SOFTWARE: PRACTICE AND EXPERIENCE
Softw. Pract. Exper. (2016)
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/spe.2426

Resource requests prediction in the cloud computing environment with a deep belief network

Weishan Zhang[1,*,†], Pengcheng Duan[1], Laurence T. Yang[2], Feng Xia[3], Zhongwei Li[1], Qinghua Lu[1], Wenjuan Gong[1] and Su Yang[4]

[1]*Department of Software Engineering, China University of Petroleum, No.66 Changjiang West Road, Qingdao 266580, China*
[2]*Department of Computer Science, St Francis Xavier University, Antigonish, Canada*
[3]*School of Software, Dalian University of Technology, Dalian 116620, China*
[4]*College of Computer Science and Technology, Fudan University, Shanghai 200433, China*

SUMMARY

Accurate resource requests prediction is essential to achieve optimal job scheduling and load balancing for cloud Computing. Existing prediction approaches fall short in providing satisfactory accuracy because of high variances of cloud metrics. We propose a deep belief network (DBN)-based approach to predict cloud resource requests. We design a set of experiments to find the most influential factors for prediction accuracy and the best DBN parameter set to achieve optimal performance. The innovative points of the proposed approach is that it introduces analysis of variance and orthogonal experimental design techniques into the parameter learning of DBN. The proposed approach achieves high accuracy with mean square error of $[10^{-6}, 10^{-5}]$, approximately 72% reduction compared with the traditional autoregressive integrated moving average predictor, and has better prediction accuracy compared with the state-of-art fractal modeling approach. Copyright © 2016 John Wiley & Sons, Ltd.

Received 21 October 2015; Revised 25 May 2016; Accepted 3 June 2016

KEY WORDS: deep belief network; prediction; cloud computing; resource request

1. INTRODUCTION

Predictions of cloud metrics such as resource requests, job lengths, hostload, and workload play critical roles in decision making for job scheduling and load balancing. Decision making is very important for many cloud applications in security, reliability, and availability and has been studied in many researches such as [1–4]. However, accurate prediction of cloud metrics is much more difficult than other systems. The underlying challenges for the accurate prediction are as follows:

- Varying interactions with cloud muti-tenants. Unlike Grid systems and HPC systems, cloud systems interact with varying number of cloud clients. Each client connection occurs at different time with different resource requirements and time lengths. The varying interactions exert unique challenges for cloud metric modeling.
- High nonlinearity in metric time series. When a cloud metric is sensed/sampled periodically from cloud system, it can be seen as a time series. Compared with other computing systems, the patterns in metric time series are extremely nonlinear. For example, Sheng [5] shows that cloud

*Correspondence to: Weishan Zhang, Department of Software Engineering, China University of Petroleum, No.66 Changjiang West Road, Qingdao 266580, China.
†E-mail: zhangws@upc.edu.cn

task lengths are only $\left[\frac{1}{20}, \frac{1}{2}\right]$ of grid task lengths and that the Google hostload is significantly more fluctuating than that of AuverGrid.[‡]

Although the metrics of Grid and other systems have been extensively explored [6–14], existing methods, such as moving averages, autoregressions, ARIMA,[§] and family and noise filters, are too simple to model the complex patterns in cloud resource requests. There has been some research regarding hostload prediction [5], workload characterization [15], task usage pattern [16], and characterizing task constraints [17]. However, there is few research addressing predicting cloud resource requests. Most such research [5, 18–22] used linear or probabilistic models with hand-crafted features. Hand-crafted features are laborious and largely based on domain knowledge and experience. There is a high likelihood that features extracted from cloud metrics contain little useful data information, thus generating inappropriate patterns. Probabilistic models fail to provide insights into the underlying patterns and tend to produce simple solutions to complex issues [5]. Eliciting prior probability for a Bayesian model is always difficult and tends to provide low prediction accuracy, whereas linear models cannot capture cloud metrics that show strong nonlinearity of varying degree over time.

Deep learning, including the deep belief network (DBN), has emerged as a promising technique for mining data patterns, surpassing existing approaches for image classification, natural language processing, and other areas. DBN consists of multiple undirected layers called restricted Boltzmann machines (RBMs). Hinton [23] showed that significantly better results could be achieved in deeper architectures when each RBM layer was pre-trained with an unsupervised learning algorithm (contrastive divergence [23]). An RBM can capture features or patterns of data. A DBN includes multiple stacked RBMs and can capture deep features. Therefore, the motivation of the proposed paper is to use DBN to model the high variant patterns in cloud metrics.

The reason why we choose DBN is that (i) compared with auto-encoder[24], DBN shows more generic because the use of stochastic binary hidden units in DBN acts as a very strong regularizer, while auto-encoder needs some hyperparameters for regularization. (ii) Convolutional neural networks[24] are a set of supervised methods, they are mostly used for image classification, while cloud metrics modeling involves unsupervised learning for prediction. Other deep architectures such as Recurrent Neural Network (RNN [25]) can be used to predict cloud resources requests; we do not choose them because in a real production environment, the prediction process should not occupy much resources in order not to influence cloud normal operations. And DBN is a very good fit because it is more generative than auto-encoder and costs less memory than RNN.

Inspired by the success of deep learning and the application of DBN to time serials prediction [26], we propose a DBN-based method for cloud resource request prediction. We have experimentally identified the most influential factors for prediction accuracy and the DBN parameter set that achieves optimal performance. The proposed approach achieves high accuracy, with mean square error (MSE) $= [10^{-6}, 10^{-5}]$.

The contributions of this paper are as follows:

- We propose a DBN-based approach for cloud resource request prediction that can be used for long-term and short-term prediction with improved accuracy compared with existing methods.

  Resource requests are extracted from Google cloud trace and are pre-processed through differential transformation, normalization, autoregression (AR), and autocorrelation tests. Each RBM from bottom to top is trained to form a DBN that is then fine-tuned using back propagation (BP)[¶] to minimize its loss function. The prediction data must then be returned to their original format because of the transformations in the pre-processing.
- We found the most influential factors and optimal parameter set for the DBN to achieve the best prediction performance. Each influential factor can have different levels with different impacts on DBN prediction performance. These are called the level set or parameter set, which we use

[‡]http://gwa.ewi.tudelft.nl/.
[§]http://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average.
[¶]http://en.wikipedia.org/wiki/Backpropagation.

DOI: 10.1002/spe

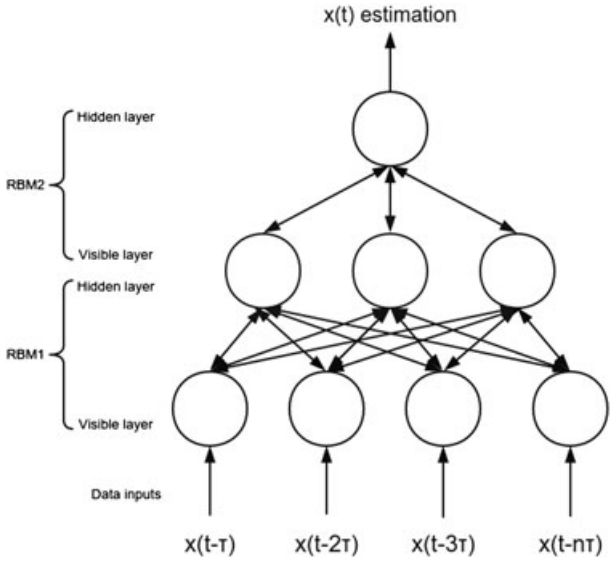

Figure 1. A deep belief network with two restricted Boltzmann machines.

interchangeably. We design a set of experiments using orthogonal experimental design (OED)[‖] methods, and the analysis of variance (ANOVA),[**] to identify the best parameter set for the DBN, and the factors that contribute most to prediction accuracy.

In Section 2, we briefly introduce the Google cloud trace data set used in this paper and the basic concept of DBN. We focus on the work flow of the proposed DBN-based work load prediction approach in Section 3. In Section 4, using OED experiments, we show evaluations on the prediction accuracy for the proposed approach, and verify our observations using the best DBN parameter set and the most influential factors using ANOVA. Our conclusions and some options for future related work completes this paper.

2. BACKGROUND OF DATA SET AND DEEP LEARNING

2.1. Google cloud trace

The Google cloud trace released in 2011 was measured on a heterogeneous 7000 machine server cluster over a 29-day period involving 672,075 jobs and more than 48 million tasks. The dataset records metrics of jobs, tasks, and machine nodes in the cluster. It contains a single table, indexed by a primary key that typically includes a timestamp. The data show the scheduling complexities that affect metrics of the Google cloud, including varieties of job types, complex scheduling constraints for some jobs, mixed hardware types, and poor user estimation of resource consumption. This paper targets predicting resource requests for upcoming tasks in a future interval.

2.2. Deep Belief Network

A DBN consists of stacked RBM as shown in Figure 1. The outputs of a bottom RBM act as inputs to the next level RBM. Compared with traditional ANNs, the key step of a DBN is pre-trained initial weights and biases using layer-by-layer unsupervised learning. After the pre-training, global supervised learning (e.g. using BP) can be employed to fine-tune and minimize its loss function.

A RBM can be regarded as a Markov Random Field (MRF), which is an undirected probabilistic graphical model. An MRF is a special joint probability distribution $p(x)$ and a graph that illustrates

[‖]http://en.wikipedia.org/wiki/Design_of_experiments.
[**]http://en.wikipedia.org/wiki/Analysis_of_variance.

DOI: 10.1002/spe

relations among the vertices in that graph. Markov Chain Monte Carlo (MCMC) technique provides a statistical method for simulating/learning the MRF, without knowing the exact formulation of the joint probability distribution. A special MRF is the RBM, which combines the probability distribution $p(x)$ with a energy function.

The joint probability distribution of an MRF $\boldsymbol{X} = (X_v)_{v\in V}$ can be written as a product of functions defined over the maximal cliques in probabilistic graphical model (PGM):

$$p(\boldsymbol{x}) = \frac{1}{Z}\prod_{c\in C}\psi_c(\boldsymbol{x}_c) = \frac{1}{Z}e^{\sum_{c\in C}\ln\psi_c(\boldsymbol{x}_c)} = \frac{1}{Z}e^{-E(\boldsymbol{x})}$$

where C is the maximal clique set in the MRF from the PGM perspective. $Z = \sum_{\boldsymbol{x}}\prod_{c\in C}\psi_c(\boldsymbol{x}_c)$ is the partition function and $\psi_c(\boldsymbol{x}_c)$ is the potential function of $\boldsymbol{x}$ in c. $E(\boldsymbol{x}) = -\sum_{c\in C}\ln\psi_c(\boldsymbol{x}_c)$ is called the energy function. However, in reality, we usually get observations of only part of the MRF (denoted by $\boldsymbol{v}$). The other part of the MRF is hidden (denoted by $\boldsymbol{h}$). Thus, $\boldsymbol{x}$ is split into $\boldsymbol{v}$ and $\boldsymbol{h}$, which are the visible units and hidden units of a RBM. So latent variables are introduced, and we only observed $\boldsymbol{v}$, so we are interested in $\boldsymbol{v}$:

$$p(\boldsymbol{v}) = \sum_{\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}) = \frac{1}{Z}\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}$$

where $Z = \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}$, which unfolds the relationship between E and Z. Tuning parameters of a MRF is usually performed through maximizing the log likelihood function of an MRF, which is

$$\ln\mathcal{L}(\boldsymbol{\theta}|S) = \ln\prod_{i=1}^{l} p(\boldsymbol{x}_i|\boldsymbol{\theta}) = \sum_{i=1}^{l}\ln p(\boldsymbol{x}_i|\boldsymbol{\theta})$$

One method for maximizing the log likelihood function is gradient ascent (GA), which is

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}^{(t)} = \boldsymbol{\theta}^{(t)} + \eta\frac{\partial}{\partial\boldsymbol{\theta}^{(t)}}\ln\mathcal{L}\left(\boldsymbol{\theta}^{(t)}|S\right) - \gamma\boldsymbol{\theta}^{(t)} + \nu\Delta\boldsymbol{\theta}^{(t-1)}$$

where γ is the regularization parameter and ν is the momentum parameter. Given a single training example $\boldsymbol{v}$, we have

$$\begin{aligned}\ln\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v}) &= \ln p(\boldsymbol{v}|\boldsymbol{\theta}) = \ln\sum_{\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}) = \ln\sum_{\boldsymbol{h}}\frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{Z}\\ &= \ln\sum_{\boldsymbol{h}}\frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}} = \ln\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} - \ln\sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}\end{aligned}$$

So the gradient should be

$$\frac{\partial}{\partial\boldsymbol{\theta}}\ln\mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v}) = \frac{\partial}{\partial\boldsymbol{\theta}}\ln\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} - \frac{\partial}{\partial\boldsymbol{\theta}}\ln\sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}$$

Deducing the gradient of first term, we have

$$\begin{aligned}&\frac{1}{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}}\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}\frac{\partial(-E(\boldsymbol{v},\boldsymbol{h}))}{\partial\boldsymbol{\theta}} = -\sum_{\boldsymbol{h}}\frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}}\frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial\boldsymbol{\theta}}\\ &= -\sum_{\boldsymbol{h}}\frac{\frac{1}{Z}e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\frac{1}{Z}\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}}\frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial\boldsymbol{\theta}} = -\sum_{\boldsymbol{h}}\frac{p(\boldsymbol{v},\boldsymbol{h})}{p(\boldsymbol{v})}\frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial\boldsymbol{\theta}} = -\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v})\frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial\boldsymbol{\theta}}\end{aligned}$$

DOI: 10.1002/spe

Deducing the gradient of second term, we have

$$\frac{1}{\sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}} \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} \frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{\theta}} = \sum_{\boldsymbol{v},\boldsymbol{h}} \frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{Z} \frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{\theta}} = \sum_{\boldsymbol{v},\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}) \frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{\theta}}$$

So we finally have the gradient of the log likelihood

$$\frac{\partial}{\partial \boldsymbol{\theta}} \ln \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v}) = -\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) \frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{\theta}} + \sum_{\boldsymbol{v},\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}) \frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{\theta}} \qquad \ldots\ldots\ldots\ldots\ldots \mathbb{G}$$

The energy function proposed in this paper is

$$E(\boldsymbol{v},\boldsymbol{h}) = \frac{||\boldsymbol{v}-\boldsymbol{b}||^2}{2\sigma^2} - \boldsymbol{c}^T\boldsymbol{h} - \frac{\boldsymbol{v}^T\boldsymbol{W}\boldsymbol{h}}{\sigma^2}$$

Insert $E(\boldsymbol{v},\boldsymbol{h})$ into $\mathbb{G}$, we have the gradients of the energy function with respect to the parameters

$$\begin{aligned}
\frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{b}} &= -\frac{\boldsymbol{v}-\boldsymbol{b}}{\sigma^2} \\
\frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{c}} &= -h \\
\frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{W}} &= -\frac{\boldsymbol{v}\boldsymbol{h}^{\boldsymbol{T}}}{\sigma^2} \\
\frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \sigma} &= -\frac{||\boldsymbol{v}-\boldsymbol{b}||^2}{\sigma^3} + \frac{2\boldsymbol{v}^{\boldsymbol{T}}\boldsymbol{W}\boldsymbol{h}}{\sigma^3}
\end{aligned}$$

Stacking multiple RBMs deduced earlier layer-by-layer into a deep neural network, we have deep belief network for modeling the Google cluster trace.

3. OVERVIEW AND WORK FLOW OF THE DBN-BASED APPROACH

This section focuses on the overall work flow of the DBN-based approach, pre-processing of the data set, and training the DBN.

The DBN-based approach consists of several steps, as shown in Figure 2. In pre-processing, we extract the resource requests from the raw trace and aggregate into a minute level (i.e., record a time stamp for every minute). We then perform a differential transformation on the aggregated data to reduce the linearity (as detailed in Section 3.1). The reason for the differential transformation is that ANNs have low learning capability for data with strong linear factors, and the differential transformation reduces any linear dependency. The transformed data is normalized into the range (0, 1), and we perform autoregression and autocorrelation analyses to check linearity and interdependency.

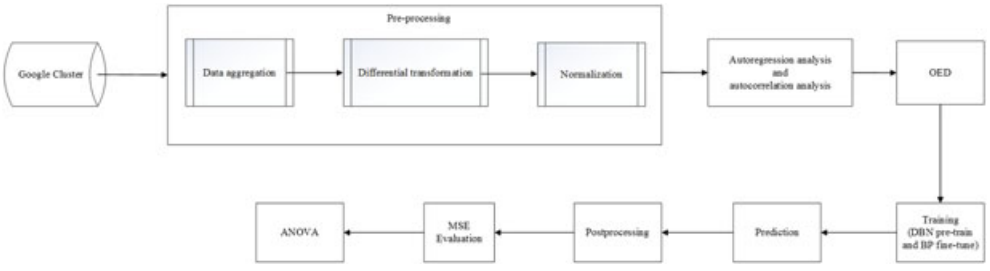

Figure 2. Deep belief network training and prediction work flow.

DOI: 10.1002/spe

Following this, we design sets of OED experiments, considering input nodes, learning rate, training sample size, and the number of RBMs. In the training step, each experiment defines a specific level for each factor, and we build the DBN for each experiment. We pre-train the initial weights and biases of the bottom RBM using the gradient-based contrastive divergence algorithm[27]. The next RBM is trained using the hidden layer of the previous RBM as the visible layer, and so forth for all RBMs from bottom to top. We fine-tune the DBN using the BP algorithm, which calculates the gradient of a loss function with respect to all the weights in the network. The gradient is provided to the optimization (gradient descent), which in turn updates the weights, to minimize the loss function. Thus, we obtain a fully trained DBN for each experiment.

The prediction step considers long-term and short-term prediction procedures, based on guidelines from the OED step.

To obtain the original format of the prediction results, we transform the prediction data to the original resource request format and calculate the MSE. Finally, we identify the most influential factors and the best parameter set of the DBN using ANOVA.

3.1. Differential transformation

In pre-processing, differential transformation is defined as $x^d(t) = x(t) - x(t-d)$, where d is the time delay, and we provide the normalized $x^d(t)$ to the DBN for training. Reconstructing the predictions of the original requests should be $\hat{x}(t) = \hat{x}^d(t) + \hat{x}(t-d)$, where $\hat{x}(t)$ is the prediction of the original request at time t. We propose d be decided by the ARIMA model [28]. The concept for this differential transformation process is to reduce correlations in resource requests. Figure 3 compares autocorrelations of CPU requests over time with their differential transformation. The dependent lag is substantially reduced, which indicates the transformed data are significantly less autocorrelated.

3.2. OED for obtaining optimal parameter set for DBN

3.2.1. Motivation of using OED and ANOVA to fine-tune a DBN. There are two points worth clarification. The first one is the parameters in the term **parameter set** do not refer to the weights and biases in the neural network, which are optimized by the contrastive divergence algorithm for a RBM, but refer to the structure elements of the network such as visible nodes, hidden nodes, number of RBMs, learning rate, momentum and proportion of training samples and test samples, which are all optimized using OED and ANOVA in this paper. The second one is on choosing methods for

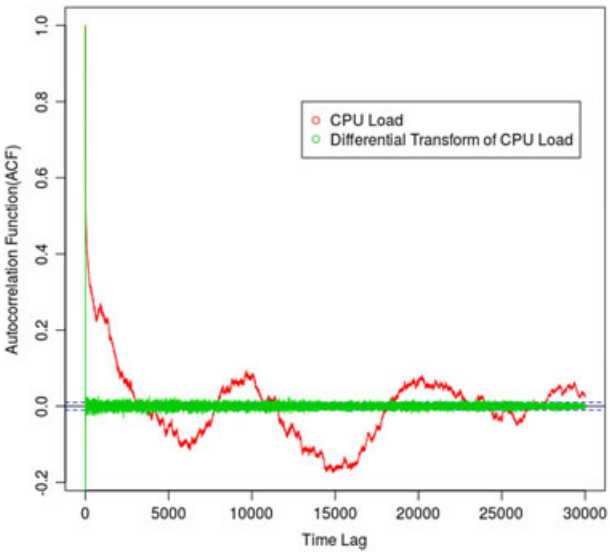

Figure 3. Autocorrelation function comparison between CPU loads and their differential transformation.

DOI: 10.1002/spe

optimizing the network structure. We choose ANOVA to achieve this goal based on the following two considerations.

- Other optimization-based methods fail in taking a broader structure elements into accounts. An example of this is when launching a PSO algorithm for searching a best parameter set, it fails in optimizing parameters such as the number of RBMs because it has been decided before the launching of PSO. However, the methodology of OED under ANOVA not only can take more structure elements into accounts but also can find out the influential rank of a neural network elements from a statistical perspective that cannot be achieved by other parameter-tuning algorithms.
- Another consideration is that optimization-based methods such as PSO could easily fall into local optima and depend sensitively on the type of problems and on the application scenarios [29], which indicates there is a high likelihood that those algorithms fail in finding out the **best** parameter set. Furthermore, it is hard to identify when those parameter-tuning algorithms fall into a local optima; however, the best parameter set can be easily discriminated from the evaluation outputs of a OED table.

3.2.2. Design orthogonal table. For univariate time series forecasting, the inputs of the prediction network are the past and lagged observations. Each input pattern is composed of a moving window of fixed length along the series. Generally, a single output artificial neural network can be regarded as a nonlinear function,

$$x_t = f\left(x_{t-1}, x_{t-2}, ..., x_{t-p}\right)$$

where x_t is the observation at time t and p is the number of the input nodes used to predict the future. The Google cloud has thousands of nodes requiring forecasting. In such a situation, an automatic forecasting algorithm such as ARIMA is required and used in this paper to gain insights into the underlying cloud resource request prediction problem.

An ARIMA consists of three orders: autoregression (OAR), integration (OI), and moving average (OMA). The three orders show the autocorrelation and interdependency in the resource requests. There have been several attempts to automate ARIMA modeling, and we use the automatic ARIMA model from [30] to generate the three orders of the resource request series, which provides guidelines for designing experiments considering the factor of input nodes. Table I compares three parts of the original with the transformed requests. The original and the corresponding transformation are less than 5, which implies the scope of the DBN input nodes is close to 5.

A DBN can be affected by several factors, including input nodes, hidden nodes, learning rate, and training sample size. To generate an optimized network topology, we propose to use the OED methodology, which is a simple but accurate method for multiple factor experiments with multiple levels. It guarantees that the effect of one factor or interaction can be estimated separately from the effect of any other factor or interaction in the model. Conducting all the possible experiments is impossible. However, OED can greatly decrease the number of experiments required when searching for the best solution. We focus on the following four factors: input nodes (IN), learning rate (LR), training sample size (TSS), and RBM numbers (RN), and build a $L_9(3^4)$ OED table, where each factor has three levels, as shown in Table II.

The autocorrelation lag and ARIMA orders are all less than 5, as discussed earlier. Therefore, the number of input nodes (NIN) should be set to a figure near 5. The number of hidden nodes is rarely larger than double the NIN, and Zhang[31] suggests that input nodes are more important than hidden

Table I. ARIMA orders of CPU and RAM requests.

	OAR	OI	OMA
Original CPU	2	1	5
Differential CPU	5	1	3
Original RAM	2	3	2
Differential RAM	4	0	5

DOI: 10.1002/spe

Table II. Level values for each factor.

Factors				
Levels	IN	RN	LR	TSS (%)
2	1	0.02	0.01	40
5	2	0.2	0.05	60
10	3	0.5	0.1	80

Table III. $L_9(3^4)$ orthogonal design table for optimizing the deep belief network.

Factor				
Experiment	Input nodes	RBM number	Learning rate	Sample size (%)
1	2	1	0.02	40.00
2	2	2	0.2	60.00
3	2	3	0.5	80.00
4	5	1	0.2	80.00
5	5	2	0.5	40.00
6	5	3	0.02	60.00
7	10	1	0.5	60.00
8	10	2	0.02	80.00
9	10	3	0.2	40.00

RBM, restricted Boltzmann machine.

nodes in a neural network model built for prediction. Hence, we do not consider the hidden nodes as a factor, and set them equal to the NIN. Training samples are taken as the top portion of the trace dataset. There is no good criteria for the percentage of training and test samples, because the training process is affected by the data themselves, but generally speaking at least one fifth of the total samples should be taken for performance evaluation. Therefore, in this paper, we separately take 40%, 60%, and 80% of the total samples for training.

Following OED, Table III details experiment parameters that follow the $L_9(3^4)$ guidelines. There are nine orthogonal experiments, each of which is assigned with one of each factor's levels, and so will identify the level set that generates the least prediction MSE.

4. EVALUATION OF THE DBN-BASED PREDICTION APPROACH

4.1. Accuracy measurement

We choose MSE to evaluate the performance of a DBN,

$$MSE(t,s) = \frac{\sum_{i=t}^{i=t+s} \left(\hat{l_i} - l_i\right)^2}{s}$$

where $MSE(t,s)$ is the MSE of the subsequent s time units starting at time t, $\hat{l_i}$ is the predicted request, and l_i is the observed load at time t.

It is essential to predict both short-term and long-term requests. Suppose we take the training samples from the beginning of the Google trace at time t_0, and we want to start prediction at some future time $t_0 + t$, then the prediction interval should be $[t_0 + t, t_0 + t + s]$. Our purpose is to investigate how the prediction accuracy changes with increments of t and s, and also how it differs with varying different factor levels. In the Google trace data case, long-term prediction is made in terms of days, that is, $t = 0, 1, 2, 3$, and the prediction interval is $s = 1, 2, 3$. Short-term prediction is made in terms of hours. We commence prediction immediately after the end of the trace (i.e., $t = 0$), and the prediction interval is $s = 1, 2, 3$.

DOI: 10.1002/spe

4.2. CPU requests

Tables IV and V show the prediction accuracy of short-term and long-term CPU requests of Google cloud trace, respectively, using the proposed DBN-based approach. The accuracy is of the order 10^{-5}, which applies to all the measurements in these two tables, where the average MSE was calculated for each experiment.

From Table IV, we can see that the best solution for predicting short-term CPU requests is the 9th experiment with 10 input nodes, two RBMs, learning rate 0.2, with 40% of the total samples used for training. The best solution for the long-term prediction (as in Table V) is also the 9th experiment.

Table IV. Short-term prediction accuracy of CPU requests.

	Experiment Num = 01					Experiment Num = 02					Experiment Num = 03			
S\T	1	2	3	Avg	S\T	1	2	3	Avg	S\T	1	2	3	Avg
0	3.156	4.038	3.950	3.715	0	0.886	1.183	1.307	1.125	0	3.407	4.009	3.023	3.480
	Experiment Num= 04					Experiment Num= 05					Experiment Num= 06			
S\T	1	2	3		S\T	1	2	3		S\T	1	2	3	
0	3.263	4.050	3.114	3.476	0	1.326	1.099	1.088	1.171	0	0.897	1.220	1.301	1.139
	Experiment Num= 07					Experiment Num= 08					Experiment Num= 09			
S\T	1	2	3		S\T	1	2	3		S\T	1	2	3	
0	1.083	1.430	1.463	1.325	0	2.381	3.323	2.545	2.750	0	1.009	0.844	0.831	0.895

Table V. Long-term prediction accuracy of CPU requests.

	Experiment Num= 01					Experiment Num= 02					Experiment Num= 03			
S\T	1	2	3	Avg	S\T	1	2	3	Avg	S\T	1	2	3	Avg
0	3.342	3.162	3.701	3.402	0	2.062	1.483	1.222	1.589	0	1.909	2.008	1.807	1.9078
1	2.985	3.883	5.653	4.174	1	0.905	0.802	0.992	0.9	1	2.106	1.756	1.596	1.8193
2	4.782	6.988	9.039	6.937	2	0.7	1.036	1.341	1.025	2	1.406	1.341	1.489	1.4118
3	9.197	11.169	11.787	10.718	3	1.372	1.661	2.001	1.678	3	1.277	1.53	1.285	1.3642
Avg	5.077	6.301	7.545	6.307		1.26	1.246	1.389	1.2981		1.674	1.659	1.544	1.6258
	Experiment Num= 04					Experiment Num= 05					Experiment Num= 06			
S\T	1	2	3		S\T	1	2	3		S\T	1	2	3	
0	2.155	2.205	2.007	2.122	0	1.261	1.316	1.155	1.244	0	2.082	1.499	1.235	1.6052
1	2.255	1.933	1.774	1.987	1	1.371	1.102	0.978	1.15	1	0.917	0.811	1.003	0.9105
2	1.612	1.533	1.706	1.617	2	0.835	0.782	0.962	0.86	2	0.706	1.046	1.354	1.0354
3	1.457	1.754	1.492	1.568	3	0.731	1.026	1.135	0.964	3	1.387	1.678	2.022	1.6955
Avg	1.87	1.856	1.745	1.823		1.049	1.057	1.057	1.0545		1.273	1.259	1.403	1.3117
	Experiment Num= 07					Experiment Num= 08					Experiment Num= 09			
S\T	1	2	3		S\T	1	2	3		S\T	1	2	3	
0	2.189	1.62	1.414	1.741	0	1.89	1.912	1.757	1.853	0	0.945	0.987	0.866	0.9329
1	1.052	1.027	1.205	1.094	1	1.935	1.691	1.558	1.728	1	1.03	0.827	0.811	0.8892
2	1.002	1.281	1.557	1.28	2	1.449	1.37	1.537	1.452	2	0.625	0.701	0.993	0.7733
3	1.568	1.838	2.14	1.849	3	1.292	1.581	1.387	1.42	3	0.778	1.178	1.317	1.0909
Avg	1.4527	1.4415	1.5788	1.491		1.6415	1.6386	1.5597	1.613		0.8445	0.9234	0.9968	0.92157

DOI: 10.1002/spe

Long-term prediction can be more essential in a production environment. To conduct a full test, the beginning of prediction time was set to be 0, 1, 2, and 3 days, in contrast to the short-term analysis.

The least accurate performance is the first experiment for both short-term and long-term prediction. This is due to there being very few input nodes and only one hidden layer. With a small number of input nodes, the DBN is unable to memorize sufficient metric histories to give a reasonable prediction, and the single hidden layer is too simple to model high variances in Google cloud traces.

4.3. RAM requests

Tables VI and VII show the prediction accuracy of short-term and long-term RAM requests of Google cloud trace. The best solution for predicting the RAM requests in both cases is the fifth experiment with five input nodes, two RBMs, learning rate = 0.5, and 40% of the total samples

Table VI. Short-term prediction accuracy of RAM requests.

	Experiment Num= 01					Experiment Num= 02					Experiment Num= 03			
S\T	1	2	3	Avg	S\T	1	2	3	Avg	S\T	1	2	3	Avg
0	6.209	5.079	5.074	5.454	0	1.981	1.728	2.191	1.967	0	0.503	2.118	2.175	1.599
	Experiment Num= 04					Experiment Num= 05					Experiment Num= 06			
S\T	1	2	3		S\T	1	2	3		S\T	1	2	3	
0	0.738	2.304	2.593	1.878	0	1.002	0.894	0.760	0.886	0	2.018	1.714	2.174	1.969
	Experiment Num= 07					Experiment Num= 08					Experiment Num= 09			
S\T	1	2	3		S\T	1	2	3		S\T	1	2	3	
0	1.870	1.655	2.128	1.884	0	0.725	1.812	1.967	1.502	0	1.162	1.184	1.060	1.135

Table VII. Long-term prediction accuracy of RAM requests.

	Experiment Num= 01					Experiment Num= 02					Experiment Num= 03			
S\T	1	2	3	Avg	S\T	1	2	3	Avg	S\T	1	2	3	Avg
0	4.642	3.964	3.613	4.073	0	0.788	2.131	1.877	1.599	0	1.246	1.632	1.407	1.4283
1	3.288	3.1	3.393	3.261	1	3.476	2.422	2.055	2.651	1	2.019	1.488	1.749	1.7519
2	2.914	3.446	4.259	3.54	2	1.404	1.362	1.421	1.395	2	0.956	1.614	1.450	1.3398
3	3.979	4.933	5.694	4.869	3	1.320	1.429	1.535	1.428	3	2.271	1.697	1.642	1.8704
Avg	3.706	3.861	4.24	3.935		1.747	1.836	1.722	1.768		1.623	1.608	1.562	1.5976
	Experiment Num= 04					Experiment Num= 05					Experiment Num= 06			
S\T	1	2	3		S\T	1	2	3		S\T	1	2	3	
0	1.447	1.922	1.686	1.685	0	0.684	0.737	0.808	0.743	0	0.779	2.138	1.880	1.5991
1	2.397	1.805	2.101	2.101	1	0.790	0.871	0.893	0.851	1	3.498	2.431	2.057	2.662
2	1.213	1.953	1.772	1.646	2	0.952	0.944	1.120	1.006	2	1.365	1.337	1.398	1.3664
3	2.693	2.051	2.009	2.251	3	0.938	1.205	1.286	1.143	3	1.309	1.414	1.536	1.4196
Avg	1.938	1.933	1.892	1.921		0.841	0.939	1.027	0.936		1.738	1.83	1.718	1.762
	Experiment Num= 07					Experiment Num= 08					Experiment Num= 09			
S\T	1	2	3		S\T	1	2	3		S\T	1	2	3	
0	0.777	2.132	1.877	1.595	0	1.040	1.419	1.299	1.252	0	1.007	1.136	1.251	1.1314
1	3.489	2.428	2.054	2.657	1	1.798	1.429	1.661	1.629	1	1.265	1.372	1.365	1.3339
2	1.368	1.337	1.401	1.369	2	1.059	1.593	1.506	1.386	2	1.481	1.415	1.529	1.4749
3	1.307	1.417	1.537	1.42	3	2.127	1.730	1.766	1.874	3	1.350	1.553	1.563	1.4885
Avg	1.735	1.8286	1.7172	1.76		1.506	1.5426	1.558	1.536		1.2756	1.3691	1.4268	1.357

DOI: 10.1002/spe

used for training. Similar to the earlier discussion, the least accurate performance for both cases is the first experiment.

4.4. Comparing the DBN approach with the ARIMA predictor

To show the effectiveness of our proposed approach, we compare our outcomes with that of an ARIMA predictor [32], which is often a baseline for prediction research. Resource requests in Google cloud trace were analysed in Section 3, where the ARIMA orders in CPU and RAM load requests were fitted. ARIMA (2,3,5) was chosen to compare with our proposed DBN-based approach, with optimal parameters set under each circumstance, with the outcome as shown in Figure 4. Short-term prediction shows MSE reduction of 76% and 61%, and long-term prediction shows 83% and 67% reduction for CPU and RAM requests, respectively.

Thus, the proposed DBN-based approach has significantly improved accuracy than that of the baseline for both short-term and long-term prediction, providing approximately 72% MSE reduction on average for all prediction cases.

4.5. Discussion

Fractal modeling technique is proposed in [18] to predict cloud resource requests, which achieves MSE reduction of 72% and 59% for CPU requests and RAM requests, respectively, when compared with an ARIMA (16,0,0) predictor. As shown in the previous sections, the DBN method achieves higher MSE reduction for both CPU and RAM requests.

Contrasting our proposed method for predictions of resource requests in the Google cloud trace to ARIMA and Fractal modeling technique, we observe that

- The proposed method achieves MSE in the range $\left[10^{-6}, 10^{-5}\right]$. The best solution in all experiments achieves MSE = $9.216E^{-6}$ (long term), and $8.95E^{-6}$ (short term) for prediction of CPU requests, and MSE = $9.36E^{-6}$ (long term) and $8.86E^{-6}$ (short term) for prediction of RAM requests.
- Compared with the state-of-the-art work in [18] where fractal modeling technique is used, the DBN-based approach has higher accuracy for both long-term and short-term prediction.
- The most accurate prediction for CPU requests arises from the 9th, whereas that for the RAM requests arises from is the 5th experiment. This indicates that CPU requests in the Google cloud trace show a more complex pattern than RAM requests, requiring longer input nodes and more RBMs. However, the using 40% of the available samples as the training set fits for both CPU and RAM requests.

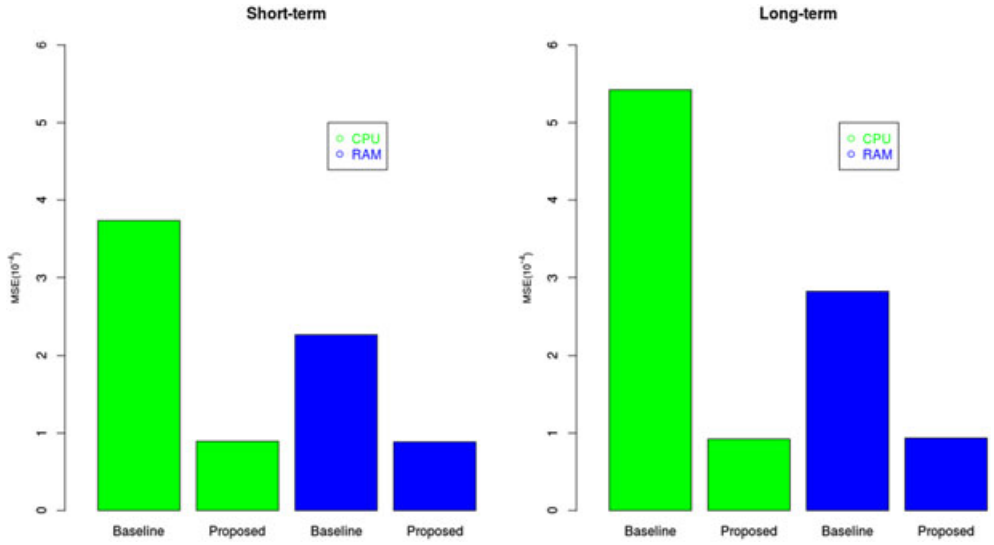

Figure 4. Comparing the proposed deep belief network (DBN) approach with ARIMA(2,3,5).

DOI: 10.1002/spe

4.6. Analysis and verification

We also want to ascertain which factors are most important and what combination of different factors is the best combination for DBN prediction accuracy. The average MSE (Tables IV, V, VI, and VII) show the best level combination to be the 9th and 5th experiments for CPU and RAM requests, respectively.

However, this is not rigorous. We need to conduct a statistical analysis on these factors to identify their relative effects on prediction performance. We derive an ith experiment score

$$s(i) = \frac{Max\left\{\sqrt{AMSE}\right\} - Min\left\{\sqrt{AMSE}\right\}}{\sqrt{AMSE(i)}},$$

where AMSE is the average MSE. Using ANOVA on the OED experiments, each of which is assigned a score, we can estimate each factor's effect, as shown in Tables VIII and IX, where F is the factor, L the level, and EN the experiment number. For a specific factor, k_i represents the average score of each level of the factor. R_i is the data range of the ith factor, which is the difference between the maximum and the minimum in the 3 k_is of that factor, and represents the factor's importance. For a factor, a superior level (SL) is the level with the highest average score in the 3 k_is, representing the level that achieves the best prediction performance. MF(main influence) represents the main influence factor rank, obtained by sorting R_i to identify the most influential factor and how much it affects prediction performance.

The superior levels are $[IN3 = 10, RN3 = 3, LR2 = 0.2, TSS1 = 40\%]$ (the 9th experiment) for both short-term and long-term CPU requests (Table VIII). Similarly, RAM request (Table IX) shows the best solution is $[IN2 = 5, RN2 = 2, LR3 = 0.5, TSS1 = 40\%]$, the 5 experiments. Therefore, the simplistic mean RMS assessment is verified. For short-term prediction, the main contribution factors are the training sample size (TSS) for CPU requests, and the number of RBMs for RAM requests. For long-term prediction, the number of RBMs is the single key role for both CPU and RAM requests.

Through the whole paper, we treated Google cluster trace as time series, analyzed it with automatic ARIMA and deep parameter learning combined with statistical methods. Given a time series, ARIMA automatically determines the number of points in the time series that the current point depends on. The dependent number of points is then naturally treated as the input of the first layer of DBN. Then statisctical methods are applied into the determination of parameters (IN, RN, LR and

Table VIII. Experimental results for CPU requests.

Short-term analysis						Long-term analysis					
F / L	IN 1	RN 2	LR 3	TSS 4	Score	F / L	IN 1	RN 2	LR 3	TSS 4	Score
EN						EN					
1	2	1	0.02	40.00%	0.759	1	2	1	0.02	40.00%	0.854
2	2	2	0.2	60.00%	2.507	2	2	2	0.2	60.00%	4.148
3	2	3	0.5	80.00%	0.810	3	2	3	0.5	80.00%	3.312
4	5	1	0.2	80.00%	0.811	4	5	1	0.2	80.00%	2.954
5	5	2	0.5	40.00%	2.408	5	5	2	0.5	40.00%	5.107
6	5	3	0.02	60.00%	2.475	6	5	3	0.02	60.00%	4.105
7	10	1	0.5	60.00%	2.128	7	10	1	0.5	60.00%	3.611
8	10	2	0.02	80.00%	1.025	8	10	2	0.02	80.00%	3.338
9	10	3	0.2	40.00%	3.151	9	10	3	0.2	40.00%	5.843
k1	1.359	1.233	1.419	2.406		k1	2.771	2.473	2.766	3.995	
k2	1.898	1.980	2.156	2.370		k2	4.055	4.198	4.315	3.955	
k3	2.101	2.145	1.782	0.882		k3	4.264	4.420	4.010	3.201	
R	0.742	0.912	0.737	1.488		R	1.490	1.950	1.549	0.754	
SL	IN3	RN3	LR2	TSS1		SL	IN3	RN3	LR2	TSS1	
MF	3	2	4	1		MF	3	1	2	4	

DOI: 10.1002/spe

Table IX. Experimental results for RAM requests.

Short-term analysis						Long-term analysis					
F / L	IN 1	RN 2	LR 3	TSS 4	Score	F / L	IN 1	RN 2	LR 3	TSS 4	Score
EN						EN					
1	2	1	0.02	40.00%	0.838	1	2	1	0.02	40.00%	0.762
2	2	2	0.2	60.00%	2.322	2	2	2	0.2	60.00%	1.697
3	2	3	0.5	80.00%	2.857	3	2	3	0.5	80.00%	1.878
4	5	1	0.2	80.00%	2.432	4	5	1	0.2	80.00%	1.562
5	5	2	0.5	40.00%	5.156	5	5	2	0.5	40.00%	3.205
6	5	3	0.02	60.00%	2.320	6	5	3	0.02	60.00%	1.703
7	10	1	0.5	60.00%	2.425	7	10	1	0.5	60.00%	1.704
8	10	2	0.02	80.00%	3.041	8	10	2	0.02	80.00%	1.953
9	10	3	0.2	40.00%	4.025	9	10	3	0.2	40.00%	2.211
k1	2.001	1.898	2.066	3.340		k1	1.446	1.343	1.473	2.060	
k2	3.303	3.506	2.926	2.356		k2	2.157	2.285	1.823	1.701	
k3	3.164	3.067	3.480	2.776		k3	1.956	1.878	2.262	1.797	
R	1.302	1.608	1.414	0.980		R	0.711	0.942	0.789	0.359	
SL	IN2	RN2	LR3	TSS1		SL	IN2	RN2	LR3	TSS1	
MF	3	1	2	4		MF	3	1	2	4	

TSS). Those parameters and the parameters ($\boldsymbol{c}$, $\boldsymbol{b}$, $\boldsymbol{W}$, σ) in the DBN can be very different for diverse types of time series in that different time series may be generated from different sources and reflect intrinsic attributes of distrinct phenomena. Therefore, the obtained DBN parameters in the paper are effective for Google cluster trace, but not necessarily effective for other time series. However, applying the methods proposed in this paper, we can automatically generate the best parameters for a specific time series data.

5. RELATED WORK

There are some similar work on Google cluster metric predictions. Sheng [5] proposed a Bayesian-based model to predict host loads and build long-term and short-term patterns of expectation, predictability, and trends. However, eliciting the priors probabilities of the Bayesian model is always difficult, and consequently the model tends to achieve low prediction accuracy. Rodrigo [33] introduced the prediction based on the ARIMA model and evaluate its accuracy of future workload prediction using real traces of requests to web servers. Chen [18] predicted cloud resource requests using fractal modeling influenced by self-similarity in the data. Therefore, the achieved high prediction accuracy was largely based on the fact that Google cloud resource requests must be self-similar. It achieves MSE reduction of 72% and 59% for CPU requests and RAM requests, respectively, when compared with an ARIMA (16,0,0) predictor. As shown in the previous sections, the DBN method achieves higher MSE reduction for both CPU and RAM requests. Hu [19] proposed using support vector regression (SVR), with the radial basis function as the kernel, to predict Google Cluster metrics. However, the choice of SVR kernel is highly depending on the domain specification, and adjusting the parameters of the kernel remains rather arbitrary, which place significant limitations on the generalization of the model. Roy [20] developed a model-predictive algorithm (ARIMA model) for workload prediction, which was then used for resource auto-scaling. However, as discussed earlier, linear models, that is, the ARIMA family, are inferior foe fitting highly variant patterns such as the Google cloud trace.

Except for prediction researches, there are also some other work on analysis of Google cluster trace. Di [34] proposed characterizing run-time task resource usage for CPU, memory, and disk to reproduce the performance of historical workload traces in terms of key performance metrics. Sheng [35] analysed the classification of applications via a K-means clustering algorithm with an optimized number of sets, based on task events and resource usage and build a model that simulated

DOI: 10.1002/spe

Google jobs, tasks, and dynamic events, in accordance with the Google trace dataset. Khan [15] characterized the temporal correlations in virtual machine clusters and predicted variations of workload patterns using hidden Markov modeling (HMM). However, while all these proposals provide cloud workload characterization, task clustering, and comparison between cloud and Grid patterns, they do not focus on resource request prediction in the cloud.

In contrast to using DBN for cloud computing metric prediction, researchers are also developing predictions in other areas. For example, Kuremoto [26] proposed to predict the artificial time series data using DBN with particle swarm optimization [36]. Busseti [37] used deep feed-forward neural networks and deep recurrent neural networks (RNNs) to forecast power demands. The current paper aimed to obtain short and long-term prediction patterns of Google cloud resource requests using DBN. The outcomes provide insights into extending this work using other deep learning techniques, such as RNN.

6. CONCLUSION AND FUTURE WORK

In this paper, we propose a DBN-based approach for cloud resource request prediction, using the dataset from the Google cloud trace. We first pre-process the Google cloud trace and feed the pre-processed data into a DBN for training, fine-tuning using BP to minimize the loss function. The proposed approach achieves average MSE $[10^{-6}, 10^{-5}]$. Compared with an ARIMA predictor, the MSE reduction is 76% and 61% for short term, and 83% and 67% for long-term CPU and RAM predictions, respectively. This significantly outperforms the current best model [18] using fractal modeling techniques.

A set of OED experiments were performed and assessed for ANOVA to identify the main factors and the level set that produced the best prediction performance. The main contribution factor was the training sample size (TSS) and the number of RBMs for short-term and long-term CPU request predictions, respectively, whereas the main contribution factor is the number of RBMs for both short-term and long-term RAM request predictions. The corresponding best level sets for predicting CPU and RAM requests were $[IN = 10, RN = 3, LR = 0.2, TSS = 40\%]$ and $[IN = 5, RN = 2, LR = 0.5, TSS = 40\%]$, respectively.

In the future, we plan to do a comprehensive study on the comparison among mutiple deep learning networks. The studies mainly focus on two aspects. One is to conduct the prediction accuracy comparison among different deep networks. One is to conduct the resource load comparison among them.

ACKNOWLEDGEMENTS

This research was supported by the National Natural Science Foundation of China (Grant No. 61309024 and 61402533) and Natural Science Foundation of Shandong Province (Grant No. ZR2014FM038), "Key Technologies Development Plan of Qingdao Technical Economic Development Area". Weishan Zhang was supported through the "Academic Top-Notch Professors in China University of Petroleum" program.

REFERENCES

1. Menzel M, Wang L, Khan SU, Chen J, et al. Cloudgenius: a hybrid decision support method for automating the migration of web application clusters to public clouds. *IEEE Transactions on Computers* 2015; **64**(5): 1336–1348.
2. Ma Y, Wang L, Zomaya AY, Chen D, Ranjan R. Task-tree based large-scale mosaicking for massive remote sensed imageries with dynamic dag scheduling. *IEEE Transactions on Parallel and Distributed Systems* 2014; **25**(8): 2126–2137.
3. Liu C, Ranjan R, Yang C, Zhang X, Wang L, Chen J. Mur-dpa: top-down levelled multi-replica merkle hash tree based secure public auditing for dynamic big data storage on cloud. *IEEE Transactions on Computers* 2015; **64**(9): 2609–2622.
4. Zhang Q, Chen Z, Yang LT. A nodes scheduling model based on markov chain prediction for big streaming data analysis. *International Journal of Communication Systems* 2015; **28**(9):1610–1619.
5. Sheng D, Kondo D, Cirne W. Host load prediction in a google compute cloud with a bayesian model. *IEEE/ACM 24th International Conference for High Performance Computing, Networking, Storage and Analysis (SC'12)*, Los Alamitos, CA, 2012; 21: 1–11.

DOI: 10.1002/spe

6. Liang J, Nahrstedt K, Zhou Y. Adaptive multi-resource prediction in distributed resource sharing environment. *IEEE International Symposium on Cluster Computing And The Grid, 2004. CCGrid 2004*, IEEE, Washington, D.C., 2004; 293–300.
7. Dabrowski C, Hunt F. Using markov chain analysis to study dynamic behaviour in large-scale grid systems. *Proceedings of the Seventh Australasian Symposium On Grid Computing And E-Research-Volume 99*, Australian Computer Society, Inc., Darlinghurst, 2009; 29–40.
8. Dinda PA, O'Hallaron DR. Host load prediction using linear models. *Cluster Computing* 2000; **3**(4):265–280.
9. Carrington L, Snavely A, Gao X, Wolter N. A performance prediction framework for scientific applications. In *Computational Science-ICCS 2003,* Springer, 2003; 926–935.
10. Tirado Juan M, Higuero Daniel, Isaila Florin, Carretero Jesus. Multi-model prediction for enhancing content locality in elastic server infrastructures. *International Conference on High Performance Computing (HiPC), 2011 18th*, IEEE, Washington, D.C., 2011; 1–9.
11. Akioka S, Muraoka Y. Extended forecast of cpu and network load on computational grid. *IEEE International Symposium on Cluster Computing and the Grid, 2004. CCGrid 2004*, IEEE, Washington, D.C., 2004; 765–772.
12. Zhang Y, Wei S, Inoguchi Y. Cpu load predictions on the computational grid. *IEICE TRANSACTIONS on Information and Systems* 2007; **90**(1):40–47.
13. Wu Y, Hwang K, Yuan Y, Zheng W. Adaptive workload prediction of grid performance in confidence windows. *IEEE Transactions on Parallel and Distributed Systems* 2010; **21**(7):925–938.
14. Wolski R, Spring N, Hayes J. Predicting the cpu availability of time-shared unix systems on the computational grid. *The Eighth International Symposium on High Performance Distributed Computing, 1999. Proceedings*, IEEE, Washington, D.C., 1999; 105–112.
15. Khan A, Yan X, Tao S, Anerousis N. Workload characterization and prediction in the cloud: A multiple time series approach. *Network Operations and Management Symposium (NOMS), 2012 IEEE*, IEEE, Washington, D.C., 2012; 1287–1294.
16. Zhang Q, Hellerstein JL, Boutaba R. Characterizing task usage shapes in google's compute clusters. *Large Scale Distributed Systems and Middleware Workshop (LADIS'11)*, 2011.
17. Sharma B, Chudnovsky V, Hellerstein JL, Rifaat R, Das CR. Modeling and synthesizing task placement constraints in google compute clusters. *Proceedings of the 2nd ACM Symposium on Cloud Computing*, ACM, New York, 2011; 3. 3:1–3:14.
18. Chen S, Ghorbani M, Wang Y, Bogdan P, Pedram M. Trace-based analysis and prediction of cloud computing user behavior using the fractal modeling technique. *International Congress on Big data (BigData Congress), 2014 IEEE*, IEEE, Washington, D.C., 2014; 733–739.
19. Hu R, Jiang J, Liu G, Wang L. Efficient resources provisioning based on load forecasting in cloud. *The Scientific World Journal,* New York, 2014; **2014**:1–14.
20. Roy N, Dubey A, Gokhale A. Efficient autoscaling in the cloud using predictive models for workload forecasting. *International Conference on Cloud Computing (CLOUD), 2011 IEEE*, IEEE, Washington, D.C., 2011; 500–507.
21. Saripalli P, Kiran G, Shankar RR, Narware H, Bindal N. Load prediction and hot spot detection models for autonomic cloud computing. *International Conference on Utility and Cloud Computing (UCC), 2011 Fourth IEEE*, IEEE, Washington, D.C., 2011; 397–402.
22. Jiang Y, Perng C-s, Li T, Chang R. Asap: A self-adaptive prediction system for instant cloud resource demand provisioning. *International Conference on Data Mining (ICDM), 2011 IEEE 11th*, IEEE, Washington, D.C., 2011; 1104–1109.
23. Hinton G, Osindero S, Teh Y-W. A fast learning algorithm for deep belief nets. *Neural Computation* 2006; **18**(7):1527–1554.
24. Bengio Y. Learning deep architectures for ai. *Foundations and Trends® in Machine Learning* 2009; **2**(1):1–127.
25. Connor JT, Martin RD, Atlas LE. Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks* 1994; **5**(2):240–254.
26. Kuremoto T, Kimura S, Kobayashi K, Obayashi M. Time series forecasting using a deep belief network with restricted boltzmann machines. *Neurocomputing* 2014; **137**:47–56.
27. Carreira-Perpinan MA, Hinton GE. On contrastive divergence learning. *Artificial Intelligence & Statistics*, New York, 2005; 33–40.
28. Gardner Jr ES, McKenzie E. Note-seasonal exponential smoothing with damped trends. *Management Science* 1989; **35**(3):372–376.
29. Aote SS, Raghuwanshi M, Malik L. A brief review on particle swarm optimization: Limitations & future directions. *International Journal of Computer Science Engineering (IJCSE)* 2013; **14**(1):196–200.
30. Hyndman RJ, Kh Y. Automatic time series forecasting: The forecast package for r. *Journal of Statistical Software* 2008; **27**(3):1–22.
31. Zhang GP, Patuwo BE, Hu MY. A simulation study of artificial neural networks for nonlinear time-series forecasting. *Computers & Operations Research* 2001; **28**(4):381–396.
32. Hamilton JD. *Time series analysis*, Vol. 2. Princeton University Press: Princeton, 1994; 68–86.
33. Calheiros RN, Masoumi E, Ranjan R, Buyya R. Workload prediction using arima model and its impact on cloud applicationsâĂá qos. *IEEE Transactions on Cloud Computing* 2015; **3**(4):449–458.

DOI: 10.1002/spe

34. Di S, Kondo D, Cirne W. Characterization and comparison of cloud versus grid workloads. *IEEE International Conference on Cluster Computing (CLUSTER), 2012*: IEEE, 2012; 230–238.
35. Di S, Kondo D, Cappello F. Characterizing and modeling cloud applications/jobs on a google data center. *The Journal of Supercomputing* 2014; **69**(1):139–160.
36. Kennedy J. Particle swarm optimization. In *Encyclopedia of Machine Learning*. Springer, 2010; 760–766.
37. Busseti E, Osband I, Wong S. Deep learning for time series modeling. *Technical Report*, Stanford University, 2012.

DOI: 10.1002/spe