Cluster Computing (2023) 26:25–35
https://doi.org/10.1007/s10586-020-03214-y

Workload time series prediction in storage systems: a deep learning based approach

Li Ruan[1,2] · Yu Bai[1,2] · Shaoning Li[1,2] · Shuibing He[3] · Limin Xiao[1,2]

Received: 19 March 2019 / Revised: 13 November 2020 / Accepted: 23 November 2020 / Published online: 13 January 2021

Abstract
Storage workload prediction is a critical 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 method called CrystalLP based on deep learning. CrystalLP includes workload collecting, 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). Furthermore, to improve the efficiency of LSTM, we study the sensitivity of the hyperparameters in LSTM. Extensive experimental results show that CrystalLP can obtain performance improvement compared with three classic time series prediction algorithms.

Keywords Workload prediction · LSTM · Time series

1 Introduction

In the big data era, storage system performance becomes a critical bottleneck of many data-intensive applications. [27] reported that in the data-centric model, meeting the data characteristics of different workloads is a necessity of storage system (e.g. PFS) shared by all of the compute resources at the same time. Different workload patterns can significantly impact application runtime of simulations [31], waste of energy due to inefficient resource utilization [3] or the responsiveness of interactive analysis workloads [22], which result in resource shortages and application issues such as delays or over-provisioning. With the diversity of such applications increasing, simply scaling up a single server or scaling out the cluster can not completely solve the storage bottleneck issue. Accurate workload prediction provides a critical approach to address this issue because it helps to achieve fine-grained runtime-sensitive load balancing and job scheduling [10]. For example, inevitable hot data phenomenon will usually cause performance hungry for some servers while the others would waste their hardware. 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.

Previous researches on server workload prediction can mainly be classified into two categories [15]. The first category focuses on constructing the relationship between the time and the workload by using regression models [28]. The second one pays more attention to analyze the characteristics of current workload with neural network for future workload prediction [32]. Decision trees and clustering [1] are commonly used prediction methods. However, due to the increasing complexity of input workloads, existing methods are hard to extract the underlying relationship of the workload.

Due to the merits of modeling nonlinear relationship between input and output, deep learning method gains

✉ Li Ruan
ruanli@buaa.edu.cn

Yu Bai
19373215@buaa.edu.cn

Shuibing He
heshuibing@zju.edu.cn

1 State Key Laboratory of Software Development Environment, Beihang University, Beijing 100191, China

2 School of Computer Science and Engineering, Beihang University, Beijing 100191, China

3 College of Computer Science and Technology, Zhejiang University, Zhejiang 310027, China

increasing attention for real world problem modeling with the success in many fields, such as pattern recognition, natural language processing, etc. When dealing with similar time series prediction problems, deep learning methods perform better than traditional regression models. However, to the best of our knowledge, how to perform a deep learning based storage workload prediction has not been studied yet. So in this paper, we propose a storage workload prediction method called CrystalLP based on LSTM neural networks. And results show that CrystalLP can obtain performance improvement compared with three classic time series prediction algorithms.

Comparing with previous work, our contributions are as following:

- We propose a practical storage workload prediction method called CrystalLP which includes workload collecting, data preprocessing, time series prediction based on a long short-term memory network(LSTM), and data postprocessing phase. To the best of our knowledge, we are the first to introduce the deep learning method into practical storage workload analysis.
- We study the sensitivity of the hyperparameters in LSTM and conduct extensive experiments to show the superiority of CrystalLP to three classic time series prediction algorithms.

The rest of this paper is organized as follows. Section 2 introduces the detailed framework and components of our workload time series prediction model called CrystalLP. Section 3 evaluates the proposed method via stimulate experiment with real-world data source. In Sect. 4, we analyse the current related work on workload prediction method. Finally we conclude this paper in Sect. 5.

2 The workload time series prediction framework based on deep learning

Our practical storage system workload time series prediction method (Fig. 1) called CrystalLP includes workloads collection, data preprocessing, time series prediction based on long short-term memory network(LSTM) and data postprocessing.

2.1 The storage workload time series model

How to establish the server workload time series model from the practical traces is the first critical problem that should to be solved. According to the time locality principle [25], within a period of time, as programs tend to run the same code segment to access the same data, thus the same files in a storage system are likely to be requested repeatedly. In other words, clients accessing a particular data server still have a high probability of accessing the same data server next time. We define this phenomenon as the workload exhibits certain underline patterns which will be helpful for precise and adaptive scheduling and load balancing. Therefore, in this paper, we model the workload prediction problem as a univariate time series predict problem.

The storage workload is defined as the size (e.g., bytes) of requested data for a data server during a fixed period. Therefore, given a time series of the request data size, it is denoted as $\mathbf{x} = (x_1, \ldots, x_i, \ldots, x_w)$. The goal is to learn function $\mathbf{y} = f(\mathbf{x})$ where $\mathbf{y} = (x_w, \ldots, x_{w+j}, \ldots, x_{w+T})$. And x_i is the time series point at time i, w is the history horizon which means how many historical data points used to predict the future data and T is the forecast horizon which means how many future data points we want to predict. In this paper, for the convenience of clearly introducing our method, we use single step prediction as an example. Therefore, the forecast horizon T is set to 1. But surely, the prediction horizon T can be set to the suitable p according to the user's practical prediction requirements.

2.2 Autocorrelationn analysis of workload time series

To demonstrate the local principle and explain the reason for modeling the workload prediction problem in our scenario as a sequential problem, we analyze the data from a popular search engine's file system workload called WebSearch. Details of the workload will be introduced in Sect. 3.1. We mainly use autocorrelation as a mathematical tool to analyze the features of the storage time series. The autocorrelation (Eq. 1) which describes the correlation of a signal with a delayed copy of itself is a good tool to demonstrate the idea proposed above.

$$\rho(t, s) = \frac{E(X_t - u_t)(X_s - u_s)}{\sqrt{DX_t \bullet DX_s}} \tag{1}$$

For instance, Fig 2 shows three different autocorrelations of time series. In Fig 2a values are closed to zero except the value at time lag 0, that means there is neither correlation between adjacent time nor bigger lags. The signal is closer to white noise, which is hard to predict. In Fig 2b values are closed to zero except the value at time lag 0 and 1, which means there is a strong correlation between adjacent times. And in Fig 2c, the values are bigger in general, which means there is a correlation between adjacent time and bigger time lags. For the correlation displayed in Fig 2a, c, we can use some functions to capture the underlying relationship between past data and future ones. So that's our intuition to predict the workload using

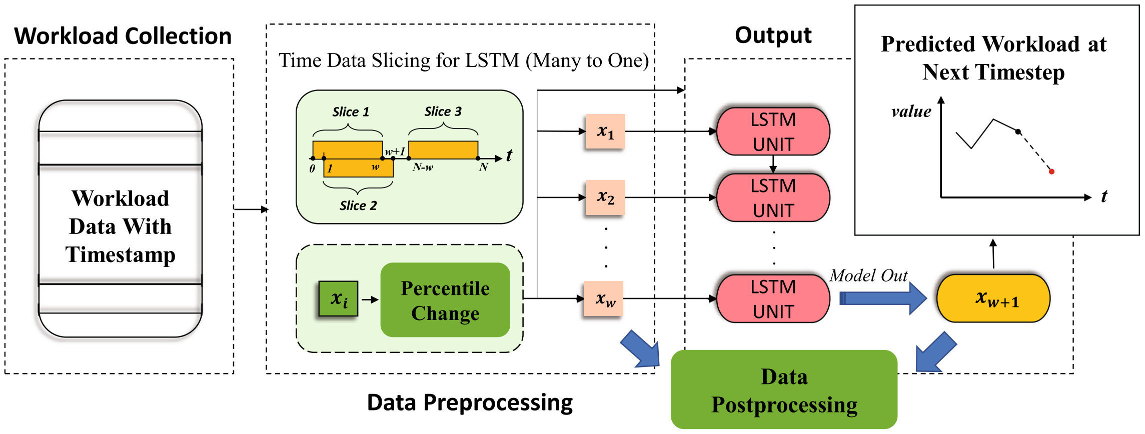
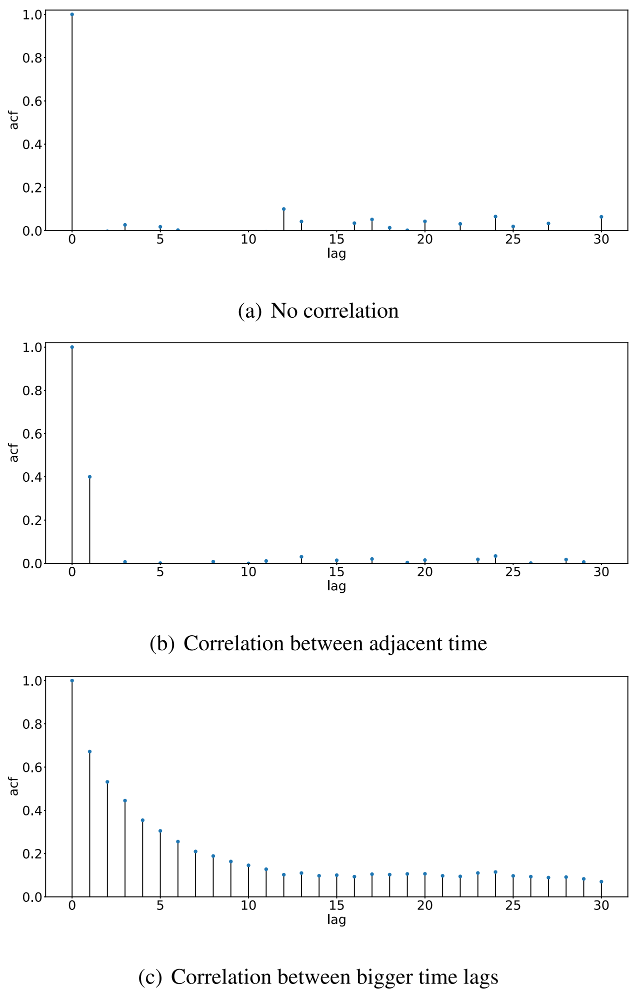


Fig. 1 CrystalLP: the workload time series prediction framework based on deep learning



(a) No correlation

(b) Correlation between adjacent time

(c) Correlation between bigger time lags

Fig. 2 Correlation analysis

sequential model for data servers in large scale storage system.

2.3 The workload prediction model based on deep learning network

After we have established an univariable single step time series prediction model for the workload prediction problem in Sect. 2.1, how to establish a prediction model using a proper deep learning model suitable for the server's workload feature is a critical step. As is analyzed above, the workloads tend to exhibit complex patterns. Because the long short time memory neural network as a many-to-one unit has the merits of suitable for complex workloads patterns, it is suitable for capturing the inner relationship between the history horizon and future value. Therefore, we choose the LSTM network as an example to estimate the load function $\mathbf{y} = f(\mathbf{x})$. The x is the sequence of historical time workload data,i.e. $\mathbf{x} = (x_{t-T}, \ldots, x_t)$ and each x_t is the workload size at time t where T is set to 1. The output $\hat{y}$ of the whole model represents the data at the next time step x_{t+1}, that's to say we use the LSTM model as a many-to-one network. Moreover, in order to get a 1-dim vector at the last time step, we also add a fully connected layer at the end of the LSTM network. As is shown in Fig. 3, the basic

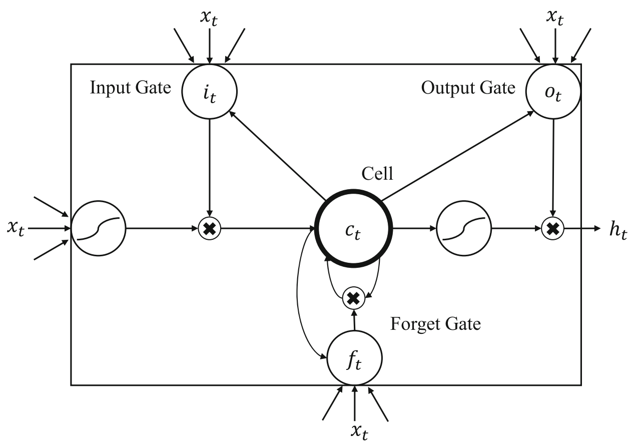


Fig. 3 The structure of LSTM unit

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 and t means the time step.

The state of each gate and cell in a LSTM unit with the input of x_t, h_{t-1}, c_{t-1} is updated according to the Eq. 2.

$$\begin{cases} z_t &= \tanh(W^z x_t + U^z h_{t-1}) \\ i_t &= \sigma(W^i x_t + U^i h_{t-1}) \\ f_t &= \sigma(W^f x_t + U^f h_{t-1}) \\ o_t &= \sigma(W^o x_t + U^o h_{t-1}) \\ c_t &= i_t \odot z_t + f_t \odot c_{t-1} \\ h_t &= o_t \odot \tanh(c_t) \end{cases} \tag{2}$$

The $x_t \in \mathbb{R}^T$ 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}. The selected parameters for each layer of the LSTM model can be found in Table 1:

where ω is denoted as the history horizon, H and L respectively represent the number of hidden units and layers, d is the the dropout probability while training and λ is the L_2 regularization multiplier to avoid overfitting.

2.4 Workload collection

During the workload collection stage, the workload time series data which consists of workloads at different time steps are collected. As our model is mainly for single step workload prediction, it means that for every prediction process the input data is the data within the history horizon, and the output of the model is the predicted workload at next time step.

As for the continuous forecasting process, we use a rolling procedure shown in Algorithm 1 to handle it:

```
Algorithm 1: Continuous CrystalLP
  Input: storage workload time series $\mathbf{x} = (x_1, ...x_i, ...x_w)$
  Output: the predicted workload $\mathbf{y} = (x'_{w+1}, ...x'_{w+j}, ...x'_{w+T})$
  for $i \leftarrow 1; i <= T; i \leftarrow i+1$ do
      $\mathbf{x} \leftarrow$ preprocess input data $\mathbf{x}$;
      $x'_{w+i} \leftarrow$ workload prediction using preprocessed data $\mathbf{x}$;
      $x'_{w+i} \leftarrow$ postprocess the predicted data $x'_{w+i}$;
      $x_{w+i} \leftarrow$ collect the real workload at the predicted time
        step;
      $\mathbf{x} \leftarrow (x_{i+1}, ...x_i, ...x_{w+i})$
  end
  $\mathbf{y} \leftarrow (x'_{w+1}, ...x'_{w+j}, ...x'_{w+T})$
  return $\mathbf{y}$
```

where storage workload time series **x** is donated as the input of Alg.1 and the predicted workload **y** is the wanted output. The T loop represents the adopted continuous rolling prediction model which consists of 3 steps: First, the input data are preprocessed and transferred to prediction module. Then the results of workload prediction are obtained and sent to postprocessing module. Finally, the real load value corresponding to this prediction is collected, the historical observation window is moved forward, the load corresponding to the front edge of the window is added to the observation window and the original data of the back edge is removed to start the next prediction. As the whole loop comes to an end, which means all time steps of storage workload time series **x** are processed, the wanted predicted workload **y** is eventually obtained. This algorithm is also used in our experiment for the test set.

2.5 Preprocessing and postprocessing of workloads

The first challenge we meet during the workload time series prediction is how to preprocess the workload data to construct a suitable data structure for the time series prediction.

During the data preprocessing stage, the workload data will be sliced by a history time window of length W. With the time window sliding along the time axis, the workload time series data structure suitable for LSTM model is established.

As in the practical storage system, the workload trace is usually collected at variable time period, therefore, certain normalization operations to the original trace to make it well prepared for further feature analysis and prediction needs to be performed at the prepocessing stage. We thus design and perform several normalization operations to the workload traces, such as normalizing the workload traces into the trace with same time periods, observing and analyzing the stability of the trace to better find a sub-trace for further feature mining and parameter tuning, etc. In our empirical study, we designed a preprocessing module which slices the sequential data with fixed sliding window size, and then performs scaling transformation to construct suitable data as an input. The function to train the model based on the input to output is designed as Eq. 3.

$$x_{t+1} = f(x_{t-w} ... x_t) \tag{3}$$

As is shown in Eq. 4, a time window with the fixed length $T+1$ was used to slice the time series data to construct the input and output data needed for model training.

$$\{ < (x_{t-w} ... x_t), x_{t+1} > \mid t \in [w+1, N-1] \} \tag{4}$$

However, I/O workload has such a strong volatility,i.e., within the time window there exists big diversity between different input workload value which leads to model

Table 1 Parameters for each layer

Parameter	Explanation
ω	The history horizon
H	The number of hidden units
L	The number of layers
d	The dropout probability
λ	The L_2 regularization multiplier

convergence difficulties. In order to eliminate the gaps of workload data value in different time window and make it easier for the model to learn, we employ a normalization method based on window, as shown in Eq. 5, which makes all data in the same window take the first data as a reference and transform the forecast amplitude into the forecast growth rate, thus reducing the fluctuation of the training data.

$$x_i = \frac{x_i}{x_{t-w}} - 1, i \in [t-w, t] \tag{5}$$

The postprocessing adopts the reverse shrinkage operation to recover the data, and the predicted value is finally obtained.

2.6 Model training

To achieve good model training results, recently it is proposed to use a combined model to train the deep learning model. In our model, the minibatch stochastic gradient descent(SGD) together with the Adam optimizer [17] is employed to train the storage workload model. The size of the minibatch is set to 64. The learning rate is set to 0.001 which is recommended according to [17]. The parameters are learned by standard back propagation with mean squared error as the objective function as is shown in Eq. 6.

$$\begin{aligned} j_i &= mse(y_i, \hat{y_i}) = (y_i - \hat{y_i})^2 \\ J(\theta) &= \frac{1}{N}\sum_{i=1}^{N} j_i + \lambda \sum_{\mathrm{k}} \theta_k^2 \end{aligned} \tag{6}$$

where θ is the parameter for training, N is the batch size, the second term in the last equation represent the $L2$ regularization which will help to avoid overfitting. And an early stopping on the validation set is used to make sure the model is trained for a proper epoch.

3 Experiments

3.1 Introduction of I/O data source

During our test, the first challenge is to choose a suitable storage I/O trace to validate our method. Although there are many publicly released workload traces like google trace, Alibaba trace, they are more on resource utility related workload traces which are not suitable for storage I/O performance analysis.

As most of the I/O traces are not publicly released, to test our proposed model, we use a data set called WebSearch[1] which records a Search Engine's I/O. The WebSearch archive is saved in a special format called SPC TRACE FILE FORMAT which is designed for the better analysis of trace[1].

According to the Umass trace repository document(2007)[1] each record in the trace file represents an I/O command and consists of four fields. The first field is called Application specific unit(ASU), ASU is a positive integer that represents an application-specific unit. The second field is called Logical block address(LBA), The LBA field is a positive integer that describes the ASU block offset for the data transfer for this record. The third field is the size field, which is a positive integer that describes the number of bytes transferred for this record. The fourth field is called Opcode which records the operation corresponding read or write command. The fifth field is the timestamp, which is a positive number that represents the offset of this I/O operation from the start of the trace. All of the fields' types are described in Table 2.

3.2 Data preprocessing and model training

We use the *size* field which describes the number of bytes transferred for a record as the target series. Note that the *size* field is both for read and write operation request, and is traced by the *timestamp*.

Firstly, because the workload trace is not naturally organized as a storage I/O time series, we choose the archives from the trace data to establish a time series. In our experiment, we use the first archived file in the trace as an example and organize the traces by field of *timestamp*. To normalize the time traces, we sum up the requested data's size for $ASU0$ per 1.0 second to generate a univariate time series of requests.

After the workload traces have been formatted well as a storage workload time series, the distribution of fields (e.g. *size* and *timestamp*) are analyzed. The analysis results of the original $ASU0$'s data are shown in Fig. 4. Figure 4a shows there are four kinds of request size in the original data, and the largest one is 8192 byte. The smallest one is 24576 byte. The medium ones are 32768 bytes and 16384 bytes.

During our analysis, the arrival time is critical info too. Thus, to get the requests' arrival time, we divide the entire data set into 1000 bins uniformly according to the requests'

[1] http://traces.cs.umass.edu/index.php/Storage/Storage.

Table 2 Data type description

Field	Type
ASU	Positive integer
LBA	Ppositive integer
Size	Real number
Opcode	0 or 1
TimeStamp	Real number
Options	None

time stamp, and calculate out the counts of request per bins. Fig. 4b shows the distribution of the *timestamp*. We can see that the most request counts per bins lies in the range of [320,400], and the variance of the data could be huge, which is hard for prediction. The above analysis verified the necessity of the preprocessing of the storage workload traces. Moreover, we can see our above analysis is the basis for the storage workload time series prediction.

Finally, the training set which consists of the first 2488 data points, the validation set which consists of the following 312 data points and test set which consists of the last 351 data points are established. The data set is shown in Fig 5. Its mean value is 1711868.02, the standard deviation is 415512.83, the max value is 2932736.00, the min value is 262144.00.

As is described above, the network is to learn the function $x_{t+1} = f(x_{t-w} \dots x_t)$. Therefore, we first cut the time series by the fixed window size $T + 1$, and use Eq. 4 as our input and out pairs. In order to scale the values to be much smaller to make it easier for the network to learn and also bound the magnitude of the data, we then use a window based data normalization method in Eq. 5.

3.3 Metrics

To validate the effectiveness of our method, three different evaluation metrics in Eq. 7 are considered, i.e. the root mean squared error (RMSE), mean absolute error (MAE) and mean absolute percentage error (MAPE), where the output of the model is denoted as $\hat{y}$ and the ground truth is denoted as y.

$$
\begin{aligned}
RMSE &= \sqrt{\frac{1}{N}\left(\sum_i (y_i - \hat{y_i})^2\right)} \\
MAE &= \frac{1}{N}\left(\sum_i |y_i - \hat{y_i}|\right) \\
MAPE &= \frac{1}{N}\left(\sum_i \frac{|y_i - \hat{y_i}|}{y_i}\right)
\end{aligned}
\tag{7}
$$

3.4 Results: time series prediction

To validate our method, three classic time series prediction models (Sect. 4) are implemented by us for the comparison baseline.

For the ARIMA based workload prediction model, we implement it with python package statsmodels of version 0.8.0. We first analyze the workload stationary using Augmented Dickey-Fuller test. Then we plot the auto-correlation and partial auto-correlation of the workloads to determine the approximate range of parameter p, i, q. Next, the *Akaike information criterion*(*AIC*) is used to find the proper values of the three parameters. When we get the proper values of p, i, q according to the training set, we fit an ARIMA model then predict a future value in the test set, then the ground truth of the predicted data is collected into the known data set to expand the training data, then a new ARIMA model is fitted according to the expanded training data. The loop will stop until there is no data to be predicted in the test set. So we get the predicted value of the entire test set.

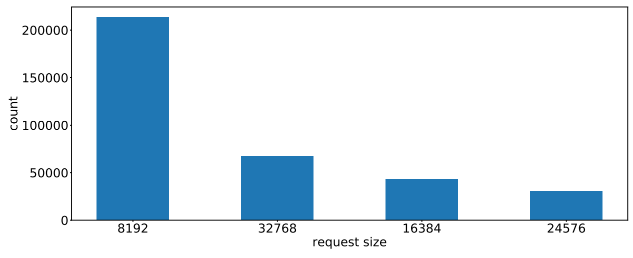
For the SVR model, we use the Radial Basis Function as the kernel method which is a mainstream kernel method and implemented it with python package scikit-learn of version 0.19.1. The same procedures to the workload prediction method discussed above are used except for the prediction model for fair comparison.

For our method (Section model 2), we implement it with python in Keras of version 2.2.2, and for the Simple-RNN model baseline we just replace our LTSM cell with RNN cell.

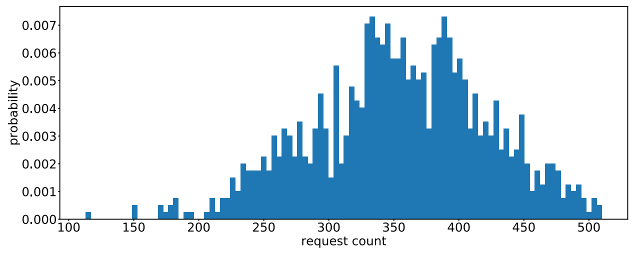
All the model are implemented using python 3.6.6. Finally, the experimental results among the three models are shown in Table 3.

The predicted workload value from CrystalLP can be used as an input during the adaptive I/O resources scheduling [22] [27].

We compare our CrystalLP's performance with three classic models, e.g. Arima, SVR, SimpleRNN, in three metrices, e.g. RMSE, MAE and MAPE. From the experimental results in Fig. 3, we can find that Arima achieves RMSE of 307629.58 which is of lower performance than CrystalLP by 0.89, MAE of 250214.06 which is of lower performance than CrystalLP by 0.28, MAPE of 0.1846 which is of lower performance than CrystalLP by 2.46. SVR achieves RMSE of 304278.19 which is of lower performance than CrystalLP by 0.20, MAE of 248345.99 which is of lower performance than CrystalLP by 1.03, MAPE of 0.1840 which is of lower performance than CrystalLP by 2.20. SimpleRNN achieves RMSE of 314267.07 which is of lower performance than CrystalLP by 2.98, MAE of 258297.24 which is of lower performance than CrystalLP by 2.86, MAPE of 0.1837 which is of lower performance than CrystalLP by 1.10. CrystalLP achieves

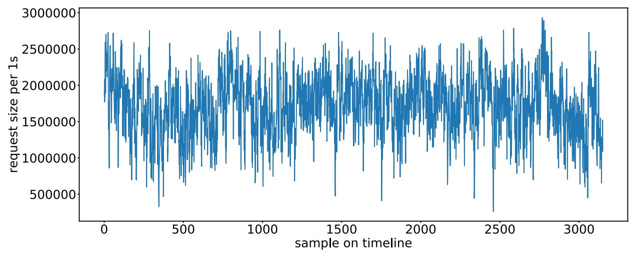


(a) Distribution of the *size* field

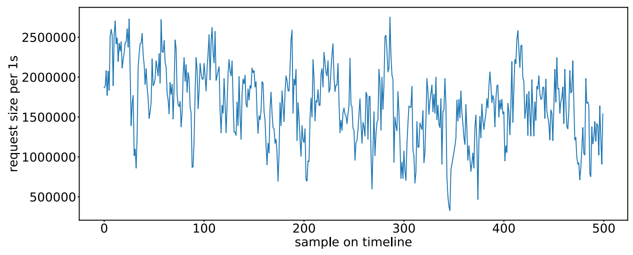


(b) Distribution of the *timestamp* field

Fig. 4 Original data



(a) The whole data set



(b) First 500 samples of the whole data set

Fig. 5 Dataset analysis

RMSE of 304894.7, MAE of 250915.78, MAPE of 0.1800. As CystalLP is the baseline, there is no value with () for it. To sum up, the prediction results (Fig. 3) show that CrystalLP is superior to all the baselines with at least 1.10% improvement in *MAPE*, and also better performance in *MAE* and *MAPE*.

As is shown in Fig. 6, we plot the predicted workload(request size per 1s) trends to compare the model prediction vivid effects of original real storage workload with ARIMA/SVR/Simple-RNN/LSTM(CrystalLP). the line which shows the predicted value, it can be seen that CrystalLP can well capture the trend of the real time data. Considering that our time series data is the size of a server's requested data within 0.2s which has high-frequency and complex correlation, we think it's impressive to achieve this improvement.

3.5 Results: parameter sensitivity analysis

We study the sensitivity of the important hyper-parameters in LSTM i.e., *hidden size*. We plot the relationship between hidden size and the metrics RMSE,MAE and MAPE, and the results are presented in Fig. 7.

From Fig. 7a, we can see that with the ascent of *hiddensize* from 210 to 290, RMSE increases from around 305000 to 307000. With the ascent of *hiddensize* from 210 to 290, MAE increases from around 250500 to 253500. With the ascent of *hiddensize* from 210 to 290, MAPE increases from around 0.181 to 0.184. To sum up, it can be seen that with the increase of the hidden size, the metrics have a tendency to decrease and increase. Moreover, RMSE achieves best performance when the hidden size is between 250 and 260 when we compare the three experiments in Fig. 7. Therefore, the parameter sensitivity analysis results prove that the idea of simply increasing the size of the network to improve model performance is not reasonable. The layered parameter combining coarse-grained search and fine-grained search just as what we have proposed in CrystalLP is critical to be suitable for the establishment of deep learning model for high-variance of storage workload during the storage workload analysis.

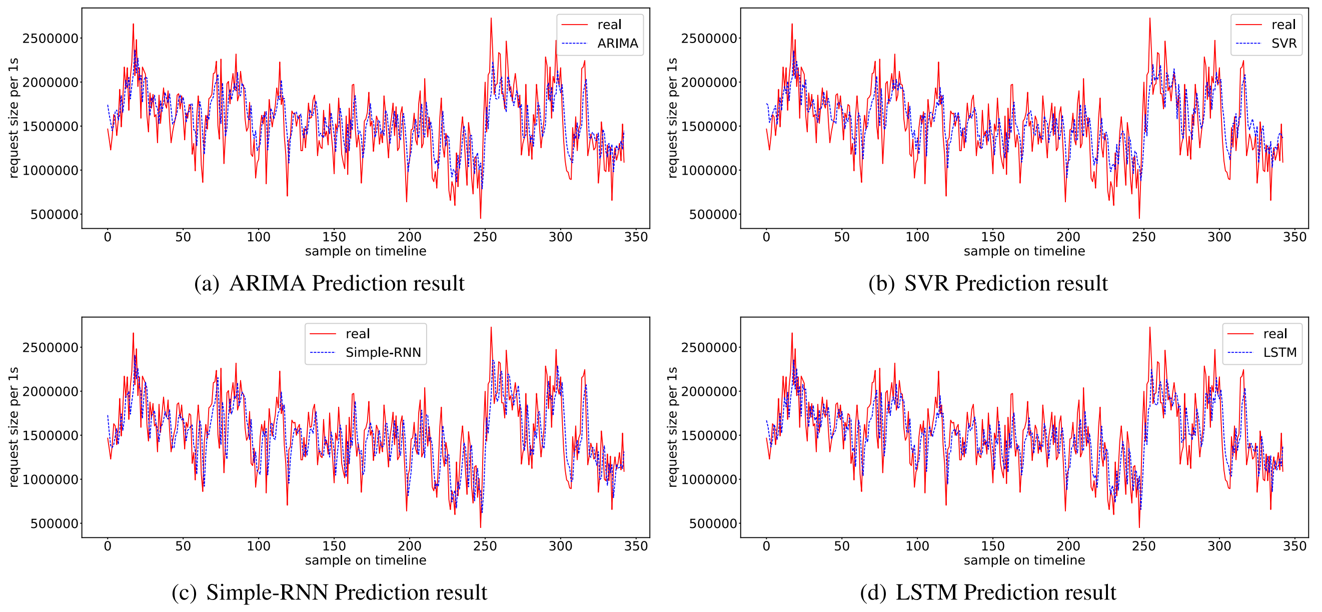
4 Related work

Many scholars have attempted to overcome the problem in workload prediction. In the past decade, there have been many studies using deep learning algorithms or statistics-

Table 3 Model performance in three metrics

Model	RMSE	MAE	MAPE
Arima	307629.58 (0.89)	250214.06 (– 0.28)	0.1846 (2.46)
SVR	304278.19 (– 0.20)	248345.99 (– 1.03)	0.1840 (2.20)
SimpleRNN	314267.07 (2.98)	258297.24 (2.86)	0.1837 (1.10)
CrystalLP	304894.71	250915.78	0.1800

[a] Data in () means deteriorative percentage compared with CrystalLP



(a) ARIMA Prediction result (b) SVR Prediction result (c) Simple-RNN Prediction result (d) LSTM Prediction result

Fig. 6 Prediction result in test set

based algorithms to predict workload. As is explained in the above section, the workload prediction problem can be regarded as a time series prediction problem using workload trace. In this section,we first review some classic methods for cloud time series analysis. Then, we present AI methods for the time series prediction.

4.1 Classic methods in time series analysis and prediction

Workload time series analysis is a critical branch of workload based platform performance analysis approaches, such as resource utility analysis [19, 23], task failure analysis [16, 29], etc. However, there are seldom methods focusing on deep learning based storage workload analysis. This paper proposed a deep learning based on storage analysis.

4.2 Classic methods in AI based workload analysis

The existing classic methods for workload time series prediction can be classified into the following three categories of statistical, machine learning based, and deep learning based methods. [2]

4.2.1 The statistical method

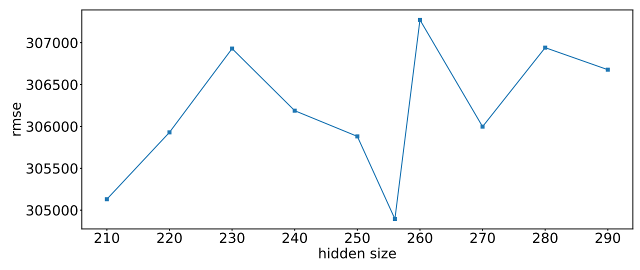
Among this category, the autoregressive moving average model (ARMA) [14] is the most famous one. This model can capture the linear relationship in the stationary sequence and establish the relationship between the historical data and the future data through the form of difference equation. And to model the non-stationary time series, the autoregressive integrated moving average (ARIMA) [5] model has been proposed as an extension of ARMA, which can tackle nonstationary time series forecasting by differencing techniques. We have developed an online variant of the ARIMA model to predict I/O workload in our previous work [8]. Di et al. [7] proposed a Bayesian model based on statistical features within the observation window to predict server's CPU usage in Google Cloud service cluster. And CrystalLP is our new try using a deep learning model.

4.2.2 The machine learning method

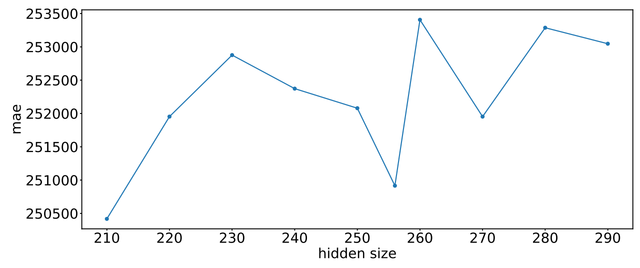
With the development of machine learning, time series prediction has been formulated as a regression problem and usually can be solved by support vector regression (SVR). SVR can capture the relationship between input and output by mapping input space to feature space through nonlinear transformation. Besides, with kernel methods (e.g. RBF), SVR can easily deal with the curse of dimension problem [4] and make better use of the high dimension feature space. Furthermore, Neelima et al. [21] proposed a novel adaptive dragonfly algorithm(ADA) to accomplish the NP-hard optimization problem in cloud load balancing.

4.2.3 The deep learning method

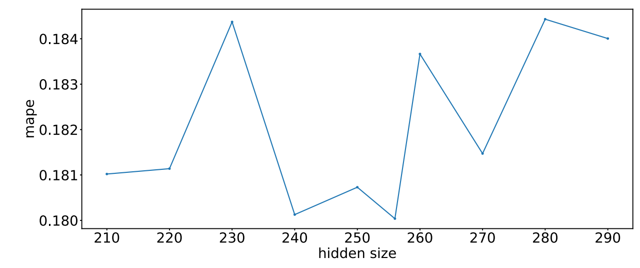
Long short-term memory(LSTM) network [26] has overcome the problem of vanishing gradients and thus can capturing long-term dependencies. Recently, LSTM has



(a) RMSE change along with hidden size



(b) MAE change along with hidden size



(c) MAPE change along with hidden size

Fig. 7 Parameter sensitivity analysis

been applied in predicting the CPU usage in cloud environment. In 2018, Zhang et al. [35] used time series clustering algorithm combined with the bidirectional long-term and short-term memory network to predict CPU utility of the server and In 2019, Duggan et al. [9] proposed a recurrent neural network based server CPU and network bandwidth occupancy prediction to aid virtual machine migration in a cloud environment. Gupta et al. [13] firstly use an online sparse Bi-LSTM to tackle the same cloud workload prediction problem and Gao et al. [11] adopted the similar model to identify task and job failures in the cloud. Recently, some more complex model structure combined with deep learning have been proposed. Peng et al. [24] developed a novel prediction framework called *encoder – decoder* to make multi-step-head prediction for cloud workload. Zhang et al. [34] proposed an efficient deep learning model based on the canonical polyadic decomposition to predict the cloud workload for industry informatics. [18] proposed a workload prediction model using neural networks and self adaptive differential evolution algorithm. [6] proposed a deep learning based prediction algorithm for cloud workloads, which integrates top-sparse auto-encoder and gated recurrent unit block with RNN to improve forecast accuracy. Xia [30] in 2018 proposed a turning point prediction model called WSVM and it performed well in predicting sudden changes in virtual machine request number. Zhang et al. [33] developed an Interference-Aware Workload Parallelization (IAWP) method to better perform scheduling actions. Geng et al. [12] proposed a two-stage workload parallelization approach based on Deep Neural Network(DNN). By now the workload prediction researches using trace analysis methods are mainly focus on platform [20, 29] A cloud cluster trace called *Google cluster trace* is often studied for CPU, memory workload prediction, while the traces in storage area are rarely been studied. To the best of our knowledge, we are the first to incorporate LSTM in our prediction process.

5 Conclusion

In this paper, we proposed a practical deep learning based approach called CrystalLP to predict the workload in the storage system. Extensive experimental results verified that CrystalLP achieved performance improvement than three classic time series prediction algorithms in *RMSE*, *MAE* and *MAPE*. To the best of our knowledge, we are the first to introduce the deep learning method into server load analysis. According to our practice, we predict how to apply AI methods, e.g. deep learning methods, to big system's performance mining will be a promising research direction. The design of other deep learning based storage workloads analysis is in our on-going and future work.

Acknowledgements This work is by supported by the National Key R&D Program of China under Grant No. 2017YFB0202004, the National Science Foundation of China under Grant No. 61772053, the fund of the State Key Laboratory of Software Development Environment under Grant No. SKLSDE-2020ZX-15.

References

1. Abbasi, M., Shokrollahi, A.: Enhancing the performance of decision tree-based packet classification algorithms using CPU cluster. Clust. Comput. pp. 1–17 (2020)
2. Ahmad, I., Khalil, M.I.K., Shah, S.A.A.: Optimization-based workload distribution in geographically distributed data centers: a survey. Int. J. Commun. Syst. p. e4453 (2020)
3. Azizi, S., Li, D., et al.: An energy-efficient algorithm for virtual machine placement optimization in cloud data centers. Clust. Comput. pp. 1–14 (2020)

4. Bengio, Y., Delalleau, O., Roux, N.L.: The curse of dimensionality for local kernel machines. Tech. Rep. (2006)
5. Box, G.E.P., Jenkins, G.M.: Time series analysis, forecasting and control, holden-day. J. R. Stat. Soc. **134**(3), 229–240 (1976)
6. Chen, Z., Hu, J., Min, G., Zomaya, A.Y., El-Ghazawi, T.: Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning. IEEE Trans. Parallel Distribut. Syst. **31**(4), 923–934 (2019)
7. Di, S., Kondo, D., Cirne, W.: Host load prediction in a google compute cloud with a bayesian model. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, p. 21. IEEE Computer Society Press (2012)
8. Dong, B., Li, X., Wu, Q., Xiao, L., Li, R.: A dynamic and adaptive load balancing strategy for parallel file system with large-scale i/o servers. J. Parallel Distribut. Comput. **72**(10), 1254–1268 (2012)
9. Duggan, M., Shaw, R., Duggan, J., Howley, E., Barrett, E.: A multitime-steps-ahead prediction approach for scheduling live migration in cloud data centers. Softw. Pract. Exp. **49**(4), 617–639 (2019)
10. Firoz, J.S., Zalewski, M., Lumsdaine, A., Barnas, M.: Runtime scheduling policies for distributed graph algorithms. In: IEEE International Parallel and Distributed Processing Symposium, pp. 640–649 (2018)
11. Gao, J., Wang, H., Shen, H.: Task failure prediction in cloud data centers using deep learning. IEEE Trans. Serv. Comput. pp. 1–1 (2020). https://doi.org/10.1109/TSC.2020.2993728
12. Geng, X., Zhang, H., Zhao, Z., Ma, H.: Interference-aware parallelization for deep learning workload in GPU cluster. Clust. Comput. pp. 1–14 (2020)
13. Gupta, S., Dileep, A.D., Gonsalves, T.A.: Online sparse blstm models for resource usage prediction in cloud datacentres. In: IEEE Transactions on Network and Service Management pp. 1–1 (2020)
14. Hamilton, J.D.: Time series analysis, vol. 2. Princeton University Press Princeton, NJ (1994)
15. Huang, Z., Peng, J., Lian, H., Guo, J., Qiu, W.: Deep recurrent model for server load and performance prediction in data center. Complexity **2017**(99), 1–10 (2017)
16. Jassas, M.S., Mahmoud, Q.H.: Failure characterization and prediction of scheduling jobs in google cluster traces. In: 2019 IEEE 10th GCC Conference & Exhibition (GCC), pp. 1–7. IEEE (2019)
17. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. Comput. Sci. (2014)
18. Kumar, J., Singh, A.K.: Workload prediction in cloud using artificial neural network and adaptive differential evolution. Future Generat. Comput. Syst. **81**, 41–52 (2018)
19. Lu, Y., Sun, N.: An effective task scheduling algorithm based on dynamic energy management and efficient resource utilization in green cloud computing environment. Clust. Comput. **22**(1), 513–520 (2019)
20. Masdari, M., Khoshnevis, A.: A survey and classification of the workload forecasting methods in cloud computing. Clust. Comput. pp. 1–26 (2019)
21. Neelima, P., Reddy, A.R.M.: An efficient load balancing system using adaptive dragonfly algorithm in cloud computing. Clust. Comput. pp. 1–9 (2020)
22. Oral, S., Simmons, J., Hill, J., Leverman, D., Wang, F., Ezell, M., Miller, R., Fuller, D., Gunasekaran, R., Kim, Y., Gupta, S., Vazhkudai, D.T.S.S., Rogers, J.H., Dillow, D., Shipman, G.M., Bland, A.S.: Best practices and lessons learned from deploying and operating large-scale data-centric parallel file systems. In: SC ’14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 217–228 (2014)
23. Pang, P., Chen, Q., Zeng, D., Guo, M.: Adaptive preference-aware co-location for improving resource utilization of power constrained datacenters. IEEE Trans. Parallel Distribut. Syst. **32**(2), 441–456 (2020)
24. Peng, C., Li, Y., Yu, Y., Zhou, Y., Du, S.: Multi-step-ahead host load prediction with gru based encoder-decoder in cloud computing. In: 2018 10th International Conference on Knowledge and Smart Technology (KST), pp. 186–191. IEEE (2018)
25. Ping, L.: Analysis and development of the locality principle. Adv. Intell. Soft Comput. **133**(7), 211–214 (2012)
26. Sundermeyer, M., Ney, H.: From Feedforward to Recurrent LSTM Neural Networks for Language Modeling. IEEE Press, Oxford (2015)
27. Tang, K., Huang, P., He, X., Lu, T., Vazhkudai, S.S., Tiwari, D.: Toward managing HPC burst buffers effectively: draining strategy to regulate bursty i/o behavior. In: 2017 IEEE 25th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 87–98 (2017)
28. Tang, X., Liao, X., Zheng, J., Yang, X.: Energy efficient job scheduling with workload prediction on cloud data center. Clust. Comput. **21**(3), 1581–1593 (2018)
29. Wang, B., Wang, C., Song, Y., Cao, J., Cui, X., Zhang, L.: A survey and taxonomy on workload scheduling and resource provisioning in hybrid clouds. Clust. Comput. pp. 1–26 (2020)
30. Xia, B., Li, T., Zhou, Q.F., Li, Q., Zhang, H.: An effective classification-based framework for predicting cloud capacity demand in cloud services. In: IEEE Transactions on Services Computing (2018)
31. Xu, M., Buyya, R.: Brownout approach for adaptive management of resources and applications in cloud computing systems: A taxonomy and future directions. ACM Comput. Surv. **52**(1), 26–41 (2019). https://doi.org/10.1145/3234151
32. Yu, Y., Jindal, V., Bastani, F., Li, F., Yen, I.L.: Improving the smartness of cloud management via machine learning based workload prediction. In: 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), vol. 2, pp. 38–44. IEEE (2018)
33. Zhang, H., Geng, X., Ma, H.: Learning-driven interference-aware workload parallelization for streaming applications in heterogeneous cluster. IEEE Trans. Parallel Distribut. Syst. **32**(1), 1–15 (2020)
34. Zhang, Q., Yang, L.T., Yan, Z., Chen, Z., Li, P.: An efficient deep learning model to predict cloud workload for industry informatics. IEEE Trans Indust. Inform. **14**(7), 3170–3178 (2018)
35. Zhang, Z., Tang, X., Han, J., Wang, P.: Sibyl: Host load prediction with an efficient deep learning model in cloud computing. In: Algorithms and Architectures for Parallel Processing-18th International Conference, ICA3PP 2018, Guangzhou, China, November 15-17, 2018, Proceedings, Part II, pp. 226–237 (2018)

Li Ruan received her PhD degree in computer science and technology from Institute of Software, Chinese Academy. She is currently an assitant professor in state key laboratory of software developement environment, China and school of computer science and engineering at Beihang University. She is a senior membership of China Computer Federation. Her main research areas are AI security, cloud computing, computer system software, high performance computer.

Yu Bai was born in He Nan, China in 1994. He received the B.S. degrees in Electronic information science and technology from Zhengzhou University. He is currently pursuing the M.S. degree in computer science and technology at Beihang University, China. His research interest includes the deep learning and computer architecture.

Shaoning Li born in September 2001, is an undergraduate of Shie College, Beihang University. His main research interests include deep learning and network security.

Shuibing He received the PhD degree in computer science and technology from Huazhong University of Science and Technology, China, in 2009. He is now a ZJU100 Young Professor in the College of Computer Science and Technology at Zhejiang University. His research areas include parallel I/O systems, file and storage systems, high-performance and distributed computing.

Limin Xiao PhD, Professor of Beihang University, CCF senior member and IEEE member. His research interests include computer architecture and system software, high performance computer and server system, system virtualization and cloud computing, big data storage and distributed file system, computer system security. He authored more than 200 papers on international journals and conferences.