ISSN 2063-5346*Section A-Research paper*

SLA-Aware Predictive Autoscaling for Containerized Microservices

Hirenkumar Ramanbhai Patel
Research Scholar, Computer/IT Engineering, Gujarat Technological University, Chandkheda, Ahmedabad, Gujarat, India.Email: scethiren@gmail.com
Dr. Tejas P. Patalia
Head & Professor, Computer Engineering Department,VVP Engineering College, Rajkot, Gujarat, India.Email: pataliatejas@rediffmail.com

ABSTRACT

Cloud Elasticity can be used for the provisioning and de-provisioning of cloud resources to match incoming workloads with cloud resources in an automatic manner. Elasticity can be achieved using auto-scaling. In literature, major auto scalers discussed can only handle one microservice but a case of multi-service was discussed in a limited fashion. In this paper, we proposed a proactive auto-scaling algorithm to manage the auto-scaling of multi-microservices-based application. compared to the baseline auto scaler (which is a reactive approach and auto scale containers based on the response time of application) our proposed work improves SLA achievement, reduces request failures, and also quickly reacts during heavy incoming workloads.

Keywords: Cloud Computing, Autoscaling, Microservices, Containers, Service Level Agreement, Docker Swarm

1.0 Introduction

Cloud computing is a new paradigm that is widely used to deploy and manage applications[1]. Cloud service can beavailed as Iaas (Infrastructure as a service), Paas (Platform as a service), and Saas (Software as a service). Pay-as-you-use / Pay-as-you-go and on-demand resource provisioning are the most interesting features of cloud computing which attracts established organizations and startups to adopt cloud computing. It helps organizations to provide services to their clients and also to reduce infrastructure operational costs. It is a feature to pay infrastructure bills/costs based on resource usage. On-demand resource provisioning helps cloud users to get/use extra resources as and when required by the cloud service provider.

Resource utilization costs can be minimized by the cloud service user by managing the cloud resources based on their usage. Cloud resources can be managed by considering peak workload or average workload. if resources are fixed based on peak workload, then during average workload resources may be underutilized and result in unnecessary over cost to service users. If resources are fixed based on the average workload then in case of peak workload resources may be overutilized and results in client dissatisfaction[2].

Service level agreement (SLA) can be defined as a mutual contract between the service provider and consumer, which determines the agreed service level objective (SLO)[3]. Service level agreement is very important to pursue a profitable business relationship between the service user and the service provider. Common performance SLOs include availability, response time, capacity, etc.[4].

Cloud elasticity is a very useful feature of cloud computing that handles the provisioning and de-provisioning of cloud resources according to the incoming workload and tries to map resources with the incoming workload. Elasticity can be achieved using auto scaling[5].

ISSN 2063-5346Section A-Research paper

Auto-scaling is a mechanism to add or remove resources based on certain criteria or conditions. Auto-scaling decisions can be taken based on infrastructure-level metrics like CPU and memory utilization or based onapplication-level metrics like application response time. Auto-scaling approaches can be classified into reactive approaches and proactive approaches.

The following listed statistics motivated us to work on SLA-aware resource management considering the case of multiple microservices-based applications.

- Microsoft Bing found that increasing page load time by a mere 1-2 seconds had a dramatic impact on both user satisfaction and revenue per user.

Page Load Time	User Satisfactions	Revenue/User
1000ms	-1.6%	-2.3%
2000ms	-3.8%	-4.3%

- Making the Barack Obama Campaign site 60% faster-increased donation conversion by 14%.
- Walmart noted that for every 1 second of improvement, they experienced a 2% increase in conversions for every 100ms of improvement, they grew incremental revenue by 1%.
- Amazon also experienced the same 1% in increased revenue for every 100ms of improvement. On the flip side, a 1-second loss for Amazon is about $1.6 billion in sales a year.
- Shopzilla sped up its average page load time from 6 seconds to 1.2 seconds and increased revenue by 12% and page views by 25%.
- Firefox reduced its average load time by 2.2 seconds and increased downloads by 15.4% or 60 million more downloads a year.
- Google showed that showing their search engine results by 400 milliseconds reduced the number of searches by 8000,000 per day.
- Yahoo increased traffic by 9% for every 400 milliseconds of improvement [27].

2.0 Back ground

2.1 Docker

It is an open-source tool that provides an IaaS Platform for executing client applications. Docker consists of all the required software components to run an application. It provides an isolated environment between containers and provides a namespace to run applications [6].

Docker container consists of three components docker-daemon, docker-client, and docker-hub[7]. Docker-client is used to send related commands to docker-daemon or docker API.

2.2 Container

Container provides operating system-level virtualization. It contains an executable software package that includes user code, system tools, runtime environment, and system libraries. the developer may create and run the code inside the container along with its dependency [8]. Cloud providers use existing infrastructure and use Virtual Machines to run containers on it.

Auto-scaling problem is a classical automatic control problem that requires a controller to dynamically manage the number of resources allocated to reach a certain goal[9].

In this paper, auto-scaler is used to manage several containers to achieve the SLA Parameter which is response time. MAPE loop was proposed first by IBM for architecting a self-adaptive system which is

widely used in the autonomic system[10]. It contains a loop of four steps namely Monitor(M), Analyze(A), Plan(P), and Execute(E). these steps continuously repeat themselves over time. All steps significantly impact the auto-scaler efficiency. We explained all the steps briefly and also discussed the key challenges of auto-scaler designers in particular steps.

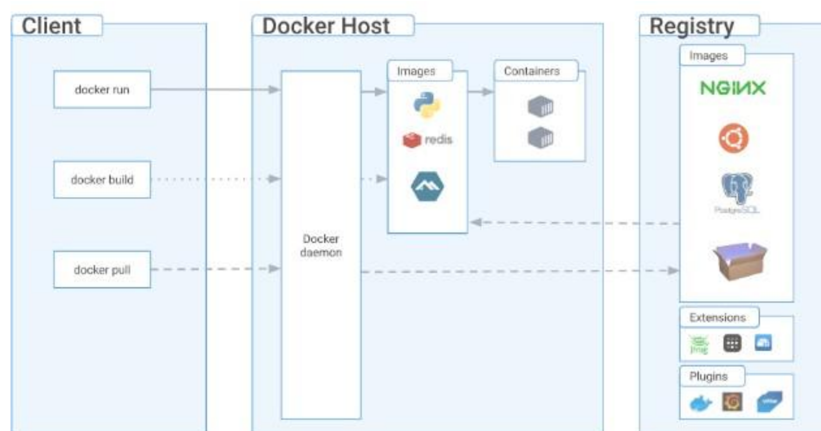

Figure 1 Architecture of Docker.

Monitoring (M): In this step monitoring information for scaling like nodes/containers resource utilization, HTTP requests rate, and application response time is collected to make scaling decisions. The key information is monitored from low-level metrics at the node/container level like CPU utilization, and memory utilization, and/or high-level metrics at the application level like request rate and average response time, etc. challenge here is to use suitable metrics for auto-scaling decisions.

Analysis (A): In the analysis step, an analysis of the present status of the system is performed and a decision about scaling operations is taken. Key decisions about scaling time, Oscillation mitigation, and adaptivity are made during this step.

Planning(P): With respect to the Analysis step the decision regarding the auto-scaling is planned in this step. Based on the present or future workload available from the analysis step, the planning step estimates the number of resources to be provisioned or de-provisioned in the next scaling operation to meet SLA agreement and to minimize utilization cost. Decision about resource estimation and scaling methods are made during this step.

Execution(E): during this step planned scaling operations are executed where auto-scaler uses cloud provider's API to execute scaling commands.

3.0 Related Work

In this section we review the recent studies related to container auto-scaling using reactive and proactive method.

Mahendra Pratap,Rohit Yadav, Dharmendra Kumar [2] proposed an auto-scaler that uses proactive approach to addresses the issue of optimal resource management to fulfil auto-scaling strategies of Kubernetes, oracle cloud, google cloud, Microsoft Windows Azure and Amazon EC2.experiment was performed using docker swarm tool. Used HA-Proxy for load balancing, used Node.js based containerized application. for workload used actual web service logs from the Complutense University of Madrid which contains time-stamp and number of hits. Used Support Vector Regression based on RBF model and IBM MAPE-K principle to predict incoming workload to perform horizontal elasticity for docker containers. Prediction accuracy of SVR-RBF was compared with ARIMA and LR using metrics like MAE and

ISSN 2063-5346 *Section A-Research paper*

RMSE. Auto scaler uses predicted workload to find minimum number of containers to handle future workload. the experimental results show that SVM prediction keep the performance of the system sustainable with fluctuating workload.

Mahendra Pratap , Pal, Nisha Yadav, Dharmendra Kumar [11]proposed reactive hybrid auto scaler that uses container-based virtualization for allocating computing resources to running container. Proposed approach is beneficial to both provider and consumer as it provides better resource utilization and minimizes subscription costs. For scaling decisions infrastructure level metrics CPU utilization and Memory utilization of container was used. Experiment was performed using docker swarm tool and HA-Proxy as a load balancer. For workload generation Apache bench tool was used. Some of the limitations of the proposed work are i) it not predicts the computing resources in advance ii) it does not perform live migration iii) assumed that sufficient number of resources are available iv) workload generated using dummy tool. Some of the advantages of the proposed work are i) it utilizes the resources in fine-grained fashion, ii) improve the response time ii) consume minimum number of resources in comparison to other available approaches. The experimental results show that as compared with horizontal elasticity, vertical elasticity and docker the proposed model dynamically allocates resources to applications instantly and maintains higher resources utilization.

Abdullah, Muhammad Iqbal, Waheed Berral, Josep Lluis Polo, Jorda Carrera, David [12] proposed a novel burst aware auto-scaling method to minimize response time that detects burst in dynamic workload using workload forecasting. it performs resource prediction and take scaling decision accordingly. Experiment was performed using docker swarm. Used Elastic Net Regression for workload prediction and Decision Tree Regression for resource prediction. Evaluated the proposed approach using trace driven simulation using World cup 98, Wikipedia, Nasa, clarinet server logs with load generation tool httperf for containerized microservice. Experiment results show that in proposed method there is an increase of x1.09 in total processed requests, a reduction of x5.17 for SLO violations and an increase of x.0767 cost with compared to baseline methods.

Coulson, Nathan Cruz Sotiriadis, Stelios Bessis, Nik [13] proposed a prototype auto-scaling system for microservice based web application that learns from past service experience. in reviewed auto scalers a key concern is the decision on which microservice to scale to increase performance. proposed system addressed a problem of selecting bottleneck microservice among the microservices. Contribution of the work can be divided into two parts i) developing a pipeline for microservice auto-scaling and ii) hybrid sequence evaluation and supervised learning model for recommending scaling actions. Researchers have used docker and docker swarm container technologies, NGINX load balancer to log requests and responses, Locust as a workload generator. For workload prediction used stacked LSTM supervised learning. The result of a hybrid model shows the importance of using a supervised model for selecting microservice for scaling.

Yadav, Mahendra Pratap Raj, Gaurav Akarte, Harishchandra A. Yadav, Dharmendra Kumar[1] proposed a hybrid auto scaling approach which uses both reactive and proactive mechanism. Researchers have used docker swarm for simulation, Tsung benchmarking tool for workload generation, HAProxy as a load balancer. used both approaches to find number of containers, in reactive approach used overload threshold to count number of containers while is proactive approach used workload prediction and from that predicted workload it counted required number of containers. Reactive approach was used for scaling out container while workload increases and proactive approach was used for scaling in containers (using SVR) while workload decreases. Compared number of containers corresponding to workload. The

ISSN 2063-5346 Section A-Research paper

experimental results show that the performance of the system has improvement in terms of CPU Utilization, response time and throughput.

Yang, Yanan Zhao, Laiping Li, Zhigang Nie, Lihai Chen, Peiqi Li, Keqiu [14] presented an online service manager named Elax which minimizes the resource provisioning cost and guaranteeing tail latency requirement for containerized online services. It uses workload aware resource allocation mechanism using the collaboration of three key components 1) workload predictor 2) resource reservation 3) controller, workload predictor predicts the workload in periodic scenario using LSTM network, resource reservation uses predicted workload to allocate right number of resources using scale up and scale down operations. Controller uses feedback-based control method for tail latency requirement and also reduce resource provisioning cost by resource reclamation. Used ClarkNet and Calgary production workloads for request generation. Compared results with existing approaches or Peak, PRESS, EFRA , Kubernetes. Experimental results show that Elax can reduce the average resource over-provisioning cost by more than 32.6%.

Dang-Quang, Nhat Minh Yoo, Myungsik [15] proposed system architecture based on Kubernetes with a proactive custom auto-scaler which uses deep neural network model Bi-LSTM. Proposed system architecture uses MAPE loop. Main focus of the research was on analysis and planning phases. In analysis phase used Bi-LSTM for HTTP workload prediction and in Planning phase a cooling-down period was implemented for oscillation problem mitigation, also resource removal strategy was proposed to remove resource when workload decreases. Used two realistic server workload and used LSTM, Bi-LSTM and ARIMA methods for prediction. compared accuracy of prediction methods to select Bi-LSTM as a best prediction method. In experiment setup used Apache-Jmeter for workload generation, HAProxy as a load balancer and Prometheus as monitoring server. Experimental results show that proposed system achieved better performance in accuracy and speed than the default horizontal pod auto scaler (HPA) of the Kubernetes when provisioning and de-provisioning resources.

Moreno-Vozmediano, Rafael Montero, Rubén S. Huedo, Eduardo Llorente, Ignacio M. [16] presented and evaluated a novel predictive auto-scaling mechanism for time series analysis that is based on machine learning technique and queuing theory. Proposed mechanism predicts the processing load of a distributed server and estimate the number of resources required to optimize service response time and fulfill the SLA. Used SVM regression model for server processing workload and queue-based performance model for determining the number of resources that must be provisioned based on the predicted load. The results show that the proposed model achieves better forecasting accuracy with compared to other classical models and makes a resource allocation closer to the optimal case.

Y. Al-Dhuraibi, F. Paraiso, N. Djarallah, and P. Merle [17] proposed ElasticDocker that is rule-based and works autonomously which is the first system powering vertical elasticity of docker containers. ElasticDocker was designed to scale up and down both CPU and memory assigned to each container as per the application workload. It adjusts memory, CPU time, and vCPU cores according to workload. It directly modifies the cgroup file system of docker contains to implement scaling decisions. live migration was also added as container resources are limited to VM resources. Experiments show that ElasticDeocker helps to reduce expenses for users, and better resource utilization for container providers. It improves QoS for application end-users. Elastic docker outperforms Kubernetes elasticity by 37.63%. infrastructure level metrics like average memory utilization and CPU utilization are used for scaling decisions.

Meng, Yang Rao, Ruonan Zhang, Xin Hong, Pei [18] presented CRUPA a resource prediction algorithm based on a time series analysis model (ARIMA) combined with docker container technique. Designed comparison experiment to evaluate the proposed algorithm. Result is that proposed model forecast is only

6.5% in the short term, which is much lower than threshold based (16.9%) on the same dataset. Experimental result shows that proposed model has high prediction accuracy and also scales the resource well.

4.0 Problem Statement

From literature survey we found that auto scaling of multiple microservices based application is explored in very limited fashion. in case of sla violations which microservice to scale and how many containers to scale for quickly reacting to varying incoming workload is a research problem.

5.0 System Architecture

Container orchestration frameworks provide support for deploying and managing microservice based application as a set of containers on a cluster of nodes[10]. Docker swarm is most popular and prominent orchestration framework used for containerized application. A high-level architecture of a simplified containerized system deployed on a cluster of nodes is shown in fig. 1.

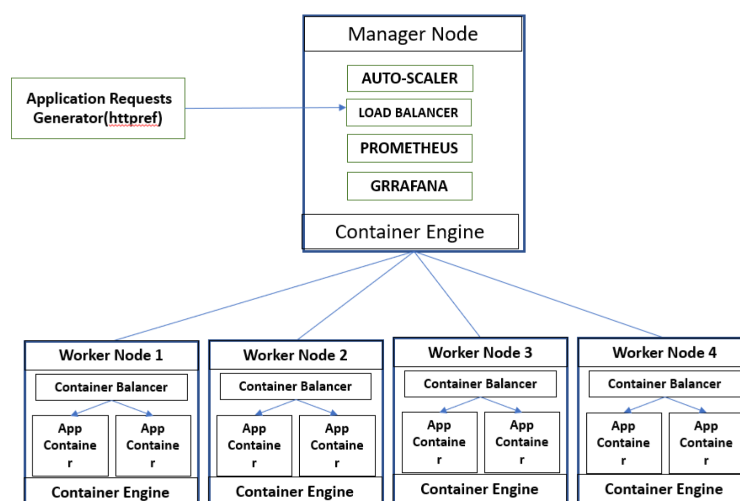

Figure 2 System Architecture.

The system contains several connected nodes orchestrated by Docker Swarm. In our system architecture one node act as a Manager Node while rest of the nodes are in role of Worker Nodes. Each node runs a Docker container engine to host containers.

Manager Node orchestrates the cluster and deployed containers. The manager node continuously checks for the join requests from the worker node and maintains the state of the cluster[19]. It also manages tasks of starting, stopping containers and changing container replicas. In our architecture Master Node hosts 1) auto-scaler which adjust containers replicas of bottleneck microservices with reference to predicted future workload. 2) Load Balancer (ngnix) which is used to redirect the user requests to containers. It also stores details about user requests and response in log file. 3) Prometheus which is a time series database. It is used to store infrastructure level and application-level metrics. It also fetches and store user requests and response details from load balancer log. Auto-Scaler regularly fetch this stored time series data from Prometheus for scaling decisions. 4) Grafana which is a tool to query, visualize, alert on and understand the data. It is used here to get data from Prometheus and visualize the number of requests, average response time, number of nodes, number of containers, etc.

ISSN 2063-5346Section A-Research paper

Worker nodes are used to host the replicas of containerized multiple microservice-based applications. Requests from the users are managed by load balancer where the workload will be distributed across container replicas running on that node. Application container requires resources such as CPU and Memory to process user requests.5) microservices based application. In our architecture we used our custom multiple microservice based application, which use CPU and memory of the containers. It is designed in such a way that we may apply CPU and memory utilization by passing numerical parameter from URL. E.g http://ip:5005/num here if we first pass 5 and then 10 it will use more CPU and memory in case of 10. We are using multiple microservices' so to apply different CPU and memory utilization to different microservices during simulation we managed to do it from url like http://ip:5005/1/3/5.

Table 1Summary of Container Auto-Scaling work

Work	Virtualization Technology	Monitoring Indicators	Method	Technique	Scaling Method	Application Scaled	Workload Generation Tool Used	Type Of Workload
[2]	Docker	CPU,Mem.	Proactive	SVR model with RBF Kernel	Horizontal	Node.js web application	-	Real Workload Trace
[11]	Docker	CPU,Mem.	Reactive	-	Hybrid (horizontal + vertical)	Web Application	apache-bench	Synthetic
[12]	Docker	No. of requests, avg. Response time, no of violations, scaling operations	Proactive	DTR, Elastic Net Regression	Horizontal	FFT Microservice	httperf	Real Workload Trace
[13]	Docker	Response time, no of containers in microservices	Proactive	RNN_LSTM	Horizontal	microservice based web application	Locust.io	Synthetic
[1]	Docker	Containers vs workload, responserate, CPU	Proactive	SVR	Horizontal	Web application	Tsung	Synthetic
[14]	Docker	Prediction error,slo guarantee, scaling operations	Proactive	LSTM with SSA Preprocessing	Horizontal	redis servcie and ecommerce application	-	Real Server Trace
[15]	Kubernetes	Prediction error, no. of pods,	Proactive	Bi-LSTM	Horizontal	Simple web application	Apache-JMeter	Real Server Trace
[17]	Kubernetes	CPU, Mem.	Reactive	-	Vertical	-	httperf	Synthetic
[18]	Kubernetes	CPU	Proactive	ARIMA	Horizontal	-	-	synthetic

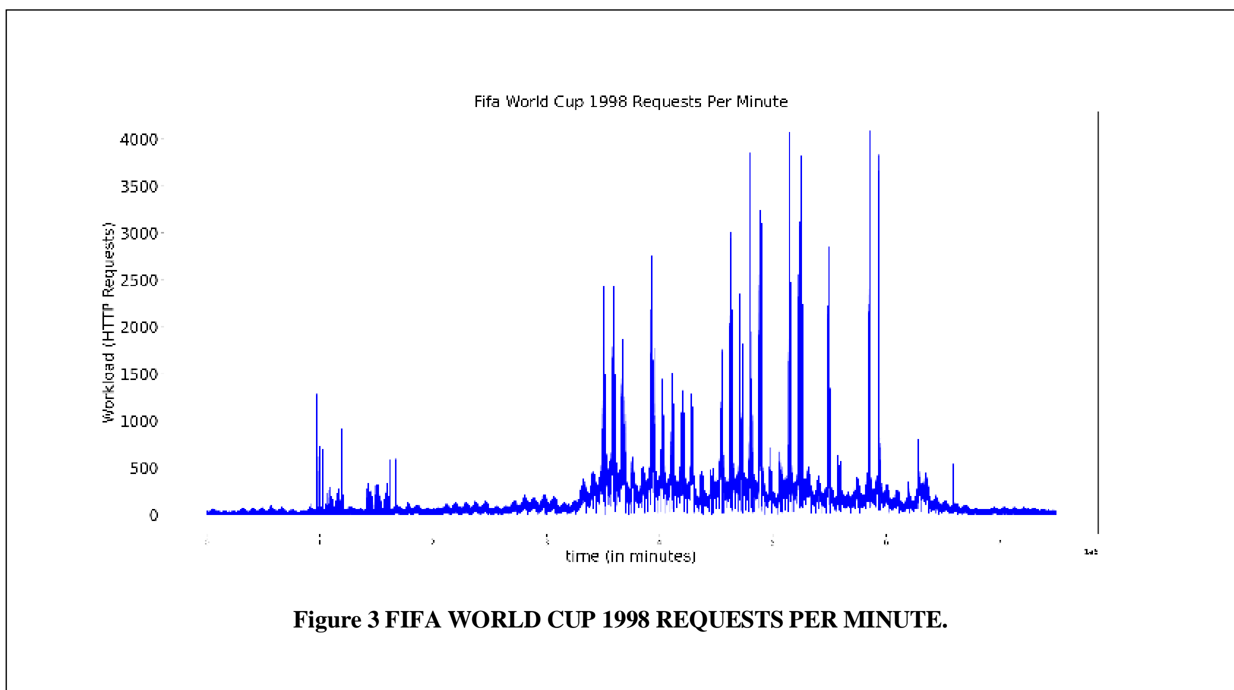

Figure 3 FIFA WORLD CUP 1998 REQUESTS PER MINUTE.

ISSN 2063-5346Section A-Research paper

6.0 Auto-scaler Architecture

Figure 3 shows the architecture of the auto-scaler. It contains four main steps as discussed in the previous section background of this paper. It also contains a time series database. For the experiment we used Prometheus which is monitoring and time series database.

this database stores data about Node/Container CPU and Memory utilization, Requests to application, Average Response Time which is utilized by monitor and analyzer units. Next will discuss the details of each unit.

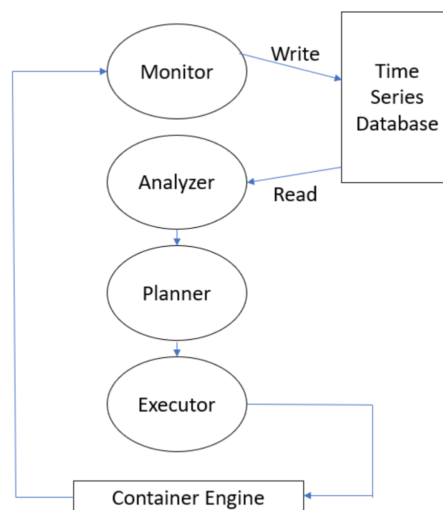

Figure 4 Auto-Scaler Architecture.

6.1 Monitor

The monitor unit keep collecting data at a fixed interval of time using the Prometheus monitoring tool that can be used in analyzer and plan units. It collects data like requests per second, the average response time of application from the load balancer, and CPU and memory utilization of all nodes and containers.

6.2 Analyzer

In the Analyzer unit collected data is analyzed periodically to get the status of the system. Data is pre-processed for the prediction model. The prediction model takes time series data as input and predicts the next sequence. Bi-LSTM neural network-based model is used in our implementation to predict future HTTP requests. Analyzer unit is very important as it help in predicting future requests based on which auto scaler can add or remove container replicas to minimize the delay in container starting time for better managing SLA agreement.

6.3 Planner

The Analyzer unit provides the predicted workload as output to the Planner unit. based on the provided predicted workload planner unit decides about a number of container replicas to increase or decrease.

We considered multiple microservices-based applications. So, for a container of a particular microservice, we need to estimate its replicas based on the maximum workload that can be handled by each container per second of that microservice. For simplicity, we used three microservices with different resource requirements. The application is designed in such as way that microservice_1 calls microservice_2 and microservice_2 calls microservice_3.

So estimated replicas for particular microservice are determined using the below formula.

Rmsi_estimated=(Wtotal_predicted/ Wmsi_max_container) (1)

Where Wtotal_predicted is total predicted incoming workload to the system which is obtained from the analyzer and Wmsi_max_container is the maximum workload that can be handled by each container per minute of the microservice i which can be obtained using stress testing during the development stage.

6.4 Executor

In Executor planned auto-scaling operations in the planning unit are executed. An executor is mainly responsible for getting commands from the planning unit for updating container replicas by communicating with the container engine (Docker daemon). planner uses API command of Docker Swarm for executing scaling commands.

Algorithm 1: Auto-Scaling Algorithm

Input: *rt_ut, rt_lt,* $W_{total_predicted}$*, scale_down_factor, cdt_scale_up, cdt_scale_down, rtlt_perc,min_replicas*

Output: *Scaling Action*

1 *Initialization*
2 ***While*** *the system is running* ***do***
3 ***If*** *avg_resp_time > rt_ut* ***then***
4 *btnk_ms = find_bottleneck_ms()*
5 $R_{ms_estimated} = W_{total_predicted} / W_{ms_max_container}$
6 *If* $R_{ms_estimated} > R_{current}$
7 *sendCommand(Scale,Up, btnk_ms,*$R_{ms_stimated} - R_{current}$*)*
8 *restart cdt_scale_up*
9 ***else if*** *avg_resp_time * rtlt_perc <= rt_lt* ***then***
10 *ms_sdown = find_sdown_ms()*
11 ***if*** $ms_sdown_{replicas}$ *> min_replicas* ***then***
12 *sendCommand(Scale,Down,ms_sdown,* $ms_down_{replicas}$ ** scale_down_factor)*
restart cdt_scale_down
13 ***else***
14 *sendCommand(None)*
15 ***end if***
16 ***end while***

7.0 Dataset Description

We used the FIFA Worldcup98 dataset to compare the accuracy of machine learning-based models. It contains HTTP requests logs of 1,352,804,107 requests Between April 30 and July 26 [20].

The dataset's total size is around 8.14 GB. The log contains the Unix time stamp, client ID, methods, and status. For evaluating auto-scalers Worldcup98 dataset has been used intensively [21],[22],[23].

7.1 Preprocessing

FIFA Worldcup 98 dataset is available from URL https://ita.ee.lbl.gov/html/contrib/WorldCup.html in binary log format in the form of day wise249 archive files. Preprocessing of these files is required before using them. The pre-processing steps are shown below.

1. Convert file from binary format to text format using the tool provided at URL https://ita.ee.lbl.gov/html/contrib/WorldCup.html.
2. Remove additional details from the log file and prepare a file that only contains the UNIX timestamp and request count.
3. In step, number 2 file contains a UNIX time stamp with an interval of microseconds. To convert the interval in the form of seconds from microseconds so it contains the number of requests per second.
4. In step number 3 file contains details of requests per second. Converted it to request per minute.

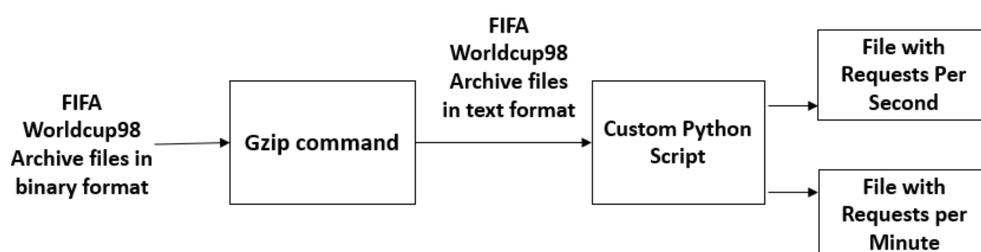

Figure 5 FIFA Worldcup98 File Preprocessing.

For preprocessing of log files, we created our own Python script that uses the Unix cat command with specific options.

The dataset converted to per-second requests contains a total of 7499630 requests from which 90% of requests were used for training and the rest requests used for testing. So, for the training set S1 uses 6749667, and for the testing set S2 uses 749963. The per-second dataset can further be simplified by converting it into the per-minute format as it represents the same pattern and also speeds up to training process. In per minute format, it contains a total of 1,25,300 requests from which 90% of requests are used for training purposes and it is named as set M1 which contains a total of 1,12,770 requests while the testing dataset named M2 contains a total of 12,530 requests. These details are described in the table below.

The per-minute requests are also shown in the table below. With training and testing size.

Table 2 FIFA Worldcup98 Dataset

Reference	Dataset	Size
Sec1	Workload per second	67,49,667
Sec2		7,49,963
Min1	Workload per minute	1,12,770
Min2		12,530

8.0 Prediction Models

8.1 ANN

An artificial neural network (ANN) contains three layers Input Layer, Hidden Layer, and Output Layer. The name itself suggests input layer accepts inputs. The hidden layer presents between the input layer and output layers. it does all the calculations to find hidden features and patterns. The input passes through a series of transformations using the hidden layer and that finally results in output[24]. The ANN computes the weighted sum of the inputs and includes a bias which can be represented using the below function.

ISSN 2063-5346 Section A-Research paper

$$O = \sum_{i=1}^{n} Wi * Xi + b \quad (2)$$

Here weighted total is passed as an input to the activation function for producing output. The activation function selects if the node should fire or not. The ones only which are fired make it to the output layer.

8.2 LSTM

Long short-term memory (LSTM) is a special type of recurrent neural network (RNN). RNN is having vanishing gradient drawback and LSTM overcame it. In LSTM series of LSTM cells are connected to each other in order to build up the LSTM network. The cell state stores incoming data. LSTM units contain three gates namely forget, input, and output which control the amount of data passing through them [25].

8.3 BI-LSTM

Bi-LSTM is a special type of LSTM that contains two LSTM layers. These layers run the input data in opposite directions which makes the model preserve information from the past and the future. So Bi-LSTM is capable of better accumulating knowledge and also it improves the prediction accuracy[26].

8.4 Prediction Model Evaluation Metric

Metrics available to evaluate prediction models are mean square error (MSE), mean absolute error (MAE) and root mean square error (RMSE), mean absolute percentage error (MAPE). Expression forms of all error types are given below.

Where X_t represents time series of observed data, $\bar{X}_t$ represent the time series of predicted data. n represents length of the series.

Mean Square Error (MSE) formula can be given be given as below.

$$\text{MSE} = \frac{1}{n}\sum_{t=1}^{n}(\bar{X}_t - X_t)^2 \quad (3)$$

Mean Absolute Error (MAE) formula can be given be given as below

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n}|\bar{X}_t - X_t| \quad (4)$$

Root Mean Square Error (RMSE) formula can be given be given as below.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(\bar{X}_t - X_t)^2} \quad (5)$$

Mean Absolute Percentage Error (MAPE) formula can be given as below.

$$\text{MAPE} = \frac{1}{n}\sum_{t=1}^{n}\frac{|\bar{X}_t - X_t|}{X_t} \quad (6)$$

ISSN 2063-5346*Section A-Research paper*

8.5 Prediction and Evaluation

The prediction models discussed above sections were trained and evaluated by using the Min1 dataset. Scaling between -1 and 1 is applied to the dataset. For validation, 10% of the Min1 dataset is used. For training 90% and evaluation 10% of requests per minute dataset is used.

Models are trained to predict workload in the next minute provided workloads in the last 10 minutes. All prediction models are implemented using the Keras library powered by the TensorFlow backend. The models were trained on the local machine with 16 GB RAM and a core i5 processor.

Table 3 ANN model configuration

Input Size	10
Output Size	1
Number of hidden layers	1
Number of hidden neural cells	30
Activation function	Relu
Loss function	MSE
Optimizer	Adam
Batch Size	64
Number of epochs	50
Early Stopping Patience	2

Table 4 LSTM model configuration

Input Size	10
Output Size	1
Number of LSTM layers	1
Number of LSTM units	30
Kernal initializer	Locum uniform
Loss function	MSE
Optimizer	Adam
Batch Size	64
Number of epochs	50
Early Stopping Patience	2

Table 5 Bi-LSTM model configuration

Input Size	10
Output Size	1
Number of LSTM layers	1
Number of LSTM units	30
Kernal initializer	Locum uniform
Loss function	MSE
Optimizer	Adam
Batch Size	64
Number of epochs	50
Early Stopping Patience	2

Table 6 Comparison of ANN, LSTM & Bi-LSTM Accuracy

Model	**ANN**	**LSTM**	**Bi-LSTM**
RMSE	0.52477	0.00965	**0.00929**

ISSN 2063-5346 *Section A-Research paper*

9.0 Experiment Description

To perform the experiment, we used The Digital Ocean cloud and implemented an auto scaler in the docker swarm environment.

The system architecture is shown in Figure 2. We collected a sample workload from the FIFA Worldcup1998 dataset and used our custom Python script which reads the number of requests from the workload file and use httpref to generate workload for Multiple microservices-based application. We used httpref rate option to simulate the number of requests at every interval of 1 min.

We used one manager node and five worker nodes. The manager node contains the source of our custom microservices-based application, auto scaler, Prometheus, and Grafana. We used Nginx as a load balancer along with the swarm default load balancer. Auto scaler is implemented using Docker SDK and Python Programming language. It includes some of the important functions to get Container average CPU Utilization, Microservice average response time, find boatneck microservices, etc. We also created a custom log writer script to record a log of simulation which can be used to analyze the outcome of the experiment.

We used Prometheus monitoring toolkit to record all the metrics during the simulation. We configured it to fetch metrics with every interval of 15 seconds.

Worker nodes are part of the docker swam and are used to run containers. To store and analyze data from worker nodes data exporters are also required to run on worker nodes.

The master node and worker node had 8GB of RAM and 4 vCPUs.The proxy node had 1 GB of RAM and 1 vCPU Below table shows system configurations.

We kept 0.100 seconds (average response time) as our upper response time threshold and 80% of it as lower response time threshold. We kept 30 seconds as scale up and 15 seconds as scale down cooldown time to avoid frequent scaling operations. For prediction we used last 10 min workload as input to predict next minute workload.

Table 7 System Configuration

Name	Version
OS	Ubuntu 18.04.3 LTS
Docker	20.10.21
httpref	0.9.0
Nginx	1.22.0
Prometheus	2.0.0

10. Experiment Results

In this section we are going to discuss experiment results.

We gone through multiple simulations to find the per minute capacity of each microservice container.

We started our simulation by taking 60 requests per minute waited for a minute to made the simulation stable. We applied the linear workload as shown in figure 6.

From the simulation log we compared the average response time maintained during the simulation of 15 min. in proactive approach auto scaler gain improvements in maintaining the response time as shown in

ISSN 2063-5346 *Section A-Research paper*

figure 7. During the simulation we kept increasing the requests per minute to test the proactive auto scaler capacity to handle the number of requests. Figure 8 demonstrates error in handling requests and indicates that proactive approach can handle a greater number of requests as compared with reactive approach.

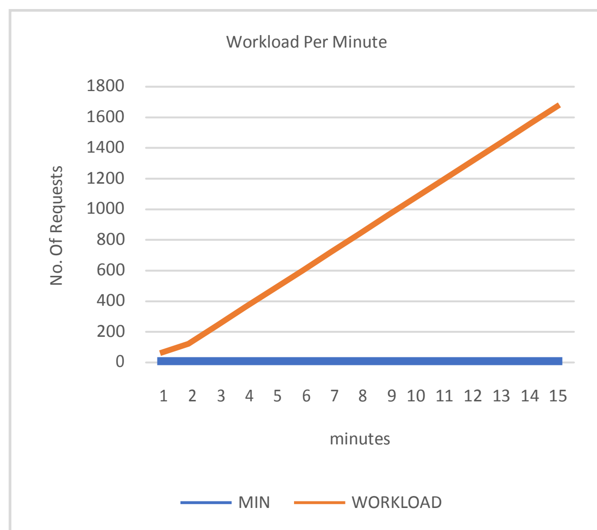

Figure 6 Workload Per Minute.

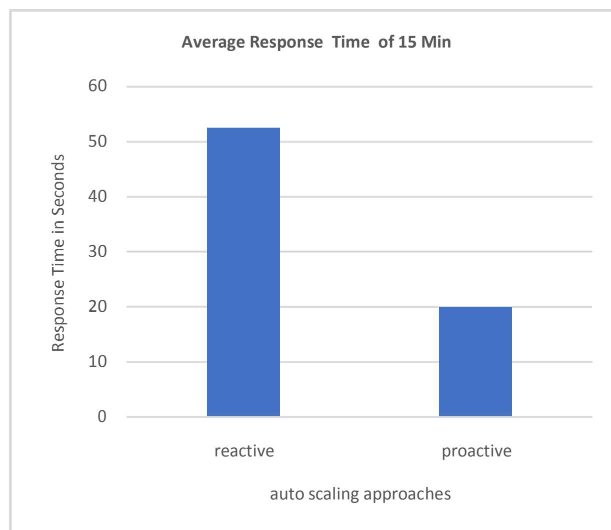

Figure 7 Average Response Time of 15 Min.

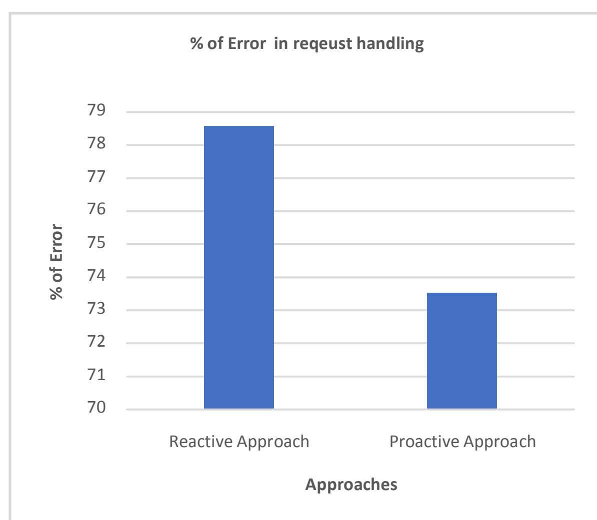

Figure 8 % of errors in request handling.

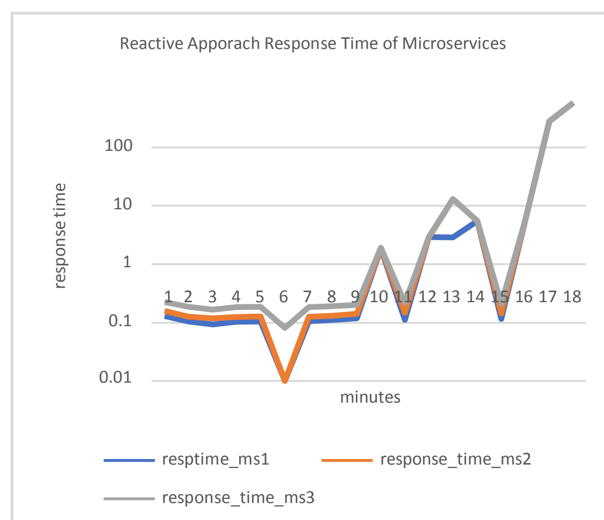

Figure 9 ReactiveResponse Time of Microservices.

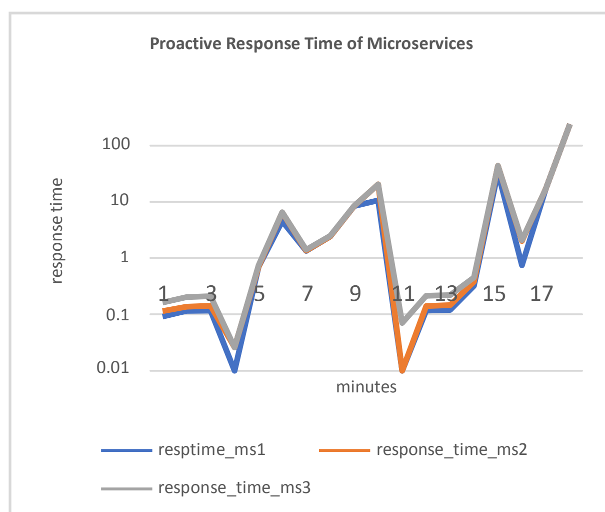

Figure 10 Proactive Response Time of Microservices.

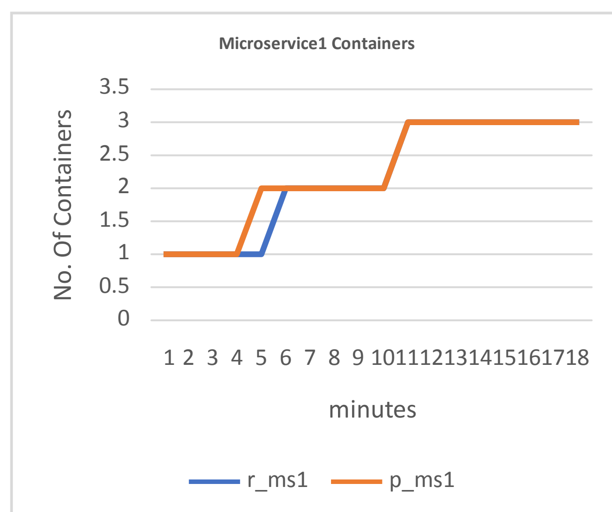

Figure 11 MS-1 Containers in Both Approaches.

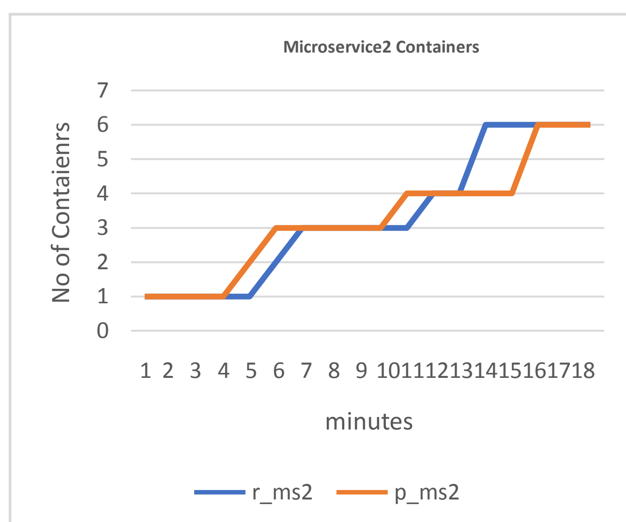

Figure 12 MS-2 Containers in Both Approaches.

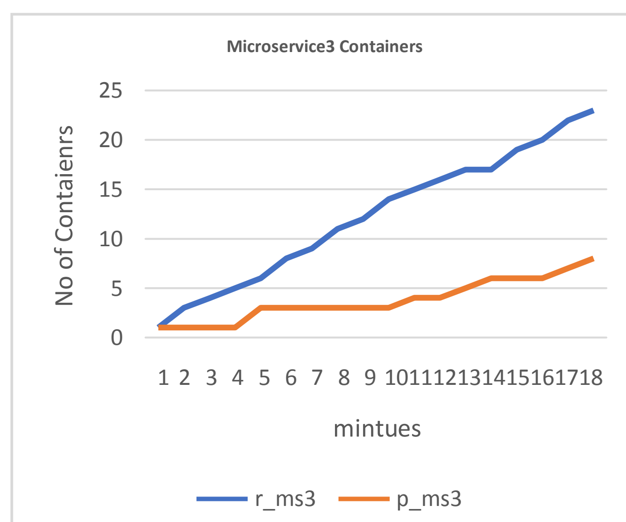

Figure 13 MS-3 Containers in Both Approaches.

Figure 9 and 10 shows the response time of microservices for reactive and proactive approach. Workload after 10-minute increases in large and proactive approach can quickly react and maintain the response time for longer time.

Figure 11,12 and 13 shows the microservices container auto scaling in action. We can see that in proactive approach it scales up the containers in advance to quickly react to the incoming workload.

11. Conclusion and Future Work

In this article, we considered an application that is a multiple microservices-based. We did simulation for 15 min on linear workload our first experiment was reactive auto-scaling in which scaling decisions were based on application average response time. our next experiment was proactive auto-scaling using the Bi-LSTM prediction model and also scaling of containers of bottleneck microservices. The results of our approach improve in SLA achievement, reduces request failures and also quickly reacts during heavy incoming workloads.

Future directions of this work are listed below.

1. In the current work, we used horizontal elasticity.it can be extended by using both horizontal and vertical elasticity (hybrid elasticity).
2. Work can be extended by using workload prediction in multiple time frames. Better auto-scaling decisions can be taken using this approach.
3. Work can be extended by adding an additional layer of Virtual Machine along with containers.
4. Work can be extended by using more accurate prediction models.
5. We will explore the approach with various workload patterns.

References

[1] M. P. Yadav, G. Raj, H. A. Akarte, and D. K. Yadav, "Horizontal scaling for containerized application using hybrid approach," *Ing. des Syst. d'Information*, vol. 25, no. 6, pp. 709–718, 2020, doi: 10.18280/isi.250601.

[2] M. P. Yadav, Rohit, and D. K. Yadav, "Maintaining container sustainability through machine learning," *Cluster Comput.*, vol. 24, no. 4, pp. 3725–3750, 2021, doi: 10.1007/s10586-021-03359-4.

[3] A. Mosallanejad, "A HIERARCHICAL SELF-HEALING SLA FOR CLOUD COMPUTING," *Int. J. Digit. Inf. Wirel. Commun.*, vol. 4, no. 1, pp. 43–52, 2014, doi: 10.17781/P001082.

[4] S. Chowhan, A. Kumar, and S. Shirwaikar, "Data Driven Resource Provisioning for Efficient Utilization of Cloud Resources," *Int. J. Inf. Technol. Electr. Eng.*, vol. 8, no. 6, pp. 45–49, 2019.

[5] N. Ma, A. Maghari, N. Sarkissian, and W. Clark Lambert, "Elasticity in Cloud Computing: What It Is, and What It Is Not," *Skinmed*, vol. 8, no. 6, pp. 361–362, 2010, [Online]. Available: http://sdqweb.ipd.kit.edu/publications/pdfs/HeKoRe2013-ICAC-Elasticity.pdf

[6] A. Zhao, Q. Huang, Y. Huang, L. Zou, Z. Chen, and J. Song, "Research on Resource Prediction Model Based on Kubernetes Container Auto-scaling Technology," in *IOP Conference Series: Materials Science and Engineering*, Aug. 2019, p. 052092. doi: 10.1088/1757-899X/569/5/052092.

[7] O. Adam, Y. C. Lee, and A. Y. Zomaya, "Stochastic resource provisioning for containerized multi-tier web services in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 7, pp. 2060–2073, 2017, doi: 10.1109/TPDS.2016.2639009.

[8] M. Abdullah, W. Iqbal, and F. Bukhari, "Containers vs Virtual Machines for Auto-scaling Multi-tier Applications Under Dynamically Increasing Workloads," *Commun. Comput. Inf. Sci.*, vol. 932, no. October, pp. 153–167, 2019, doi: 10.1007/978-981-13-6052-7_14.

[9] R. Moudjari, Z. Sahnoun, and F. Belala, "Towards a Fuzzy Bigraphical Multi Agent System for Cloud of Clouds Elasticity Management," *Int. J. Approx. Reason.*, vol. 102, pp. 86–107, 2018, doi: 10.1016/j.ijar.2018.07.012.

[10] M. S. Aslanpour, A. N. Toosi, J. Taheri, and R. Gaire, "AutoScaleSim: A simulation toolkit for auto-scaling Web applications in clouds," *Simul. Model. Pract. Theory*, vol. 108, no. January, 2021, doi: 10.1016/j.simpat.2020.102245.

[11] M. P. Yadav, N. Pal, and D. K. Yadav, "Resource provisioning for containerized applications,"

Cluster Comput., vol. 5, 2021, doi: 10.1007/s10586-021-03293-5.

[12] M. Abdullah, W. Iqbal, J. L. Berral, J. Polo, and D. Carrera, "Burst-Aware Predictive Autoscaling for Containerized Microservices," *IEEE Trans. Serv. Comput.*, vol. 15, no. May, pp. 1–1, 2020, doi: 10.1109/tsc.2020.2995937.

[13] N. C. Coulson, S. Sotiriadis, and N. Bessis, "Adaptive microservice scaling for elastic applications," *IEEE Internet Things J.*, vol. 7, no. 5, pp. 1–1, 2020, doi: 10.1109/JIOT.2020.2964405.

[14] Y. Yang, L. Zhao, Z. Li, L. Nie, P. Chen, and K. Li, "ElaX: Provisioning Resource Elastically for Containerized Online Cloud Services," in *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*, IEEE, 2019, pp. 1987–1994. doi: 10.1109/hpcc/smartcity/dss.2019.00274.

[15] N. M. Dang-Quang and M. Yoo, "Deep learning-based autoscaling using bidirectional long short-term memory for kubernetes," *Appl. Sci.*, vol. 11, no. 9, 2021, doi: 10.3390/app11093835.

[16] R. Moreno-Vozmediano, R. S. Montero, E. Huedo, and I. M. Llorente, "Efficient resource provisioning for elastic Cloud services based on machine learning techniques," *J. Cloud Comput.*, vol. 8, no. 1, 2019, doi: 10.1186/s13677-019-0128-9.

[17] Y. Al-dhuraibi, F. Paraiso, N. Djarallah, and P. Merle, "Autonomic Vertical Elasticity of Docker Containers with ElasticDocker To cite this version : Autonomic Vertical Elasticity of Docker Containers with ELASTICDOCKER," *Proc. 10th IEEE Int. Conf. Cloud Comput. IEEE CLOUD 2017*, 2017.

[18] Y. Meng, R. Rao, X. Zhang, and P. Hong, "CRUPA: A container resource utilization prediction algorithm for auto-scaling based on time series analysis," in *2016 International Conference on Progress in Informatics and Computing (PIC)*, IEEE, Dec. 2016, pp. 468–472. doi: 10.1109/PIC.2016.7949546.

[19] F. Rossi, M. Nardelli, and V. Cardellini, "Horizontal and vertical scaling of container-based applications using reinforcement learning," *IEEE Int. Conf. Cloud Comput. CLOUD*, vol. 2019-July, pp. 329–338, 2019, doi: 10.1109/CLOUD.2019.00061.

[20] M. Arlitt and T. Jin, "Workload characterization study of the 1998 World Cup Web site," *IEEE Netw.*, vol. 14, no. 3, pp. 30–37, 2000, doi: 10.1109/65.844498.

[21] A. Bauer, N. Herbst, S. Spinner, A. Ali-Eldin, and S. Kounev, "Chameleon: A Hybrid, Proactive Auto-Scaling Mechanism on a Level-Playing Field," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 4, pp. 800–813, Apr. 2019, doi: 10.1109/TPDS.2018.2870389.

[22] A. Ilyushkin *et al.*, "An Experimental Performance Evaluation of Autoscalers for Complex Workflows," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 3, no. 2, pp. 1–32, 2018, doi: 10.1145/3164537.

[23] Q. Zhang, H. Chen, Y. Shen, S. Ma, and H. Lu, "Optimization of virtual resource management for cloud applications to cope with traffic burst," *Futur. Gener. Comput. Syst.*, vol. 58, pp. 42–55,

ISSN 2063-5346 Section A-Research paper

2016, doi: 10.1016/j.future.2015.12.011.

[24] B. S. Kamath and R. D'Souza, "An ANN based classification scheme for QoS negotiation to achieve overall social benefit in cloud computing," *Int. J. Appl. Eng. Res.*, vol. 12, no. 10, pp. 2262–2271, 2017.

[25] S. Siami-Namini, N. Tavakoli, and A. Siami Namin, "A Comparison of ARIMA and LSTM in Forecasting Time Series," *Proc. - 17th IEEE Int. Conf. Mach. Learn. Appl. ICMLA 2018*, no. December, pp. 1394–1401, 2019, doi: 10.1109/ICMLA.2018.00227.

[26] Z. Wu, "The comparison of forecasting analysis based on the ARIMA-LSTM hybrid models," *Proc. - 2021 Int. Conf. E-Commerce E-Management, ICECEM 2021*, pp. 185–188, 2021, doi: 10.1109/ICECEM54757.2021.00044.

[27] https://www.youtube.com/watch?v=ADjRj6qPF7o