server logs and the server logs over a 3-month period time for the FIFA World CUP 98 web server. The 2 data sets were split 70% and 30% and were used for training and testing respectively for each of the LSTM, ARIMA, and Bi-LSTM models. There a 10 min workload was used to predict the workload of the next minute. Several evaluation metrices such as Mean Sum Error(MSE) , Root Mean Square Error(RMSE) , Mean Absolute Error (MAE) and the coefficient of determination were proposed to evaluate the models. The results were as follows

Model Type	ARIMA 1 Step	BI-LSTM 1 Step	ARIMA 5 Steps	BI-LSTM 5 Steps
MSE	196.288	**183.642**	237.604	**207.313**
RMSE	14.010	**13.551**	15.414	**14.39**
MAE	10.572	**10.280**	11.628	**10.592**
R^2	0.692	**0.712**	0.628	**0.675**
Prediction speed (ms)	2300	**4.3**	2488	**45.1**

Bold values indicate the best results.

From this, it is possible to conclude that the prediction speed is significantly higher and the error of MSE, RMSE,MAE are comparatively lower in almost all the considered cases when the number of steps in the Bi-LSTM increases. Through this, it the authors of the study argue that the Bi-LSTM is more suitable for estimating perditions in the Kubernetes cluster than statistical ARIMA and feed-forward LSTMs. The error of one-step predictions is reduced by 3.28% and the error of 5-step predictions is reduced by 6.64% for Bi-LSTMs compared to ARIMA models based on the RMSE metric.

Although ARIMA model and LSTM based models are inferior in most use cases for load prediction in most of the use cases, [6] argues how LSTM enabled dynamic Stackelberg game-theoretic method can be used for resource allocation in cloud-based environments. In this paper, the author proposes how a cloud resource allocation model can be developed using time series data corresponding to the cloud infrastructure matrices and the load at time t to predict the corresponding load at time $t + 1$.

The previously discussed solutions mainly focused on implementing a load prediction approach based on cluster matrices at an orchestrator level. [2] proposes a mechanism to implement an autoscaling mechanism at a container level. It manages to tackle the fluctuating loads at small time scales and evaluates the provided container level matrices to determine the most important factors for the system, therefore identifying possible bottlenecks. The system also utilizes the capabilities of Bi-LSTMs to achieve load prediction. This study discusses how traditional time series-based prediction models such as ARIMA[4-6] and algorithms like Hidden Markov Model[7-9] fail to encapsulate the fluctuating trends in time series and how they assume the future pattern of time series data will follow the past patterns. It further elaborates how resource requirements of the container workloads can fluctuate drastically and how Bi-LSTMs manage to encapsulate this.

The solution implements a metrics selection module (MSM) that is responsible for reducing the dimensionality of the metrics by understanding the patterns between the metrics and filtering out useless ones and a Bi-LSTM to make predictions

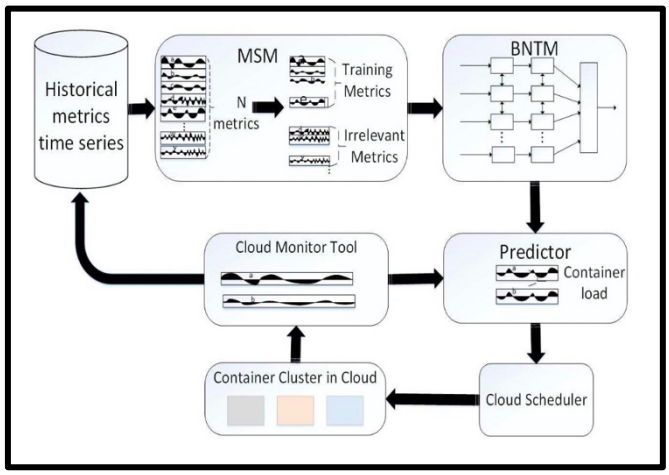
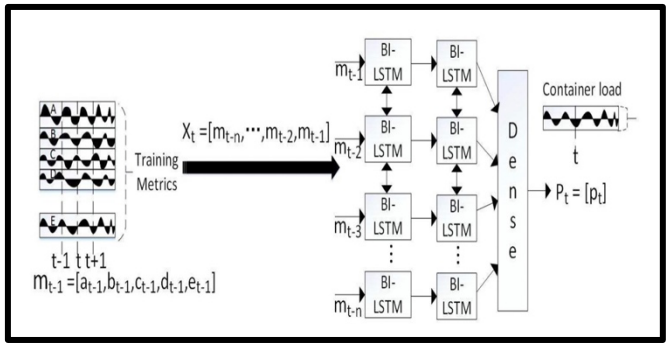


Figure III-3:Software architecture of the BRNN based autoscaler

The working process of the fisher's approach queries historical metrics of time series to train the MSM and understand the most important features. The features are categorized as training metrics and irrelevant metrics. Only the training metrics are used which will improve the efficiency and the accuracy of the predictions made by the Bi-LSTM while reducing the cost of monitoring unwanted matrices. According to the study, the system uses a k-shape (a novel algorithm that can be used for shape-based time series clustering) [3]

After the training matrices are identified a module called BNTM is trained using the identified matrices. Since Bidirectional LSTM networks are capable of processing information both forward and backward using 2 separate hidden layers, a higher accuracy can be acquired.



Similar to [2] the input vector of X_t is given as the input neuron layer to the Bi-LSTMs. After transformation through

several hidden layers, the outputs are combined together in a dense layer to make the prediction vector for the container workload P_t.

The study also dives into the important feature vector that is identified as the inputs for the Bi-LSTM. Using the k-shape clustering algorithm, the following set of features were identified as the important features for each metric.

Metric type	Total Metrics	Selected Metrics	Detail of Selected Metrics
CPU	7	2	cpu_usage_seconds_total, cpu_busy
Memory	27	4	memory_active_file, memory_hierarchical_memory_limit, memory_rss, mem_memused
Disk	6	2	blkio_service_bytes_Async, blkio_service_bytes_Read
Network	19	2	network_tcp_TimeWait, net_if_in_bytes
Total	59	10	

From the above feature set the model was evaluated in the study against other prediction methodologies and it was conclude that this approach was the most accurate.

TABLE II

RMSE ERRORS FOR THE THREE MODELS ON EACH TYPE OF SERVICE

Application type	ARIMA	LSTM	Fisher
Apache load	0.56	0.33	0.11
Nginx load	0.35	0.15	0.09
Mysql load	0.31	0.32	0.15
Redis load	0.23	0.22	0.1
MongoDB load	0.20	0.06	0.06
Average RMSE	**0.33**	**0.21**	**0.1**

A similar approach to this is provided in [5] where matrices such as CPU utilization and RTT is used for load prediction in data centers using LSTM based prediction models to optimize resource utilization.

IV. Conclusion

From the research studies conducted in the domain, it is quite apparent that there are mainly 3 most common approaches for time series-based load predictions in distributed systems. They are mainly ARIMA, feed-forward RNN, and Bi-LSTMs. Among them, the Bi-LSTM model is the most accurate and efficient in most use cases. This ability to accurately predict the workload is extremely important in distributed systems where overprovisioning of resources may lead to the unwanted cost being associated with resources while underprovisioning may result in the system working at a sub-optimal level. With the accuracy increase, the system is possible to build auto scalers that dynamically scale the system up and down based on the expected workload.

The main reason behind this accuracy is that compared to the feed-forward RNNs where time-series information from the past is used to predict future values in a forward direction, Bii-LSTMs process time-series information in both directions and account for the past data and future data which makes it highly accurate for sequential data predictions.

V. Improvement

The existing proposed solutions only consider the metrics either at an orchestrator level or a cluster level to make time-series predictions for the workload. But in essence, a combination of them plays a role in the performance hindrance of the system. Therefore it is important that a method that collectively takes into account all the metrics at all abstraction levels of virtualization is taken into account before predictions are made.

As discussed under [2] an MSM component can be extremely important to any prediction system which uses categorization algorithms such as k-shape to identify the most important features for the prediction algorithm. This logic can be extended further to identify the training features not only in container level metrics but also in metrics gathered at a cluster level to create the input vector.

Further, a proactive mechanism for the auto scaler to dynamically scale up the clusters can be proposed since the bidirectional LSTM uses a width of w and identifies past trends and future tend to predict the values which helps in accurately predicting the load throughout the operation time.

Also, it was observed that in some scenarios, as the number of steps is lower, the ARIMA-based models yield high accuracy than LSTM based approaches. Therefore, a mechanism can be implemented to predict the workload using a combination of the models such that the accuracy is always high in all circumstances.

VI. References

[1] M. Schuster and K. Paliwal, "idirectional recurrent neural networks.," *IEEE tans.*

[2] X. Tang, L. Qiuyang, Y. Dong, J. Han and Z. Zhang, "Fisher: An Efficient Container Load Prediction Model with Deep Neural Network in Clouds," 2018.

[3] J. Paparrizos and L. Gravano, " k-Shape:Efficient and Accurate Clustering of Time Series".

[4] S. S. Namini, N. Tavakoli and A. S. Namin, "The Performance of LSTM and BiLSTM in Forecasting Time Series".

[5] T. Kundjanasith, K. Ichikawa, K. Takahashi and H. Iida, "Improving Resource Utilization in Data Centers using an LSTM-based Prediction Model," 2015.

[6] Y. Liu, L. L. Njilla, J. Wang and H. Song, "An LSTM Enabled Dynamic Stackelberg Game Theoretic Method for Resource Allocation in the Cloud".

[7] "How to Develop LSTM Models for Time Series Forecasting.," [Online]. Available: https://machinelearningmastery.com/how-to-develop-lstm-models-for-time-series-forecasting/. [Accessed 22 Mar 2022].

[8] H. Fernandez, G. Pierre and K. Kielmann, "Autoscaling Web Applications in Heterogeneous Cloud Infrastructures".

[9] H. Ciptaningtyas, B. Santoso and M. Razi, "Resource elasticity controller for Docker-based web applications.".

[10] M. Imdoukh, L. Ahmad and M. Alfailakawi, " 16. Imdoukh, M.; Ahmad, I.; Alfailakawi, M.G. Machine learning-based auto-scaling for containerized applications," *Neural Comput. Appl.*

[11] "Two Month's Worth of All HTTP Requests to the NASA Kennedy Space Center," [Online]. Available: fttp://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html. [Accessed 22 Mar 2022].

[12] R. Calheiros, E. Masoumi, R. Ranjan and R. Buyya, "Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications".

[13] N.-M. Dang-Quang and M. Yoo, "Deep Learning-Based Autoscaling Using Bidirectional Long Short-Term Memory for Kubernetes".