Building Resource Auto-Scaler with Functional-Link Neural Network and Adaptive Bacterial Foraging Optimization

Thieu Nguyen[1][0000−0001−9994−8747], Binh Minh Nguyen[1⋆][0000−0003−1328−3647], and Giang Nguyen[2][0000−0002−6769−0195]

[1] School of Information and Communication Technology
Hanoi University of Science and Technology, Hanoi, Vietnam
nguyenthieu2102@gmail.com, minhnb@soict.hust.edu.vn
[2] Institute of Informatics, Slovak Academy of Sciences, Bratislava, Slovakia
giang.ui@savba.sk

Abstract. In this paper, we present a novel intelligent proactive auto-scaling solution for cloud resource provisioning systems. The solution composes of an improvement variant of functional-link neural network and adaptive bacterial foraging optimization with life-cycle and social learning for proactive resource utilization forecasting as a part of our auto-scaler module. We also propose several mechanisms for processing simultaneously different resource metrics for the system. This enables our auto-scaler to explore hidden relationships between various metrics and thus help make more realistic for scaling decisions. In our system, a decision module is developed based on the cloud Service-Level Agreement (SLA) violation evaluation. We use Google trace dataset to evaluate the proposed solution well as the decision module introduced in this work. The gained experiment results demonstrate that our system is feasible to work in real situations with good performance.

Keywords: Proactive auto-scaling · Functional-link neural network · Adaptive bacterial foraging optimization · Multivariate time series data · Cloud computing · Google trace dataset.

1 Introduction

Cloud technologies bring many benefits for both vendors and users. From the view of providers, they can maximize physical server utilization via time-sharing of using resources among multiple customers. Meanwhile, from the view of users, they do not need to care about infrastructure deployment costs, and only pay for what they used conforming to the pay-as-you-go model. Consequently, cloud hosting services are becoming more and more popular today.

One of the main outstanding features of cloud computing is the capability of elasticizing resources provided for applications. This mechanism thus supports

⋆ Corresponding author.

to increase the availability as well as Quality of Service (QoS) for those applications [4]. Usually, the resource elasticity function is offered to users through an auto-scaling service that operates based on two techniques covering threshold and prediction. While the first already has been used in clouds, the second technique is still an attractive research problem today because there are barriers while applying the prediction-based auto-scaling system to clouds in practice. Firstly, accuracy of the prediction models must reach a certain level while having the ability of processing multiple resource metrics at the same time to meet practical demands. Secondly, the prediction models must be simple to deploy and operate but keep the effectiveness in forecast. Thirdly, the prediction-based auto-scaling system also must have a scaling decision mechanism which ensures QoS defined in Service-Level Agreement (SLA) signed between cloud customers and vendors.

In this paper, we focus on developing a cloud prediction-based auto-scaler, which can simultaneously resolve the barriers presented above. In this way, our auto-scaler use a simple prediction model while still achieving the same level of accuracy and stability as compared with other complex methods. For the prediction module of the auto-scaler, we propose a novel variant of functional-linking neural network (FLNN) using adaptive bacterial foraging optimization with life-cycle and social learning (in short from here ABFOLS or Adaptive BFOLS) to train forecast model. Our prediction module also can simultaneously process multiple resources based on several data preprocessing mechanisms. To build scaling decision module for the auto-scaler, we exploit SLA-awareness to make decisions as well as evaluate these scaling actions. Our designed system is experimented and assessed with a real cluster trace dataset published by Google in 2011 [18], and [17]. The gained outcomes show that our auto-scaler archives good performance and can be applied to practice.

The structure of this paper is as follows. In Section 2, we classify and analyze existing studies to highlight our work contributions. Section 3 presents our cloud auto-scaler proposal (in short from here FLABL) with the intelligent core built based on the combination of FLNN and ABFOLS. The preprocessing raw data mechanisms, which helps the prediction module to exploit multiple data metrics is also described here. In the section 4, we present tests and evaluations for the proposed auto-scaler to prove its effectiveness. The last section 5 concludes and defines our future work directions.

2 Related Work

As mentioned in the previous Section, there are two cloud resources scaling mechanism types, namely reactive (i.e. systems will response to unpredictable resource consumption changes using usage thresholds) and proactive [7] (i.e. systems attempt to predict resource requirements to make scaling decisions in advance). According to that classification, we focus on analyzing related works in the proactive technique category.

Resource consumption prediction. In [3], several predictive time series models were proposed and demonstrated for cloud workload forecast problem such as ARMA (autoregressive-moving average), non-stationary, long memory, three families of seasonal, multiple input-output, intervention and multivariate ARMA models. As in [22], the authors used two datasets collected from the Intel Netbatch logs and Google cluster data center to compare and evaluate prediction models together, including first-order autoregressive, simple exponential smoothing, double exponential smoothing, ETS, Automated ARIMA, neural network (NN) autoregression. Although these studies investigate methods for resource usage prediction, there is still lack of investigation of applying of FLNN variance in combination with evolutionary optimization in that domain like in [13].

Functional-link neural network (FLNN). Structurally, FLNN has only single neuron, which was proposed by Pao in [14] for pattern-recognition task. The network is quite simple but brings a good effective performance in the aspects of accuracy and training speed. Consequently, this learning model has been applied to many domains covering stock market prediction [10], and money exchange rate forecast [8]. In [5], the authors used FLNN for four datasets of wine sales. The gained results prove that FLNN is more efficient in comparison with random walk model and feed-forward neural network (FFNN). The authors of [19] used FLNN with Chebyshev and Legendre polynomials to predict impact on tall building structure under seismic loads. The experiments presented in that work shown FLNN yields better outcomes as compared with multi-layer neural network (MLNN) also in terms of accuracy and computation time. However, the main disadvantage of FLNN is the use of back-propagation (BP) mechanism with gradients descent to train the learning model ([10], [8], [5], and [19]) .

Adaptive Bacterial Foraging Optimization (ABFOLS). Recently, a new evolutionary computing technique called Bacterial Foraging Optimization (BFO) has been proposed in the work [15]. It is inspired based on principle of bacterial movement (i.e. tumbling, swimming or repositioning) to food-seeking. Their behavior is achieved through a series of three processes on a population of simulated cells: "Chemotaxis", "Reproduction", and "Elimination-dispersal" [15]. BFO has been applied to several industry applications like PID controller tuning [6], power system [1], stock market [9]. Unfortunately, BFO has certain shortages e.g., the appropriate time and method for dispersal and reproduction must be carefully selected, otherwise the stability of population may be destroyed [23]. Therefore, the authors of [23] proposed an improvement for BFO including life-cycle of bacteria, social learning and adaptive search step length (ABFOLS), which offer significant improvements over the original version in complexity and competitive performances as compared with other algorithms (e.g. GA) on higher-dimensional problems. At present, there are no works that deal in optimizing FLNN with ABFOLS algorithm, especially in cloud auto-scaling issue.

Cloud resources provision under SLA conditions. In [16], the authors presented a job scheduling system using machine learning to predict workloads and allocate resources complying with SLA. The authors of [20] used SLA to

estimate the amount of resources required for making scaling actions. They argued that their algorithm is able to reduce the number of SLA violations up to 95% and decrease resource requirements up to 33%. In [21], the authors proposed a proactive cloud scaling system that enables to estimate SLA violations based on multiple metric parameters. Although the works [16], and [20] already introduced SLA evaluation models, they operate based on single workload metric such as CPU usage, response time, or job execution deadline. Meanwhile, the work [21] allows processing multiple metrics, SLA violation still is evaluated based on resource usages (e.g. CPU, and memory usage). Conversely, in this study, we assess SLA to ensure QoS using the number of provided virtual machines (VMs). This is scaling unit, which cloud customers often care when making resources increase or decrease decisions.

In comparison with the existing works analyzed above, the differences and contributions of our work are as follows.

1. Proposing an improvement (called FLABL) for FLNN, in which the network is trained by ABFOLS instead of back-propagation mechanism.
2. Proposing an auto-scaler that uses FLABL, which can process multivariate metric data and predict resource consumptions.
3. Proposing an SLA violation evaluation module that operates based on VM number rather than resource metrics in order to make scaling decisions for the proposed auto-scaler.

3 Resource auto-scaler using functional-link adaptive bacterial foraging optimization neural network

The designs for our resource auto-scaling module are built based on underlying Cloud System. The module consists of 3 main phases, including Extraction, Learning and Scaling. The Scaling has two components namely Forecasting and Decision.

Raw resource monitoring data is collected from VMs in Cloud system. There are a lot of available monitoring services (e.g. CloudWatch, IBM cloud monitoring, and Rackspace Monitoring, Nagios, Prometheus and Zabbix and so forth) that can be used or deployed on cloud infrastructures for monitor problem. Based on the monitoring services, we gather diverse VM metrics such as CPU, memory utilization, disk IO, network IO, etc.). The detail descriptions of Extraction, Learning and Scaling phases are described in the following subsections.

3.1 Extraction phase

There are seven mechanisms deployed in Extraction phase to pre-process raw data and prepare normalized data for Scaling phase in our system. As shown in Fig. 1, those mechanisms cover:

1. Using collected raw monitoring data from the underlying Cloud System.

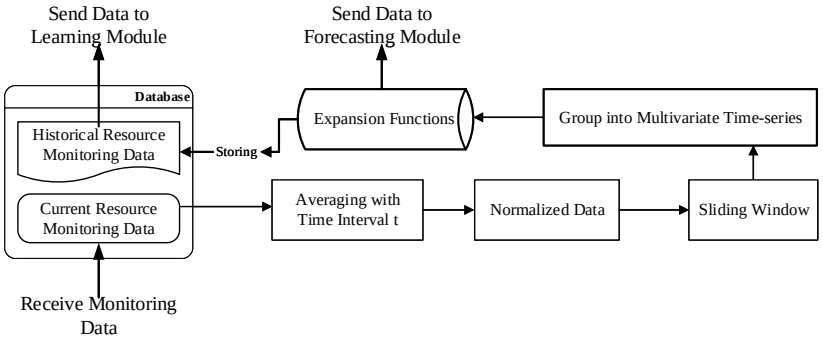

Fig. 1. FLABL data extraction process

2. Transforming current raw data in the predefined period for model training into corresponding time series $r_i(t)(i = 1, 2, ..., M)$ with time interval ρ.
3. Averaging values of given metrics in the given interval of ρ for each point in time series $r_i(t)$.
4. Normalizing the time series data in the range of [0, 1].
5. Transforming the normalized time series data to supervised data using sliding technique with window width k, which is the number of used values before the time t to predict a value at the time t.
6. The supervised data undergoes through an expansion function (e.g. Chebyshev or Power series) to enable the ability of catching nonlinear relationship between inputs and outputs.
7. Finally, the output data is put into a database in form of historical resources data, which is used to build prediction model in Learning phase. The data also is provided for Forecasting module in Scaling phase to predict the resource consumption.

3.2 Learning phase

Generated data from Extraction phase is divided into three different sets, namely training, validating, and testing. While the first two sets are employed to learn and select parameters, the third set is used for validating trained prediction model. A novel method is proposed in our Learning phase based on FLNN and trained by ABFOLS algorithm to speed up the convergence and increase the prediction accuracy. Due to combinations, our learning method is called by Functional-Link Adaptive Bacterial Foraging Lifecycle Neural Network (called by FLABL).

The final trained model of Learning is used in Forecasting module that belongs to Scaling phase. Meanwhile, the real-time monitoring data collected from cloud VM after preprocessing will be use as Real-time monitoring resource usage before putting into our Final Model of Forecasting module. The gained predictive results then will be denormalized to be able to use in the next module.

Algorithm 1 FLABL Learning phase

Input: S, N_s, P_{ed}, C_s, C_e, N_{split}, N_{adapt}
Output: The trained model

1: Normalizing all current resource time series of M type of resource consumption: $r_1(t), r_2(t), ..., r_i(t), ..., r_M(t)$
2: Initializing sliding window with p consecutive points as the inputs $(X_1(t), X_1(t-1), ..., X_1(t-p+1)), ..., (X_M(t), X_M(t-1), ..., X_M(t-p+1))$ and the next observation $X_1(t+k), ..., X_M(t+k)$ as the outputs $(t = 1, 2, ..., n)$
3: Grouping all column inputs data of M type resource into multivariate data $X(t) = [X_1(t), X_1(t-1), ..., X_1(t-p+1), ..., X_M(t), X_M(t-1), ..., X_M(t-p+1)]$
4: Applying expansion functions to multivariate data:

$$\begin{aligned} x(t) = [&x_1^1(t), x_1^2(t), ..., x_1^5(t), x_1^1(t-1), x_1^2(t-1), ..., x_1^5(t-p+1), ..., \\ &x_M^1(t), x_M^2(t), ..., x_M^5(t), x_M^1(t-1), x_M^2(t-1), ..., x_M^5(t-1), ..., \\ &x_M^1(t-p+1), x_M^2(t-p+1), .., x_M^5(t-p+1)] \end{aligned} \quad (1)$$

5: Training the constructed FLNN (from step 1 to 4) by ABFOLS (from step 6 to 31)
6: Initialize population $Pop = \{Cell_1, ..., Cell_s\}$ with S cells (*bacterium*), where $Cell_i = (c_{i1}, , c_{id})$, $c_{ij} \in [-1, 1]$ with d-dimensions, $d = length(x(t) + 1)$
Initialize $Nutrient(i) \leftarrow 0$ for all $Cell_i$
7: **while** (termination conditions are not met) **do**
8: $S \leftarrow size(Pop)$; i $\leftarrow 0$
9: **while** $(i < S)$ **do**
10: $i \leftarrow i + 1$; $fitLast \leftarrow fitCurrent(i)$
11: Generate a tumble angle for $Cell_i$ by Eq. 4
12: Update the position of $Cell_i$ by Eq. 3
13: Recalculate $fitCurrent(i)$ and $Nutrient(i)$
14: Update personal best ($Cell_{pBest}$) of ith cell and global best ($Cell_{gBest}$)
15: **for** `m = 1 to` N_s **do**
16: **if** $fitCurrent(i) < fitLast$ **then**
17: $fitLast \leftarrow fitCurrent(i)$
18: Run one step using Eq. 3
19: Recalculate $fitCurrent(i)$ and $Nutrient(i)$
20: Update $Cell_{pBest}$ and $Cell_{gBest}$
21: **else**
22: **break**
23: **if** $Nutrient(i) > threshold_{split}$ (Eq. 6 is True) **then**
24: Split $Cell_i$ into two *cell*
25: **break**
26: **if** $Nutrient(i) < threshold_{dead}$ (Eq. 7 is True) **then**
27: Remove $Cell_i$ from Pop
28: **break**
29: **if** $Nutrient(i) < 0$ and $random(0, 1) < P_e$ **then**
30: Move $Cell_i$ to a random position
31: Passing the **trained model** ($Cell_{gBest}$) to Forecasting module.
32: Repeating step 1 to step 30 in every time period T

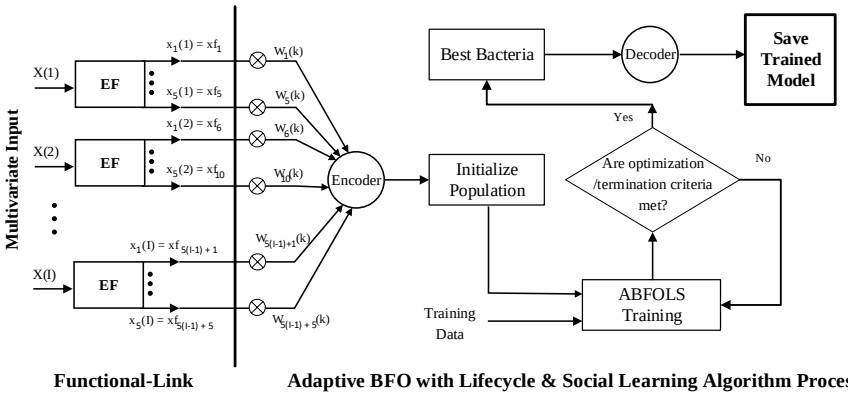

Fig. 2. FLABL Learning phase with FLNN and ABFOLS

The details of our proposal FLABL are as follows: Used parameters in Algorithm 1 are explained in Table 1. While$Nutrient(i)$ is the nutrient of ith bacterium, $fitCurrent(i)$ is the current fitness value of ith bacterium, $fitLast$ is the fitness value of the last position of ith bacterium, $Cell_{pBest}$ is the past position of ith, bacterium which gained the best fitness so far. $threshold_{split}$ and $threshold_{dead}$ is coresponding to right side of Eq. 6 and Eq. 7.

We use Mean Absolute Error (MAE) for bacterial fitness function according to the Eq. 2. In chemotactic steps, a bacterium will update its position based on the direction which created by combination of information of its personal best position and the populations global best position in Eq. 3 and Eq. 4. The gain of nutrients process is updated when bacteria move, so if the new position is better than the last one (the fitness higher), it is regarded that the bacterium will gain nutrient from the environment and the nutrient is added by one. Otherwise, it loses nutrient in the searching process and its nutrient is reduced by one Eq. 5.

In an intelligence optimization algorithm, it is important to balance its exploration ability and exploitation ability. In the early stage, we should enhance the exploration ability to search all the areas. In the later stage of the algorithm, we should enhance the exploitation ability to search the good areas intensively. So in BFO, the step size which bacterium used to move play an important role. We use the decreasing step length based on bacterium's fitness (Eq. 8). In the early stage of BFOLS algorithm, larger step length provides the exploration ability. And at the later stage, small step length is used to make the algorithm turn to exploitation. Besides, we also deploy an adaptive search strategy, which change the step length of each bacteria based on their own nutrient, which calculated using (Eq. 9). The higher nutrient value, the bacteriums step length is shortened further. This is also in accordance with the food searching behaviors in natural. The higher nutrient value indicates that the bacterium is located in potential

nutrient rich area with a larger probability. So, it is necessary to exploit the area carefully with smaller step length.

$$fit = MAE = \frac{\sum_{i=1}^{N} |forecast(i) - actual(i)|}{N} \tag{2}$$

$$\theta_i^{t+1} = \theta_i^t + C(i) * \frac{\Delta_i}{\sqrt{\Delta_i * \Delta_i^T}} \tag{3}$$

$$\Delta_i = (\theta_{gBest} - \theta_i) + (\theta_{i,pBest} - \theta_i) \tag{4}$$

$$Nutrient(i) = \begin{cases} Nutrient(i) + 1, & \text{if } fitCurrent(i) > fitLast \\ Nutrient(i) - 1, & \text{if } fitCurrent(i) < fitLast \end{cases} \tag{5}$$

$$Nutrient(i) > max\Big(N_{split}, N_{split} + \frac{S^i - S}{N_{adapt}}\Big) \tag{6}$$

$$Nutrient(i) < min\Big(0, 0 + \frac{S^i - S}{N_{adapt}}\Big) \tag{7}$$

$$C = C_s - \frac{(C_s - C_e) * nowFit}{totalFit} \tag{8}$$

$$C(i) = \begin{cases} \dfrac{C}{Nutrition(i)}, & \text{if } Nutrient(i) > 0 \\ C, & \text{if } Nutrient(i) \leq 0 \end{cases} \tag{9}$$

Name	Description
S	The number of cells (bacteria) in the population
d	d-dimension / problem size
N_s	Swimming length after which tumbling of bacteria will be undertaken in a chemotactic loop
P_{ed}	Probability for eliminate bacteria
C_s, C_e	The step size at the beginning and the end of chemotactic process
N_{split}, N_{adapt}	Parameters used to control and adjust the split criterion and dead criterion.

Table 1. FLABL parameters used for ABFOLS optimization

3.3 Scaling phase

The Scaling phase contains of Forecasting and Scaling components. The functionalities and collaboration of these two components are based on SLA-violation auto-scaling strategy as described in Algorithm 2.

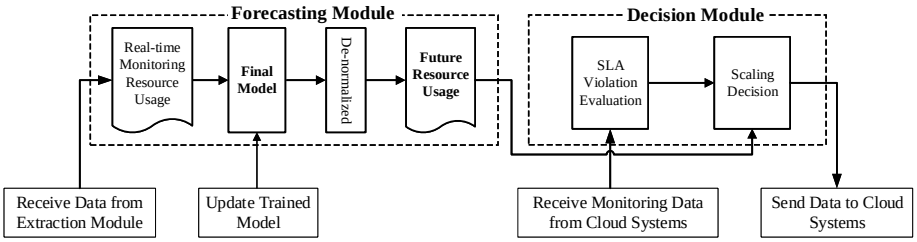

Fig. 3. FLABL Scaling phase

Forecasting module Forecasting phase uses real-time monitoring data (is pre-processed by Preprocessing phase) as inputs for the trained model to predict new values (i.e. resource consumption) in advance. Then, the obtained outputs are de-normalized into the real values. In every time period T, learning algorithm is updated with new monitoring data.

Decision module Decision module is responsible for calculating the number of provided VMs according to the predictive resource consumption. In addition, we develop SLA Violation Evaluation component that operates based on the number of VMs allocated at previous time and with the VM numbers predicted in the future to decide the how many VMs will be provisioned as in Equations 10, 11, and 12. Finally, predictive scaling information will be sent to Resource Scaling Trigger in Cloud System to create appropriate actions. With resource usages predicted by Forecasting module, we assume that cloud system can provision unlimited amount of VMs like the systems presented in [12], which has the same hardware configuration. We also do not consider VM scheduling policies in our scaling strategy in this work.

In terms of QoS guarantee, the number of predicted VMs will be multiplied by s ($s \geq 1$). The larger s, the more VMs allocated to applications and this reduces the SLA violations. In this direction, the total number of SLA violations in an earlier period with length L is used to determine the increases (or decreases) of allocated VMs.

4 Experiments

To evaluate our proposed auto-scaler, we carry out two experiments, including:

1. Comparing prediction accuracy and the system run time between various neural network models with our FLABL, involving ANN, MLNN, traditional FLNN, FL-GANN and FL-BFONN using univariate and multivariate data.
2. Evaluating scaling performance of our proposed auto-scaler under SLA violation measurement.

Algorithm 2 Auto-scaling strategy based on SLA-violations

1: At each timepoint t, calculate $F_1^{pred}(t+1), F_1^{pred}(t+1), .., F_1^{pred}(t+1)$ by FLABL then de-normalize results back to get resources consumption at the next timepoint: $r_1^{pred}(t+1), r_2^{pred}(t+1), ..., r_i^{pred}(t+1)$

2: Calculating the number of VMs predicted at time $(t+1)$ as

$$n_{VM}^{pred}(t+1) = max\Big\{\frac{r_1^{pred}(t+1)}{C_1}, \frac{r_2^{pred}(t+1)}{C_2}, .., \frac{r_i^{pred}(t+1)}{C_i}\Big\} \tag{10}$$

3: **for** $(z \leftarrow t - L + 1)$ to t calculate the number of VMs violations

$$n_{VM}^{violation}(t) = max\big\{0, n_{VM}^{actual}(t) - n_{VM}^{alloc}(t)\big\} \tag{11}$$

4: Calculating the number of allocated VMs

$$n_{VM}^{alloc}(t+1) = s * n_{VM}^{pred}(t+1) + \frac{1}{L}\sum_{z=t-L+1}^{t} n_{VM}^{violation}(z) \tag{12}$$

5: **if** $n_{VM}^{alloc}(t+1) > n_{VM}^{alloc}(t)$ **then**
6: Making decision to instantiate $n_{VM}^{alloc}(t+1) - n_{VM}^{alloc}(t)$ VMs
7: **else**
8: Making decision to destroy $n_{VM}^{alloc}(t) - n_{VM}^{alloc}(t+1)$ VMs
9: Repeating from step 1 to 8 at the next point of time.

4.1 Experimental setup

Dataset. In our experiment, we use dataset gathered by Google from one of their production data center clusters. The log records operations of approximate 12000 servers for one month [18], [17]. This log thus contains a lot of jobs submitted to the cluster. Each job is a collection of multiple tasks that are run simultaneously on many machines. Resource utilization of tasks is measured by several metrics such as CPU, and memory usage, disk I/O, and so on with more than 1233 million data items. To evaluate our predictive model as well as the auto-scaling strategy, we chose a long-running job with ID 6176858948, which consists of 60171 divergent tasks during the 20-day period.

We assume that the cloud system has enough VMs to process the chosen long-running job. VM capacity is equal to the minimum configuration of a machine in Google cluster with $C_{CPU} = 0.245$ and $C_{RAM} = 0.03$ (normalized values). We also assume that time required to instantiate a new VM is five minutes. Therefore, the resource usages in the incoming five minutes are predicted to make scaling decisions in advance. In this way, average time t is set by 5 minutes, forecast horizon $k = 1$ in all experiments. While, the collected data from 1^{st} to 14^{th} day is used to train the networks, data from 15^{th} to 17^{th} day employed to validate network's parameters, and data from 18^{th} to 20^{th} day is used to test the prediction performance. We select both CPU and memory metric types (visualization in Fig. 4) in the Google dataset as multivariate data for our proposed system. Before we group CPU and memory data to be multivariate

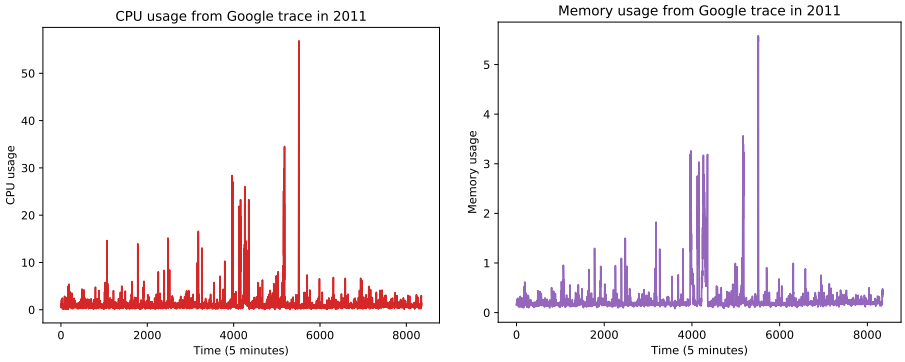

Fig. 4. CPU(left) and memory(right) from Google trace dataset (Google normalized data)

data, we make sure the data range in the same by normalized step in the Extraction phrase. The prediction accuracy is assessed by Root Mean Square Error $RMSE = \sqrt{\frac{\sum_{i=1}^{N}(forecast(i)-actual(i))^2}{N}}$.

Test models. Our tested ANN model is cofigured with three layers (one input, one hidden, and one output), MLNN model is configured with five layers (one input, three hidden and one output). Meanwhile, FLNN, FL-GANN, FL-BFONN and FLABL have only one input and output layer with structure $(x, 1)$. The polynomial is set as functional-link for all models. Here, x is the sliding window value used in the Extraction phase in our system. Activation function used for our tested networks are Exponential Linear Unit (ELU) [2].

Based on previous research in [13], we use standard GA algorithm for FL-GANN model. The GA parameters are set as follows. The population size $p_s = 500$, the maximum number of generations $g_{max} = 700$, the probability of two individuals exchanging crossovers $p_c = 0.95$ and the probability of individual mutation $p_m = 0.025$. For BFO algorithm, the parameters are configured in the same way as works described in [15] which are set as follows: The length of the lifetime of the bacteria as measured by the number of chemotactic steps they take during their life $N_c = 50$, swimming length after which tumbling of bacteria will be undertaken in a chemotactic loop $N_s = 4$, maximum number of reproduction to be undertaken $N_{re} = 4$, the number of elimination-dispersal steps $N_{ed} = 8$, probability for eliminate bacteria $P_e = 0.25$, and $Step_{size} = 0.1$. Finally, for our ABFOLS algorithm, the parameter settings are similar to [23] work which are $N_s = 4$, $P_e = 0.25$ (same as BFO configurations), $N_{split} = 30$, $N_{adapt} = 5$, the step size at the beginning of process $C_s = 0.1(U_b$ - $L_b)$, the step size at the end of the process $C_e = 0.00001(U_b$ - $L_b)$, where U_b and L_b are referred to the upper and lower bound of the variables.

SLA evaluation mechanisms. We use SLAVTP (SLA Violation Time Percentage) proposed in [21] to evaluate our proposed VM calculation mechanism (formula 12 $SLAVTP = \frac{T_{under-provisioning}}{T_{execution}}$, where $T_{under-provisioning}$ is the time

when at least one allocation resource causes "under-provisioning" (i.e. lack of RAM or CPU core), and $T_{execution}$ is total time of the application running in cloud system.

To evaluate performance of our auto-scaling strategies, we use ADI (Auto-scaling Demand Index) measurement [11], which considers the difference between actual and desired resource utilization. In other words, the total distance between u_t and $[L, U]$. In which u_t is the utilization level of the system, L and U correspond to the lower and upper limits of reasonable resource use, with $0 \leq L \leq U \leq 1.0$. The ADI is denoted by the variable $\sigma = \sum_{t \in T} \sigma_t$ where, $\sigma_t = L - u_t$ if $u_t \leq L$; $\sigma_t = 0$ if $L < u_t < U$; $\sigma_t = u_t - U$ if otherwise. For each time t, according to the ADI formula, the optimization strategy will yield a minimum of σ.

4.2 FLABL forecast accuracy and runtime

In this test, we evaluate the efficiency of FLABL against FL-BFONN, FL-GANN, traditional FLNN, MLNN and ANN in forecasting resources consumption. For each model, we use both univariate (single input metric) and multivariate (multiple input metrics) data. We also change sliding windows size k from 3 to 5 (k = 3, 4, 5) in the experiment to test effectiveness fluctuation. Our achieved outcomes are given in Table 2.

Input Type	Model	CPU			RAM		
		k = 3	k = 4	k = 5	k = 3	k = 4	k = 5
Univariate	ANN	0.4962	0.4952	0.5054	0.0344	0.0352	0.0354
	MLNN	0.4903	0.4930	0.4966	0.0345	0.0349	0.0355
	FLNN	0.5171	0.510	0.5253	0.038	0.0356	0.0373
	FL-GANN	0.4892	0.4762	0.4877	0.0389	0.0376	0.0375
	FL-BFONN	0.4777	0.4872	0.4798	0.0342	0.0431	0.035
	FLABL	**0.469**	**0.4726**	**0.4732**	**0.0335**	**0.0338**	**0.0338**
Multivariate	ANN	0.4884	0.4863	0.507	0.0343	0.0336	0.0348
	MLNN	0.4913	0.4814	0.5026	0.0359	0.0355	0.0356
	FLNN	0.4877	0.5179	0.5043	0.0367	0.0365	0.0366
	FL-GANN	0.49	0.491	0.488	0.037	0.0361	0.036
	FL-BFONN	0.4976	0.4762	0.4963	0.0334	0.0332	0.0337
	FLABL	**0.4678**	**0.4705**	**0.4793**	**0.0333**	0.0337	0.0344

Table 2. RMSE comparison between FLABL and other models

According to the achieved results, there are some observations that can be made as follows. In almost all cases, RMSE accuracy of FLABL are smaller than ANN, MLNN, traditional FLNN, FL-GANN and FL-BFONN model with different sliding window values as well as input types. Concretely, for univariate data input, FLABL brings the best results as compared with others, also for

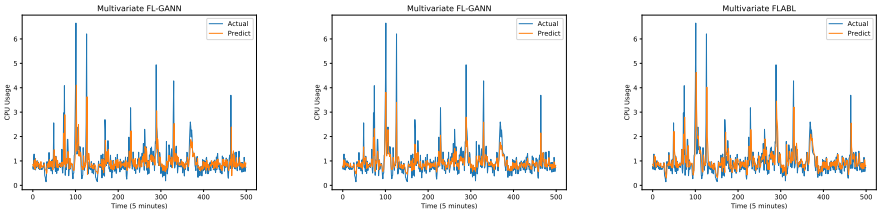

Fig. 5. CPU prediction outcomes of FL-GANN (first), FL-BFONN (second) and FLABL (third) with multivariate data, sliding window = 5

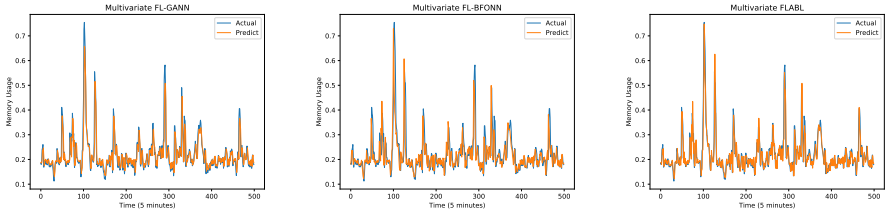

Fig. 6. Memory prediction outcomes of FL-GANN (first), FL-BFONN (second) and FLABL (third) with multivariate data, sliding window = 5

multivariate data input except $k = 4$ and 5 in memory prediction case. However, the difference is trivial for the test scenarios (0.0337 (FLABL) compares with 0.0332 (FL-BFONN) with $k = 4$, and 0.0344 (FLABL) compares with 0.0337(FL-BFONN) with $k = 5$). This shows the advantage of FLABL in prediction in comparison with other models. Figure 5 and 6 illustrate the prediction outcomes of CPU and memory metrics in accuracy comparison tests among experimental models.

In this test, we compare the speed of those prediction models, which based on 3 factor includes: t_e is average time for 1 epoch (second/epoch), t_p is time to predict test data (second) and t_s is the total time of all system (preprocessing, training and testing - second). Because each model has different epoch configuration, so t_e should be choice instead of the total time of training process. Our speed comparison are given in Table 3.

4.3 Auto-scaling strategy

In this test, we evaluate auto-scaling decisions made based on prediction results and SLA violation assessments as presented in Subsection 3.3. Figure 7 shows the number of VMs calculated using the formula 12. An observation can be made from the obtained outcomes as follows. With $s < 1.3$, there are still some under-provision VMs as compared with resource requirements. For $s \geq 1.3$, VMs are allocated sufficiently for the demand usages. It's also shown in below test when we consider the lack and over-provision of resources.

Input Type	Model	CPU			RAM		
		t_e	t_p	t_s	t_e	t_p	t_s
Univariate	ANN	0.0379	0.0319	198.4	0.044	0.0321	220.3
	MLNN	0.0583	0.0587	291.58	0.0644	0.0686	322.15
	FLNN	0.023	0.0003	114.83	0.0248	0.0003	124.03
	FL-GANN	0.3154	0.0004	220.79	0.2612	0.0004	182.89
	FL-BFONN	3.473	0.0006	2778.5	3.289	0.0014	2521.5
	FLABL	0.1095	0.0004	**71.16**	0.1229	0.0007	**122.89**
Multivariate	ANN	0.0406	0.0324	202.95	0.0418	0.0325	209.15
	MLNN	0.0605	0.0598	302.48	0.054	0.0589	270.25
	FLNN	0.0245	0.0021	122.53	0.0263	0.0004	131.45
	FL-GANN	0.3944	0.0004	276.15	0.4176	0.0004	292.35
	FL-BFONN	3.166	0.0007	2532.6	3.177	0.0034	2541.5
	FLABL	0.1708	0.0008	170.82	0.1419	0.0009	141.91

Table 3. System run time (second) comparison between FLABL and other models with sliding window = 5

In order to assess the lack or over-provision of resources, we use SLAVTP and ADI measurements. Concretely, table 4 shows SLAVTP estimations of various models in VM allocation process with window size $p = 3$. In general, our proposed model gains smaller SLAVTP values than other models when the scaling coefficient is changed. For example, when $s = 2.0$, FLABL univariate model has only 0.18% violation, while the multivariate model violates 0.12%. Especially, when $s = 2.5$ both of univariate and multivariate model reaches 0% violation. Of course, in these cases, the number of VMS provisioned is quite large.

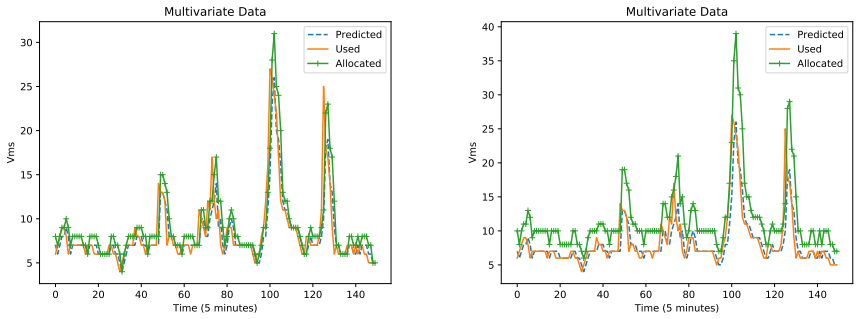

Fig. 7. The number of predicted, allocated, used VMs with sliding window = 3, adaptation length $L = 5$, and scaling coefficient $s = 1$ (left), $s = 1.3$(right)

Based on the ADI measurement, table 5 shows the ADI evaluation of different predictive models, with the desired utilization [60%, 80%] and window size $p =$

Input Type	Model	scaling coefficient						
		s = 1.0	s = 1.3	s = 1.5	s = 1.7	s = 2.0	s = 2.2	s = 2.5
Univariate	ANN	10.3	1.87	1.02	0.54	0.18	0.12	0.06
	MLNN	10.24	1.81	0.96	0.6	0.18	0.12	0.06
	FLNN	9.28	1.81	0.96	0.54	0.24	0.24	0.12
	FL-GANN	11.33	1.69	0.9	0.48	0.24	0.18	0.06
	FL-BFONN	11.81	1.75	0.84	0.54	0.24	0.06	0
	FLABL	10.3	1.93	1.02	0.54	**0.18**	**0.06**	**0**
Multivariate	ANN	15.96	7.59	5.42	4.28	3.13	2.53	1.99
	MLNN	10.42	1.69	0.9	0.6	0.18	0.06	0.06
	FLNN	11.08	1.75	0.84	0.54	0.12	0.06	0
	FL-GANN	10.3	1.69	1.27	0.48	0.24	0.12	0.06
	FL-BFONN	9.7	1.81	0.84	0.6	0.18	0.12	0.06
	FLABL	**9.76**	**1.69**	0.9	0.54	**0.12**	**0.06**	**0**

Table 4. Violation percentage in comparison among various models with adaptation length = 5, and sliding window = 3

Input Type	Model	scaling coefficient						
		s = 1.0	s = 1.3	s = 1.5	s = 1.7	s = 2.0	s = 2.2	s = 2.5
Univariate	ANN	154.01	15.41	37.88	112.63	227.11	293.53	372.4
	MLNN	176.59	3.81	4.87	78.94	205.75	270.58	353.31
	FLNN	184.6	4.41	11.29	82.46	207.8	277.2	357.88
	FL-GANN	176.67	3.86	15.9	89.18	214.8	283.6	363.81
	FL-BFONN	179.62	5.21	4.42	77.31	202.46	267.21	350.03
	FLABL	189.27	**3.7**	5.54	74.73	**199.9**	**267**	**349.5**
Multivariate	ANN	342.11	320.86	356.43	394.72	456.56	491.43	539.16
	MLNN	203.73	10.03	7.84	67.19	192.52	254.28	337.38
	FLNN	183.2	8.63	14.53	83.34	203.7	271.43	352.82
	FL-GANN	195.78	9.79	10.24	74.45	192.43	259.98	341.88
	FL-BFONN	219.81	40.24	36.47	93.23	197.92	260.28	340.58
	FLABL	192.27	**8.22**	**5.5**	72.16	**183.9**	259.61	342.76

Table 5. ADI in comparison among various models with adaptation length = 5, and sliding window = 3

3. It is easy to observe that ADI of FLABL for both univariate and multivariate data types is smaller than others. Specifically, in the test, optimal ADI is at 3.7 when $s = 1.3$. The results demonstrate significant effect of our prediction as well as decision solutions. As mentioned above, when s is increased, SLA violation decreases but resource over-provision phenomenon occurs.

5 Conclusion and future work

In this paper, we presented our designs for a novel cloud proactive auto-scaler with complete modules from prediction to decision making. Thus, functional-link

neural network is used for the forecast phase. To overcome back-propagation drawback, we integrate adaptive bacterial foraging life-cycle and social learning optimization with the artificial neural network. This improvement brings better accuracy for our proposed model. The auto-scaler designed in this paper also enables the capability of analyzing multiple monitoring metrics at the same time. This mechanism supports our system to be able to discover the implicit relationships among metrics types and thus help make scaling decisions more precisely. For the decision module, we proposed an efficient way to calculate the number of VMs that are provided for cloud-based applications using SLA violation measurement. We tested the auto-scaler with a real dataset generated by Google cluster. The obtained outcomes show that our system can work efficiently as compared with other methods and it can be applied to practice in clouds. For the future, we would like to implement the auto-scaler in private cloud middleware like Openstack or OpenNebula. Based on the infrastructures, we will test the proposal with applications under real conditions.

Acknowledgments

This research is supported by Vietnamese MOETs project "Research on developing software framework to integrate IoT gateways for fog computing deployed on multi-cloud environment" No. B2017-BKA-32, Slovak APVV-17-0619 "Urgent Computing for Exascale Data", and EU H2020-777536 EOSC-hub "Integrating and managing services for the European Open Science Cloud".

References

1. Ali, E., Abd-Elazim, S.: Bacteria foraging optimization algorithm based load frequency controller for interconnected power system. International Journal of Electrical Power & Energy Systems **33**(3), 633–638 (2011)
2. Clevert, D.A., Unterthiner, T., Hochreiter, S.: Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289 (2015)
3. Hipel, K.W., McLeod, A.I.: Time series modelling of water resources and environmental systems, vol. 45. Elsevier (1994)
4. Hluchỳ, L., Nguyen, G., Astaloš, J., Tran, V., Šipková, V., Nguyen, B.M.: Effective computation resilience in high performance and distributed environments. Computing and Informatics **35**(6), 1386–1415 (2017)
5. Khandelwal, I., Satija, U., Adhikari, R.: Forecasting seasonal time series with functional link artificial neural network. In: Signal Processing and Integrated Networks (SPIN), 2015 2nd International Conference on. pp. 725–729. IEEE (2015)
6. Kim, D.H., Cho, J.H.: Adaptive tuning of pid controller for multivariable system using bacterial foraging based optimization. In: International Atlantic Web Intelligence Conference. pp. 231–235. Springer (2005)
7. Lorido-Botrán, T., Miguel-Alonso, J., Lozano, J.A.: Auto-scaling techniques for elastic applications in cloud environments. Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09 **12**, 2012 (2012)

8. Majhi, B., Rout, M., Majhi, R., Panda, G., Fleming, P.J.: New robust forecasting models for exchange rates prediction. Expert Systems with Applications **39**(16), 12658–12670 (2012)
9. Majhi, R., Panda, G., Sahoo, G., Dash, P.K., Das, D.P.: Stock market prediction of s&p 500 and djia using bacterial foraging optimization technique. In: Evolutionary Computation, 2007. CEC 2007. IEEE Congress on. pp. 2569–2575. IEEE (2007)
10. Majhi, R., Panda, G., Sahoo, G.: Development and performance evaluation of flann based model for forecasting of stock markets. Expert systems with Applications **36**(3), 6800–6808 (2009)
11. Netto, M.A., Cardonha, C., Cunha, R.L., Assunçao, M.D.: Evaluating auto-scaling strategies for cloud computing environments. In: Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2014 IEEE 22nd International Symposium on. pp. 187–196. IEEE (2014)
12. Nguyen, B.M., Tran, D., Nguyen, G.: Enhancing service capability with multiple finite capacity server queues in cloud data centers. Cluster Computing **19**(4), 1747–1767 (2016)
13. Nguyen, T., Tran, N., Nguyen, B.M., Nguyen, G.: A resource usage prediction system using functional-link and genetic algorithm neural network for multivariate cloud metrics. In: 2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA). pp. 49–56. IEEE (2018)
14. Pao, Y.: Adaptive pattern recognition and neural networks (1989)
15. Passino, K.M.: Biomimicry of bacterial foraging for distributed optimization and control. IEEE control systems **22**(3), 52–67 (2002)
16. Reig, G., Alonso, J., Guitart, J.: Prediction of job resource requirements for deadline schedulers to manage high-level slas on the cloud. In: Network Computing and Applications (NCA), 2010 9th IEEE International Symposium on. pp. 162–167. IEEE (2010)
17. Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A.: Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In: Proceedings of the Third ACM Symposium on Cloud Computing. p. 7. ACM (2012)
18. Reiss, C., Wilkes, J., Hellerstein, J.L.: Google cluster-usage traces: format+ schema. Google Inc., White Paper pp. 1–14 (2011)
19. Sahoo, D.M., Chakraverty, S.: Functional link neural network learning for response prediction of tall shear buildings with respect to earthquake data. IEEE Transactions on Systems, Man, and Cybernetics: Systems **48**(1), 1–10 (2018)
20. Souza, A.A.D., Netto, M.A.: Using application data for sla-aware auto-scaling in cloud environments. In: Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2015 IEEE 23rd International Symposium on. pp. 252–255. IEEE (2015)
21. Tran, D., Tran, N., Nguyen, G., Nguyen, B.M.: A proactive cloud scaling model based on fuzzy time series and sla awareness. Procedia Computer Science **108**, 365–374 (2017)
22. Vazquez, C., Krishnan, R., John, E.: Time series forecasting of cloud data center workloads for dynamic resource provisioning. JoWUA **6**(3), 87–110 (2015)
23. Yan, X., Zhu, Y., Zhang, H., Chen, H., Niu, B.: An adaptive bacterial foraging optimization algorithm with lifecycle and social learning. Discrete Dynamics in Nature and Society **2012** (2012)