Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A mixture of HMM, GA, and Elman network for load prediction in cloud-oriented data centers*

Da-yu XU†[1,2], Shan-lin YANG[1], Ren-ping LIU[2]

([1]*MOE Key Laboratory of Process Optimization and Intelligent Decision-Making, Hefei University of Technology, Hefei 230009, China*)

([2]*Information and Communication Technologies Centre, CSIRO, Sydney 2122, Australia*)

†E-mail: xdyhfut@163.com

Received Apr. 25, 2013; Revision accepted Sept. 16, 2013; Crosschecked Oct. 15, 2013

Abstract: The rapid growth of computational power demand from scientific, business, and Web applications has led to the emergence of cloud-oriented data centers. These centers use pay-as-you-go execution environments that scale transparently to the user. Load prediction is a significant cost-optimal resource allocation and energy saving approach for a cloud computing environment. Traditional linear or nonlinear prediction models that forecast future load directly from historical information appear less effective. Load classification before prediction is necessary to improve prediction accuracy. In this paper, a novel approach is proposed to forecast the future load for cloud-oriented data centers. First, a hidden Markov model (HMM) based data clustering method is adopted to classify the cloud load. The Bayesian information criterion and Akaike information criterion are employed to automatically determine the optimal HMM model size and cluster numbers. Trained HMMs are then used to identify the most appropriate cluster that possesses the maximum likelihood for current load. With the data from this cluster, a genetic algorithm optimized Elman network is used to forecast future load. Experimental results show that our algorithm outperforms other approaches reported in previous works.

Key words: Cloud computing, Load prediction, Hidden Markov model, Genetic algorithm, Elman network
doi:10.1631/jzus.C1300109 **Document code:** A **CLC number:** TP391

1 Introduction

Demand for high-performance computing infrastructures from modern resource intensive enterprises and scientific applications has been growing recently. Server-side computing and the exploding popularity of Internet services have rapidly made large-scale data centers an integral part of the Internet fabric (Marston *et al*., 2011). Cloud computing, which has been introduced recently, is a paradigm that streamlines on-demand provisioning of software, hardware, and data services. It also provides end users with flexible and scalable services that are accessible through the Internet (Armbrust *et al*., 2009). It allows organizations to outsource their computation needs to the cloud and eliminates the need for these organizations to maintain their own computing infrastructure (Vaquero *et al*., 2009). To ensure high scalability, flexibility, and cost effectiveness, cloud platforms need to be able to quickly plan and provide resources, which will ensure that supporting infrastructures can closely match the needs of various applications. Cloud platforms require mechanisms to continuously characterize and predict their loads.

Load prediction is a crucial issue for efficient resource utilization in a dynamic cloud computing environment. Based on future load prediction and an estimate of the future performance of each virtual machine (VM), management middleware in cloud can allocate enough resources for running services while avoiding costly over provisioning. Hence, the problem for elastic resource management involves

* Project (No. 71131002) supported by the National Natural Science Foundation of China

deciding when, how much, and whether to scale VM up or down. Scaling can be done horizontally by increasing or decreasing the number of allocated VMs. Scaling can also be performed vertically by changing the hardware configuration for the CPU, memory, and input/output, among others, of already running VMs. Given the scale of the current and future loads in cloud data centers and services, autonomic management can be implemented for clouds (Calheiros *et al*., 2011). Effective load prediction will help administrators take appropriate actions in preventing the system suffering from traffic surge or the Slashdot effect (Bauer and Adams, 2012), which is caused by high load.

In the past few years, some studies have been devoted to load prediction in cloud computing environments. Caron *et al*. (2010) presented a Knuth-Morris-Pratt (KMP) based string matching algorithm to forecast the on-demand cloud computing resource workload. KMP uses a set of historical data to identify similar load patterns in a current window for records that occurred in the past. The algorithm predicts cloud workload by interpolating what follows after the identified patterns from the historical data. Duy *et al*. (2010) proposed a strategy for energy conservation in cloud infrastructures using a neural predictor to forecast future load. They attempted to reduce the number of running servers to minimize energy consumption while ensuring that the service level agreement (SLA) requirements are met by using accurate load prediction. Saripalli *et al*. (2011) used a load tracker function to obtain a representative view of the cloud workload trend from measured raw data. They used cubic spline interpolation to predict loads at any given timestamp (in the future). This algorithm can predict future resource loads and detect hot spots under real-time constraints. Islam *et al*. (2012) presented empirical prediction models for adaptive resource prediction in a cloud environment and developed load prediction based resource measurement and provisioning strategies. They employed a neural network and linear regression model to predict upcoming resource demands. Their method offers more adaptive resource management for applications that can be achieved through on-demand resource allocation in cloud computing. Khan *et al*. (2012) proposed a new means of characterizing correlated workload patterns across servers resulting from the dependencies of applications that run on these servers. They treated server performance data samples as multiple time series and introduced a co-clustering algorithm to identify server groups and time periods when certain workload patterns appear in a group. A hidden Markov model (HMM) was used to explore temporal correlations in workload pattern changes. Finally, server behavior could be predicted based on the server groups. The proposed method takes into account the load characteristics in cloud computing and provides a better prediction performance.

We argue that the key to accurate load prediction in cloud computing is proper modeling of the relationship between historic data and future values, and a proper understanding of cloud computing backend workloads. Benson *et al*. (2010), Mishra *et al*. (2010), Di *et al*. (2012a), and Reiss *et al*. (2012) analyzed the characteristics of cloud load and studied the differences between the cloud data center and other Grid or high performance computing (HPC) systems from the perspectives of resource utilization, resource requests, and job execution time, among others. On the basis of their work, we summarized the main differences between cloud computing and other Grid or HPC systems: (1) Most jobs in the cloud are small, and they are generally slighter than those in Grid or HPC; (2) The execution time of cloud jobs is shorter than that of Grid or HPC jobs; (3) Load in clouds has higher noise and changes frequently in shorter periods, whereas the host load of Grid or HPC systems is more stable during relatively long periods. These unavoidable dynamic load changes increase barriers to better prediction performance in a cloud computing environment.

The motivation for our work is to develop an approach that can discover load patterns automatically, classify the load objectively, and forecast future load accurately. We take the perspective of a cloud service provider who hosts multiple sites with vast amounts of virtualized servers, such as the Amazon Elastic Compute Cloud (EC2). We focus on short-term load prediction for real-time cloud computing resource allocation and management. Based on short-term prediction of future load (Bennani and Menasce, 2005; Ardagna *et al*., 2012), the cloud administrator will be able to implement appropriate measures at a more fine-grain time scale (e.g., from 5 to 10 min). As shown in Fig. 1, we propose a hybrid approach to

forecast future load in cloud computing. A hybrid model has both linear and nonlinear modeling abilities and thus can be a good alternative for predicting time series data. Combining various models can capture different aspects of underlying patterns (Zhang, 2003). Recently, researchers proved that hybrid approaches integrating classification methods achieve greater prediction accuracy than do single models (Yu *et al*., 2008; Hassan *et al*., 2012; Khashei *et al*., 2012).

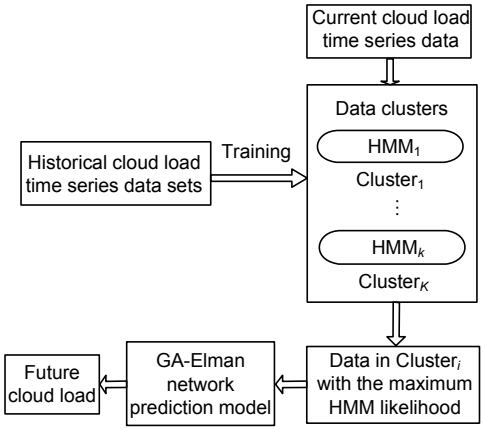

Fig. 1 Block diagram of the proposed prediction process

Our hybrid approach is composed of three major phases: load subsequence extraction, load subsequence clustering, and future load prediction. First, we adopt the fixed size overlapping sliding window (FSOSW) (Arasu and Manku, 2004) to extract load subsequences from historical time series data. Second, we employ HMM based unsupervised data clustering to identify natural groupings of cloud computing load data from a large set of historical traces to concisely represent the system behavior. Many clustering techniques, such as *K*-means, *C*-means, and the fuzzy theory based clustering method (Kaur *et al*., 2013), have been studied by researchers in various areas, such as statistics, pattern recognition, and machine learning. However, most existing clustering approaches were assumed to have a pre-specified number of clusters. This assumption is not true for every situation, because we have insufficient knowledge about data structures from different domains before clustering. Furthermore, different clusters in a partition should represent different dynamic patterns and structures. The predetermined clustering partition is not selected objectively. Therefore, an objective criterion should be employed to partition data into homogeneous groups, and models that can best explain the phenomena associated within each group should be derived.

HMM is an effective tool for modeling time varying processes and capturing dynamic properties of temporal data (Rabiner, 1989). It also adopts probability measures to model sequential data represented by observation sequences. HMM has been extensively used in various applications, such as speech signal recognition (Liang *et al*., 2007), finance (Zhang, 2004; Hassan *et al*., 2007), DNA sequence analysis (Andersson *et al*., 2012), and time series data pattern discovery and prediction (Niu *et al*., 2009; Wang *et al*., 2011). In the present study, the Bayesian information criterion (BIC) (Weakliem, 1999) and Akaike information criterion (AIC) (Bozdogan, 1987) are adopted to find the optimal HMM model size and cluster partition. BIC and AIC are widely used model selection criteria to find the best fitting model to the data (Burnham and Anderson, 2004). They select the optimal model that can minimize negative likelihood penalized by the number of parameters. Similar data falls in the same cluster after data clustering. Thus, the current load data will have a likelihood value produced by HMMs in each cluster. From these clusters, the historical data that has the maximum HMM likelihood value and the next time interval data will be chosen for prediction model training and future load forecasting.

Finally, a genetic algorithm (GA) optimized Elman network is used to forecast required load in the data during the next time period. The Elman network is a recurrent neural network, which is used to predict nonlinear and dynamic time series data in various areas (e.g., traffic flow, electric power load, weather, and cloud computing host load). These networks are self-studying, data driven, self-organizing, and self-adaptive, and they possess associative memory. Artificial neural networks can learn from patterns and capture hidden functional relationships in given data, even if the relationships are unknown or difficult to identify. Consequently, unseen inputs can be fed to the trained networks to generate appropriate outputs (Palit and Popovic, 2005; Hirose, 2012). However, the Elman network, which learns the network by modifying the weight and threshold values, starts at the output layer, and then moves backward through

the hidden layers. By applying a gradient method, the Elman network finds better weights and thresholds, but it is prone to local minimum problems and slow, with unsteady convergence during its training procedure. To overcome these weaknesses, we adopt a GA to optimize the weights and thresholds. GA is a stochastic general search method, which performs a global search and can effectively explore large search spaces. GA can help discover better network connection weights and parameters, thus improving the prediction accuracy of the Elman network. We name our model HGE, and experimental results show that the proposed method can perform more accurate load prediction in a cloud computing environment compared with other models.

2 An unsupervised offline load clustering approach

In this section, we introduce the proposed unsupervised offline load clustering method. In Section 2.1, we describe the fixed size overlapping sliding window (FSOSW) based load subsequence extraction technique and the dual time scale new arrival load time series data updating method. The HMM based unsupervised load subsequence clustering method is detailed in Section 2.2.

2.1 FSOSW based load subsequence extraction

Given an L cloud computing load time series data, we use FSOSW to divide the total length of L historical load into n subsequences of equal length d. We then obtain n d-dimensional data points $\boldsymbol{x}_i$ (i=1, 2, …, n, n=L/d) for clustering; i.e., we will treat d-length load subsequences as d-dimensional data points when they are classified into different clusters by using HMM. Fig. 2 shows the whole load subsequence extraction process.

The sliding window size is equal to d+h, where h is the length of the load subsequence that needs to be predicted; i.e., d-dimensional data point $\boldsymbol{x}_i$ and their next h-dimensional data points are extracted simultaneously to establish a rational connection between current loads and future loads. The d-dimensional and h-dimensional data points, which represent d-length and h-length load subsequences respectively, are the training and testing data for the Elman network respectively, which can help logically build the network structure. In accordance with the structure of the Elman network, the length of subsequence d in our study is an integer multiple of the prediction length h (i.e., d=γh). The number of input layer nodes is d, and h is the number of output layer nodes in the Elman network. Hence, d and h have a significant impact on the prediction performance of the Elman network. Coefficient γ cannot be determined by existing experience or formula, so it will be studied in later experiments.

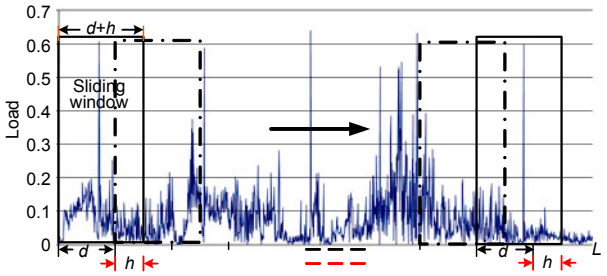

Fig. 2 The fixed size overlapping sliding window based cloud load subsequence extraction

We make use of d-dimensional data point $\boldsymbol{x}_i$ in the same fuzzy cluster and the next h-dimensional data as the training data to build the GA-Elman network predictors with the best fitness. Length-d subsequences will be placed into different clusters by using the HMM clustering algorithm, which will be based on their load characteristics. When the new load arrives, the newcomers will be divided into small load subsequences with a length of d by using FSOSW. Dual time scales are available for data updating. At a short time scale, for less than 6 or 12 hours, the new data with the maximum HMM likelihood values is placed into existing clusters. At a long time scale, for more than 12 hours, we re-train the whole historical data to obtain new HMMs and clusters. In this way, the proposed method can adapt to a dynamic load in real time by updating and re-training with the latest data.

2.2 HMM based unsupervised load subsequence clustering method

Clustering involves grouping data patterns into clusters so that patterns within a cluster bear strong similarity to one another but are very dissimilar to those in other clusters. These clusters are central to

many knowledge discovery and data mining tasks. The clustering problem addressed in this study can be described as follows:

Given n cloud computing load time series subsequences $X=\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$ of equal length, with index set $I=\{1, 2, \ldots, n\}$ and a fixed integer $K<<n$, a partition Cluster=(Cluster$_1$, Cluster$_2$, …, Cluster$_K$) of I and HMMs, $\lambda_1, \lambda_2, \ldots, \lambda_K$ that meet specific criteria are computed. The unsupervised HMM clustering method proposed in this study is a nested algorithm. We aim to build models based on the data structure and explain dynamic phenomena in a manner that is easy to interpret. Objective techniques should be employed when partitioning data into homogeneous groups. The proposed clustering algorithm searches for the optimal set of clusters that represents the best data partition based on the AIC, and builds the most appropriate HMM model for each cluster (i.e., determining the optimal HMM model states) according to the BIC. This refinement procedure starts with an initial model configuration and incrementally grows or shrinks the model through HMM based splitting and merging operations for choosing the right model size; e.g., the process starts with one cluster, and the number of clusters is increased in steps, or it starts with a configured maximum number of clusters, and the number of clusters is decreased based on their chosen criterion. In this study, we apply the model expansion approach which starts with the minimum model size, and the model size is increased until the best model is found. There are two search steps to find the optimal HMM states for each cluster and determine the optimal number of clusters in a partition. Both search steps are described in the following subsections.

2.2.1 Searching for the optimal HMM states

Searching for the optimal HMM states for each cluster can obtain the HMM with the optimal number of states for time series data grouped in the same cluster. The goal is to seek a model that can better account for the data, i.e., having a higher model posterior probability. The Bayesian model is adopted to find the optimal HMM structure for each cluster.

The Bayesian model merging criterion trades the model likelihood against the bias to a simpler model. A prior probability of a fully parameterized HMM model $\lambda=(\pi, \boldsymbol{A}, \boldsymbol{B})$ is assumed, where π represents the initial state probabilities. The probability of each state is the initial state of a given data sequence, where $\boldsymbol{A}$ is the transition matrix that defines the probability of transition from state i at time t to state j at the next time step, and $\boldsymbol{B}$ is the emission probability matrix that defines the probability of generating feature values at each state. The model structure λ and the model parameter θ are uniformly distributed. Given data X, the posterior probability of the model $P(\lambda, \theta|X)$ can be described by using the Bayesian rule as

$$P(\lambda,\theta|X)=\frac{P(\lambda,\theta)P(X|\lambda,\theta)}{P(X)}, \tag{1}$$

where $P(\lambda, \theta)$ and $P(X)$ are the prior probabilities of the model and the data, respectively, $P(X|\lambda, \theta)$ is the marginal likelihood of the data, and the prior probability of the data will not change across different models. As a consequence, we will have $P(\lambda, \theta|X)\propto P(\lambda, \theta)P(X|\lambda, \theta)$. We can assign an equal prior probability $P(\lambda, \theta)$ to all models. Thus, $P(\lambda, \theta|X)\propto P(X|\lambda, \theta)$. The posterior probability is proportional to the marginal log-likelihood of the given data. Bayesian model selection finds the model that has the highest marginal log-likelihood, which can be computed as

$$\begin{aligned}&\mathrm{BIC}(\log P(\lambda_k,\theta_k|X_k),d_k)\\&\approx\log P(X_k|\lambda_k,\hat{\theta}_k)-\frac{d_k}{2}\log N_k,\end{aligned} \tag{2}$$

where d_k is the dimensionality of the model parameter space, N_k is the object in cluster k, θ_k is the maximum likelihood (ML) configuration of the model, $\hat{\theta}_k$ is the re-estimated value of θ_k obtained by using the Baum-Welch procedure (Bilmes, 1997), $\log P(X_k|\lambda_k,\hat{\theta}_k)$ is the log-likelihood with a negative value, and $(d_k\log N_k)/2$ is the model complexity penalty term that tends to simplify models with fewer parameters. BIC seeks the best model for the data by trading off log-likelihood and penalty terms. The log-likelihood increases as the size of the HMM model increases until it reaches a peak value. It starts to decrease because of the increase in the model complexity penalty. The peak value obtained by combining the log-likelihood and penalty terms corresponds to the optimal model size.

2.2.2 Searching for the optimal cluster number

We use first-order continuous density HMMs to model the data. These datasets come from a combination of the underlying mixture Gaussian distribution. Each component of the mixture represents a cluster. A mixture HMM model M associated with K clusters λ_1, λ_2, …, λ_K is given. From this model, the log-likelihood of an object is computed as the combination of weighted log-likelihoods from all component models in the mixture. In this study, to make the computation feasible, we further approximate the log-likelihood value. The log-likelihood value of the given data from the mixture model is expressed as

$$\begin{aligned}
&\log P(X\left|\hat{\theta},M\right.)\\
&=\log\left(\prod_{i=1}^{N}\sum_{k=1}^{K}P_k f(\boldsymbol{x}_i\left|\boldsymbol{x}_i\in\lambda_k,\hat{\theta}_k,\lambda_k\right.)\right)\\
&=\sum_{i=1}^{N}\log\left(\sum_{k=1}^{K}P_k f(\boldsymbol{x}_i\left|\boldsymbol{x}_i\in\lambda_k,\hat{\theta}_k,\lambda_k\right.)\right)\\
&\approx\sum_{i=1}^{N}\sum_{k=1}^{K}[\log P_k+\log f(\boldsymbol{x}_i\left|\boldsymbol{x}_i\in\lambda_k,\hat{\theta}_k,\lambda_k\right.)]\\
&=\sum_{i=1}^{N}\sum_{k=1}^{K}\log P_k+\sum_{i=1}^{N}\sum_{k=1}^{K}\log f(\boldsymbol{x}_i\left|\boldsymbol{x}_i\in\lambda_k,\hat{\theta}_k,\lambda_k\right.), \quad (3)
\end{aligned}$$

where $f(\boldsymbol{x}_i\left|\boldsymbol{x}_i\in\lambda_k,\hat{\theta}_k,\lambda_k\right.)$ is the probability that an object $\boldsymbol{x}_i$ belongs to the kth component λ_k. Parameters are described by θ_k, and P_k is the prior probability of the component model. Searching for the optimal number of clusters yields the best mixture model for the data. To achieve this, we find λ_k the partition model that has the highest partition posterior probability $P(M|X)$. We exploit AIC for this task, which we revise and can be expressed as

$$\mathrm{AIC}_C=\mathrm{LL}-d-\frac{d(d+1)}{N-d-1}, \quad (4)$$

where d is the total number of free parameters that are estimated in the mixture HMM model M, i.e., $d=\sum_{k=1}^{K}d_k$, N is the total number of samples, and LL is the negative log-likelihood which the model uses to represent the observations. The following is used to compute the $P(M|X)$ for a clustering partition model with K clusters by using the AIC approximation:

$$\begin{aligned}
&\mathrm{AIC}(\log P(M\left|X\right.),d_k)\\
&\approx\log P(X\left|\hat{\theta},M\right.)-d-\frac{d(d+1)}{N-d-1}\\
&=\sum_{i=1}^{N}\sum_{k=1}^{K}\log P_k+\sum_{i=1}^{N}\sum_{k=1}^{K}\log f(\boldsymbol{x}_i\left|\hat{\theta}_k,\lambda_k\right.)\\
&\quad-\frac{N(K+\sum_{k=1}^{K}d_k)}{N-(K+\sum_{k=1}^{K}d_k)-1}, \quad (5)
\end{aligned}$$

where d_k is the number of free parameters in cluster k as mentioned above. The optimal cluster number is determined by increasing K between a minimum and maximum limit in increments and by the difference between the minimum and maximum limits. Similar to the BIC based optimal state searching, AIC finds the best partition for the data by trading off the log-likelihood term $\log P(X\,|\,\hat{\theta},M)$ and the penalty term $\frac{N(K+\sum_{k=1}^{K}d_k)}{N-(K+\sum_{k=1}^{K}d_k)-1}$.

In our study, HMM model state selection is performed after HMM clustering to automatically determine the optimal HMM states for load subsequences in each cluster. Accurately estimating the sizes for individual cluster models improves the overall partition model quality.

3 Cloud computing load prediction approach

3.1 HMM clustering based cloud load prediction

After unsupervised load subsequence clustering, all load subsequences are placed in different clusters, where each cluster has a related HMM model. Next, we find the historical load subsequences that have the most similar characteristics to the current load subsequence. To do this, the likelihood value for the observation sequence on the current load subsequence is obtained. In our experiments, the current load subsequence is the last subsequence of the whole historical load, i.e., $\boldsymbol{x}_n$ mentioned in Section 2.1. For clarification, we assume that the HMM likelihood value of the current resource load observation subsequence is LC. From the historical dataset clusters,

load subsequences are located in different HMM clusters and the one that produces the maximum likelihood value of LC will be selected. Then, the whole length of d load subsequences in this cluster and their next length of h time series subsequences are treated as the training and testing data for the Elman network, respectively. Finally, the GA-Elman network prediction model can be used to obtain the future load.

3.2 Genetic algorithm optimized Elman network prediction model

The Elman neural network is a partial recurrent network model which was first proposed by Elman (1990) and is considered a cross between a classic feed-forward perception and a pure recurrent network. In contrast to the feed-forward loop that consists of input, hidden, and output layers with variable weights connecting two neighboring layers, the back-forward loop employs a context layer that is sensitive to the historical input data; thus, the connections between the context layer and hidden layer are fixed. The dynamic characteristics of the Elman neural network are provided only by internal connections, so it does not need to use the state as an input or training signal, which makes the Elman network widely used in dynamic systems. Fig. 3 shows the structure of the Elman network.

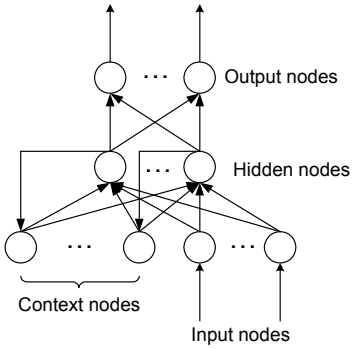

Fig. 3 Structure of the Elman network

The network contains four layers: input layer, hidden layer, context layer, and output layer. The context layer is used to remember the output of the hidden layer, which can be treated as a step delay operator. The association among these four layers is sensitive to historical data, and the internal feedback network increases the ability to process dynamic time varying data. In general, n nodes are present in the input layer, m nodes in the output layer, and r nodes in the hidden and context layers. The weight of the input layer to the hidden layer is w_1; the weight of the context layer to the hidden layer is w_2; the weight of the hidden layer to the output layer is w_3. $u(t)$ is set as the input at time t; $x(t)$ is the output of the hidden layer; $x_c(t)$ is the output of the context layer; and $y(t)$ is the output of the neural network. Therefore,

$$x(t)=\xi(w_2x_c(t)+w_1(u(t-1))), \tag{6}$$

where $x_c(t)=x(t-1)$. ξ is the transfer function of the hidden layer. Then, we use the common S-type function

$$\xi(x)=(1+e^{-x})^{-1}. \tag{7}$$

The output of the Elman neural network is

$$y(t)=\zeta(w_3x(t)), \tag{8}$$

where ζ is the transfer function of the output layer, which is a linear function. In this study, we employ a GA to optimize the network weights and thresholds simultaneously to produce better prediction performance.

GA maintains a population of chromosomes (individuals), which represent potential solutions to a problem to be solved, that is, the optimization of a function, which is generally very complex. Each individual in the population has an associated fitness, which indicates the utility or adaptation of the solution that it represents. GA starts with a population of randomly generated chromosomes and advances toward better chromosomes by applying genetic operators that are modeled on genetic processes which occur in nature. During successive iterations, called generations, the chromosomes are evaluated as possible solutions. Based on these evaluations, a new population is formed by using a selection mechanism and applying genetic operators such as crossover and mutation. Using GA to optimize the Elman network includes chromosome coding, fitness function adaptation, and defining the network structure. In this study, GA-Elman optimization is an adaptive scheme that can automatically adjust the Elman network's connection weights and thresholds (Fig. 4).

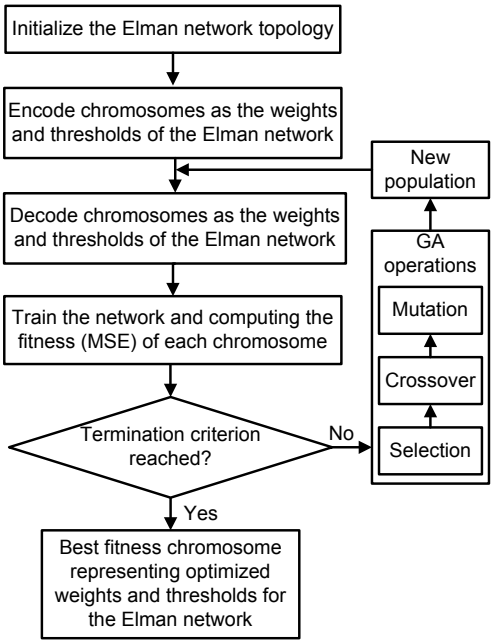

Fig. 4 GA-optimized Elman network

The operation of the GA-Elman algorithm is described as follows:

Step 1: Encode the Elman network's weights and thresholds.

Step 2: Decode step 1 and compute a different network from the current population.

Step 3: Train the network with given training data. If the current network satisfies the problem requirements, then stop; otherwise, continue to the next step.

Step 4: Determine the individual fitness by the objective function and training results, and choose a number of individuals with the smallest fitness to inherit to the next generation. The fitness function is the mean square error (MSE) in this study.

Step 5: Generate a new population by applying the genetic operators of crossover and mutation to this intermediate population.

Step 6: Return to step 2 until a certain termination criterion is met or the maximum training generations.

4 Evaluations and experiments

Experiments were conducted to validate our modeling strategy in terms of CPU load prediction in cloud computing. The proposed load predictive approach was evaluated by using real-world load trace from the Google Data Center. In 2011, Google published a new sample dataset of resource usage information from a Google production cluster. This cluster trace consists of data collected from approximately 12 500 servers which have a 29-day duration and contain nearly 650 000 jobs (Reiss *et al*., 2012). By leveraging Google's machine event trace which contains each host's capacity (re-scaled), we collected about a week's data, and calculated 600 000 relative CPU loads by dividing the absolute values according to corresponding capacities (Di *et al*., 2012b). The formula for data normalization is

$$\text{Average CPU load} = \frac{1}{C}\sum_{c=1}^{C}\sum_{j=1}^{J}\text{Load}_c^j, \tag{9}$$

where C is the total number of running machines at a given time point, and Load_c^j (c=1, 2, …, C) is the jth task's CPU load value of one particular running machine c, which is obtained from dividing the absolute values by corresponding capacities. We also used a week's load from NASA (Duy *et al*., 2010; Prevost *et al*., 2011) to conduct our experiments.

Both the evaluation and comparison of the proposed model for load prediction will be presented to compare the performance of our prediction strategy with those of other methods. We evaluated and compared our prediction method based on two classic metrics, MSE and the mean absolute error (MAE), which are defined as follows:

$$\text{MSE} = \sum_{t=1}^{t=T}(y_t - \hat{y}_t)^2 \Big/ \sum_{t=1}^{t=T} y_t^{\,2}, \tag{10}$$

$$\text{MAE} = \frac{1}{T}\sum_{t=1}^{t=T}\left|y_t - \hat{y}_t\right|, \tag{11}$$

where $\hat{y}_t$ represents the load prediction value, y_t is the actual measurement, and T is the number of load time series data points that need to be predicted.

4.1 Model evaluation

For practical resource management in cloud computing, setting up new VM instances on demand takes 5–10 min (Li *et al*., 2010; Islam *et al*., 2012). Hence, the resource management middleware needs to request new virtual instances about 10 min earlier to accommodate increased resource requirement in

the cloud. The one-step-ahead prediction interval T in our experiment was set to 300 data points (5 min), i.e., h=300, and the number of output layer nodes of the Elman network was also 300. As described in Section 2.2, we should determine the best value of γ, that is, the optimal number of input layer nodes for the Elman network. We applied FSOSW to extract load subsequences with different lengths from the historical data and trained the network by using the whole load trace without clustering. The maximum number of iterations for GA was 20, and the maximum number of epochs for the Elman network was 1500, because in all of our experiments, GA and the Elman network reached their own convergence status before those maximum numbers of epochs. These numbers are also the termination criteria and experimental parameter setting, respectively, for the next experiments. Table 1 presents the training MSE of different γ's.

Table 1 Training MSE of different γ's

γ	MSE	
	Google	NASA
1	0.0311	0.0342
2	0.0395	0.0467
3	0.0455	0.0703
4	0.0639	0.0481
5	0.0772	0.1135
6	0.0924	0.1597

Table 1 shows that when γ increased, the training errors of both datasets increased. The best value of γ in our experiment was 1, i.e., d=h=300, and the number of output layer nodes was also 300. Next, we used the proposed unsupervised HMM clustering to find the optimal number of clusters and the best HMM state for each cluster. Fig. 5 shows the characteristics of the AIC measure for clustering the partition model selection by using historical load data.

As the cluster number increases, the log-likelihood value and the penalty increase monotonically. AIC increased initially as the cluster number increased until it reached a peak value and then began to decrease because of the trade-off between the log-likelihood term and the model complexity penalty term. AIC reached the peak when the number of clusters was four for the Google trace and five for the NASA trace, which means the best partitions that can represent data structure were four and five clusters for the Google and NASA traces in our experiments, respectively. After cluster number selection, the proposed unsupervised method needs to find the best HMM state for each cluster. In this procedure, BIC was used to seek the optimal HMM state for each cluster. The final transition matrices of the clusters are shown in Table 2.

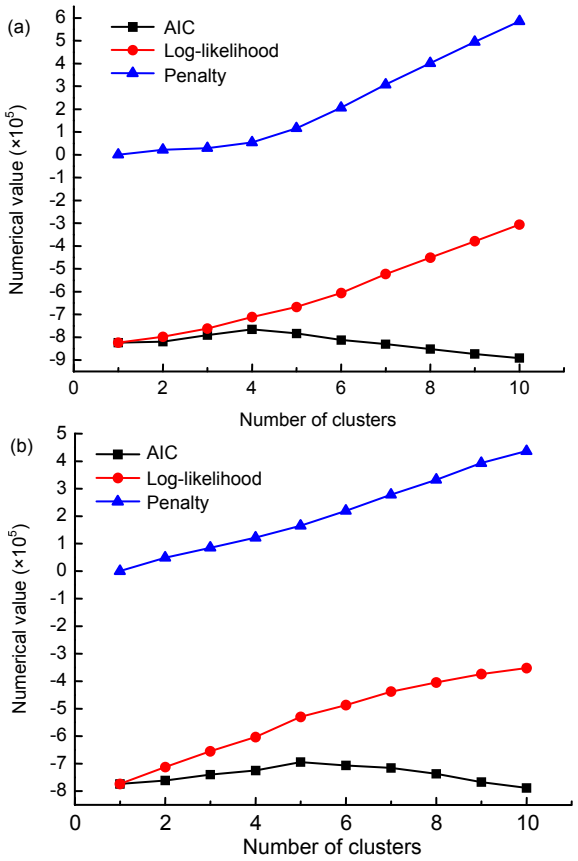

Fig. 5 AIC measure for model selection of the Google trace (a) and NASA trace (b)

Table 2 The transition matrix of each cluster

Transition matrix	Google	NASA
A_{HMM_1}	$\begin{pmatrix} 0.6178 & 0.3822 \\ 0.4119 & 0.5881 \end{pmatrix}$	$\begin{pmatrix} 0.8147 & 0.1853 \\ 0.2019 & 0.7981 \end{pmatrix}$
A_{HMM_2}	$\begin{pmatrix} 0.4019 & 0.2372 & 0.3609 \\ 0.3721 & 0.4683 & 0.1596 \\ 0.3561 & 0.2174 & 0.4265 \end{pmatrix}$	$\begin{pmatrix} 0.6324 & 0.2100 & 0.1576 \\ 0.0975 & 0.7505 & 0.1520 \\ 0.2085 & 0.2446 & 0.5469 \end{pmatrix}$
A_{HMM_3}	$\begin{pmatrix} 0.6379 & 0.2585 & 0.1036 \\ 0.3210 & 0.5255 & 0.1535 \\ 0.2553 & 0.1521 & 0.5926 \end{pmatrix}$	$\begin{pmatrix} 0.5146 & 0.4854 \\ 0.4218 & 0.5782 \end{pmatrix}$
A_{HMM_4}	$\begin{pmatrix} 0.5161 & 0.4839 \\ 0.3785 & 0.6215 \end{pmatrix}$	$\begin{pmatrix} 0.6078 & 0.3922 \\ 0.4118 & 0.5882 \end{pmatrix}$
A_{HMM_5}	NULL	$\begin{pmatrix} 0.5519 & 0.1712 & 0.2769 \\ 0.1545 & 0.7075 & 0.1380 \\ 0.3018 & 0.1034 & 0.5948 \end{pmatrix}$

Table 3 reports the HMM log-likelihood value of the current load subsequence from every cluster, as well as the number of load subsequences in each cluster.

Table 3 Log-likelihood of each HMM for current load

HMM	Number of load subsequences		Log-likelihood	
	Google	NASA	Google	NASA
HMM_1	18	346	−307	−324
HMM_2	996	708	−320	−286
HMM_3	856	102	−293	−357
HMM_4	130	169	−385	−318
HMM_5	NULL	675	NULL	−302

Table 3 shows that the maximum log-likelihood value was from HMM_3 and HMM_2 for the Google and NASA traces, respectively, which means that load subsequences in clusters 3 and 2 were most similar to their current loads. These load subsequences will be fed into the GA-Elman network as training data to forecast future load in cloud. The final step of our model evaluation was to find the optimal number of hidden layer nodes for the Elman network. No existing approach or rule was available to search for the best hidden layer nodes. Duy *et al*. (2011) employed artificial neural networks to forecast the host load on computational grids. They followed the trial-and-error method for obtaining the optimal network architecture by testing several numbers of hidden layer nodes, which could promise both high performance and low overhead. Hence, we tested different numbers of hidden layer nodes and empirically chose the best one with the minimum training MSE. Table 4 illustrates the MSE results of the different numbers of hidden layer nodes. The optimal numbers of hidden layer nodes were 35 and 30 for the Google and NASA traces, respectively.

Table 4 Training MSE of various numbers of hidden layer nodes

Number of nodes in the hidden layer	MSE	
	Google	NASA
15	0.0461	0.0491
20	0.0412	0.0424
25	0.0320	0.0399
30	0.0303	0.0317
35	0.0294	0.0362
40	0.0376	0.0388
45	0.0404	0.0416

After load subsequence clustering and optimal Elman network structure determination, the proposed prediction model HGE was employed to forecast future load from a historical trace. Fig. 6 displays the prediction results of our model. The predicted load was very close to the actual load, which demonstrates the effectiveness of the method. In the next subsection, we will make a comprehensive comparison between HGE and other prediction models.

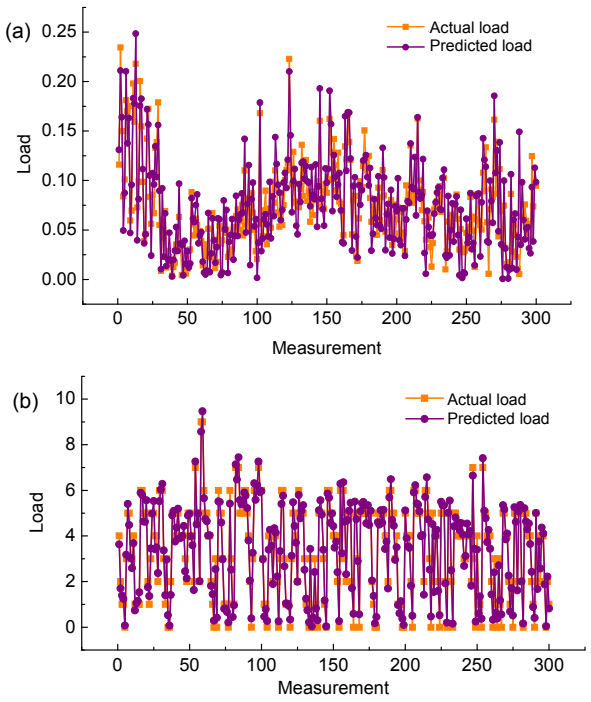

Fig. 6 Load prediction results of the Google trace (a) and NASA trace (b)

4.2 Comparison with other prediction models

In this subsection, we implemented other time series prediction algorithms, including GM(1, 1), auto regression (AR, of order 9), moving average (MA), exponential smoothing (ES, with α=0.5), BP neural network, the recently proposed polynomial fitting (PF) (Zhang *et al*., 2006), and ANFIS (Atique and Ali, 2007), for a comprehensive comparison with our HGE prediction model. We compared the models using the parameters that give the best prediction performance for every model. The predictive models were divided into two categories. GM, AR, MA, and ES belong to the first category. These are traditional prediction models that will be adopted to directly

forecast future cloud computing load from historical load time series data without clustering. BP neural network, PF, and ANFIS, which are state-of-the-art prediction methods that forecast future load, belong to the second category.

First, we conducted one-step-ahead load prediction. Table 5 reports the MSE and MAE results of all the prediction methods. As shown in Table 5, our prediction model showed advantages over the traditional models. MSE was several times smaller than those of traditional models. The PF, ANFIS, and HGE models exhibited similar forecasting accuracies and performed better than the BP neural network. Hence, GA improves the performance of the Elman network.

Table 5 MSE and MAE results of one-step-ahead prediction

Prediction model	MSE		MAE	
	Google	NASA	Google	NASA
AR	0.071	0.083	0.075	0.965
MA	0.092	0.107	0.083	1.174
ES	0.067	0.080	0.074	0.828
GM	0.083	0.095	0.097	1.227
BP	0.046	0.048	0.049	0.467
PF	0.039	0.044	0.036	0.412
ANFIS	0.035	0.037	0.035	0.336
HGE	0.029	0.032	0.031	0.294

To further evaluate the forecasting performance of the HGE model, we tested the performance of our model by performing iterative multi-step-ahead prediction based on one-step-ahead prediction. Fig. 7 shows the comparative MAE results of HGE and other models by performing steps 2 to 6 in the operation of the GA-Elman algorithm. Table 6 shows the corresponding MSE results. As the prediction step increases, the prediction errors of all models increased. Our model's prediction error remained acceptable, which indicates its effectiveness and stable performance. Compared with traditional prediction models, our model's superior prediction ability was apparent. The HGE model had far lower prediction errors than traditional models. Models in the second category were effective in dynamic time series data prediction and had lower prediction errors than models in the first category. In most situations, our proposed model outperformed both the first and second category models. As shown in Fig. 7, from two to six iterative prediction steps, the MAEs of our model were 0.036 to 0.067 and 0.358 to 0.559 for the Google and NASA traces, respectively.

4.3 Computational cost

In practical applications, the running time of a prediction model is a significant issue. In this

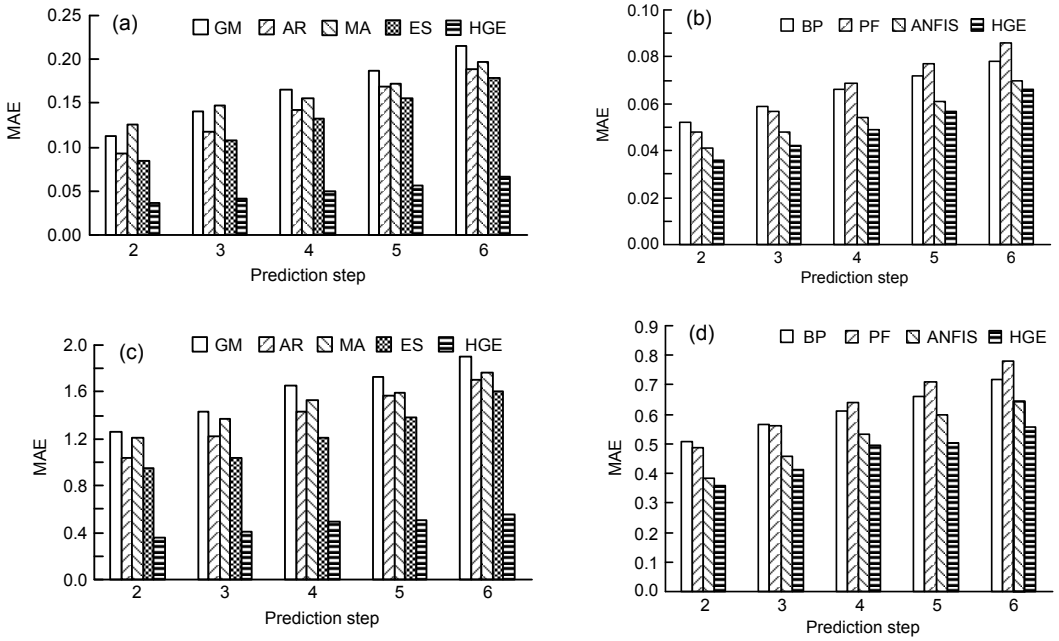

Fig. 7 Mean absolute error results of iterative multi-step-ahead prediction for Google trace (a–b) and NASA trace (c–d)

Table 6 MSE results of multi-step-ahead prediction for steps 2–6

Prediction model	MSE									
	Google					NASA				
	2	3	4	5	6	2	3	4	5	6
AR	0.076	0.082	0.085	0.093	0.104	0.087	0.094	0.097	0.104	0.111
MA	0.088	0.092	0.102	0.107	0.112	0.101	0.103	0.110	0.116	0.121
ES	0.073	0.075	0.084	0.095	0.106	0.086	0.091	0.095	0.101	0.108
GM	0.099	0.103	0.108	0.117	0.125	0.114	0.116	0.123	0.128	0.135
BP	0.050	0.053	0.059	0.064	0.070	0.052	0.062	0.070	0.077	0.085
PF	0.046	0.052	0.062	0.068	0.077	0.049	0.056	0.068	0.079	0.093
ANFIS	0.037	0.040	0.047	0.053	0.061	0.041	0.047	0.053	0.060	0.066
HGE	0.034	0.042	0.045	0.049	0.056	0.036	0.043	0.046	0.057	0.061

subsection, the computational cost of our load prediction model is given. In our approach, HMM based unsupervised clustering can be done offline. Newly arriving data will be added and processed incrementally. For Google and NASA traces with 600 000 time series data points, after FSOSW based load subsequence extraction, the HMM based clustering time for Google and NASA traces were 8.7 s and 9.3 s, respectively. Therefore, the prediction time of the GA-Elman network is what we are mostly concerned about. Our experiments were conducted on a computer equipped with an Intel i3 2.20 GHz CPU. The prediction time of our model is shown in Table 7.

The prediction time of AR, MA, ES, and GM was much longer than that of BP, PF, ANFIS, and HGE, because AR, MA, ES, and GM use the whole trace to predict future load, whereas BP, PF, ANFIS, and HGE apply only a certain part of the load trace. Compared with BP, PF, and ANFIS, our model had a longer prediction time, because GA requires a little more time to optimize the Elman network. However, the prediction time of HGE remained acceptable.

Table 7 Running time of all prediction models

Prediction model	Prediction time (s)	
	Google	NASA
AR	154	152
MA	132	128
ES	147	140
GM	159	149
BP	38	32
PF	41	36
ANFIS	58	55
HGE	68	64

For all the prediction models, data size is 600 000 and prediction size is 300

5 Conclusions

Our study is a contribution to short-term load prediction modeling in a cloud computing environment. The present approach represents the scheme of the hybrid system, which combines the unsupervised HMM clustering algorithm and genetic algorithm optimized Elman network. First, historical load time series data from the cloud computing platform is divided into small subsequences, and then we use the unsupervised HMM clustering method to put the similar subsequences together according to the distance between them. Second, we employ a genetic algorithm optimized Elman network to forecast the future load from the cluster with the maximum log-likelihood for the current load. We evaluated our model using load from a real world cloud computing trace. Experimental results show that our model outperforms other algorithms for dynamic cloud load prediction. In future work, we plan to consider the correlation between CPU, memory, and disk and make a multivariate prediction for better resource management in a cloud computing platform.

References

Andersson, S., Yamagishi, J., Clark, R.A.J., 2012. Synthesis and evaluation of conversational characteristics in HMM-based speech synthesis. *Speech Commun.*, **54**(2):175-188. [doi:10.1016/j.specom.2011.08.001]

Arasu, A., Manku, G.S., 2004. Approximate Counts and Quantiles over Sliding Windows. 23rd ACM SIGMOD-SIGACT-SIGART Symp. on Principles of Database Systems, p.286-296. [doi:10.1145/1055558.1055598]

Ardagna, D., Casolari, S., Colajanni, M., Panicucci, B., 2012. Dual time-scale distributed capacity allocation and load redirect algorithms for cloud systems. *J. Parall. Distr. Comput.*, **72**(6):796-808. [doi:10.1016/j.jpdc.2012.02.014]

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R.H., Konwinski, A., Lee, G., Patterson, D.A., Rabkin, I.S.A., Zaharia, M., 2009. Above the Clouds: a Berkeley View of Cloud Computing. Available from http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.pdf.

Atique, M., Ali, M.S., 2007. A Novel Adaptive Neuro Fuzzy Inference System Based CPU Scheduler for Multimedia Operating System. Int. Conf. on Neural Networks, p.1002-1007. [doi:10.1109/IJCNN.2007.4371095]

Bauer, E., Adams, R., 2012. Reliability and Availability of Cloud Computing. Wiley-IEEE Press, New Jersey, USA.

Bennani, M.N., Menasce, D.A., 2005. Resource Allocation for Autonomic Data Centers Using Analytic Performance Models. 2nd IEEE Int. Conf. on Autonomic Computing, p.229-240. [doi:10.1109/ICAC.2005.50]

Benson, T., Akella, A., Maltz, D.A., 2010. Network Traffic Characteristics of Data Centers in the Wild. 10th ACM SIGCOMM Conf. on Internet Measurement, p.267-280. [doi:10.1145/1879141.1879175]

Bilmes, J., 1997. A Gentle Tutorial on the EM Algorithm and Its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. Technical Report ICSI-TR-97-02, University of Berkeley, CA.

Bozdogan, H., 1987. Model selection and Akaike's information criterion (AIC): the general theory and its analytical extensions. *Psychometrika*, **52**(3):345-370. [doi:10.1007/BF02294361]

Burnham, K.P., Anderson, D.R., 2004. Multimodel inference understanding AIC and BIC in model selection. *Sociol. Methods Res.*, **33**(2):261-304. [doi:10.1177/0049124104268644]

Calheiros, R.N., Ranjan, R., Buyya, R., 2011. Virtual Machine Provisioning Based on Analytical Performance and QoS in Cloud Computing Environments. Int. Conf. on Parallel Processing, p.295-304. [doi:10.1109/ICPP.2011.17]

Caron, E., Desprez, F., Muresan, A., 2010. Forecasting for Grid and Cloud Computing On-Demand Resources Based on Pattern Matching. 2nd IEEE Int. Conf. on Cloud Computing Technology and Science, p.456-463. [doi:10.1109/CloudCom.2010.65]

Di, S., Kondo, D., Cirne, W., 2012a. Characterization and Comparison of Cloud versus Grid Workloads. IEEE Int. Conf. on Cluster Computing, p.230-238. [doi:10.1109/CLUSTER.2012.35]

Di, S., Kondo, D., Cirne, W., 2012b. Host Load Prediction in a Google Compute Cloud with a Bayesian Model. IEEE Int. Conf. on High Performance Computing, Networking, Storage and Analysis, p.1-11. [doi:10.1109/SC.2012.68]

Duy, T.V.T., Sato, Y., Inoguchi, Y., 2010. Performance Evaluation of a Green Scheduling Algorithm for Energy Savings in Cloud Computing. IEEE Int. Symp. on Parallel & Distributed Processing, Workshops and PhD Forum, p.1-8. [doi:10.1109/IPDPSW. 2010.5470908]

Duy, T.V.T., Sato, Y., Inoguchi, Y., 2011. Improving accuracy of host load predictions on computational grids by artificial neural networks. *Int. J. Parall. Emerg. Distr. Syst.*, **26**(4):275-290. [doi:10.1080/17445760.2010.481786]

Elman, J., 1990. Finding structure in time. *Cogn. Sci.*, **14**(2):179-211. [doi:10.1207/s15516709cog1402_1]

Hassan, M.R., Nath, B., Kirley, M., 2007. A fusion model of HMM, ANN and GA for stock market forecasting. *Expert Syst. Appl.*, **33**(1):171-180. [doi:10.1016/j.eswa.2006.04.007]

Hassan, R., Nath, B., Kirley, M., Kamruzzaman, J., 2012. A hybrid of multiobjective evolutionary algorithm and HMM-fuzzy model for time series prediction. *Neurocomputing*, **81**:1-11. [doi:10.1016/j.neucom.2011.09.012]

Hirose, A., 2012. Complex-Valued Neural Networks, Vol. 400. Springer, New York, Dordrecht, Heidelberg, London.

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]

Kaur, P., Soni, A.K., Gosain, A., 2013. A robust kernelized intuitionistic fuzzy c-means clustering algorithm in segmentation of noisy medical images. *Pattern Recogn. Lett.*, **34**(2):163-175. [doi:10.1016/j.patrec.2012.09.015]

Khan, A., Yan, X., Tao, S., Anerousis, N., 2012. Workload Characterization and Prediction in the Cloud: a Multiple Time Series Approach. 3rd IEEE Int. Workshop on Cloud Management, p.1287-1294. [doi:10.1109/NOMS.2012.6212065]

Khashei, M., Zeinal Hamadani, A., Bijari, M., 2012. A novel hybrid classification model of artificial neural networks and multiple linear regression models. *Expert Syst. Appl.*, **39**(3):2606-2620. [doi:10.1016/j.eswa.2011.08.116]

Li, A., Yang, X., Kandula, S., Zhang, M., 2010. CloudCmp: Comparing Public Cloud Providers. 10th ACM SIGCOMM Conf. on Internet Measurement, p.1-14. [doi:10.1145/1879141.1879143]

Liang, K.C., Wang, X.D., Anastassiou, D., 2007. Bayesian basecalling for DNA sequence analysis using hidden Markov models. *IEEE Trans. Comput. Biol. Bioinf.*, **4**(3):430-440. [doi:10.1109/tcbb.2007.1027]

Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., Ghalsasi, A., 2011. Cloud computing: the business perspective. *Decis. Support Syst.*, **51**(1):176-189. [doi:10.1016/j.dss.2010.12.006]

Mishra, A.K., Hellerstein, J.L., Cirne, W., Das, C.R., 2010. Towards characterizing cloud backend workloads: insights from Google compute clusters. *ACM SIGMETRICS Perform. Eval. Rev.*, **37**(4):34-41. [doi:10.1145/1773394.1773400]

Niu, D.X., Kou, B.E., Zhang, Y.Y., 2009. Mid-long Term Load Forecasting Using Hidden Markov Model. 3rd Int. Symp. on Intelligent Information Technology Application, p.481-483. [doi:10.1109/IITA.2009.422]

Palit, A.K., Popovic, D., 2005. Computational Intelligence in Time Series Forecasting: Theory and Engineering Applications. *In*: Advances in Industrial Control. Springer-Verlag New York, Inc., Secaucus, NJ.

Prevost, J.J., Nagothu, K., Kelley, B., Jamshidi, M., 2011. Prediction of Cloud Data Center Networks Loads Using

Stochastic and Neural Models. 6th Int. Conf. on System of Systems Engineering, p.276-281. [doi:10.1109/SYSOSE.2011.5966610]

Rabiner, L.R., 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE*, **77**(2):257-286. [doi:10.1109/5.18626]

Reiss, C., Tumanov, A., Ganger, G.R., Katz, R.H., Kozuch, M.A., 2012. Towards Understanding Heterogeneous Clouds at Scale: Google Trace Analysis. Proc. 3rd ACM Symp. on Cloud Computing, p.7.

Saripalli, P., Kiran, G.V.R., Shankar, R.R., Narware, H., Bindal, N., 2011. Load Prediction and Hot Spot Detection Models for Autonomic Cloud Computing. 4th IEEE Int. Conf. on Utility and Cloud Computing, p.397-402. [doi:10.1109/UCC.2011.66]

Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M., 2009. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Comput. Commun. Rev.*, **39**(1):50-55. [doi:10.1145/1496091.1496100]

Wang, P., Wang, H., Wang, W., 2011. Finding Semantics in Time Series. ACM SIGMOD Int. Conf. on Management of Data, p.385-396. [doi:10.1145/1989323.1989364]

Weakliem, D.L., 1999. A critique of the Bayesian information criterion for model selection. *Sociol. Methods Res.*, **27**(3):359-397. [doi:10.1177/0049124199027003002]

Yu, L., Wang, S., Lai, K.K., 2008. Credit risk assessment with a multistage neural network ensemble learning approach. *Expert Syst. Appl.*, **34**(2):1434-1444. [doi:10.1016/j.eswa.2007.01.009]

Zhang, G.P., 2003. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, **50**:159-175. [doi:10.1016/S0925-2312(01)00702-0]

Zhang, Y., 2004. Prediction of Financial Time Series with Hidden Markov Models. PhD Thesis, Simon Fraser University, Burnaby, Canada.

Zhang, Y., Sun, W., Inoguchi, Y., 2006. CPU Load Predictions on the Computational Grid. 6th IEEE Int. Symp. on Cluster Computing and the Grid, p.321-326. [doi:10.1109/CCGRID.2006.27]