J Supercomput (2015) 71:4235–4259
DOI 10.1007/s11227-015-1520-y

A hybrid heuristic-based tuned support vector regression model for cloud load prediction

Masoud Barati[1] · **Saeed Sharifian**[1]

Published online: 14 September 2015

Abstract Cloud computing elasticity helps the cloud providers to handle large amount of computation and storage demands in an efficient manner. Proactively provisioning cloud workload is essential in order to keep the cloud utilization and service-level agreement at an acceptable level. Problems such as new virtual machine start-up latency, energy minimization and efficient resource provisioning, requires to predict resource demands for a few minutes ahead. Since the Cloud workloads have a very dynamic nature, CPU/memory usage varies considerably in the cloud. Also, existing prediction methods have considerable prediction error and erroneous results. So we propose a novel tuned support vector regression (TSVR) scheme that carefully selects three SVR parameters by a hybrid genetic algorithm and particle swarm optimization method. A chaotic sequence is devised into the algorithm to improve prediction accuracy and simultaneously avoid premature converging. To demonstrate the prediction accuracy of our TSVR model, we conduct a simulation study using Google cloud traces. The simulation results show that the proposed TSVR model achieves better prediction performance than conventional models in terms of standard metrics.

Keywords Cloud computing · Forecasting · SVR · GA · PSO · VM

1 Introduction

Rapidly growing cloud computing in recent years and its pay-as-you-go model which charges users according to their resource consumption besides the dynamic resource

✉ Saeed Sharifian
sharifian_s@aut.ac.ir

Masoud Barati
masoud_barati@aut.ac.ir

[1] Department of Electrical Engineering, Amirkabir University of Technology, 15914 Tehran, Iran

provisioning feature has resulted in wide adoption of this technology by many enterprises, businesses and governments. The computing model provides access to businesses and users to the vast computation and storage resources on demand in any platform such as mobile devices [31,42] and from anywhere in the world [6,21,37] with three different service models, namely infrastructure as a service (IaaS), platform as a service (PaaS) and software as a service (SaaS) [40]. The IaaS that provides scalable resources such as physical resource and VMs is a basic form of cloud computing. The PaaS service that is a cloud platform such as programming-language environment, operating system, and database is delivered by application developers to run their programs on a cloud. The SaaS is a service that provides application and database on the Internet.

Many large datacenters around the world are built and equipped with massive computation and storage capabilities that consume large amounts of energy. The clients are charged according to their resource and services usages in the datacenter. An important factor in the bill of a cloud provider is cost of electrical energy. Datacenters have a considerable role in the total electrical energy consumption of a country. For example, up to 2 percent of the total electrical energy consumption in the US in 2010 have been used by datacenters [32]. Hence, any reduction in datacenter energy consumption results in lower total cost of datacenter ownership [18].

One of the main concerns in today's datacenter provisioning is the resource efficiency. Static provisioning as traditional model causes undesirable over-provisioning and under-provisioning effects [27]. Under-provisioning leads to a poor quality of experience (QoE) and the large amount of SLA violation penalty for the cloud providers [35]. On the other hand, over-provisioning causes the waste of resources and high energy consumption in datacenter. One of the research conducted on resource utilization and the energy efficiency for more than 5000 Google's servers shows that idle state energy consumption is about 50 percent of peak power demand in datacenter [3]. Another result shows that there is a difference between the actual task demand for memory and CPU resources and the allocated amount in the datacenter which should be considered in cloud computing dynamic resource provisioning [19]. Therefore, developing better resource management can lead to better energy management in datacenters and solve the problem of static resource provisioning.

Many previous works have been addressed resource provisioning to solve this problem in the Cloud Computing [30]. Among different issues such as dynamic resource provisioning [33], virtual machine migration [36], server consolidation and energy management; in recent years proactive resource provisioning have been attracting a lot of attention. In this regard, prediction of cloud resource demands instead of reaction to anomaly occurrence such as overloading can benefit resource allocation and improve resource utilization. Therefore, proactive scheme in the form of predictive elastic resource provisioning can minimize the energy consumption and leads to an efficient use of resources. So an accurate cloud load prediction is an essential requirement for any proactive method. Therefore, to minimize the total datacenter energy consumption and simultaneously satisfy the constraints specified in the SLAs, total CPU and memory load must be predicted.

Load prediction in cloud computing leads to dynamic resource provisioning based on the pay-as-you-go model that provides a cost-effective resource management

scheme. Also, time variation of VM boot-up, taking 1–5 min [34], is a crucial issue in dynamic resource provisioning. A good dynamic provisioning system should carefully predict VM addition/deletion requirement about 5 min before its actual demand [2]. Due to the observed shortcoming of the current load prediction model for cloud resource demand prediction; in this paper, we propose a new machine learning algorithm for effective cloud prediction. Our load prediction method is based on SVR, which has lower average prediction error with low time consumption. The SVR model parameters are carefully selected by a combination of two Evolutionary Algorithms: genetic algorithm and particle swarm optimization. Also a chaotic sequence is proposed as random numbers to decrease computation time.

The proposed hybrid GA–PSO model is named TSVR and aims to predict CPU and memory usage in the cloud. This hybrid model has an efficient result in experimental load prediction. The predicted demands are considered to provision the utilization of datacenter workloads and push the distribution of utilization as much as possible close to the more energy efficient values. The rest on this paper is organized as follows: Sect. 2 introduces related work. An overview on the proposed method is presented in Sect. 3. Also Sect. 4 describes the basic SVR model. The TSVR model and its hybrid evolutionary algorithm and performance metrics for time series prediction are presented in Sect. 5. Section 6 presents an analysis of the Google dataset by means of auto-correlation and partial auto-correlation functions. We present the experimental results and comparison in Sect. 7. Finally, the conclusion is described in Sect. 8.

2 Related work

Most of the recent papers in CPU and memory load prediction, works on the host load prediction in the grid computing [15,47]. In contrast to the jobs that have been used in grid, the jobs in a cloud are more interactive, shorter and noisier on average [43,44]. Since the cloud workload is nearly 20 times noisier than the load in a grid, predicting the cloud load is more complicated than that of the grid. Still, not an enough accuracy achieved from recent methods when applying them to the cloud [45]. In [41], several challenges of load prediction are discussed and a workload prediction algorithm is proposed. The model-predictive algorithm is used for resource allocation in a way that satisfies the application QoS and reduces operational costs. For this purpose, a second-order autoregressive moving average (ARMA) model is used to predict the workload. Also, fractional autoregressive integrated moving average (FARIMA) is employed to predict a traffic model in [29,35]. The proposed model predicts network traffic considering its long-term and short-term dependent behaviors. In [49] a two-level algorithm is proposed that considers the repeating pattern of the resource usage and then the consequent behavior is predicted by Markov chain [1,13] in host load prediction. Also, the work which is presented in [38] uses data mining algorithms to extract the repeating pattern of the raw data.

Other known models are presented in the literature for the performance prediction, but each one has its shortcoming. For example, gray model [9] has the overshooting problem which causes so much error in the output, especially in the points where the slope suddenly changes from negative to positive and vice versa. Also the ARMA

model and its family, such as FARIMA or seasonal autoregressive integrated moving average (SARIMA) [5,35], have a weak dynamic learning mechanism so fail in the prediction of rapid varying workloads. Artificial neural network (ANN) has under-fitting and over-fitting problems which results in inefficient trained weights [16,20,27]. So the ANN architecture and its parameters should be selected by an expert. Another approach is using adaptive neuro-fuzzy interface system (ANFIS), but it requires a large amount of training data. Since training is not well done in real time, its prediction accuracy is low [28]. To overcome these problems, hybrid models are proposed. The model proposed in [10] refines the performance prediction by combining nonlinear generalized autoregressive conditional heteroskedasticity (NGARCH) and ANFIS models.

Support vector regression (SVR) is also used in a wide range of applications [12]. In [25] a Kalman smoothed SVR load forecasting method is proposed for efficient resources provisioning in the cloud. The authors conclude that the smoothed SVR has slightly better performance in prediction in comparison to the traditional SVR method. Also a dynamic resource provisioning scheme based on the predicted workload is proposed for the cloud. The SVR method has some internal parameters which are not determined accurately in this work and have considerable effect on SVR accuracy. In previous works on SVR these parameters were tuned by trial and error. As a solution, heuristic algorithms are used for SVR parameter tuning [23]. Literature [4] has used optimum SVR to predict network links load at short time scales. PSO heuristic is used for parameter selection in SVR-based time series forecasting [22,26] and genetic algorithm [14,24] is used for this purpose, too. In this paper, we propose a hybrid GA–PSO-based SVR parameter selection method which is used for cloud workload prediction. Also, we inject chaotic time series samples as a booster to the hybrid GA–PSO algorithm for better convergence. The proposed method increases the prediction accuracy of real cloud workload in comparison to the previous algorithms.

3 Overview of the proposed method

In this section, an overview of the proposed method for dynamic resource provisioning in the cloud is provided. As illustrated in Fig. 1 the main component of the system is a cloud controller (CC) which is doing dynamic resource provisioning for the entire cloud infrastructure. Our proposed resource demand prediction (TSVR) model is implemented inside the CC called predictor. This component predicts the workload for the next time, and then estimates the number of resources which needs to be allocated. Then, allocator receives number of necessary resources and accordingly increases or decreases VMs with sufficient resources [18,41,46].

After a user submits his/her request to the cloud provider, the cloud provider assigns the required VMs to the user and prepares connection between the user and his/her VMs. The CC functions is a periodic manner. It includes two main components, namely predictor and allocator. Since the CPU and memory are main bottleneck resources of cloud, we have considered both of them in the resource prediction module for a better and more efficient resource allocation. In each control time interval, the predictor module gathers CPU and memory usages of VMs in cloud. The new data samples are

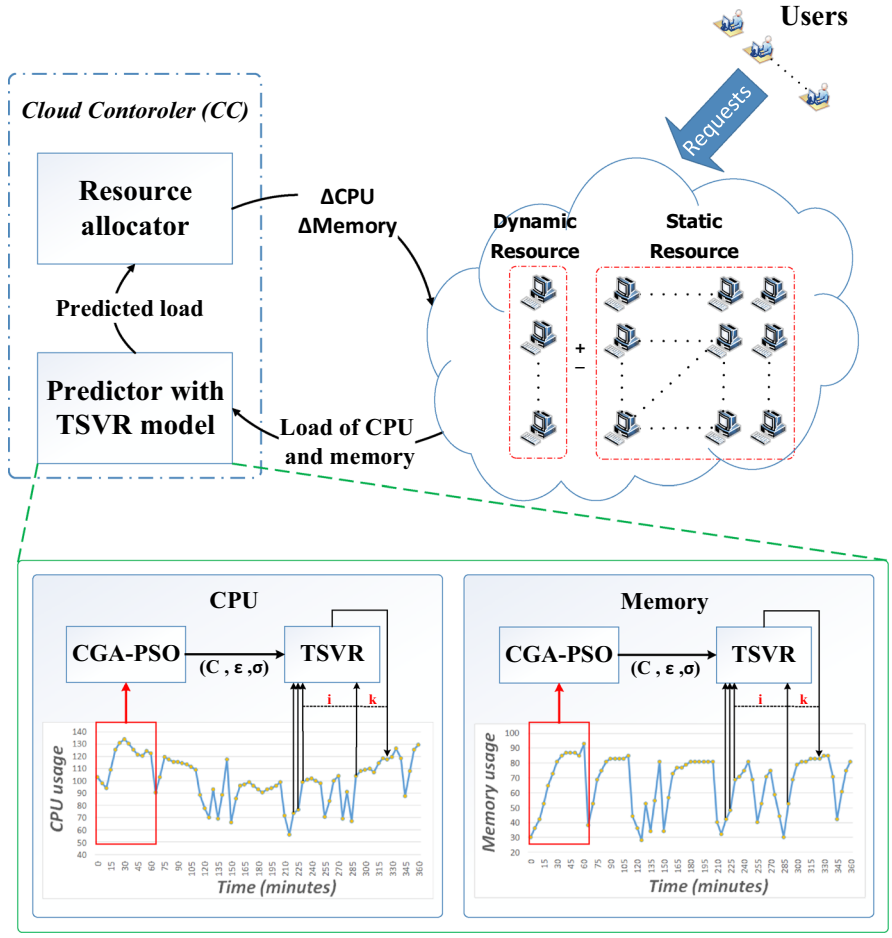

Fig. 1 The proposed method for dynamic resource provisioning in cloud

added to a sliding window buffer which stores i previous load samples of CPU and memory usage separately.

The predictor module executes TSVR prediction algorithm and predict CPU and memory load for k step ahead time interval. According to the difference between predicted and actual usage, the CPU and memory demands (ΔCPU, Δmemory) are calculated and provided in the allocator module. The allocator module decides that how many VMs should be added/deleted to/from the current VMs pool. Also the allocator module determines that how many physical machines should be powered on or powered off during the k step ahead time intervals.

As mentioned before the predictor module employs the TSVR model to predict CPU and memory load separately. As shown in the bottom of Fig. 1, we use real CPU and memory workload from Google cluster. Total length of CPU and memory time series in this study is 6 h and 15 min [11].

The proposed TSVR model is composed of two phases, the parameter-tuning phase and the prediction phase. The goal of the parameter-tuning phase is optimizing three main parameters of SVR model. A combination of GA and PSO as a hybrid evolutionary algorithm is used to carefully tune these essential parameters. The tuning phase, considerably boosts the prediction performance of SVR model. As shown in Fig. 1, for both memory and CPU load time series only a few samples from the beginning of the time series are used in tuning phase. After the tuning phase, TSVR model can enter into the prediction phase and predict a few steps ahead of time series just like the ordinary SVR model. In the following sections more details about the TSVR algorithm are presented.

4 Basics of SVR model for prediction

In this section, we first review the basics of SVR model for time series prediction. The symbols and notations which are used for SVR mathematical representation are shown in Table 1.

The theory of support vector regression first proposed in [12]. SVR is a machine learning tool which uses a nonlinear mapping, Φ, to transform input data $\{(t_i, y_i)\}_{i=1}^{N}$ into a higher-dimensional (probably infinite-dimensional) feature space, in which a theoretically linear function exists to formulate the relationship between input data and output data. This is shown in (1) as follows:

$$f(x) = w^T \phi(t) + b \tag{1}$$

Typically SVR uses a ε-insensitive loss function. This type of loss function allows for deviation from the true target by at most ε. The ε-SVR algorithm does not recognize

Table 1 Symbols and notations of SVR

Symbol	Description
t_i	The input vector
$f(t)$	The prediction values
N	The total number of data set
$\Phi(t_i)$	The feature of inputs
w	Weight coefficients of the SVR function
b	Constant coefficient of the SVR function
ε	The value of epsilon in the insensitive loss function
σ	The value of sigma in the Gaussian kernel
C	The trade-off between the empirical risk and the model flatness
$\zeta\ \zeta^*$	The distance from actual values to the corresponding boundary values of ε-tube
w^*	The optimal weight vector of the regression hyperplane
$\beta_i\ \beta_i^*$	The Lagrangian multipliers
$K(t_i, t_j)$	The kernel function

errors as long as they fall within $\pm\varepsilon$ boundary of the learned function. The SVR model not only fits the training data, but also maintains the highest possible degree of generality so it will be applicable to the future unseen data.

The terms w and b represent weight coefficients and constant coefficient, respectively, which have been learned by the SVR algorithm, also $f(t)$ is the predicted value. The $f(t)$ should loosely fit the training data to avoid over-fitting problem. The SVR accomplishes this by forcing the function to be as flat as possible. Equation (1) is considered as a flat function if w was chosen as small as possible, which can be achieved by minimizing the norm value of w. Hence, the entire problem can be formulated as the convex optimization problem that shown in Eq. (2).

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\|w\|^2 \\ &\text{subject to} && \begin{cases} f(t_i) - y_i \le \varepsilon \\ y_i - f(t_i) \le \varepsilon \end{cases} \end{aligned} \tag{2}$$

This optimization problem assumes that, a function exists which approximates every data pair (t_i, y_i) with an acceptable ε accuracy. To deal with instances where this is not the case, and to account for errors in the training data, Vapnik introduced two slack variables ζ_i and ζ_i^*. These slack variables compute the error for underestimating and overestimating of the actual values. With the addition of these two variables, the optimization problem in Eq. (2) becomes a new optimization problem which is shown in Eq. (3). This problem can be optimized using multi-objective optimization methods such as Pareto optima [7]. However, we have used Lagrange function which is described below.

$$\begin{aligned} &\text{minimize} && \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\left(\zeta_i^* + \zeta_i\right) \\ &\text{subject to} && \begin{cases} y_i - w^T\phi(t_i) - b \le \varepsilon + \zeta_i \\ w^T\phi(t_i) + b - y_i \le \varepsilon + \zeta_i^* \\ \zeta_i, \zeta_i^* \ge 0 \end{cases} \end{aligned} \tag{3}$$

This is the primary form of the objective function for linear SVR. Here, C is a constant known as the penalty factor, which controls the trade-off between the complexity of the function and the frequency in which errors are allowed when estimating the actual value of the training patterns. The large value of C indicates the greater the influence of the empirical error component in the optimization problem. The $(\zeta_i + \zeta_i^*)$ term shows the amount of difference between the estimated value and the target value, which is greater than ε. A dual description of the problem is obtained by constructing a Lagrange function, which introduces dual variables. This Lagrange function is solved to obtain the parameter vector w that is shown in Eq. (4).

$$f(x) = \sum_{i=1}^{N}\left(\beta_i^* - \beta_i\right) K\left(t_i, t_j\right) + b, \tag{4}$$

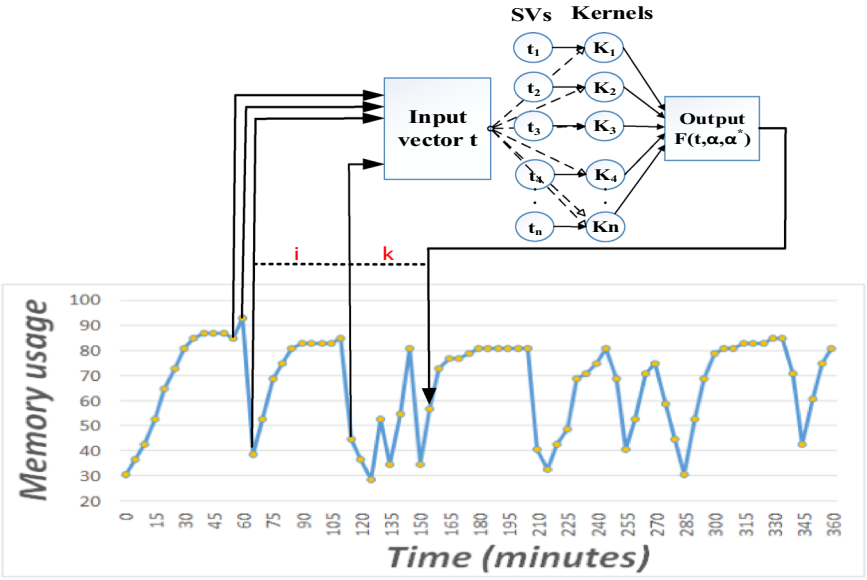

Fig. 2 SVR model for time series prediction (for example, in this figure we used memory workload of Google)

where β_i and β_i^* are Lagrange multipliers and the samples with positive and non-zero β_i and β_i^* are called the support vectors (SVs) and also the $K(t_i, t_j)$ is called the kernel function. The kernel function used in this paper is radial basis function (RBF) as shown in Eq. (5).

$$K\left(t_i, t_j\right) = \exp\left(-0.5 \left\| t_i - t_j \right\|^2 / \sigma^2\right) \tag{5}$$

The SVR scheme for time series prediction is illustrated in Fig. 2.

It should be notified that selection of features and kernel parameter (C, σ and ε) of SVR in the training process can indubitably affect the estimation accuracy [22–24]. Hence inappropriate selection of parameters will greatly degrade the regression performance. For example, if a small value is chosen for ε, then more SVs are employed and the regression function is too complicated (less flat). Also very large value of C, regardless of the model complexity, only considers the minimization of empirical risk. The σ parameter has to be picked out carefully because it defines the structure of the kernel function and thus controls the complexity of the model. Since the exploration space for selection of these parameters is very large; we consider heuristic approaches for suitable SVR parameters selection. Actually a hybrid GA–PSO algorithm is proposed and the obtained results are compared to the results of each GA, PSO and the other methods. It can be observed that the proposed method has the best prediction accuracy. Moreover, we inject a chaotic random series to the TSVR to decrease the probability of premature convergence to a local optimum, and also increasing the diversity of individuals [22–24].

5 The proposed TSVR model

In this section, the details of the proposed TSVR model for time series prediction are explained. First the chaotic sequence is discussed and next the hybrid GA–PSO algorithm with details of genetic and PSO are provided. The symbols and notations which are used for TSVR model mathematical representation are shown in Table 2.

The CGA and the PSO are the two consisting parts of the previously mentioned hybrid CGA–PSO algorithm. The CGA–PSO algorithm is briefly described in this section as illustrated with the flowchart in Fig. 3 and the steps as described below:

1. After generation of the initial population, the CGA algorithm evaluates the initial population by mean absolute percentage error (MAPE) criteria and conducts basic operators (crossover and mutation) to produce new population.

Table 2 Symbols and notations which are used for TSVR model

Symbol	Description
i	Number of iteration
$x^{(i)}$	The value of the chaotic variable x at the ith iteration
μ	The bifurcation parameter of the chaos system
$q_{\max}$	The maximum iteration
W	Inertia weight
$W_{\min}$	Minimum inertia weight
$W_{\max}$	Maximum inertia weight
P_k^g	The global best position among all particles in the swarm
$P_k^{(i)}$	The own best position of the ith particle, $k = C, \sigma, \varepsilon$
$f_{\text{local}}^{(i)}$	The local cost value of ith particle
f_{global}	The global cost value of each iteration
$C1$	Acceleration coefficient
$C2$	Acceleration coefficient
$X_{\min}^{(k)}$	The minimum value of the kth parameters, $k = C, \sigma, \varepsilon$
$X_{\max}^{(k)}$	The maximum value of the kth parameters, $k = C, \sigma, \varepsilon$
$V_{\min}^{(k)}$	The maximum velocity of the kth parameters, $k = C, \sigma, \varepsilon$
$V_{\min}^{(k)}$	The maximum velocity of the kth parameters, $k = C, \sigma, \varepsilon$
$X_k^{(i)}$	The three hyper parameters of an SVR model, $k = C, \sigma, \varepsilon$
$x_k^{(i)}$	The chaotic variables after normalizing $X_k^{(i)}$, $k = C, \sigma, \varepsilon$
$\hat{X}_k^{(i)}$	The three after-crossover hyper parameters of an SVR model, $k = C, \sigma, \varepsilon$
$\hat{x}_k^{(i)}$	The chaotic variables after normalizing $\hat{X}_k^{(i)}$, $k = C, \sigma, \varepsilon$
$\tilde{x}_k^{(i)}$	The chaotic mutation variables, $k = C, \sigma, \varepsilon$ the annealing mutation operation
$V_k^{(i)}$	The three velocity of an SVR model, $k = C, \sigma, \varepsilon$
numpop	Number of chromosomes or particles

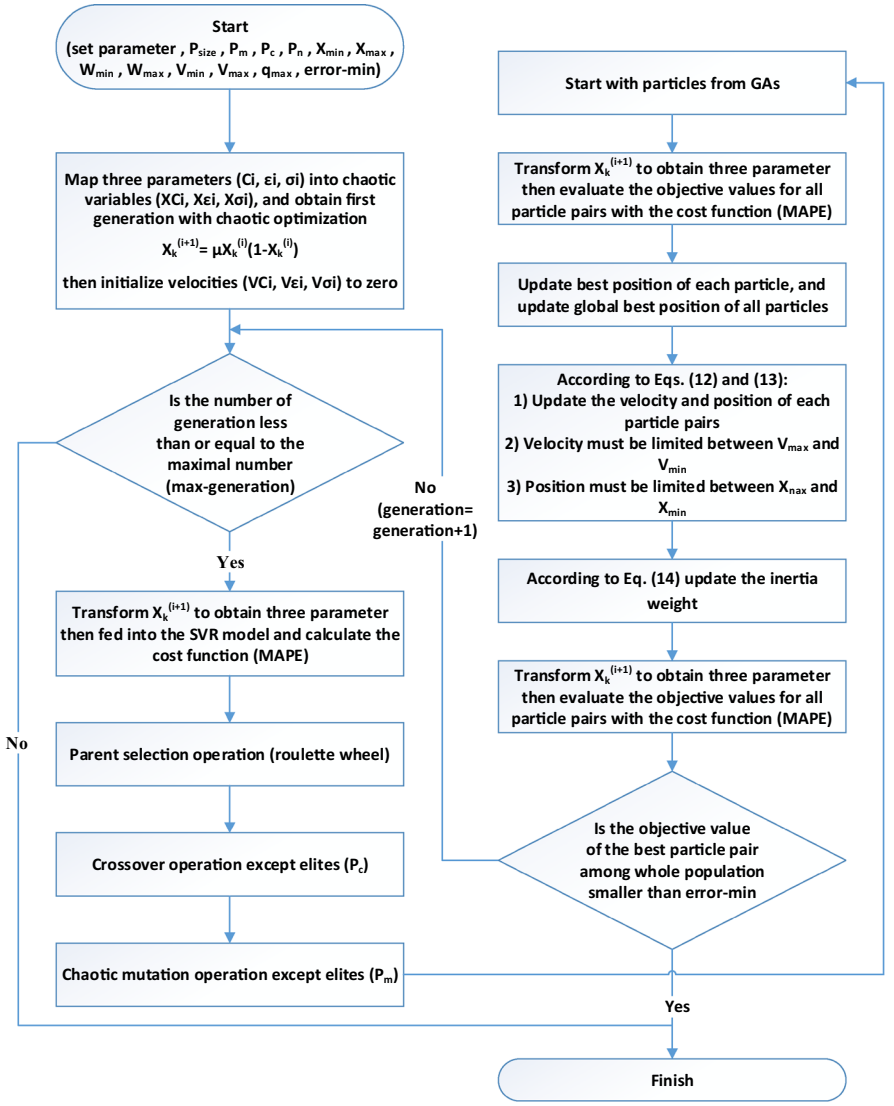

Fig. 3 The hybrid CGA–PSO algorithm's flowchart

2. Next the selected individuals are delivered to the PSO algorithm for more processing.
3. The modified individuals will be sent back to the CGA for the next generation of population.
4. The steps 1–4 are repeated until the stop criteria of the algorithm is reached.

5.1 Generation of chaotic sequence

Logistic map is used to generate a chaotic sequence. The logistic map is a polynomial mapping with degree 2. A chaotic system is an uncorrelated and unpredictable system

Chaotic Sequence

Begin

Initialize: After mapping the chromosomes into the interval [0, 1], population is initialized using the following code;

For each chromosome (i=1 to numpop)

For each of the three parameters of SVR (k=1 to 3)

If $x_k^{(i)}$ is 0.25 or 0.5 or 0.75

$x_k^{(i)} = x_k^{(i)} + 0.01$;

End if

$x_k^{(i)} = 4 * x_k^{(i)} * (1 - x_k^{(i)})$;

Update k^{th} parameter (C, ε, or σ) of SVR model in i^{th} chromosome;

End for

End for

End

Fig. 4 Pseudo-code for chaotization process of a chaotic sequence

that exhibits noise-like behavior through its sensitive dependence on its initial value. It produces chaotic behavior from simple nonlinear dynamical equation. Mathematically, a logistic map can be represented by the one-dimensional function as defined in Eq. (6).

$$x^{(i+1)} = \mu x^{(i)} \left(1 - x^{(i)}\right)$$
$$x^{(i)} \in (0, 1), \quad i = 0, 1, 2, \ldots, \tag{6}$$

where $x^{(i)}$ is a number between zero and one, and represents the value of the chaotic variable x at the ith iteration. Hence, $x^{(0)}$ represents the initial value of chaotic sequence that future behavior of the system is dependent on this value. Also the result changed completely after a very short period with a slight change in the initial value. Further, μ is called the bifurcation parameter within the range of 2.8–4. The dynamic of this system is highly dependent on the value of μ, exhibiting periodicity or chaos. In this paper, we consider $\mu = 4$ and $x \neq 0.25, 0.5$, and 0.75 for production of random numbers [22–24]. For example, the following pseudo-code as shown in Fig. 4 is used to initialize population (step 2 of CGA). Besides, this chaotic sequences are used for mutation (step 7 of CGA).

5.2 The chaotic genetic algorithm (CGA)

An important disadvantage of GA is the reduction of population diversity in each next coming generation, so that a premature convergence condition occurs. The lack of diversity in the initial population of the GA has an effective role in premature convergence. Even if the initial population distributed uniformly, the global optimum

will not be guaranteed to be achieved. In this regard, the chaotic sequence is taken into account in this paper instead of random numbers.

Another drawback of GA is its slow convergence. Thus, the chaotic sequence of this paper is used in two steps of population generation and mutation to improve the speed of convergence in GA [24]. The steps of CGA algorithm are explained as follows:

Step 1: Initialize the main parameters of algorithm (P_{size}, P_m, P_c, P_n, $X_{\min}$, $X_{\max}$, $W_{\min}$, $W_{\max}$, $V_{\min}$, $V_{\max}$, $q_{\max}$, error-min).

Step 2: The three SVR parameters (C, ε and σ) are normalized into the interval [0, 1] by Eq. (7). Then, the population is initialized according to Fig. 4. In this study, the maximum number of population members (chromosomes) has been set 20 (numpop = 20). Three genes for each solution are defined as $X_k^{(i)}$, $k = C, \varepsilon, \sigma$; i is iteration number. In this paper, the range of C is defined as [10, 1000], the range of ε is defined as [0, 10], and the range of σ is defined as [0, 25].

$$x_k^{(i)} = \frac{X_k^{(i)} - X_{\min}^{(k)}}{X_{\max}^{(k)} - X_{\min}^{(k)}}, \quad k = C, \varepsilon, \sigma, \quad i = 1, 2, \ldots, \text{numpop}, \tag{7}$$

where $X_{\max}^{(k)}$ and $X_{\min}^{(k)}$ are the maximum and the minimum of each parameter, respectively. After initializing 20 different solutions with chaotic function, we calculate the decimal format of genes for the next steps by using Eq. (8). Also, we initialize the velocity (V_{ci}, $V_{\varepsilon i}$, and $\mathrm{V}_{\sigma i}$) of particles for the PSO algorithm to zero.

$$X_k^{(i+1)} = X_{\min}^{(k)} + x_k^{(i+1)} \left(X_{\max}^{(k)} - X_{\min}^{(k)} \right) \tag{8}$$

Step 3: If the maximum number of iterations ($q_{\max} = 200$ in this work) is reached, then the best solution would be determined and the algorithm finished; otherwise, go to Step 4.

Step 4: In order to perform natural selection, every individual i is evaluated in terms of its cost value, determined by the cost function. The cost value measures the accuracy of each solution and enables them to be compared. Different cost functions such as MAE, MAPE, RMSE and MSE are discussed in Sect. 7.4. Since the MAPE is a scale-independent measure and does not show the direction of error (i.e., it is in absolute value), we have used this metric as a cost function to evaluate the performance of TSVR model.

Step 5: Selecting individuals for reproduction has a crucial effect on the efficiency of the GA. In this paper, the roulette wheel [24] is used to select chromosomes with less costs.

Step 6: Crossover is the main operator in the GA, which simulates a recombination between two parents. It works on a pair of selected solutions (chromosomes) and recombines them in a certain way in order to generate two offspring. The functionality of the crossover is illustrated in Fig. 5. In this study, probability of crossover is set to 0.6 ($pc = 0.6$).

Step 7: Chaotic mutation is applied to a group of solutions (chromosomes) which are selected with a certain probability ($pm = 0.4$). Before obtaining the chaotic

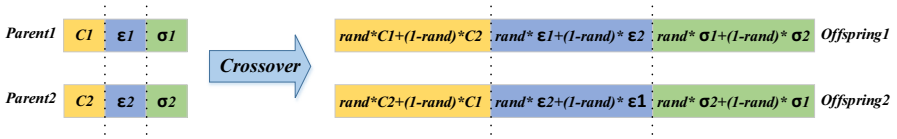

Fig. 5 The crossover mechanism in GA

mutation, the crossover parameters, $\hat{X}_k^{(i)}$, must be normalized into the interval [0, 1] as Eq. (7), which then is showed with $\hat{x}_k^{(i)}$. The chaotic variable, $x_k^{(i)}$, which is defined in Eq. (6) is used as random number in mutation function. Therefore, the ith chaotic mutated variable, $\tilde{x}_k^{(i)}$, is calculated by Eq. (9). It should be noticed that, mutation function does not operate on elites (the number of elites in this method is 2).

$$\tilde{x}_k^{(i)} = \hat{x}_k^{(i)} + \delta\left(2x_k^{(i)} - 1\right), \tag{9}$$

where δ is set to 0.5 in this study. Eventually, the three parameters after-chaotic mutation will be transformed to the actual range according to Eq. (8). After this step, all individuals are transferred to the PSO algorithm for further processing which is described in the next subsection.

5.3 The PSO algorithm

The next steps of the hybrid CGA–PSO employed the PSO algorithm by executing the following steps:

Step 1: Start with the individuals (chromosomes) from the GA as particle positions ($X_k^{(i)}$) of PSO algorithm ($P_k^{(i)}, k = C, \sigma, \varepsilon$) and also initialize the velocities ($V_k^{(i)}$).
Step 2: Calculate the next positions $X_k^{(i+1)}$ for each of the three parameters by PSO algorithm then evaluate the MAPE cost function for all particles.
Step 3: Update the local best position ($f_{\text{local}}^{(i)}$) and the global best position (f_{global}) for each particle. For example, if the cost value in this iteration is better than the prior local best, then update the local best position ($P_k^{(i)}, k = C, \sigma, \varepsilon$) and also its cost value. Determine the best particle pair in the whole population based on the lowest cost value. If the cost value is smaller than f_{global} then update global best position ($P_k^g, k = C, \sigma, \varepsilon$) and its cost value.
Step 4: The velocity of particles are updated according to the Eq. (10):

$$V_k^{(i+1)} = W^* V_k^{(i)} + C1^* \text{rand}^* \left(P_k^{(i)} - x_k^{(i)}\right) + C2^* \text{rand}^* \left(P_k^{(i)} - x_k^{(i)}\right) \tag{10}$$

Then, we should limit the velocity of particles between $V_{\max}$ and $V_{\min}$ values if required and update the position of particles according to the Eq. (11):

$$x_k^{(i+1)} = x_k^{(i)} + V_k^{(i+1)} \tag{11}$$

Also, the position of particles must be limited between $X_{\max}$ and $X_{\min}$ if required.
Step 5: The inertia weight is updated according to the Eq. (12) and must be limited between $W_{\max}$ and $W_{\min}$ if required.

$$W = W_{\min} + \left(\frac{q_{\max} - i}{q_{\max}}\right) * (W_{\max} - W_{\min}) \tag{12}$$

Step 6: Transform $X_k^{(i+1)}$ to obtain the three parameters, then calculate the MAPE cost values for all the particle pairs.
Step 7: If the goal value is achieved or we reached the maximum number of iterations, then the global best position and its cost value would be determined and the algorithm finished; otherwise, go back to the genetic algorithm for the next round.

6 Correlation analysis of Google workload

In order to have a valid prediction, the raw data set should have temporal dependency. Two popular methods for illustration of existence temporal dependency are the auto-correlation function (ACF) and partial auto-correlation function (PACF). These analyses also let us to recognize the possibility of achieving short-term or long-term prediction [8]. In time series analysis, ACF and PACF are more beneficial only for a stationary (i.e., non-seasonal or fixed mean and variance values) time series. A highly noisy data set reduces the accuracy of the prediction. Usually a prediction window with the size of i samples is used. The accuracy of the prediction algorithm depends on the size of prediction window. It is better to determined size of sliding widow dynamically [27]. The suitable size of prediction window can be determined by the auto-correlation function and partial auto-correlation analysis [20]. Figure 6 shows the plots relating to the ACF and PACF of the Google cluster dataset. Usually the correlation analysis such as ACF and PACF are applied to time series to determine the order of the ARMA model of the time series [17]. Regarding Fig. 6, ARMA (1, 1) model is best fitted to the raw data and this model has been used in this paper.

The sliding window technique uses i sample of input workload and then predict k intervals ahead using prediction algorithm. The predictive accuracy have been compared to different sizes of the input window according to the MAPE performance metric (Fig. 7). The MAPE performance of the CPU workload shows better result for a window with smaller size (sizes between 2 to 3) and then deteriorates with increases in the window size. In contrast, according to MAPE in the memory workload as the window size increases, the prediction error decreases.

7 Experimental results

In this section, the experimental results are presented and the performance of the proposed TSVR model is compared with the performances of four rival algorithms namely SVR [23,25,39], ANN [27], ARMA [35,48] and GARCH [10] . A comparison is also

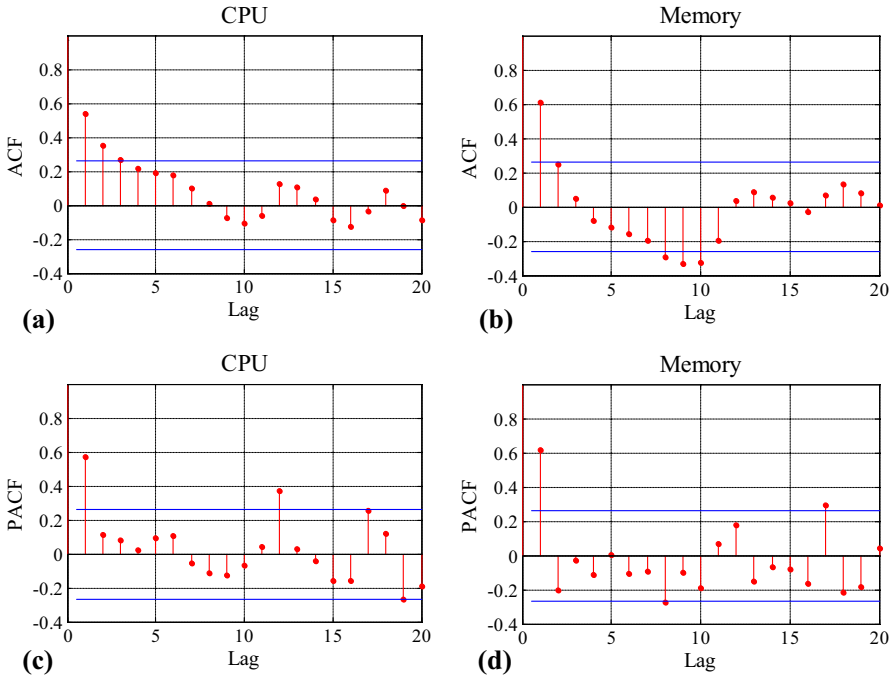

Fig. 6 ACF (**a**, **b**) and PACF (**c**, **d**) analysis for Google's CPU and memory, respectively

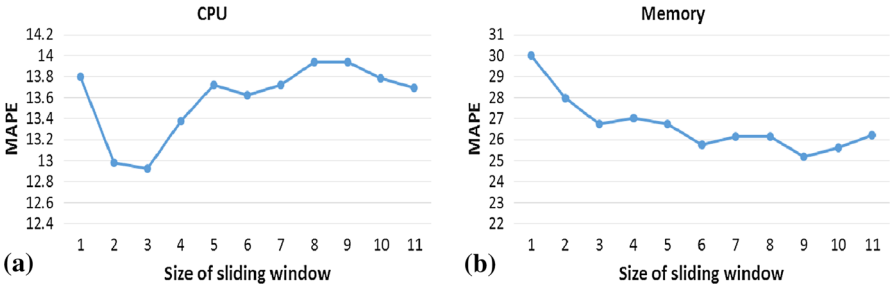

Fig. 7 MAPE vs. size of sliding window

made by the standard performance metrics as presented in Sect. 5.3. As mentioned before, CPU and memory real workloads of Google clusters have been sampled every 5 min. The duration of sampling took 6 h and 15 min. For security reasons, the real CPU and memory usage have been normalized with a linear unknown transformation [11]. To investigate the proposed TSVR algorithm, we implement the algorithm and the other comparable algorithms on a system with Intel dual core T3400 CPU and 2 GB of RAM. In the upcoming subsections, we first show decreasing trend of cost function in CGA–PSO algorithm. Then, we evaluate the prediction efficiency of TSVR algorithm and rival algorithms. Finally, the computational costs of different algorithms are provided.

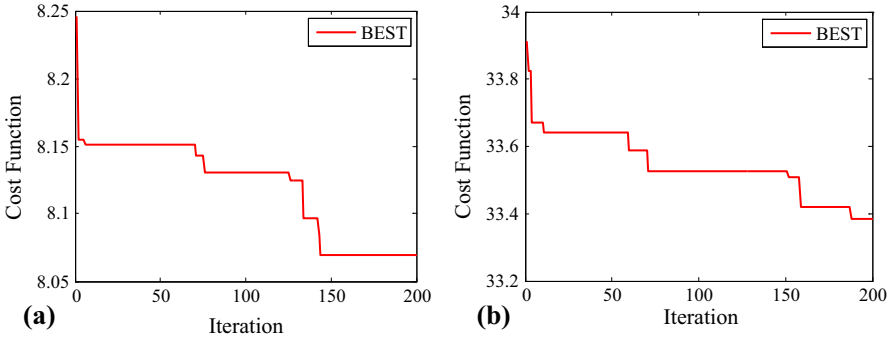

Fig. 8 Cost function of CGA–PSO for CPU time series prediction error (**a**) and memory time series prediction error (**b**)

7.1 Convergence of CGA–PSO algorithm

As shown in Fig. 8 the MAPE cost functions for CPU and memory time series prediction both have a decreasing trend for the best individuals in the population. We conclude that the CGA–PSO optimization algorithm has a suitable convergence rate. We have implemented the experiment for 10 times each of which for both CPU and memory. The results of all of the 10 experiments show decreasing trend which are mostly similar to that in Fig. 8. Hence, we have sufficed to show one of the simulation results as depicted in Fig. 8 to illustrate the convergence of TSVR model. As mentioned, a CPU and a memory of google cluster are used as benchmarks [11].

7.2 Prediction performance of TSVR algorithm

In order to verify the prediction performance of the TSVR model, a simulation study was conducted. We use Google memory and CPU workloads as shown in Fig. 9a, b for the simulation. The predicted values clearly track the actual values for both CPU

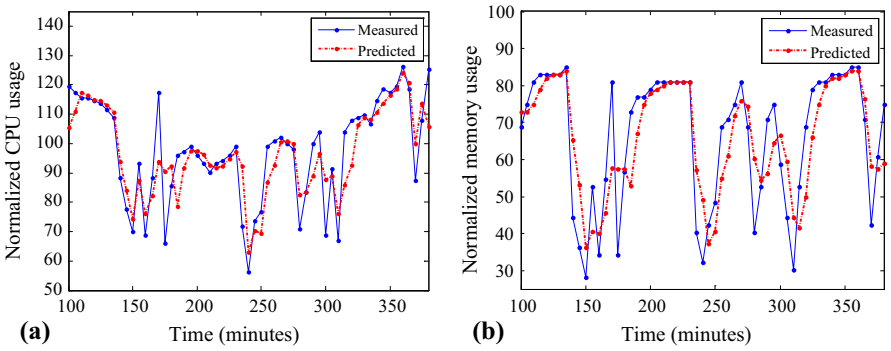

Fig. 9 Predicted CPU and memory loads

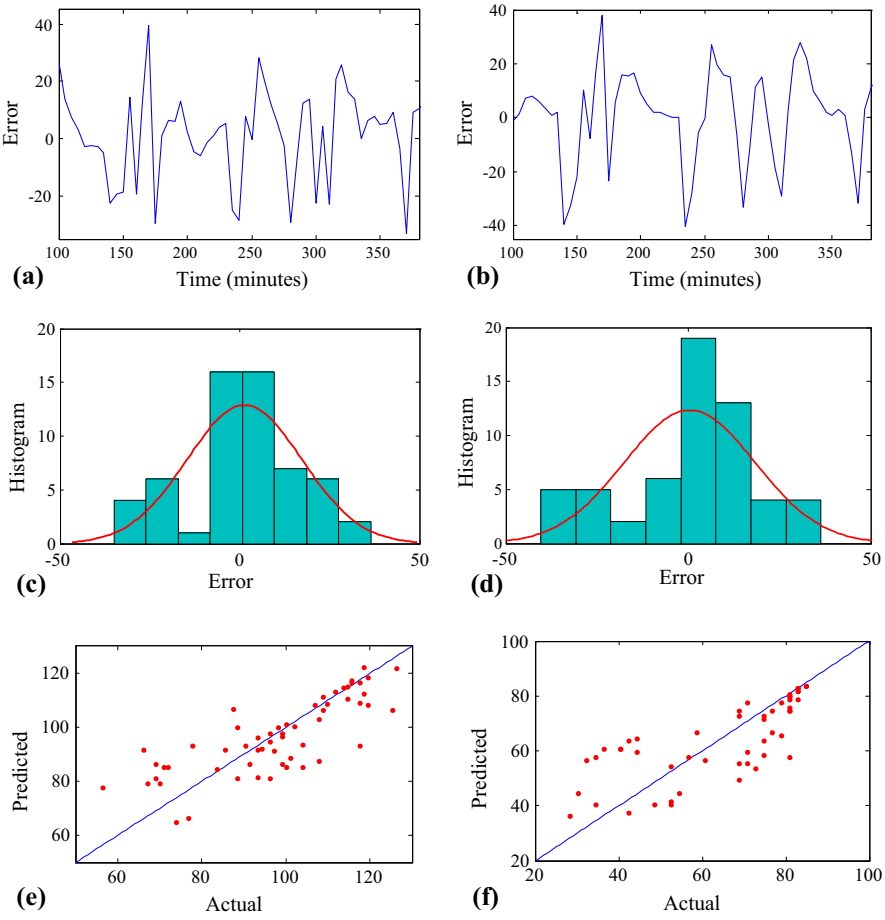

Fig. 10 Error time series (**a**, **b**), its histogram (**c**, **d**) and its dispersion for CPU and memory loads

and memory. However, some sharp variations in workloads have not been predicted accurately. This phenomenon is related to the large variation in workload during a small time interval.

The difference between the predicted and the actual values (error) are presented in Fig. 10a, b. Also, error distributions are presented in Fig. 10c, d. The error histogram of CPU load as shown in Fig. 10c has symmetric shape and more errors are concentrated near zero. A Gaussian distribution function is fitted on this histogram and has the mean value of 1.3224 and the standard deviation of 15.9960. In addition, an error histogram of memory load is shown in Fig. 10d and its Gaussian fitted distribution function has the mean value of 0.3831 and the standard deviation of 17.9907. In Fig. 10e, f, the dispersion error of memory and CPU time series are presented.

The simulation results show that the TSVR model is suitable for workload prediction, but we should compare the TSVR performance with other algorithms which are described in the next section.

7.3 Comparison results between the prediction algorithms

Several time series prediction algorithms are presented in recent papers. Among these algorithms ARMA (AR = 1, MA = 1) [35,48], GARCH ($p = 1, q = 1$) [10], ANN (input layer = 4, hidden layer = 2, output layer = 1) [27], and SVR (RBF kernel function) [23,25,39] have been simulated in this paper. The prediction results for both CPU and memory workloads are presented in Figs. 11 and 12, respectively.

The TSVR model achieves better prediction results in comparison to the other algorithms. As can be seen in Figs. 11 and 12, the red line relating to the TSVR model prediction results is in higher conformity with the black line (which is related to real data). The ANN can be considered as the second best method for prediction. Except for the slightly large variations from the original time series in a few number of extremum points, the ANN tracks time series variations suitably. Also, ARMA model shows large variations around the original time series. Finally, the GARCH model damps time series variations and provides a smooth version of the original time series which leads to a large amount of error in prediction.

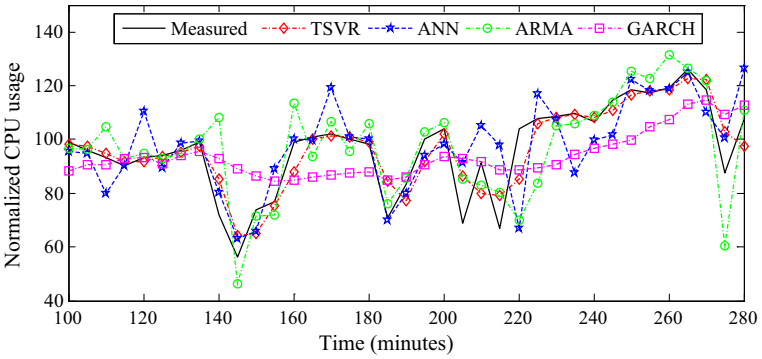

Fig. 11 CPU load prediction

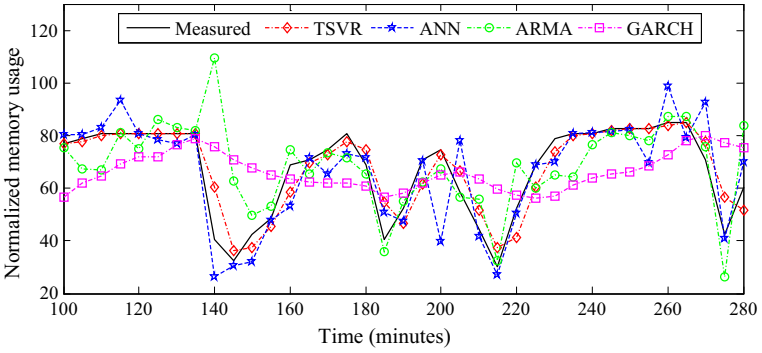

Fig. 12 Memory load prediction

7.4 Performance metrics

Each metric provides some kind of difference between actual and predicted values of the time series. In each of the forthcoming definitions n is the total number of predicted samples, a_i is the actual value and f_i is the predicted value of ith sample in the time series. These metrics shown in Table 3 are used to evaluate the performance of prediction methods [10,17,22,27].

All performance metrics described above are used to compare different algorithms as illustrated in Tables 4 and 5 for CPU and memory, respectively. Also SVRCGA and SVRCPSO in this paper are two sub-methods which are derived from the TSVR model. The SVRCGA model only considers chaotic genetic algorithm for tuning the SVR parameters and the SVRCPSO method employs only chaotic PSO algorithm for this purpose.

The results which are provided in Tables 4 and 5 show memory load prediction by the TSVR model is considerably better than CPU load prediction for Google workload. As shown in Tables 4 and 5, for example, considering MAPE; in comparison to the simple SVR, the TSVR model is 2.83 % better in CPU load prediction and 6.57 % better in memory load prediction. In addition, the TSVR model is 8.03 and 14.18 % better than GARCH in CPU and memory load prediction, respectively.

Table 3 Prediction error metrics

Performance metric	Equation
Mean absolute error (MAE)	$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} \lvert f_i - a_i \rvert$
Mean absolute percentage error (MAPE)	$\text{MAPE} = \frac{1}{n}\sum_{i=1}^{n} \frac{\lvert f_i - a_i \rvert}{a_i}$
Mean square error (MSE)	$\text{MSE} = \sum_{i=1}^{n} \frac{(f_i - a_i)^2}{n}$
Root mean square error (RMSE)	$\text{RMSE} = \sqrt{\sum_{i=1}^{n} \frac{(f_i - a_i)^2}{n}}$

Table 4 Prediction error metrics for Google CPU workload

	MAE	MAPE (%)	RMSE	MSE
ARMA (1, 1)	12.9932	14.1015	16.3727	268.0660
GARCH ($p = 1, q = 1$)	13.4259	14.8407	15.8157	250.1359
ANN (4, 2, 1)	12.6215	14.0716	15.9465	254.2917
SVR ($C = 200, \sigma = 20, \varepsilon = 5$)	12.3817	14.0465	15.9928	255.7702
SVRCGA ($C = 122.78, \sigma = 25, \varepsilon = 3.40$)	12.0943	13.8317	15.8339	250.7111
SVRCPSO ($C = 119.85, \sigma = 25, \varepsilon = 3.74$)	12.1345	13.8622	15.8543	251.3603
TSVR ($C = 127.80, \sigma = 22.90, \varepsilon = 1.92$)	11.8972	13.6493	15.6563	245.1207

Table 5 Prediction error metrics for Google memory workload

	MAE	MAPE (%)	RMSE	MSE
ARMA (1, 1)	14.0379	28.1145	20.0164	400.6561
GARCH ($p = 1, q = 1$)	14.8309	27.6339	18.5308	343.3905
ANN (4, 2, 1)	13.8488	25.5814	18.5064	342.4854
SVR ($C = 200, \sigma = 20, \varepsilon = 5$)	13.2660	25.3811	17.6112	310.1529
SVRCGA ($C = 612.95, \sigma = 16.26, \varepsilon = 5$)	13.1059	24.7306	17.4506	304.5246
SVRCPSO ($C = 576.80, \sigma = 15.97, \varepsilon = 5.11$)	13.1267	24.7723	17.4911	305.9384
TSVR ($C = 985.69, \sigma = 16.07, \varepsilon = 5.06$)	12.7413	23.7141	17.0448	290.5236

7.5 Evaluation of *k* step ahead prediction accuracy

In this section, we are going to evaluate the efficiency of the proposed TSVR model in multi-step ahead prediction. As shown in Fig. 13, we can observe the same behavior for different error metrics for both CPU and memory predicted load up to ten steps ahead. In comparison to the machine learning methods, the two statistical methods, ARMA and GARCH, have a large amount of prediction error when increasing the number of steps ahead for prediction. However, the memory load prediction error in GARCH method keeps decreasing after the 5th step up to 10th step, but never reaches to the first step error. The reason of this decrement in error can be understood from Fig. 6b. As shown in this figure, the ACF of memory time series after five steps of lag tends to decrease the same as errors in GARCH method. This phenomenon shows that the correlation between the time series samples increases up to the 10th step.

The error trends in the ANN method is above the other kernel-based machine learning methods (all SVR method that is tuned or not tuned). The simple SVR method behaves better than ANN, but its prediction error changes negligibly with different prediction steps and shows a nearly constant error rate. Three different optimization methods for SVR parameters selection have led to better prediction and lower total error in spite of changing the number of steps in ahead prediction. As illustrated in Fig. 13 the SVRCGA, SVRCPSO and TSVR methods provide better results than the other ones. Also the TSVR is slightly better than SVRCGA and SVRCPSO especially in the early steps due to the better SVR parameters selection.

7.6 Computational cost comparison

In this section, we are going to compare the computational overhead of our proposed algorithm. There is a trade-off in every prediction method between the prediction accuracy and the computational cost of prediction. Usually better methods have higher computational cost. As illustrated in Table 6, we can divide the total computational cost of the algorithm into three terms.

The evolutionary phase which is exclusively used for our heuristic methods, is a time-consuming computation. However, this term of computation is executed offline

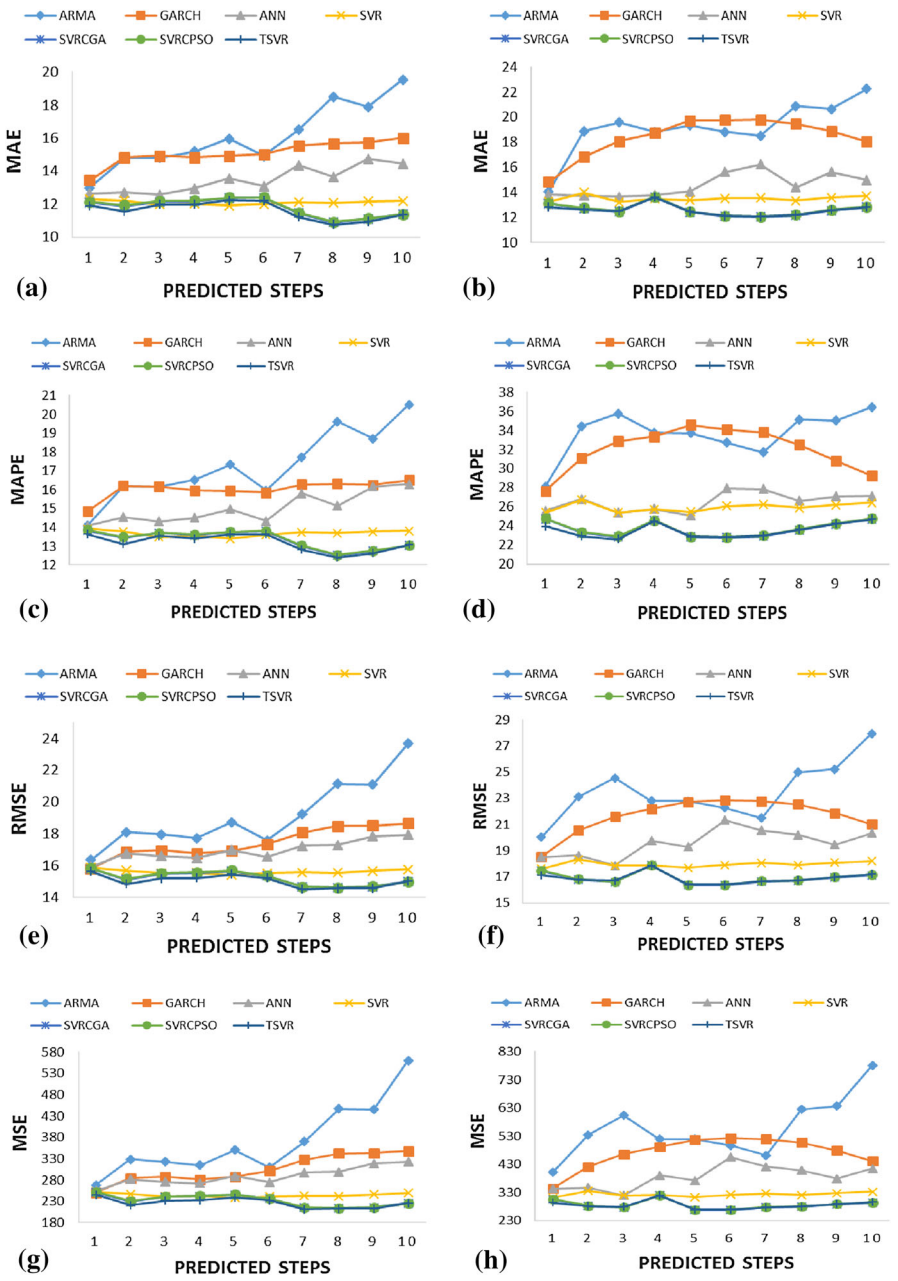

Fig. 13 Comparison of k step ahead prediction accuracy by MAE (**a**, **b**), MAPE (**c**, **d**), RMSE (**e**, **f**) and MSE (**g**, **h**) metrics for CPU and memory, respectively

and only once in the tuning stage of the algorithm. So it will be omitted and not be considered in the training and the prediction phases. Besides, the chaotic sequence has been used in our method which is less time consuming rather than the random

Table 6 Computation time (in seconds) for different algorithms in different phases

	ARMA	GARCH	ANN	SVR	SVRCGA	SVRPSO	TSVR
Evolutionary phase	NA	NA	NA	NA	25.8993	19.9319	45.9093
Train	2.6848	0.3261	0.2311	0.00039	0.00042	0.00037	0.00037
Prediction	0.1852	0.0290	0.0467	0.00017	0.00013	0.00013	0.00016

function. According to the simulation, random function takes 8.0545e−06 s while chaotic sequence takes 3.3165e−06 s for production of every random number.

In the training phase, the SVR, SVRCGA, SVRPSO and TSVR methods are trained more than 550.2381 times faster than the other methods. Especially the ARMA method spends much more time in the training phase. Also the prediction time of the SVR, SVRCGA, SVRPSO and TSVR methods are less than 170.5882 times slower than the other methods. So, we can conclude that despite the offline evolutionary phase, the TSVR method achieves better or the same computational cost in comparison to the rival methods while gains better prediction accuracy.

8 Conclusion

In this paper, a new method for host load prediction in cloud infrastructure is proposed. Kernel-based methods have been used to predict CPU and memory loads in the cloud. Especially, three different heuristic methods have been employed to carefully tune the SVR parameters. First, GA and PSO algorithms are used separately for parameter tuning. Then, a hybrid GA–PSO algorithm is proposed whose results show higher prediction performance in comparison to other rival algorithms.

In this work, the efficiency of using chaotic sequence in the original genetic algorithm is also illustrated. The chaotic sequence can help to increase the exploration and the diversity in the search space. At the same time, it reduces the computational burden of generating random numbers in comparison to the traditional genetic algorithm. We named our algorithm TSVR and implemented it to predict real Google cloud CPU and memory workloads. The only drawback of the TSVR algorithm which is common in the heuristic algorithms is that it takes a long time to select suitable SVR parameters. Since this parameter-tuning phase occurs once in the beginning of the algorithm, it does not affect the train and the prediction phase of the TSVR method. So it can be neglected easily.

Prediction results are compared to those of some popular methods which have been presented in recent papers. Standard metrics were used for comparisons. The results achieved from simulation confirm that the TSVR algorithm has higher performance in comparison to the other algorithms. For example, considering MAPE, the TSVR model is 2.83 and 6.57 % better than simple SVR for prediction CPU load and memory load, respectively. In addition, the TSVR model is 8.03 % better for prediction CPU load and 14.18 % better for memory load in comparison to the GARCH method. These results achieved due to a better parameter tuning in TSVR model. Also, the TSVR

model has less computational cost in the training phase than ANN model and provides a higher precision than simple SVR.

References

1. Akioka S, Muraoka Y (2004) Extended forecast of CPU and network load on computational grid. In: IEEE international symposium on cluster computing and the grid, 2004. CCGrid 2004, 19–22 April 2004, pp 765–772. doi:10.1109/CCGrid.2004.1336711
2. Sanaei Z, Abolfazli S, Gani A, Buyya R (2014) Heterogeneity in mobile cloud computing: taxonomy and open challenges. IEEE Commun Surv Tutor 16(1):369–392. doi:10.1109/SURV.2013.050113.00090
3. Barroso LA, Holzle U (2007) The case for energy-proportional computing. Computer 40(12):33–37. doi:10.1109/MC.2007.443
4. Hayes B (2008) Cloud computing. Commun ACM 51(7):9–11. doi:10.1145/1364782.1364786
5. Box GMJGEP, Reinsel GC (1994) Time series analysis: forecasting & control. Prentice Hall, NJ
6. Buyya R, Yeo CS, Venugopal S, Broberg J, Brandic I (2009) Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. Future Gener Comput Syst 25(6):599–616. doi:10.1016/j.future.2008.12.001
7. Caramia M, Dell'Olmo P (2008) Multi-objective management in freight logistics: Increasing capacity, service level and safety with optimization algorithms. Springer, Berlin
8. Casolari S, Colajanni M (2009) Short-term prediction models for server management in internet-based contexts. Decis Support Syst 48(1):212–223. doi:10.1016/j.dss.2009.07.014
9. Islam S, Keung J, Lee K, Liu A (2012) Empirical prediction models for adaptive resource provisioning in the cloud. Future Gener Comput Syst 28(1):155–162. doi:10.1016/j.future.2011.05.027
10. Chang BR, Tsai HF (2009) Novel hybrid approach to data-packet-flow prediction for improving network traffic analysis. Appl Soft Comput 9(3):1177–1183. doi:10.1016/j.asoc.2009.03.003
11. Chen YGA-S, Griffith R, Katz R (2010) Analysis and lessons from a publicly available google cluster trace. University of California, Berkeley
12. Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297. doi:10.1007/BF00994018
13. Jing B, Zhiliang Z, Ruixiong T, Qingbo W (2010) Dynamic provisioning modeling for virtualized multi-tier applications in cloud data center. In: 2010 IEEE 3rd international conference on cloud computing (CLOUD), 5–10 July 2010, pp 370–377. doi:10.1109/CLOUD.2010.53
14. Deng S, Yoshiyama K, Mitsubuchi T, Sakurai A (2015) Hybrid method of multiple kernel learning and genetic algorithm for forecasting short-term foreign exchange rates. Comput Econ 45(1):49–89. doi:10.1007/s10614-013-9407-6
15. Dinda P, O'Hallaron D (2000) Host load prediction using linear models. Cluster Comput 3(4):265–280. doi:10.1023/A:1019048724544
16. Egrioglu E, Yolcu U, Aladag C, Bas E (2015) Recurrent multiplicative neuron model artificial neural network for non-linear time series forecasting. Neural Process Lett 41(2):249–258. doi:10.1007/s11063-014-9342-0
17. Fan J, Yao Q (2003) Nonlinear time series: nonparametric and parametric methods. Springer, Berlin
18. Dinda P, O'Hallaron D (2000) Host load prediction using linear models. Cluster Comput 3(4):265–280. doi:10.1023/A:1019048724544
19. Yongwei W, Yulai Y, Guangwen Y, Weimin Z (2007) Load prediction using hybrid model for computational grid. In: 2007 8th IEEE/ACM international conference on grid computing, 19–21 Sept. 2007, pp 235–242. doi:10.1109/GRID.2007.4354138
20. Sheng D, Kondo D, Cirne W (2012) Characterization and comparison of cloud versus grid workloads. In: 2012 IEEE international conference on cluster computing (CLUSTER), 24–28 Sept. 2012, pp 230–238. doi:10.1109/CLUSTER.2012.35
21. Hayes B (2008) Cloud computing. Commun ACM 51(7):9–11. doi:10.1145/1364782.1364786
22. Hong W-C (2009) Chaotic particle swarm optimization algorithm in a support vector regression electric load forecasting model. Energy Convers Manag 50(1):105–117. doi:10.1016/j.enconman.2008.08.031
23. Hong W-C, Dong Y, Zheng F, Wei SY (2011) Hybrid evolutionary algorithms in a SVR traffic flow forecasting model. Appl Math Comput 217(15):6733–6747. doi:10.1016/j.amc.2011.01.073

24. Hong W-C, Dong Y, Chen L-Y, Wei S-Y (2011) SVR with hybrid chaotic genetic algorithms for tourism demand forecasting. Appl Soft Comput 11(2):1881–1890. doi:10.1016/j.asoc.2010.06.003
25. Hu R, Jiang J, Liu G, Wang L (2014) Efficient resources provisioning based on load forecasting in cloud. Sci World J 2014:12. doi:10.1155/2014/321231
26. Hu W, Yan L, Liu K, Wang H (2015) A short-term traffic flow forecasting method based on the hybrid PSO-SVR. Neural Process Lett. doi:10.1007/s11063-015-9409-6
27. Islam S, Keung J, Lee K, Liu A (2012) Empirical prediction models for adaptive resource provisioning in the cloud. Future Gener Comput Syst 28(1):155–162. doi:10.1016/j.future.2011.05.027
28. Jang JSR (1993) ANFIS: adaptive-network-based fuzzy inference system. IEEE Trans Syst Man Cybern 23(3):665–685. doi:10.1109/21.256541
29. Chang B, Chen S-H, Tsai H (2006) Forecasting the flow of data packets for website traffic analysis—ASVR-tuned ANFIS/NGARCH approach. In: King I, Wang J, Chan L-W, Wang D (eds) Neural information processing, vol 4233. Lecture notes in computer science. Springer, Berlin, pp 925–933. doi:10.1007/11893257_102
30. Box GEP, Jenkins GM, Reinsel GC (1994) Time series analysis: forecasting & control. Prentice Hall, NJ
31. Khan AR, Othman M, Madani SA, Khan SU (2014) A survey of mobile cloud computing application models. IEEE Commun Surv Tutor 16(1):393–413. doi:10.1109/SURV.2013.062613.00160
32. Koomey J (2011) Growth in data center electricity use 2005 to 2010. Analytics Press, Oakland
33. Jang JSR (1993) ANFIS: adaptive-network-based fuzzy inference system. IEEE Trans Syst Man Cybern 23(3):665–685. doi:10.1109/21.256541
34. Chang BR, Tsai HF (2009) Novel hybrid approach to data-packet-flow prediction for improving network traffic analysis. Appl Soft Comput 9(3):1177–1183. doi:10.1016/j.asoc.2009.03.003
35. Nourikhah H, Akbari M, Kalantari M (2015) Modeling and predicting measured response time of cloud-based web services using long-memory time series. J Supercomput 71(2):673–696. doi:10.1007/s11227-014-1317-4
36. Hu R, Jiang J, Liu G, Wang L (2014) Efficient resources provisioning based on load forecasting in cloud. Sci World J 2014:12. doi:10.1155/2014/321231
37. Pedram M (2012) Energy-efficient datacenters. IEEE Trans Comput-Aided Des Integr Circuits Syst 31(10):1465–1484. doi:10.1109/TCAD.2012.2212898
38. Bermolen P, Rossi D (2008) Support vector regression for link load prediction. In: 4th international telecommunication networking workshop on QoS in multiservice IP networks, 2008. IT-NEWS 2008, 13-15 Feb. 2008, pp 268–273. doi:10.1109/ITNEWS.2008.4488164
39. Hong W-C (2009) Chaotic particle swarm optimization algorithm in a support vector regression electric load forecasting model. Energy Convers Manag 50(1):105–117. doi:10.1016/j.enconman.2008.08.031
40. Ros S, Caminero A, Hernández R, Robles-Gómez A, Tobarra L (2014) Cloud-based architecture for web applications with load forecasting mechanism: a use case on the e-learning services of a distant university. J Supercomput 68(3):1556–1578. doi:10.1007/s11227-014-1125-x
41. Deng S, Yoshiyama K, Mitsubuchi T, Sakurai A (2015) Hybrid method of multiple kernel learning and genetic algorithm for forecasting short-term foreign exchange rates. Comput Econ 45(1):49–89. doi:10.1007/s10614-013-9407-6
42. Sanaei Z, Abolfazli S, Gani A, Buyya R (2014) Heterogeneity in mobile cloud computing: taxonomy and open challenges. IEEE Commun Surv Tutor 16(1):369–392. doi:10.1109/SURV.2013.050113.00090
43. Yang J, Liu C, Shang Y, Cheng B, Mao Z, Liu C, Niu L, Chen J (2014) A cost-aware auto-scaling approach using the workload prediction in service clouds. Inf Syst Front 16(1):7–18. doi:10.1007/s10796-013-9459-0
44. Chen Y, Ganapathi AS, Griffith R, Katz R (2010) Analysis and lessons from a publicly available google cluster trace. University of California, Berkeley
45. Yang Q, Peng C, Zhao H, Yu Y, Zhou Y, Wang Z, Du S (2014) A new method based on PSR and EA-GMDH for host load prediction in cloud computing system. J Supercomput 68(3):1402–1417. doi:10.1007/s11227-014-1097-x
46. Yang J, Liu C, Shang Y, Cheng B, Mao Z, Liu C, Niu L, Chen J (2014) A cost-aware auto-scaling approach using the workload prediction in service clouds. Inf Syst Front 16(1):7–18. doi:10.1007/s10796-013-9459-0
47. Fan J, Yao Q (2003) Nonlinear time series: nonparametric and parametric methods. Springer, Berlin

48. Rongdong H, JingFei J, Guangming L, Lixin W (2013) CPU load prediction using support vector regression and Kalman smoother for cloud. In: 2013 IEEE 33rd international conference on distributed computing systems workshops (ICDCSW), 8–11 July 2013, pp 88–92. doi:10.1109/ICDCSW.2013.60
49. Zhang Q, Zhani MF, Zhang S, Zhu Q, Boutaba R, Hellerstein JL (2012) Dynamic energy-aware capacity provisioning for cloud computing environments. Paper presented at the proceedings of the 9th international conference on autonomic computing, San Jose, California, USA