Workload Time Series Prediction in Storage Systems: A Deep Learning based Approach

Li Ruan,Yu Bai,Limin Xiao,Rongbin Xv,Yiyang Zhang
State Key Laboratory of Software Development Environment, Beihang University
Beijing, China
ruanli@buaa.edu.cn

Shuibing He
School of Computer Science Wuhan University
Wuhan, China
heshuibing@whu.edu.cn

Hongtao You
Wuxi Jiangnan Institute of Computing Technology
Wuxi, China
edmundyou@163.com

***Abstract*—Storage workload prediction is a critic step for fine-grained load balancing and job scheduling in realtime and adaptive cluster systems. However, how to perform workload time series prediction based on a deep learning method has not yet been thoroughly studied. In this paper, we propose a storage workload prediction framework using a deep learning approach. This framework includes workload collection, data preprocessing, time series prediction, and data post-processing phase. The time series prediction phase is based on a long short-term memory network(LSTM). To improve the efficiency of LSTM, we present a hierarchical parameter searching method, which consists of a coarse searching phase and a fine-grained searching phase. Further more, we study the sensitivity of the hyperparameters in LSTM. Extensive experimental results show that our framework can obtain performance improvement compared with three classic time series prediction algorithms.**

***Keywords*—workload prediction, time series, deep learning**

I. Introduction

In the big data era, storage system performance becomes a critical bottleneck of many data-intensive applications. With the increasing diversity of such applications, simply scaling up a single server or scaling out the cluster can not completely solve the storage bottleneck issue. Precise workload prediction provides a critical approach to address this issue because it helps to achieve fine-grained load balancing and job scheduling [1]. For example, inevitable hot data phenomenon will usually cause performance hungry for some servers but cause hardware wastage for the others. If we can precisely predict each server's workload, we can migrate data from the saturate servers to the hungry servers with the predicted workload information. This can greatly increase system resource utilization and improve system performance.

Previous researches on server workload prediction can mainly be classified into two categories [2]. The first category focuses on constructing the relationship between the time and the workload by using regression models [3] [4]. The second one pays more attention to analyze the characteristics of current workload for future workload prediction [3]. Decision trees and clustering [5] are the commonly used prediction methods. However, with the increasing complexity of input workloads, existing effects are hard to extract the underlying function.

Due to the merits of modeling nonlinear relationship between input and output, deep learning method gains increasing attention for real world problem modeling with the success in many fields, such as patten recognition, natural language processing, etc. However, to the best of our knowledge, although deep learning shows promising potential in time series prediction, how to perform a deep learning based storage workload prediction has not been studied yet.

Our contributions are as following:

- We propose a practical storage system prediction framework, which includes workload collection, data preprocessing, time series prediction based on long short-term memory network(LSTM), and data post-processing phase.
- To improve the training performance of the deep learning network, we present a hierarchical parameter searching method, which consists of a coarse searching phase and a fine-grained searching phase
- We study the sensitivity of the hyperparameters in LSTM and conduct extensive experiments to show the benefits of our framework over three classic time series prediction algorithms.

II. The Server Workload Time Series Prediction Process based on Deep Learning

Our practical storage system workload time series prediction method includes workloads collection, data preprocessing, time series prediction based on long short-term memory network(LSTM) and data postprocessing. The whole model framework is shown in Fig. 1. Firstly, we collect the workload time series data to be analyzed which consist of workload at different time step. Then the workload data will be sliced by a history time window of length *W*. With the time window sliding along the time axis, we will get the appropriate data structure for LSTM model. After that a data preprocessing method is applied to make the raw data easily to be predicted, and we will describe it in the Experiments section. Then our many-to-one LSTM network is applied in the prepared data, the workload at next time step is then predicted. Finally, we use an opposite operation of the preprocess method to recover the real workload.

A. The Server Workload Time Series Prediction Model

According to the locality principle [6], programs trend to run the same code within a certain amount of time to access the same data, so some of the same files in a storage system is likely to be requested repeatedly. In other words, clients accessing a particular data server still have a high probability to accessing the same data server at the next time. So, in this paper, we model the load predict problem as a univariate time series predict problem.

In this paper, the workload is defined as the size of requested data for a certain data server during a period. Therefore, given a time series of the request data size, it is denoted as $\mathbf{x} = (x_1 ... x_i ... x_w)$. Tphe goal is to learn the function $\mathbf{y} = f(\mathbf{x})$ where $\mathbf{y} = (x_w ... x_{w+j} ... x_{w+T})$. And x_i is the time series point at time i, w is the history horizon which means how many historical data point used to predict the future data and T is the forecast horizon which means how many future data point we want to predict. In this paper we only focus on single step prediction so the forecast horizon T is set to 1.

B. The Prediction based on LSTM Network

The workload prediction problem can be regarded as a univariable single step time series prediction problem. To capture the inner relationship between the history horizon and future value, the long short time memory neural network seen as a many-to-one unit with the merits of suitable for increasing complex workloads patterns is a ideal model which can be used. Therefore, we use the LSTM network as a deep learning model example to show our method which will be used to estimate the function $\mathbf{y} = f(\mathbf{x})$ where x is the sequence of historical time data ,i.e. $\mathbf{x} = (x_{t-T} ... x_t)$ and each $x_t \in$ is the data at time t, and that means T is set to 1.the output $\hat{y}$ of the whole model is excepted to be the data at the next time step x_{t+1}, thats to say we use the LSTM model as a many-to-one network. And in order to get a 1-dim vector at the last time step we add a fully connected layer in the end of the LSTM network. As is shown in Fig. 2 the basic structure of a LSTM unit is composed of a memory cell $c_t \in \mathbb{R}^H$, and three basic gate: Input Gate $i_t \in \mathbb{R}^H$, Forget Gate $f_t \in \mathbb{R}^H$, Output Gate $o_t \in \mathbb{R}^H$, where H is the hidden neural size of the LSTM unit, t means the time step.

The formulas for updating the state of each gate and cell in a LSTM unit using the input of $\mathrm{x}_t, h_{t-1}, c_{t-1}$ are defined as follows:

$$\begin{aligned}
z_t &= \tanh(W^z x_t + U^z h_{t-1}) \\
i_t &= \sigma(W^i x_t + U^i h_{t-1}) \\
f_t &= \sigma(W^f x_t + U^f h_{t-1}) \\
o_t &= \sigma(W^o x_t + U^o h_{t-1}) \\
c_t &= i_t \odot z_t + f_t \odot c_{t-1} \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned} \tag{1}$$

Where $x_t \in \mathbb{R}^T$ is the input vector at time t, $h_{t-1} \in \mathbb{R}^H$ is the hidden state vector at time $t-1$, $c_{t-1} \in \mathbb{R}^H$ is the cell state at time $t-1$. W^z, W^i, W^f, W^o are the weight matrixes of dimension (H, T) and are for the newly input at time t,U^z, U^i, U^f, U^o are the weight matrixes for the previous hidden state vector h_{t-1}.

C. Training procedure

We use minibatch stochastic gradient descent(SGD) together with the Adam optimizer [7] to train the model. The size of the minibatch is 256. The learning rate is set to 0.001 which is recommended in this paper. The parameters are learned by standard back propagation with mean squared error as the objective function:

$$\begin{aligned}
j_i &= mse(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2 \\
J(\theta) &= \frac{1}{N} \sum_{i=1}^{N} j_i + \lambda \sum_{\mathrm{k}} \theta_k^2
\end{aligned} \tag{2}$$

where θ is the parameters for training, N is the batch size, the second term in the last equation represent the $L2$ regularization which will help to avoid overfitting.

D. A hierarchical parameter searching method

It is well known that the selection of hyper-parameters is critical for neural networks. In our model there are five hyper-parameters, i.e., the history horizon or so called the history window w,the number of hidden units H, the number of layers L,the dropout probability d, and the $l2$ regularization multiplier λ. These 5 parameters have great influence on our model. In order to get a good combination of them, we use a hierarchical parameter searching method.

The first phrase is the random search for coarse searching and the second is the grid search for fine searching. At the beginning of model training procedure, we have little information about how to set the hyper-parameters for a good performance, so the common way to tune the hyper-parameter is to set a range according to our experience, and then adjust them according to the experiment results. So in the first phrase, we will just use randomly selected subset of the parameters in order to quickly reduce the parameters range. Further more we use the Tree-structured Parzen Estimators (TPE) [8] for random search. Each iteration TPE collects new observation and at the end of the iteration, the algorithm decides which set of parameters it should try next. When we get a small range for every hyper-parameter the grid search is used to found the final best combination of them. In order to get a good result

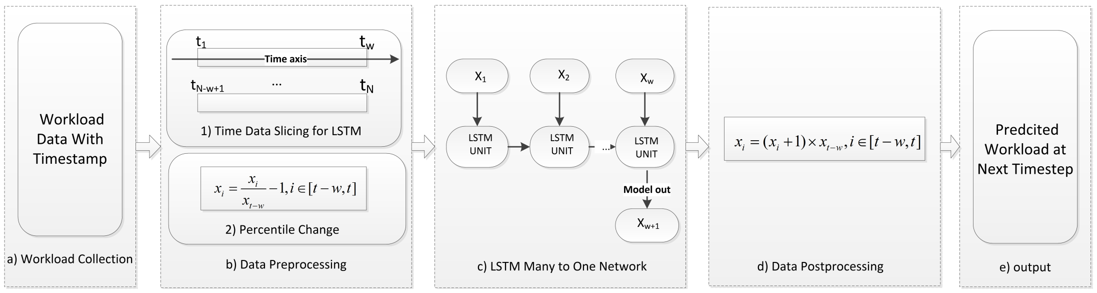

Workload Data With Timestamp
a) Workload Collection
t1
tw
Time axis
tN-w+1
...
tN
1) Time Data Slicing for LSTM
$x_i = \frac{x_i}{x_{t-w}} - 1, i \in [t-w,t]$
2) Percentile Change
b) Data Preprocessing
X1
X2
Xw
LSTM UNIT
LSTM UNIT
LSTM UNIT
Model out
Xw+1
c) LSTM Many to One Network
$x_i = (x_i + 1) \times x_{t-w}, i \in [t-w,t]$
d) Data Postprocessing
Predcited Workload at Next Timestep
e) output

Fig. 1. Our model framework.

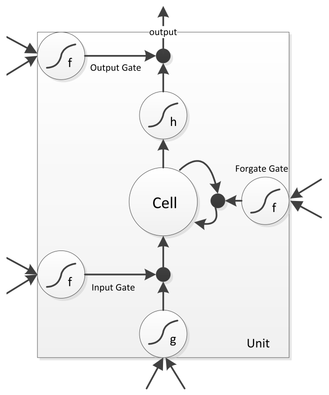


Fig. 2. LSTM unit structure.

we can set the search step to be a very small number since the range is limited in the coarse searching. And this is what we called fine searching.

III. Experiments

A. Dataset and Setup

To test our proposed model above, we use a dataset called WebSearch[1] which records a Search Engines I/O. We use the size filed which describes the number of bytes transferred for this record as the target series. Note that the size field is both for read and write operation request, and is traced by the timestamp, so we add up the number of bytes per 0.2 second to generate a univariable timeseries of request. In our experiment we use the last 3765 data point as our final target series, in order to exclude the unstable factor caused by the trace environment. So finally, we use the first 2967 data points as the training set, the following 371 data points as the validation set and the last 418 data point as the test set.

B. Data preprocessing

As is described above, the network is to learn the function $x_{t+1} = f(x_{t-w}...x_t)$, so we firstly cut the time series by the fixed window size $T+1$, and use:

$$\{(x_{t-w}...x_t), x_{t+1} | t \in [w+1, N-1]\}$$

TABLE I

HYPER-PARAMETER RANGE IN COARSE SEARCHING

Hyper-parameter name	values
history window	$x \in [4, 64]$
number of hidden units	$x \in [32, 256]$
number of layers	$x \in \{1, 2, 3\}$
dropout probe	$x \in [0, 0.5]$
l2 regularization multiplier	$x \in [0, 0.1]$

as our input and out pairs. In order to scales the values to be much smaller to makes it easier for the network to learn and also bounds the magnitude of the data. we then use a window based data normalization:

$$x_i = \frac{x_i}{x_{t-w}} - 1, i \in [t-w, t] \tag{3}$$

which is similarly used in stock price analysis [9], in the end of the model we will recover it to calculate the real error.

C. Evaluation Metrics

In order to measure the effectiveness of our method in time series prediction, three different evaluation metrics are considered. The root mean squared error (RMSE), mean absolute error (MAE), and mean absolute percentage error (MAPE). Specifically, as the output of the model is denoted as $\hat{y}$ and the ground truth is denoted as y, the RMSE, MAE, MAPE are defined as:

$$\begin{aligned} RMSE &= \sqrt{\frac{1}{N}(\sum_i (y_i - \hat{y}_i)^2)} \\ MAE &= \frac{1}{N}(\sum_i |y_i - \hat{y}_i|) \\ MAPE &= \frac{1}{N}(\sum_i \frac{|y_i - \hat{y}_i|}{y_i}) \end{aligned} \tag{4}$$

D. Parameter searching procedure

In the coarse searching phrase, we set the 5 parameters in the range shown in Table I, and use TPE algorithm implemented by the hyperopt[2] framework to select the better combination of them then train every trail for 100 epoch.

We achiever better result on the validation set when we try the trails with 1 hidden layer compared to other choices in the

[1]http://skuld.cs.umass.edu/traces/storage/

[2]http://hyperopt.github.io/

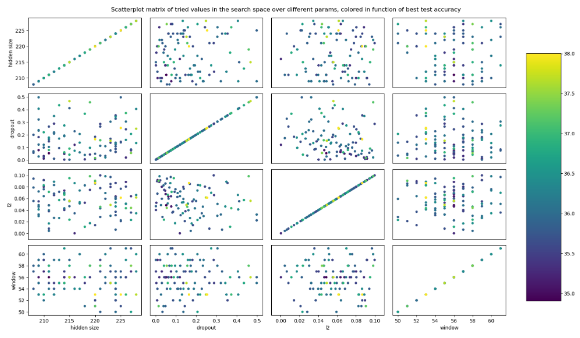


Fig. 3. Interaction between hyper-parameters.

TABLE II
HYPER-PARAMETER RANGE IN FINE SEARCHING

Hyper-parameter name	values
history window	$x \in [50, 60]$
number of hidden units	$x \in [208, 228]$
number of layers	$x = 1$
dropout probe	$x = 0.03$
l2 regularization multiplier	$x = 0.05$

first 200 trails, so we just fix the number of layers to be 1, and go on random searching in order to reduce the time cost. The searching result are shown in Fig. 3 the dot with dark color means a small RMSE.

Finally, we narrow the parameters range down to the following range shown in Table 3, and the fine searching phrase start. We use the grid search algorithm with the step set to be 1 then train every trail for 500 epoch with early stopping strategy.

It can be seen from the interaction diagram of parameters that along the axis of the window size there is a big gap between 56 and the other values, while the RMSE value along the axis of hidden size does not show any significant difference which means the history window is more important than the hidden size. When the window is set to an appropriate value, the hidden size changes in a certain range and has little impact on the final effect. In general, when the window is around 56 and the hidden size is around 214, we get the best performance in the validation set.

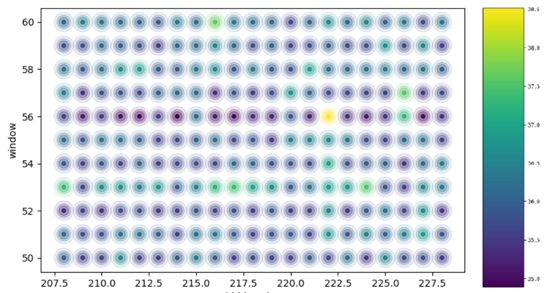


Fig. 4. Interaction between window size and hidden size.

TABLE III
MODEL PERFORMANCE IN THREE METRICS

Model	RMSE	MAE	MAPE
Arima	36.0574(3.8%)	29.0381	0.4151
SVR	35.9441(3.7%)	28.78	0.44
Simple-RNN	35.8686(3.5%)	29.15	0.46
LSTM(Our Method)	34.6535	28.13	0.41

[a] % means deteriorative percentage compared with our model.

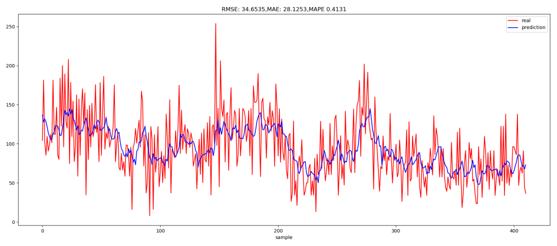


Fig. 5. Prediction result in test set.

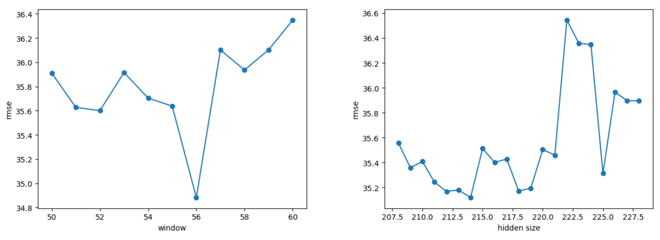
E. Results: Time Series Prediction

We choose three different time series prediction models mentioned in the Related work section as baseline. For the ARIMA model, we firstly analyze the stationarity using Dickey-Fuller test . Then we plot the auto-correlation and partial auto-correlation to determine the approximate range of parameter p, i, q, then the Akaike information criterion(AIC) is used to find the proper values of the 3 parameters. The program is implemented with python package statsmodels .For the SVR model, we use the RBF as the kernel method which is implemented with python package scikit-learn. For the Simple-RNN model we replace our LSTM cell with RNN cell in the trained model mention above which is implemented in python using Keras. The results are shown in Table III.

The result shows that LSTM model beat all the baselines with at least 3.5% improvement in RMSE, meanwhile it also achieves the best performance in MAE and MAPE. The prediction result in the test set is shown in Fig. 5. Our result in RMSE is competitive when comparing with other researchers' results in some data sets e.g. [10]. Considering that our time series data is the size of a server's requested data within 0.2s, which is high-frequency with complex correlation data, we think it's impressive to achieve this improvement.

F. Results: Parameter Sensitivity Analysis

We study the sensitivity of the two important hyper-parameters in LSTM i.e., history window and hidden size shown in Fig. 6. By fixing the hidden size to be 214,we plot the relationship between history window and RMSE. It can be found that when the window is too large or too small RMSE will deteriorate, and it can achieve better results when the window is around 5. By fixing the window to be 56 we plot the relationship between hidden size and RMSE. It can be seen as that with the increase of the hidden size, the RMSE has a tendency to increase. And RMSE achieve better result when



(a) window size (b) hidden size

Fig. 6. Parameter sensitivity analysis.

the hidden size is between 212 and 215. Therefore, the idea of simply increasing the size of the network to improve model performance is not reasonable.

IV. Related Work

As is explained before, for the time locality view, the workload prediction problem can be regarded as a time series prediction problem. The previous classic methods for workload time series prediction can be classified into the following three categories.

A. The statistical method

In this category, the autoregressive moving average (ARIMA) model [11], originated from the autoregressive model (AR) [12] and the moving average model (MA) [13] is the most famous one. This model can capture the linear relationship in the stationary sequence. We have developed an online variant to predict I/O workload in our previous work [14]. This work is the new try using a deep learning model.

B. The machine learning method

With the development of machine learning, time series prediction has been formulated as a regression problem, usually can be solved by support vector regression (SVR) [15]. SVR can capture the relationship between input and output by mapping input space to feature space through nonlinear transformation.

C. The deep learning method

Long short-term memory(LSTM) network [16], which is a typical sequence model in deep learning has the ability to capture the complex relationship between history data and future ones. LSTM has overcome the problem of vanishing gradients and thus can capturing long-term dependencies. To the best of our knowledge, how to perform a deep learning based storage system workload time series prediction has not been studied yet. In this paper, we incorporate LSTM in our prediction process.

V. Conclusion

In this paper, we proposed a practical deep learning based approach to predict the workload in storage system. The workload time series prediction method includes workloads collection, data preprocessing, time series prediction based on a deep learning network and data postprocessing. Further more, in order to train the network more efficiently, we introduced a hierarchical parameter searching method which consists of the coarse searching phrase and the fine searching phrase. Finally, we compare this model to 3 classic time series prediction algorithm and the extensive experimental results show that our framework can obtain performance improvement compared with three classic time series prediction algorithms in RMSE and the best performance in MAE and MAPE.

Acknowledgment

This work is supported by the National Key R&D Program of China under Grant NO.2017YFB0202004, the fund of the State Key Laboratory of Software Development Environment under Grant No.SKLSDE-2017ZX-10, the National Natural Science Foundation of China under Grant No.61370059.

References

[1] J. S. Firoz, M. Zalewski, A. Lumsdaine, and M. Barnas, "Runtime scheduling policies for distributed graph algorithms," in *IEEE International Parallel and Distributed Processing Symposium*, 2018, pp. 640–649.

[2] Z. Huang, J. Peng, H. Lian, J. Guo, and W. Qiu, "Deep recurrent model for server load and performance prediction in data center," *Complexity*, vol. 2017, no. 99, pp. 1–10, 2017.

[3] R. Mckenna, S. Herbein, A. Moody, T. Gamblin, and M. Taufer, "Machine learning predictions of runtime and io traffic on high-end clusters," in *IEEE International Conference on CLUSTER Computing*, 2016, pp. 255–258.

[4] R. Cao, Z. Yu, T. Marbach, J. Li, G. Wang, and X. Liu, "Load prediction for data centers based on database service," in *IEEE Computer Software and Applications Conference*, 2018, pp. 728–737.

[5] Y. Yu, V. Jindal, I. Yen, and F. Bastani, "Integrating clustering and learning for improved workload prediction in the cloud," in *IEEE International Conference on Cloud Computing*, 2017, pp. 876–879.

[6] L. Ping, "Analysis and development of the locality principle," *Advances in Intelligent & Soft Computing*, vol. 133, no. 7, pp. 211–214, 2012.

[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *Computer Science*, 2014.

[8] J. Bergstra and Y. Bengio, "Algorithms for hyper-parameter optimization," in *International Conference on Neural Information Processing Systems*, 2011, pp. 2546–2554.

[9] H. Jia, "Investigation into the effectiveness of long short term memory networks for stock price prediction," 2016.

[10] X. Zhang, F. Shen, J. Zhao, and G. H. Yang, "Time series forecasting using gru neural network with multi-lag after decomposition," in *International Conference on Neural Information Processing*, 2017, pp. 523–532.

[11] G. E. P. Box and G. M. Jenkins, "Time series analysis, forecasting and control, holden-day," *Journal of the Royal Statistical Society*, vol. 134, no. 3, 1976.

[12] G. Walker, "On periodicity in series of related terms," *Proceedings of the Royal Society of London*, vol. 131, no. 818, pp. 518–532, 1931.

[13] E. Slutzky, "The summation of random causes as the source of cyclic processes," *Econometrica*, vol. 5, no. 2, pp. 105–146, 1937.

[14] B. Dong, X. Li, Q. Wu, L. Xiao, and R. Li, "A dynamic and adaptive load balancing strategy for parallel file system with large-scale i/o servers," *Journal of Parallel & Distributed Computing*, vol. 72, no. 10, pp. 1254–1268, 2012.

[15] H. Drucker, C. J. C. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, "Support vector regression machines." *Advances in Neural Information Processing Systems*, vol. 28, no. 7, pp. 779–784, 1997.

[16] M. Sundermeyer and H. Ney, *From feedforward to recurrent LSTM neural networks for language modeling*. IEEE Press, 2015.