Hindawi
Complexity
Volume 2017, Article ID 8584252, 10 pages
https://doi.org/10.1155/2017/8584252

Research Article

Deep Recurrent Model for Server Load and Performance Prediction in Data Center

Zheng Huang,[1,2] Jiajun Peng,[1] Huijuan Lian,[1] Jie Guo,[1] and Weidong Qiu[1]

[1] *School of Cyber Security, Shanghai Jiao Tong University, Shanghai, China*
[2] *Westone Cryptologic Research Center, Beijing 100070, China*

Correspondence should be addressed to Jiajun Peng; pjj@sjtu.edu.cn

Received 31 August 2017; Accepted 2 November 2017; Published 26 November 2017

Academic Editor: Jia Wu

Recurrent neural network (RNN) has been widely applied to many sequential tagging tasks such as natural language process (NLP) and time series analysis, and it has been proved that RNN works well in those areas. In this paper, we propose using RNN with long short-term memory (LSTM) units for server load and performance prediction. Classical methods for performance prediction focus on building relation between performance and time domain, which makes a lot of unrealistic hypotheses. Our model is built based on events (user requests), which is the root cause of server performance. We predict the performance of the servers using RNN-LSTM by analyzing the log of servers in data center which contains user's access sequence. Previous work for workload prediction could not generate detailed simulated workload, which is useful in testing the working condition of servers. Our method provides a new way to reproduce user request sequence to solve this problem by using RNN-LSTM. Experiment result shows that our models get a good performance in generating load and predicting performance on the data set which has been logged in online service. We did experiments with nginx web server and mysql database server, and our methods can been easily applied to other servers in data center.

1. Introduction

In the past few decades, the World Wide Web (WWW) [1] has experienced phenomenal growth and server systems have become more and more complex and performance-hungry. With the popularization of B/S structure, data processing is further centralized to the server, which means new challenges to the management of server performance [2]. It is an important part in computer system performance management to predict the server's infrastructure resource performance (like CPU rate and throughput) and workload (user's requests) correctly and effectively, which helps improve quality of the service while minimizing the wastage in resource utilization.

Many applications of big data analytics have been developed to enhance the operation of cloud computing and web server infrastructures in recent years [3]. Previous methods for the performance prediction job basically fall into two categories: one is focusing on building the relation between performance and time, such as neural network (MLP), and linear regression [4], weighted multivariate linear regression (MVLR) [5], and recurrent neural network (RNN) [6] even LSTM [7] are used; another one does not consider the sequential effects and predicts the performance by analyzing the workload; for example, Yu et al. [8] use Clustering and Multilayer Perceptron (MLP) to do this task.

Both of these two kinds of method may not explore the essence of the problem. Since the fluctuation of server's performance is caused by user's request sequence, we argue that both user's behavior and its property of the sequence need to be considered when predicting the web server performance. And predicting the performance based on events sequence (user's requests) is our new idea.

The workload in our research is the requests loaded to the server; thus predicting the workload means predicting the requests sequence accessed by the users actually, which has few previous related studies. Previous works in this job focused on predicting the total request situation, for example, the total number of users or the number of requests in a time-window, but did not consider the detail of it, such as the research of Vercauteren et al. [9]. So it is very difficult to reproduce the workload to the server for testing the performance of servers, which can be very useful for server management.

Like the idea for performance prediction, we argue that users request to servers is the base of the total workload, so the job mentioned here is reproducing the user's access sequence, which can be considered a kind of user characterization.

Recently deep neural networks start to show their great capability in language modeling [10]. And recent research reveals that RNN significantly outperforms popular statistical algorithms [11]. As a special kind of RNN, LSTM neural network [12] is proved to be efficient in modeling sequential data like speech and text [13]. These previous researches inspires us to use LSTM in these prediction tasks, because user's request is a sequential data.

For the purpose of improving the performance of predicting web server performance and workload, we apply RNN-LSTM network with requests-to-vector to this task. Our contributions can be summarized as two points:

(1) Our work is the first one to apply RNN-LSTM network to predict the performance and workload of Web servers or data center.

(2) We proposed to investigate the relation between users' requests sequence and web server performance, which previous researches did not pay much attention to.

In a word, our research consists of two models; the workload prediction model in this paper is used to generate an analog workload which is the user's specific request sequence. And the one for performance prediction can predict the performance by analyzing the user's request sequence.

This paper is organized as follows. Section 2 introduces LSTM network and the architecture of our models. Then we introduce our training and application framework in Section 3. Section 4 shows the details and results of our experiments and compares our work with previous researches. Finally, Section 5 is the conclusion of the whole paper.

2. Model

2.1. LSTM Network. As shown in Figure 1, the basic structure of a LSTM unit is composed of a memory cell $c^t \in R^d$ and three essential gates: Input Gate $i^t \in R^d$, Output Gate $o^t \in R^d$, and Forget Gate $f^t \in R^d$.

The formulas for updating the state of each gate and cell in a LSTM unit using the input of x^t, y^{t-1}, and c^{t-1} are defined as follows:

$$\begin{aligned} z^t &= g\left(W_z x^t + R_z y^{t-1} + b_z\right), \\ f^t &= \sigma\left(W_f x^t + R_f y^{t-1} + b_f\right), \\ i^t &= \sigma\left(W_i x^t + R_i y^{t-1} + b_t\right), \\ o^t &= \sigma\left(W_o x^t + R_o y^{t-1} + b_o\right), \\ c^t &= i^t \odot z^t + f^t \odot c^{t-1}, \\ y^t &= o^t \odot h\left(c^t\right). \end{aligned} \tag{1}$$

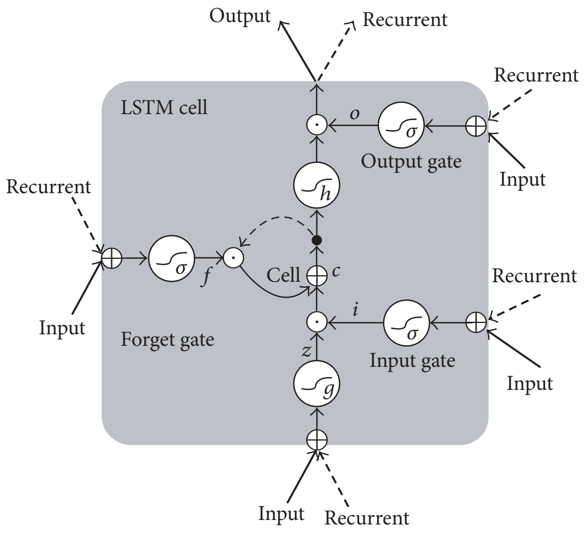


Figure 1: Schematic of LSTM unit [14].

Here x^t means the input feature vector at time t. Similarly, y^{t-1} and c^{t-1} are the output vector and cell state at time $t-1$. And each of them is a d-dimensional value. W in formulas are the weight matrices of the input parts in the gates and cell of LSTM network, and R are the ones of the recurrent parts. b means bias vectors of each formula. As for the $\odot$ mark, it means pointwise multiplication. The $\sigma(x)$, $g(x)$, and $h(x)$ functions are the activation functions of every part in LSTM, which determine the amount of information that can be passed. And we use sigmoid as the activation function of three gates ($\sigma(x)$ in the formulas) and $g(x)$, and we use the rectified linear units (ReLUs) function [15] as the function $h(x)$ in the formulas. ReLU function is a very popular new nonlinear activation function and it is defined as follows:

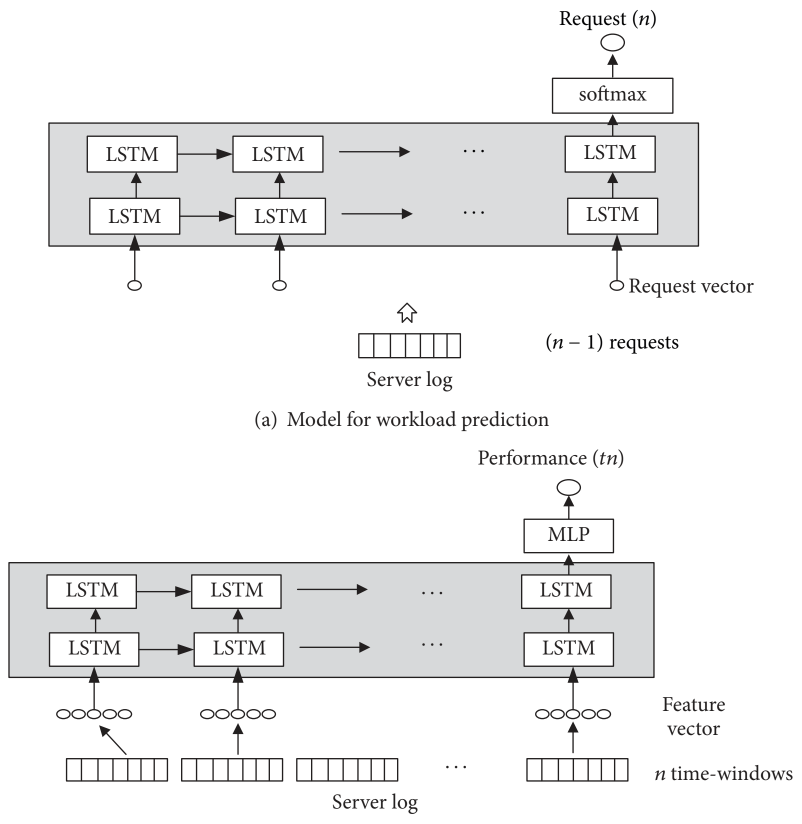
$$h(x) = \max(0, x). \tag{2}$$

And using ReLU as the activation function can make the network be trained several times faster than using equivalents with saturating neurons like tanh and sigmoid [15].

With this special structure, LSTM network is robust with respect to exploding and vanishing gradient problems [12], so it is able to learn long-term dependencies which RNN cannot perform very well and makes the model be trained without hand-generated features.

Because of the advantages of LSTM, we use LSTM as the basic part of our model to capture the sequential information of requests and then predict the workload and the performance by using the highly abstract features generated by LSTM layers.

2.2. Model for Workload Prediction. One of our preliminary ideas for this model is taking each user's request as an instance and certain length of user's request as a bag containing a number of instances, and the label of the bag is set to be the next request of the final one in the bag, so predicting users requests can be regarded as a problem of multiple-instance



(a) Model for workload prediction

(b) Model for performance prediction

FIGURE 2: The network structure for two models: (a) model for workload prediction and (b) model for performance prediction. Model (a) takes request vectors as the input and output of the network, using the latest $n-1$ requests to predict the nth request. The input of model (b) is the latest n feature vectors extracted from the log of every time-window, and its output is the value of certain kind of performance in n time-window.

learning (MIL), which has been used widely in drug discovery, text categorization, and graph classification [16]. Previous works like the researches of Wu et al. [17, 18] have shown good classification effectiveness in the area of graph classification. However, the sequential features of the requests are not considered if doing prediction in this way. So we finally selected using LSTM network in this prediction task.

Our model for workload prediction is designed to predict the request sequence of users, which is similar to natural language generation. As shown in Figure 2, the base layer of both our models is two layers of RNN-LSTM, which can capture the features of the user request sequence. And we only use the outputs of the final LSTM unit to do the prediction, which is the many-to-one model of LSTM. This means we use the previous $n-1$ request to predict the nth request (suppose the length of the sequence is n).

The output layer of the model for workload prediction is obviously a multiclassification task for every user request. So the output of the LSTM layer is designed to be passed into a hidden layer with softmax function, which finally outputs the probability of each request in the $[0,1]$ interval. Suppose the input vector of the softmax layer is $(\widehat{y_i^1}, \widehat{y_i^2}, \ldots, \widehat{y_i^k})$; we get the output vector of this layer $(p_i^1, p_i^2, \ldots, p_i^k)$ by using following formula:

$$P\left(\widehat{y_i^j}\right) = \frac{e^{\widehat{y_i^j}}}{\sum_{k=1}^{K} e^{\widehat{y_i^k}}}, \quad j \in [1, K]. \tag{3}$$

And the model is trained by minimizing the cross-entropy error between the output of the whole network and the true value. Using cross-entropy as the loss function is the most popular choice in the area of multiclassification task, which also achieves good performance. Suppose the true label is $y_i \in [1,k]$, and the true vector is one-hot, so the loss function is defined as follows:

$$J(\theta) = -\sum_{j=1}^{k} 1\{y_i = j\} \log P\left(\widehat{y_i^j}\right), \tag{4}$$

where θ means the parameters of the network, and the symbol $1\{\cdot\}$ means an indicator that equals one when the condition in the brackets returns true and it equals zero otherwise.

2.3. Model for Performance Prediction. Different from the traditional work of RNN-LSTM in NLP task and the model for workload prediction, of which the output layer is usually designed as a classifier, the output layer of this model is designed as a liner regression task for every performance. We want to predict the performance like throughput, request delay, and CPU rate, which are not classifications but certain numbers, by analyzing the requests sequence recorded by the server. So the output layer of this model is designed to be a Multilayer Perceptron (MLP), which can fit any continuous function with enough neurons in theory [19].

And ReLU is also used as the activation functions of each fully connected layer here. To prevent the neural networks from overfitting, a dropout [20] layer is connected between two fully connected layers and LSTM layers with the fixed probability $p = 0.5$. This probability means half of the units in the network will be randomly selected and then temporarily removed from the network when training the network. It has been proved by previous work that performance of the networks can be improved dramatically when applying dropout layers at multiple LSTM layers [21].

As shown in Figure 2(b), this model uses the latest requests in n time-windows to predict the performance in the nth time-window, which is a small but important difference between the models for workload, because the performance is affected by not only previous requests but also current operation of users, while the nth request is just strongly related to previous requests.

For the linear regression tasks, L1 norm (absolute differences), L2 norm (squared differences), and smooth L1 [22] are well-known loss functions. The L1 norm is not smooth when the error is close to 0, so it is seldom used. Smooth L1 is a robust L1 loss, and it is less sensitive to outliers than the L2 loss [22]. But all of our data are fed after normalization, and there is little outliers out data set; considering the simplicity of realization, we choose L2 norm as the loss function of this model.

Supposing that the input vector is x_i, $\widehat{y_i}$ is the value predicted by the model, and θ means the network parameters, we use the variance between the real value of the performance and the predicted one $(\widehat{y_i} - y_i)^2$ as the cost of $\widehat{y_i}$. So the total loss function of the model is defined as follows:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\widehat{y_i} - y_i\right)^2, \tag{5}$$

which is the mean squared error (MSE) between predicted and true value, and m is the number of sequences in a batch.

3. Training and Application Framework

3.1. Training Framework. Training framework of our models is shown in Figure 2; (a) is the model for workload prediction and (b) is the one for performance prediction.

As shown in Figure 2, log files of servers are the raw data source of our models, so the step for data processing is quite similar, and the main idea of our models is predicting the web server performance and workload by analyzing the log files. Each request has an one-to-one ID in the form of integer which is stored in a dictionary. The dictionary is generated by collecting all the unique request string in the whole data set. Using this ID, the requests can be abstracted to a one-hot vector with d-dimensional, and we call it request vector. The request vector of the request with ID 1 is $v = (1, 0, 0, \ldots)$ and the one with ID 2 is $v = (0, 1, 0, \ldots)$ and so on. The feature vectors of requests during a time-window can be generated by adding each request vector, and thus every dimension of which means the number of times user requests the server during a time-window. Take feature vector $v = (p^1, p^2, \ldots, p^d)$ as an example; p^i $i \in [1, d]$ in v means that user sends the request whose ID is i altogether p^i times in a time-window.

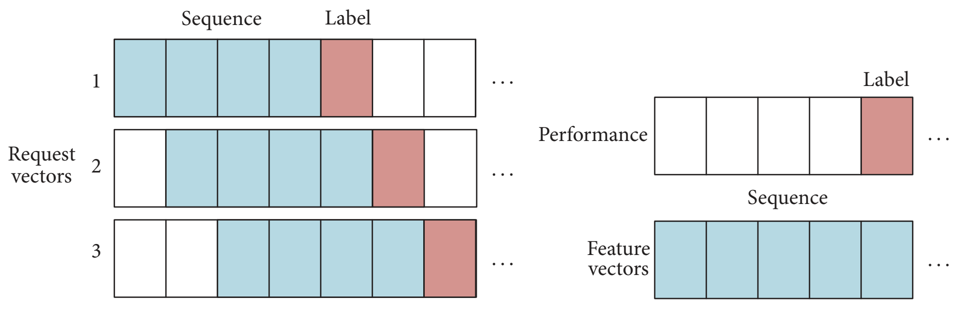
Actually, this step is not limited to abstract a certain form of user's requests. For example, the request to a database and URL record of a web site also can be done in this way. Its key point is to describe the user's request in a mathematical way, which is using the ID to represent different kinds of user's behavior. And this initialization step is regarded as request-to-vector. After the step of request-to-vector, a long sequence of feature vectors or request vectors can be generated, which is the input of our models.

For the model for workload prediction, previous $n - 1$ request vector should be fed into the network, and the nth request vector is the true label of this request vector sequence. Because of using the many-to-one model of LSTM, the output of the model for workload is the nth request, which is the same as the model for natural language generation. Although the meaning of the label output by the network is different, the principle is the same. So 1 to $n - 1$ request vector is set to be the first sequence, and 2 to n is the second one, and so on, which is similar to the effect of sliding window with the length of $n - 1$. Figure 3(a) shows the process of data set generation for this model.

As for the model for performance prediction, n feature vectors should be fed into the network; the output of the network should be the performance at time t_n. The process of data set generation for this model is shown in Figure 3(b). As the previous feature vectors and the current one both should be put into the network, the process of data set generation has a little difference between the ones shown in Figure 3(a). All of the performance values are normalized by dividing the maximum value in theory before feeding them into the network, which can improve the efficiency of training and the performance of the network [23]. Theoretically, this model is not limited to use the specific kinds of performance, CPU, and memory occupancy rate of the server and other performances also can be trained if data exits.

Finally, sequences of request vector or feature vector are put into the LSTM layer of our models. And the output vector of the LSTM is then passed to the upper layer, which is the softmax layer for the model of workload prediction and the MLP network for the one of performance prediction. And we apply the RMSPROP gradient descent algorithm when training both two networks which is an improved version of SGD algorithm and has batter performance with minibatches [24].

3.2. Application Framework. With regard to the use of the model, we propose an application framework, which consists of three kinds of model selection, as shown in Figure 4. We can use the models to predict the workload and performance



(a) Data set generation for the model for workload prediction

(b) Data set generation for the model for performance prediction

Figure 3: Data set generation for two models: (a) model for workload prediction and (b) model for performance prediction. The blue boxes are the input sequence of the model, and the red box means the label of the sequence. (a) Model for workload prediction: input sequence is a certain number of previous request vectors, and the label is the request vector right after the last request vector of the input sequence. (b) Model for performance prediction: input sequence is a certain number of previous feature vectors, and the label is the performance value of the last feature vector of the input sequence.

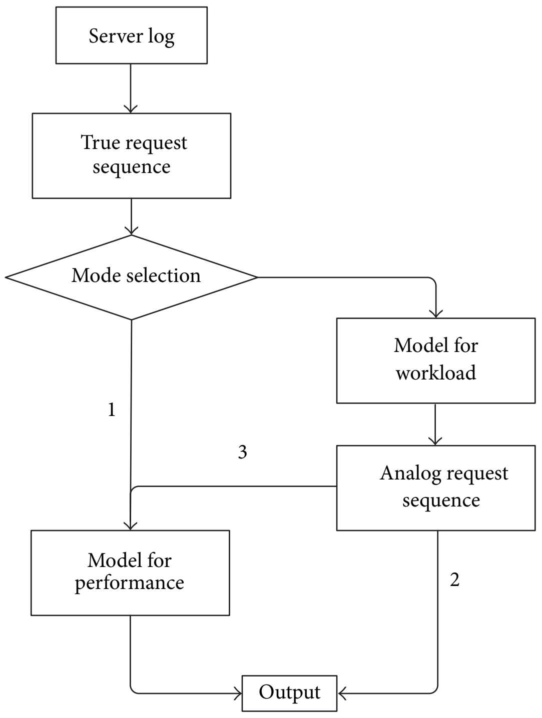


Figure 4: Application framework for two models.

of the server by analyzing new log files. The two models can be used separately, which is the basic way to use our models (1, 2 in Figure 4). After training the model for workload prediction, request sequences under different load conditions can be produced by feeding different seeds into the network. The training and application methods of this model are very similar to the usage of RNN-LSTM in the area of natural language generation (NLG). As for the model for performance prediction, it can predict the performance of server under different workload conditions.

And two models can be used in combination; the model for workload prediction can first output analog request sequence and then feed the sequence into the model for performance prediction (3 in Figure 4). By using the model for workload prediction, request sequence with the original load features can be generated. This model of usage can meet the needs such as when the data of new log is too little, and the performance under a long-term workload is required. With these 3 kinds of options to apply these two models, the models can be used more flexibly and adapted to more situations.

On the other hand, our models are not limited by the network architecture theoretically. Our models can be deployed on the load balance node to predict the operation situation of the whole network or just deployed on the service nodes like data center node or calculation node to predict the operation situation of certain node.

In a word, our models can meet many different prediction needs regardless of the network architecture, which could be helpful in the management of data center.

4. Experiment

4.1. Setup. We keep the length of LSTM network to 15 (15 seconds), because we assume that a request cannot affect the web server performance and workload after 15 seconds in general. So the requests in each 15 seconds are organized into one sequence. Our model is trained and tested on the GPU: NVIDIA GeForce GTX 1080Ti, and the model was developed on the framework of theano and lasagne with CUDA to accelerate calculation. It took about 6 to 7 hours to finish the training of the model on the GPU. As a comparison, it will take more time to complete this job on an ordinary CPU.

For the model for workload prediction, it is very hard to measure the degree of similarity between the analog load and the real load. In our research, we just do the preliminary measure, which uses the difference between real and simulated proportion of each ID of request to measure the efficiency of this model. So cosine similarity is used to do this job, and the definition is shown follows:

$$\text{similarity} = \cos(\theta) = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}, \tag{6}$$

where A and B are the proportion vectors of the true one and generated one, and A_i and B_i are components of vectors A and B, respectively, which means the proportion of the request with ID i. So the value of the similarity is $[0, 1]$, and the closer the number is to 1, the more similar the two vectors are.

As for the one for performance prediction, we use mean squared error (MSE), which is also the loss function of the network to measure the efficiency of the model.

4.2. Experiment on Data Set A

4.2.1. Data Set. Data set A for evaluating the performance of our models contains the log files of 191 web servers nodes in one day. The web servers are set up using nginx and have been deployed in production environment, which means the records in log files are real. The log files of nginx can record the request in the form of URL. Besides the URL sequences, we can also get status code, the delay of each request, and the number of bytes of data transmission for each request from the log files. We can obtain the server's error rate, throughput, and average request delay, which is the three performances in this experiment. Table 2 shows the performance we tested in this experiment and their description. We regard the maximum throughput of the network card as the maximum of throughput, which is 12.5 Mb/S. 100 ms is the maximum of request delay. As for error rate, it is in $[0, 1]$ interval originally.

Because of the limit of GPU memory and the large number of log files, we choose to use log files of several random nodes which has the biggest log files to train the model.

4.2.2. Data Processing. First of all, we filter some requests which are not user's main operation, such as the requests for a picture or a json data. Then we get 57453 valid requests, which contain 2049 different URLs. For the model for workload prediction, all of the requests were set as training set. After requests-to-vector for URL requests and calculation of the performance mentioned in Section 3.1, a one-to-one relationship with URL requests and performance can be established. Finally, the data sets in the form of sequence according to the chronological order of URL requests are generated. We split the whole data sets into training set and test set, in accordance with the ratio of four to one. And one-fifth of the training set are set as validation set.

4.2.3. Result. In the experiment for performance prediction, three performances including request error rate, throughput, and request delay are simulated. The features of the network which has the best result on the validation set are saved, and then we perform a final test to see the performance of our network by using the test set.

Figure 7 shows a declining trend of the error in training, which means our model converges quite well. Because the value of objective function $(\widehat{y}_i - y_i)^2$ is very small in this experiment, we draw the picture using the data of $\log(\widehat{y}_i - y_i)^2$ which can show the trend more clearly. After training about

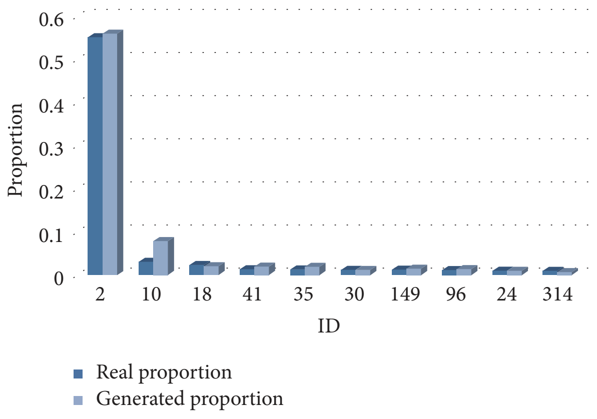


Figure 5: Proportion of top 10 ID on data set A.

200 epochs, the model gets the best result in training set and validation set.

Table 3 is the data of the final result about the objective function $(\widehat{y}_i - y_i)^2$ on the test set, compared with the results on the training set. The data of average $(\widehat{y}_i - y_i)^2$ in Table 3 shows that this model performs well in the task of predicting web server performance, which is a new area of using LSTM. On the other hand, the results of the two data sets (validation set and test set) are similar, so we think the model has a certain generalization ability.

In the experiment for workload prediction, the model generated 43077 URL requests with the same number of different request IDs. Because the number of different ID is too much, and most of the requests just appear so little times, we just compare the proportion vectors of top 10 frequent requests. Table 1 and Figure 5 show the result of this experiment. As shown in the table, the proportion of most kinds of request is less than 1%, and more than half of the requests are the request with ID: 2.

The cosine similarity is 0.996607614, which means the workload generated is very similar to the real workload and the model can capture the features of request sequence and regenerate them.

4.3. Experiment on Data Set B

4.3.1. Data Set. Because the requests in data set A concentrated in several requests, and the number of all requests is too much, we test our models on another data set, which we call it data set B. Different from A, data set B contains the log files of a database which also has been deployed in production environment. The requests of users are much more simple, and it contains only 4 types of requests. On the other hand, the performance data in this data set is recorded by other tools, so the logs of the server only provide the user request sequences and performance data is provided by the tools. Table 4 shows the performance we tested in this experiment and its description. We regard the maximum value in the data set as the maximum of average delay and

Table 1: Proportion of top 10 ID on data set A.

ID	Proportion									
	2	10	18	41	35	30	149	96	24	314
Real proportion	0.5497	0.0305	0.0226	0.0125	0.0124	0.0123	0.0123	0.0117	0.0111	0.0106
Generated proportion	0.5603	0.0789	0.0188	0.0193	0.0206	0.0128	0.0153	0.0141	0.0110	0.0083

Table 2: Performance and description on data set A.

Performance	Description
Error rate	The percentage of requests errors (like 404, 403,. . .)
Throughput	The total amount of data the server transmits
Request delay	The average delay in processing user requests

Table 3: Result of model for different performance on data set A.

Performance	Training set	Validation set	Test set
Error rate	$1.61 * 10^{-6}$	$1.71 * 10^{-06}$	$2.89 * 10^{-5}$
Throughput	$1.59 * 10^{-15}$	$1.60 * 10^{-15}$	$1.37 * 10^{-15}$
Request delay	$5.65 * 10^{-12}$	$6.33 * 10^{-12}$	$6.48 * 10^{-12}$

Table 4: Performance and description on data set B.

Performance	Description
CPU rate	The occupancy rate of the server's CPU
Average delay	Average delay of the requests
QPS	Query per second

Table 5: Result of model for different performance on data set B.

Performance	Training set	Test set
CPU rate	$1.002635 * 10^{-4}$	$8.360695 * 10^{-4}$
Average delay	$8.25704107 * 10^{-5}$	$2.9352345 * 10^{-3}$
QPS	$1.039598 * 10^{-4}$	$1.413895 * 10^{-4}$

QPS to do normalization. As for CPU rate, it is in $[0, 1]$ interval originally.

4.3.2. Result. In the experiment for performance prediction, three performances including CPU rate, average delay, and QPS are simulated. Table 5 is the data of the final result about the loss function $(\hat{y}_i - y_i)^2$, and the error of the predicted value is acceptably small.

In the experiment for workload prediction, the model generated 2000 request sequences with the length of 15 finally. The statistical result is shown in Table 6 and Figure 6, and the cosine similarity is 0.9985807, so the angle between two vectors is about 3°, which means that the workload is quite similar at the level of proportion.

The result on this data set shows that our model also performs well when the requests are in the form of the access to the database, which proves our theory that the form of the records or the requests does not affect the results of our model. And our model has the ability to adapt to a variety of situations.

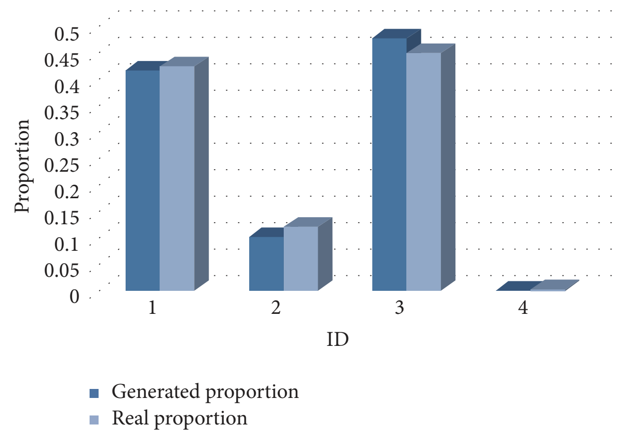


Figure 6: Proportion of each ID on data set B.

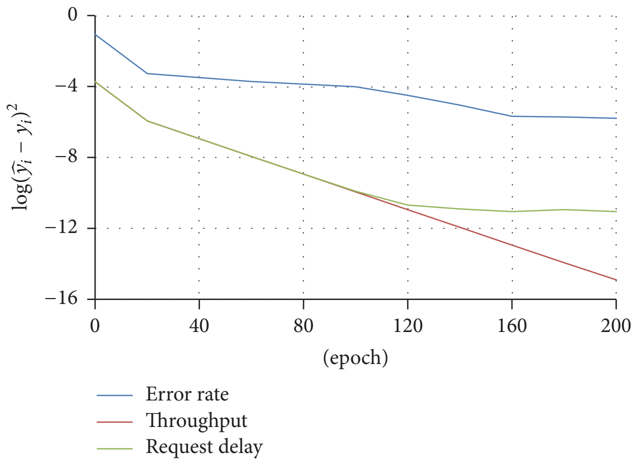


Figure 7: Illustration of declining trend of the error in training.

Table 6: Proportion of each ID on data set B.

Type (ID)	Generated proportions	Real proportions
1	0.41843333	0.4262904
2	0.10083333	0.1205442
3	0.48056667	0.4510679
4	0.00016667	0.0020976

In conclusion, the recurrent network can predict the web server performance and workload by analyzing the request sequences and turn out to have good performance both in accuracy and in generalization ability.

4.4. Related Works. Both of our models are built based on event (requests of users), which is a totally new method for

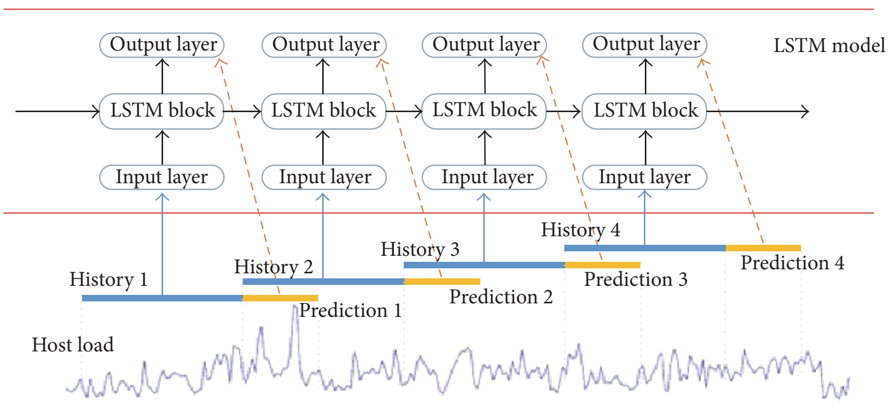


Figure 8: Model of Song et al. [7].

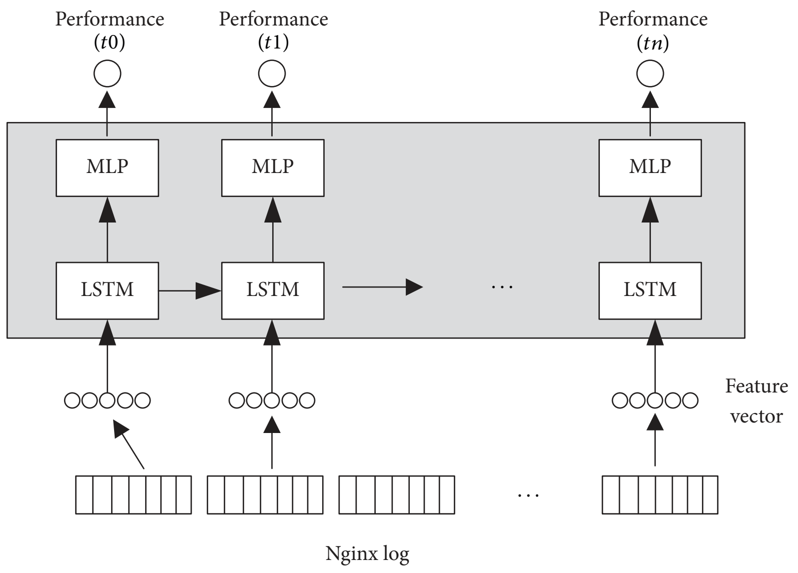


Figure 9: Model of our previous work.

Table 7: Result of model of Song et al. on data set B.

Performance	Training set	Test set
CPU rate	$3.6827354 * 10^{-3}$	$4.0699269 * 10^{-3}$
Average delay	$6.2893999 * 10^{-4}$	$2.7876459 * 10^{-3}$
QPS	$1.3815304 * 10^{-4}$	$2.3197048 * 10^{-3}$

the prediction task of the server or data center. So our model for performance prediction is very different form previous works in this prediction task. Take the work of Song et al. [7] as an example, which also uses LSTM in their work.

As shown in Figure 8, their work uses the previous performance to predict the future one, which means fitting the curve of performance with time. And it is also the key idea of previous work, but the fitting method of each is not the same.

As a comparative experiment, we did an experiment on data set B using the model of Song et al. [7]. The result is shown in Table 7, and the mean squared error (MSE) of each performance is larger than ours, which means our model achieves a better result.

As for the model for workload prediction, there is so little previous research, so we did not do the comparative experiment.

On the other hand, the research in this paper is an extension of our previous work [25], as shown in Figure 9. We change the model for performance prediction from many-to-many to many-to-one version of LSTM, and we propose the model for load prediction and the application framework of two models to improve our research.

In a word, our work is completely a new method to do this prediction job for sever and data center.

5. Conclusion

In this paper, we propose to use RNN-LSTM to predict web server performance and workload. Model for performance prediction is composed of RNN-LSTM and Multilayer Perceptron (MLP), and the one for workload prediction consists of RNN-LSTM and softmax layer. Doing the research based on events is a new way in this prediction area. The models can extract features automatically during the learning process without any prior knowledge or hand-generated features for segmentation. Experiments conducted on real data sets show that our models can achieve a good performance and generalization on predicting the performance of different kinds of severs. And the result also shows that the load generated by our model is very similar to the real one, which can be applied to test data center and other kinds of servers. Our results suggest that RNN-LSTM performs well on sequential tagging tasks; moreover RNN-LSTM with requests-to-vector is a new effective method to predict sever performance and workload which is worth further exploration. Most servers in data center has log system. As long as the log file recording the operation of the users is provided, our method can be used to generate load for the server and predict server performance under different load conditions. This can save a lot operation and maintenance work in data center.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work is supported by the National Key Research and Development Program of China under Grants 2017YFB0802704 and 2017YFB0802202 and Program of Shanghai Technology Research Leader under Grant 16XD1424400.

References

[1] T. Berners-Lee, R. Cailliau, J.-F. Groff, and B. Pollermann, "World-wide web: The information universe," *Internet Research*, vol. 20, no. 4, pp. 461–471, 2010.

[2] J.-F. Tu and R.-F. Guo, "The application reseach of mixed program structure based on client-server, browser-server and web service," in *Proceedings of the 2011 International Conference on Business Management and Electronic Information, BMEI 2011*, pp. 193–195, May 2011.

[3] R. Buyya, K. Ramamohanarao, C. Leckie, R. N. Calheiros, A. V. Dastjerdi, and S. Versteeg, "Big data analytics-enhanced cloud computing: Challenges, architectural elements, and future directions," in *Proceedings of the 21st IEEE International Conference on Parallel and Distributed Systems, ICPADS 2015*, pp. 75–84, December 2015.

[4] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155–162, 2012.

[5] I. Davis, H. Hemmati, R. C. Holt, M. W. Godfrey, D. Neuse, and S. Mankovskii, "Storm prediction in a cloud," in *Proceedings of the 2013 5th International Workshop on Principles of Engineering Service-Oriented Systems, PESOS 2013*, pp. 37–40, May 2013.

[6] B. Luo and S.-w. Ye, "Server performance prediction using recurrent neural network," *Computer Engineering and Design*, vol. 8, p. 57, 2005.

[7] B. Song, Y. Yu, Y. Zhou, Z. Wang, and S. Du, "Host load prediction with long short-term memory in cloud computing," *The Journal of Supercomputing*, pp. 1–15, 2017.

[8] Y. Yu, V. Jindal, I. Yen, and F. Bastani, "Integrating Clustering and Learning for Improved Workload Prediction in the Cloud," in *Proceedings of the 2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, pp. 876–879, San Francisco, Calif, USA, June 2016.

[9] T. Vercauteren, P. Aggarwal, X. Wang, and T.-H. Li, "Hierarchical forecasting of web server workload using sequential Monte Carlo training," *IEEE Transactions on Signal Processing*, vol. 55, no. 4, pp. 1286–1297, 2007.

[10] M. Auli, M. Galley, C. Quirk, and G. Zweig, "Joint language and translation modeling with recurrent neural networks," in *EMNLP*, vol. 3, 8 edition, 2013.

[11] M. Osawa, H. Yamakawa, and M. Imai, in *Proceedings of the Neural formation processg-23rd ternational conference, iconip 2016*, Springer Verlag, 2016.

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[13] M. Sundermeyer, H. Ney, and R. Schluter, "From feedforward to recurrent LSTM neural networks for language modeling," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 23, no. 3, pp. 517–529, 2015.

[14] X. Ma and E. Hovy, "End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1064–1074, Berlin, Germany, August 2016.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS '12)*, pp. 1097–1105, December 2012.

[16] J. Wu, S. Pan, X. Zhu, C. Zhang, and X. Wu, "Positive and Unlabeled Multi-Graph Learning," *IEEE Transactions on Cybernetics*, vol. 47, no. 4, pp. 818–829, 2016.

[17] J. Wu, S. Pan, X. Zhu, C. Zhang, and P. S. Yu, "Multiple Structure-View Learning for Graph Classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. PP, no. 99, pp. 1–16, 2017.

[18] J. Wu, S. Pan, X. Zhu, C. Zhang, and X. Wu, "Multi-instance Learning with Discriminative Bag Mapping," *IEEE Transactions on Knowledge and Data Engineering*, vol. PP, no. 99, pp. 1–14, 2017.

[19] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[20] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[21] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout Improves Recurrent Neural Networks for Handwriting Recognition," in *Proceedings of the 14th International Conference on Frontiers in Handwriting Recognition, ICFHR 2014*, pp. 285–290, September 2014.

[22] R. Girshick, "Fast R-CNN," in *Proceedings of the 15th IEEE International Conference on Computer Vision (ICCV '15)*, pp. 1440–1448, December 2015.

[23] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," *Learning*, vol. 3, 2015.

[24] G. Hinton, N. Srivastava, and K. Swersky, *Neural networks for machine learning lecture 6a overview of mini-batch gradient descent*, Coursera Lecture slides, 2012.

[25] J. Peng, Z. Huang, and J. Cheng, "A Deep Recurrent Network for Web Server Performance Prediction," in *Proceedings of the 2017 IEEE Second International Conference on Data Science in Cyberspace (DSC)*, pp. 500–504, Shenzhen, China, June 2017.