EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

A novel approach to workload prediction using attention-based LSTM encoder-decoder network in cloud environment

Yonghua Zhu[1,2], Weilin Zhang[1], Yihai Chen[1,3] and Honghao Gao[4,5*]

Abstract

Server workload in the form of cloud-end clusters is a key factor in server maintenance and task scheduling. How to balance and optimize hardware resources and computation resources should thus receive more attention. However, we have observed that the disordered execution of running application and batching seriously cuts down the efficiency of the server. To improve the workload prediction accuracy, this paper proposes an approach using the long short-term memory (LSTM) encoder-decoder network with attention mechanism. First, the approach extracts the sequential and contextual features of the historical workload data through the encoder network. Second, the model integrates the attention mechanism into the decoder network, through which the prediction for batch workloads can be carried out. Third, experiments carried out on Alibaba and Dinda workload traces dataset demonstrate that our method achieves state-of-the-art performance in mixed workload prediction in cloud computing environment. Furthermore, we also propose a scroll prediction method, which splits a long prediction sequence into several small sequences to monitor and control prediction accuracy. This work helps to dynamically guide the configuration for workload balancing.

Keywords: Workload prediction, LSTM, Encoder-Decoder Network, Attention mechanism, Cloud environment

1 Introduction

With the development of the Internet, many enterprises have accelerated and begun to include cloud-based online services. Because cloud computing can provide the capacity of on-demand network access, it enables an EIS or E-Common system to use service components without software development or refactoring, such as servers provided by Amazon, Microsoft, and Alibaba. These servers promise high availability with a probability of 99.95%, as declared in their SLA (service-level agreement). However, it is a challenge to keep their service at such a high rate while allocating as few resources as possible [1, 2]. Thus, predicting the workload helps the maintainers of the cloud-end cluster to estimate whether the current resource allocation strategy is sufficient or not [3, 4]. Based on these predictions, we can create corresponding scheduling for resource allocation or task assignment.

Existing works [5, 6] have proved that workload is a time sequence. This means that each workload at a time interval correlates its contextual workloads. Traditional statistical methods for processing time series data have been applied to workload prediction, such as auto-regressive model (AR) [6], moving average model (MA) [6], and auto-regressive integrated moving average model (ARIMA) [6]. Although these models have reasonable accuracy, they are highly dependent on the stationary form of collected data. Additionally, the model result will be changed dramatically due to different model parameters, which requires substantial manual work or experienced maintainer to adjust the parameters to fit the specific data features [7].

Recently, machine learning methods, as emerging tools, have been used to predict the workload: for example, Bayesian methods [8, 9] and k-nearest neighbor

* Correspondence: gaohonghao@shu.edu.cn
[4]Computing Center, Shanghai University, Shanghai 200444, China
[5]Shanghai ShangDa HaiRun Information System Ltd, Shanghai 200444, China
Full list of author information is available at the end of the article

(k-NN) [10]. These machine learning–based methods outperform the accuracy of traditional statistical approaches, which require only a little manual work. However, the historical workload values are considered as independent features, which mean that they ignore the relationships between the workloads. Fortunately, the recurrent neural network (RNN), a particular form of neural network, has solved this problem. RNN is designed to learn the internal correlations between data and their context in a sequence, but few RNN-based workload prediction methods have been proposed, such as echo state network [11], basic LSTM network [12], and GRU encoder-decoder network [13]. Thus, we are motivated to employ machine learning to the application areas of workload prediction.

All of the RNN-based methods have high accuracy in workload prediction of application servers. When we explore the service deployment of cloud service providers, we find that their clusters are running for both applications and batching, and the accuracy of the above methods drops when predicting such mixed workloads. Batching is an approach that divides a time-consuming task into multiple sequential subtasks to increase efficiency. We notice that, in batch workload prediction, the impact of the historical workload on the current workload is different, and these methods give the historical sequence the same weight during feature extraction [14, 15]. Therefore, we introduce an attention mechanism to address this problem. When dealing with sequence data, attention mechanism evaluates the relevancy of the historical data and gives corresponding weights. In this manner, the importance of each workload in the historical sequence can be recognized. To the best of our knowledge, attention-based RNNs have shown their power in the machine translation domain [16, 17] and have not been applied to workload prediction.

In this paper, we combine the attention mechanism with an RNN-based method and based on which LSTM encoder-decoder network with attention for workload prediction is proposed. The model contains two LSTM networks that act as the encoder and the decoder, as well as an output layer. The encoder maps the historical workload sequence to a fixed-length vector according to the weight of each time step supported by the attention module, namely, the context vector. Then, the decoder maps the context vectors back to a sequence. Finally, the output layer transforms the sequence into the final output. In this paper, our contributions are as follows:

1) Attention mechanism is applied to the RNN-based model. It enhances the prediction accuracy of batch workloads during workload prediction.
2) A scroll prediction method is proposed that divides a long prediction sequence into several small sequences to increase the accuracy of the long-term prediction method.
3) Experiments show that our approach reaches state-of-the-art performance and can achieve almost the same prediction accuracy.

The rest of the paper is organized as follows: Related Works gives a review of related work on workload prediction. Our Approach introduces the technical and conceptual details of our approach. The contrast experiment and its result and discussion are presented in Experiments. Finally, the conclusion is given in Conclusion and future work.

2 Related works

In this section, related works on predicting workload are divided into linear methods, machine learning methods, and RNN-based methods.

2.1 Linear method–based workload predictions

In the beginning, a server cluster is designed to increase the performance and availability of service [18–20]. Under such circumstances, most servers in the cluster are running the same applications, and the workload reflects how many requests are responded to on one server. The workload sequence consists of long-term trends and cyclic changes, which can be regarded as time series data [21, 22].

To explore the features of historical workload sequences, researchers have applied many linear models [6, 23–26] for processing time series to workload prediction. Dinda et al. [6] put forward a dataset that contains four types of UNIX distributed system workload traces. They use and compare AR, MA, and ARIMA models on their dataset and find that a simple AR model has the best predictive power. Wu et al. [23] combine AR model with Kalman filter for multistep-ahead workload prediction. Calheiros et al. [24] use ARIMA model in software as a service (SaaS) applications and reduce its impact on the quality of service (QoS) to its minimum.

These time series methods first transform the non-stationary time series to stationary time series through k-order difference methods, where the factor k greatly determines the final result of the model. Despite the high accuracy with a proper k in time series transformation, finding that k is difficult when the workload dataset is large, and this approach requires much manual work.

2.2 Machine learning method–based workload predictions

Cloud computing services enable one host to become multiple cloud virtual machines through virtualization technology. Such virtualization technology makes the workload much more complicated and harder to predict

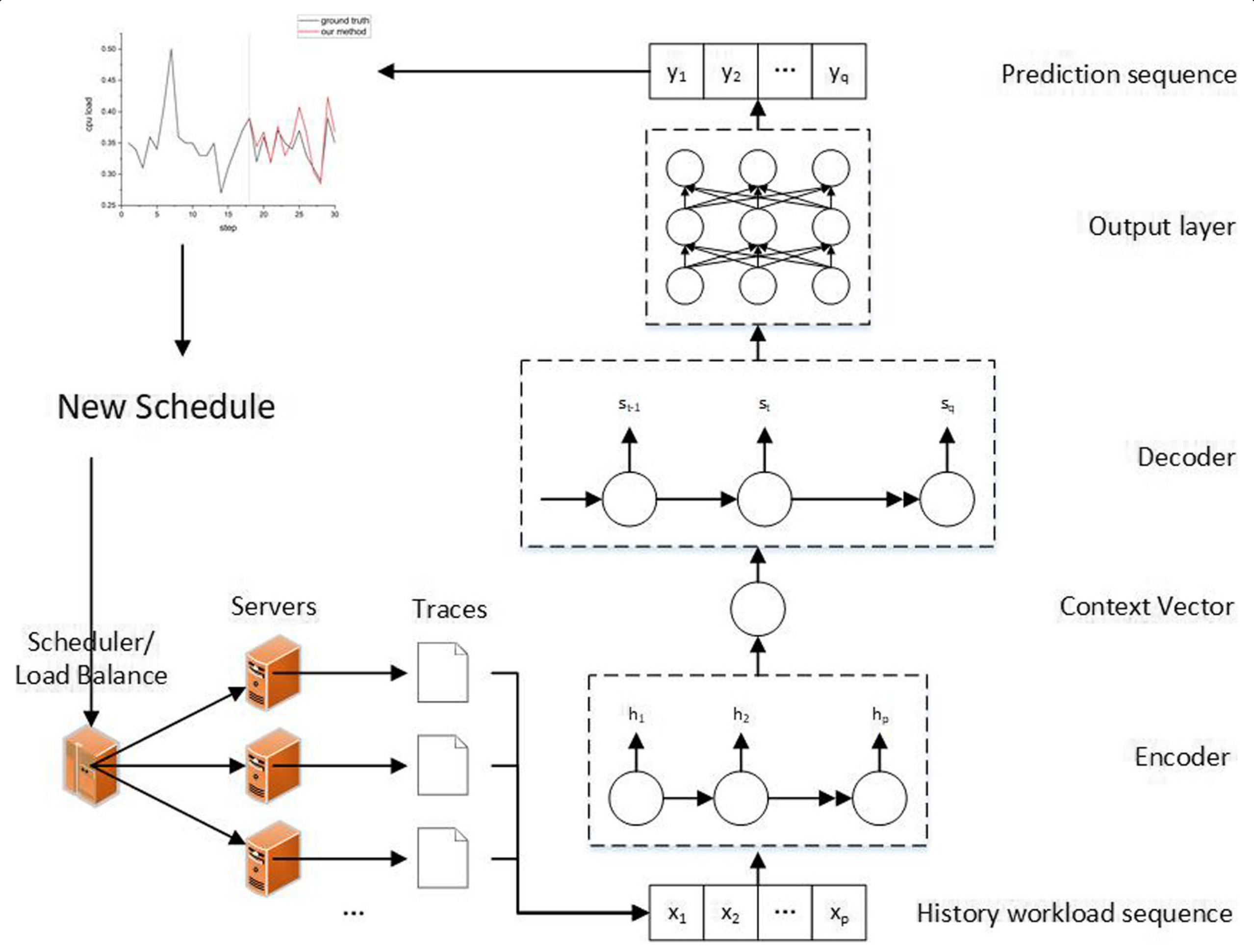

Fig. 1 The framework of the LSTM encoder-decoder network with attention for workload prediction

through linear models [27, 28]. Therefore, machine learning algorithms, which are good at nonlinear problems, have been studied by researchers to support the prediction.

Di et al. [8] introduce Bayesian model for future host load prediction. The model proposes nine features of the recent historical load to predict the mean load over consecutive time intervals. Benhammadi et al. [9] integrate fuzzy inference and Bayesian inference methods to predict CPU loads. Liang et al. [10] propose a kNN-based approach to predict long-term CPU workloads. Cao et al. [29] use ensemble learning to combine the result of several algorithms and dynamically adjust the parameters with the prediction residual. Singh et al. [30] combine ARIMA and support vector regression model (SVR) to adapt to different workload features. Kumar et al. [31] use artificial neural network to predict workload and adaptive differential evolution method to enhance the accuracy. Urgaonkar et al. [32] use dynamic queuing model to predict resources required in each tier of Internet.

Unlike the linear models, the Bayesian methods and k-NN methods directly use the history as features to build various rules mapping the historical workloads to future workloads. These methods only require a little manual work for hyper parameter adjustment and can achieve good accuracy in prediction results. However, these methods do not consider the correlation between the workload values of different time steps, which is improper for batch-workload cloud computing environments [33–35].

2.3 RNN-based method-based workload predictions

Recurrent neural network is designed to model the relationships between the items in the sequence, which makes it quite suitable to do the workload prediction tasks. Song et al. [12] use basic LSTM network to predict the multistep-ahead workload and achieve pretty good performance. Peng et al. [13] propose a GRU-based encoder-decoder network model to enhance the long-term prediction ability of

RNNs. The model encodes the historical sequence to a fixed-length vector and decodes the vector to predict the future workload value. Huang et al. [36] use RNN with long short-term memory to analyze user request logs to predict servers' performance. Additionally, they proposed a new way to reproduce user request sequences by RNN-LSTM. Kumar et al. [37] predict the number of requests with LSTM and achieve SOTA performance.

Our proposed method uses attention-based LSTM encoder-decoder network to enhance the prediction for batch workloads. According to the experimental results, our method outperforms the previous methods in both traditional distributed system and mixed cloud computing environment, and the accuracy score of the mixed cloud computing environment catches up with that of conventional distributed system. Furthermore, we put forward a scroll prediction method that helps prevent the error from being amplified when the prediction step goes long.

3 Our approach

Figure 1 shows the complete cycle of workload schedule adjustment using our workload prediction approach, and the framework of our model is on the right side in Fig. 1. First, the workload traces are collected from every server in the cluster. Our workload prediction model analyzes these traces and predicts workload change over the next period of time. A new allocation schedule is then made and is updated to the load balance server. The framework of our model is on the right side in Fig. 1. The model consists of two components: an LSTM-based encoder-decoder network and an output layer. First, the time sequence data is inputted into the encoder, where it will be encoded into the context vector. Then, the decoder iteratively generates the intermediate prediction results for the output layer. Finally, the output layer outputs the prediction values of the workload. Given the input workload value sequence of the last time, step p is $x_1, x_2, \ldots, x_p$, the model outputs the workload prediction of the future time step q as $y_1, y_2, \ldots, y_q$.

3.1 Long short-term memory

Recurrent neural network (RNN) is suitable for processing time sequence data because RNN models the relationships between the former states and the latter states. However, the vanilla RNN architecture, which is shown in Fig. 2a, suffers from "long dependency" problem, which stops the RNN from processing a long sequence [38]. Therefore, LSTM network [39], which is capable of learning the long-term dependencies, is selected in the model. The architecture of the LSTM cell is shown in Fig. 2b, where there are a hidden state and three gates in addition to the vanilla RNN cell.

At each time step t, given the input x_t, the calculations of the current hidden state h_t and the cell state C_t in the LSTM cell are as follows [39]:

$$\begin{aligned} f_t &= \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\ C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned}$$

where σ is the sigmoid function and *tanh* is the hyperbolic tangent function. The symbols i_t, f_t, and o_t denote the input gate, forget gate, and output gate, which decides whether to update the cell state with the input, forget the memory from the last time step, and output the memory, respectively. W_f, W_i, W_o, W_C and b_f, b_i, b_o, b_C are the weight matrixes and the biases of the three gates and the cell state.

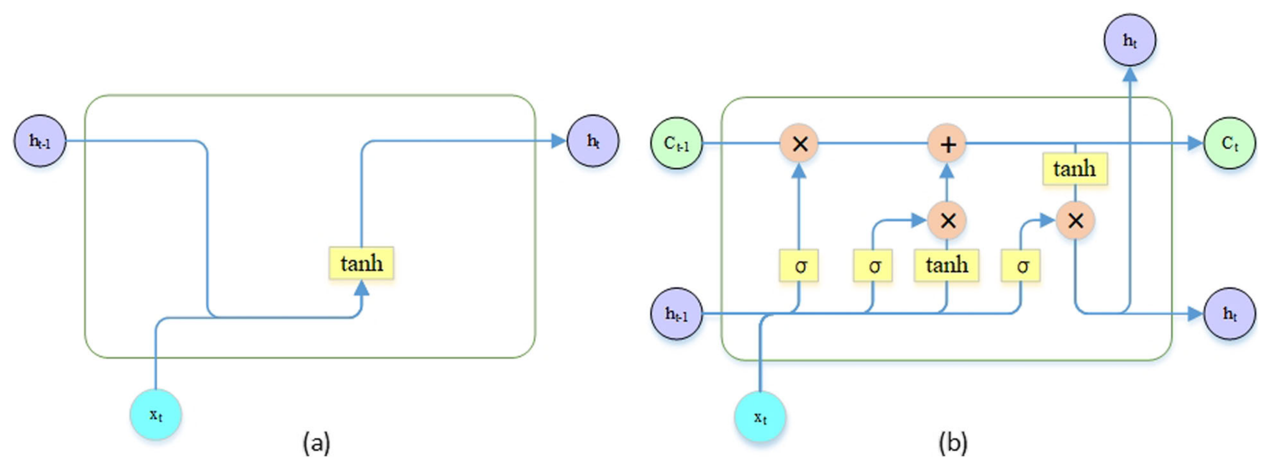

Fig. 2 The architecture of an RNN cell. **a** The architecture of a vanilla RNN cell. **b** The architecture of a LSTM cell

The hidden state vector represents the current state of the implicit variables of the workload, and the cell state vector is the accumulated change of the entire historical workload on the implicit variables. During the calculation in the LSTM as above, the hidden state and the input jointly decide how the workload at the current time step impacts the accumulated change of history, i.e., the cell state, and the three vectors together determine the updated hidden state, which is also the output of the LSTM cell.

3.2 LSTM encoder-decoder network

Figure 3 shows the unfolded architecture of the LSTM-based encoder-decoder network. The model consists of three parts, an LSTM-based encoder network, an LSTM-based decoder network, and a context vector. The encoder network encodes the input sequence into the context vector, and the decoder network decodes the context vector step by step to output the prediction value. In general, the encoder network and the decoder network are independent of each other, which means that the parameters inside the LSTM cell are not shared between the encoder and the decoder.

In the encoding stage, the input workloads are fed into the LSTM network sequentially. The hidden state and the cell state are updated when the network reads the input workload value. When the input sequence reaches its end, the hidden state and the cell state are sent to the context vector, which represents the overall encoding result of the input sequence.

The decoder network outputs the predicted sequence by iteratively decoding the context vector. There are two types of decoders, as shown in Fig. 3a and b, and they differ based on whether the context vector takes part in each time step. In the (a) model [40], the context vector works as the initial state of the decoder LSTM cell, and the output of the last time step is taken as the input of the current time step. The hidden state of the context vector carries the final state of the implicit variable, and the cell state summarizes the accumulated change of the entire history. Then, the decoder network continues to update the hidden state and the cell state, the only difference being that the input is no longer a ground-truth workload value. The initial input of the decoder network is the average workload of the historical workloads.

In the (b) model [41], the context vector is part of the input at each time step, where the output of the last time step and the context vector together form the input of the current time step. The input sequence of the decoder starts with an initial input s_0, and the rest of the inputs are the output of the decoder in the last time step. The decoding LSTM cell iteratively reads the input, updates its state and hidden state, and outputs its prediction of the current time step. The output is transformed through the output layer and is fed back to the decoder network as the next input.

Obviously, model (a) is simpler and more explainable, but it may accumulate errors in the iteration process. In addition, model (a) tends to converge with more epochs than model (b) in our early experiments. The advantage of model (b) is more about its flexibility, which allows changes to how the context vector is calculated during the sequence, whose typical representative is the attention mechanism.

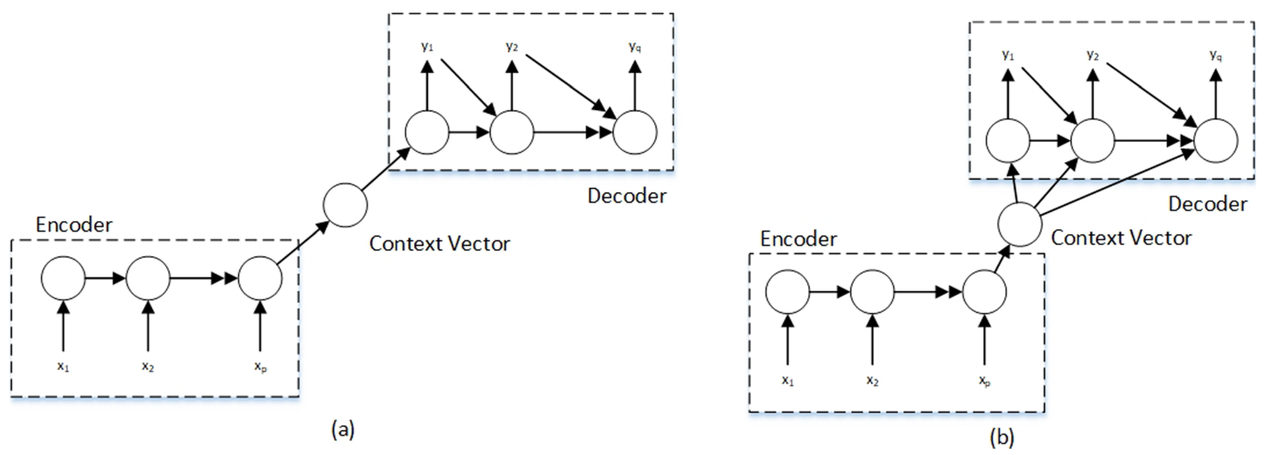

Fig. 3 The unfolded architecture of the LSTM encoder-decoder network. **a** The context vector is only connected to the first decode step. **b** The context vector is connected to each decode step

3.3 LSTM encoder-decoder network with attention

The LSTM encoder-decoder network has the ability to deal with the workload sequence prediction task when the workload at each time step is simple time series data. However, only when the hosts in the cluster are doing the same computing job or providing the same application do the workloads of the cluster become time series data. In a large cloud computing environment, compute-intensive jobs are often divided into multiple subparts, which are also known as batch workloads. In batch workloads, latter subparts must wait for the previous subparts to be finished before they can be carried out.

In such a case, each step in the historical workload sequence has a different impact on the current workload. For example, the peak workloads and the initial workloads of the latter subparts may affect it greatly, while bottom workloads may have tiny impacts. Therefore, when modeling the relationships between the current time step and its context, the historical workload sequence should be given different weight at each position rather than given the same weight. A basic LSTM encoder-decoder network gives the historical sequence the same weight, so the attention mechanism is introduced to solve the issue.

The attention mechanism is similar to human behavior when reading a sentence in that one tends not to pay the same attention to each word in the sentence but instead focus on important words. The attention mechanism evaluates how important each part is by giving a weight to each part in the sequence; the higher weight is, the more important the word is. Similar to how the attention operates in sentence processing, the attention module in our approach gives each workload in the input sequence different weight, which represents how much the workload impacts the current workload prediction.

Figure 4 shows the details of the attention module. The attention module is part of the decoder network and replaces the context vector as input. In the attention module, the context vector c_i at the i^{th} decoding time

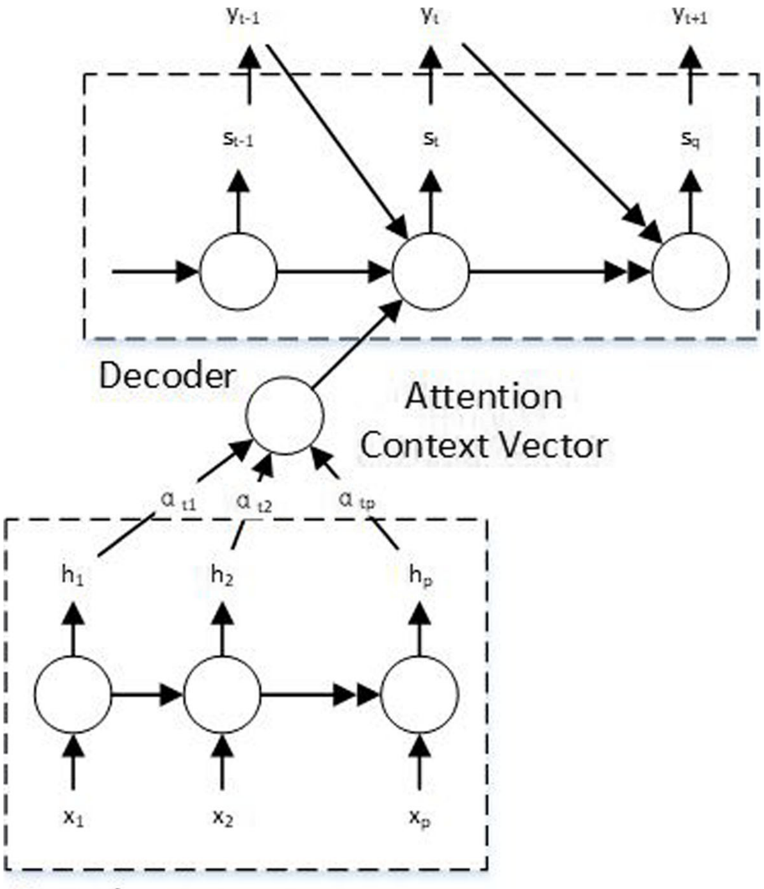

Fig. 4 The architecture of the attention mechanism in the decoder network

step is computed as a weighted sum of the hidden states of the encoder network [40]:

$$C_i = \sum_{j=1}^{T} \alpha_{ij} h_j$$

The weight α_{ij} of each hidden state h_j is calculated by:

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T} \exp(e_{ik})}$$

where

$$e_{ij} = a(S_{i-1}, h_j)$$

is the correlation value of the output at position i and the input at position j, where a denotes the scoring function that evaluates the correlation value. In our approach, global general attention is selected as the scoring function, which is computed by [42]:

$$e_{ij} = s_{i-j} W_a h_j$$

where W_a is the weight matrix of the scoring function.

From the computation above, the attention mechanism is more like a selection process. In this mode, the system regards the implicit variables of a workload as the composition of its historical workloads, and the weight of each historical workload represents its impact on the current workload. It is a more advanced form of searching for similar historical situations: during the training process, the general attention is trained to memorize how much the current workload and the historical workload are correlated under all circumstances in the training set, and the attention mechanism has learned how to select the correlated history after the training.

3.4 Deep LSTM encoder-decoder network

LSTM Encoder-Decoder Network illustrates the LSTM encoder-decoder framework, and Fig. 3 shows the two types of single-layer encoder-decoder network. However, when the relationship between the input and its context is complex, a single-layer network may not be sufficient to express the features. The deep LSTM encoder-decoder network extracts the implicit features from low level to high level with the layer going deeper, and the high-level features are synthesized by low-level features and are more likely to lead to workload change. Therefore, the encoder-decoder network can be stacked up to form a deep architecture to model the more complex features, as shown in Fig. 5.

During the encoding process, the deep layers take the output sequence of the former layer as their input sequence. As in the example of the three-layer encoder-decoder network in Fig. 5, the LSTM cell of the second layer is fed with the output of the first layer, and the third layer uses the output of the second layer as the input sequence. At each time step t, the state and the hidden state of the LSTM cell are updated from shallow layer to deep layer, where the deepest layer may contain the highest-level feature of the input sequence. After inputting the last of the input sequence, each layer of the encoder separately sends its state and hidden state to the context vector. The context vectors of the network are independent among the layers, which are the encoding of its belonging layer.

The decoder network works almost the same as the single-layer decoder network does except for the multilayer computing. There are also two types of decoder in the deep form, with the difference between them being whether the context vector only joins the first time step or joins each time step. In the multilayer decoder,

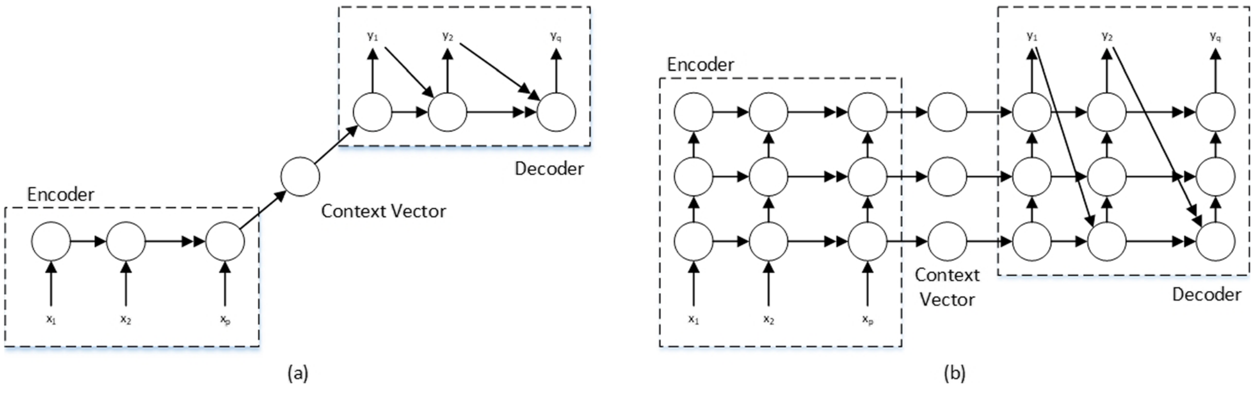

Fig. 5 The unfolded architecture of a deep LSTM encoder-decoder network. **a** Single-layer network. **b** Three-layer network

regardless of how the context vector joins the cell updating, the input is sent into the first layer, whose output works as the input of the second layer. Eventually, the output of the last layer is transformed through the output layer and is fed back to the decoder network as the next input.

There is a trade-off of the architecture, that is, the deep LSTM encoder-decoder network has better performance than the single-layer LSTM encoder-decoder network when dealing with a long sequence, but it incurs more than twice the time cost compared to the single-layer network (Fig. 5). Nevertheless, when the sequence is not so long, the deep LSTM encoder-decoder network can easily get overfitting due to its complex model. The performance of the single-layer network and multilayer network will be discussed in Experiments.

3.5 Output layer

The output layer transforms the output of the decoder network into the final prediction value of the model. Because the output layer actually works as a regression function rather than a classifier, the traditional selection of softmax function and argmax function is inappropriate in our model.

The output layer is a three-layer perceptron network. The activation function of the first two layers is a parametric rectifier linear unit (PReLU), which is calculated as follows [43]:

$$f(x) = \max(\alpha x, x)$$

where α is a parameter that is updated through the training process. PReLU is proved to have better performance than ReLU or Leaky ReLU (a special form of PReLU where the parameter α is set to 0.01), and it only adds a few parameters to the model, which may not increase the risk of overfitting. The third layer is activated by the sigmoid function to constrain the prediction value to the range between 0 and 1:

$$y = f(x) = \frac{1}{1 + e^{-\theta x}}$$

where y is the final prediction value of the future workload.

3.6 Model training

The goal of the encoder-decoder network is to estimate the conditional probability of the output sequence when given the input sequence. The attention module does not change the goal of the entire encoder-decoder network; it only impacts the context vector. Denoting the context vector of the decoder network at position t as c_t, the conditional probability of the output sequence is [41]:

$$p\left(y_1, y_2, \dots, y_q | x_1, x_2, \dots, x_p; \theta\right) = \prod_{t=1}^{q} p(y_t | c_t, y_1, y_2, \dots, y_{t-1}; \theta)$$

The encoder and decoder are jointly learned by maximizing the log-likelihood of the output sequence, which is:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \sum \log\left(p\left(y_1, y_2, \dots, y_q | x_1, x_2, \dots, x_p; \theta\right)\right)$$

The parameters inside the LSTM cell are updated through a backpropagation algorithm. The encoder network and the decoder network are jointly learned, but their parameters are independently updated. Denoting the loss function as L, the parameter updating at time step T is as follows [39]:

$$\frac{\partial L}{\partial W} = \sum_{t=0}^{T} \frac{\partial L^t}{\partial W}$$

where W can be W_f, W_i, W_o, and W_C. The detailed formulas of the update process of the four matrixes are the following [39]:

$$\begin{aligned}
\frac{\partial L}{\partial W_f} &= \sum_{t=0}^{T} \frac{\partial L}{\partial C_t} \frac{\partial C_t}{\partial f_t} \frac{\partial f_t}{\partial W_f} = \sum_{t=0}^{T} [\delta C_t \odot C_{t-1} \odot f_t \odot (1-f_t)] h_t^T \\
\frac{\partial L}{\partial W_i} &= \sum_{t=0}^{T} \frac{\partial L}{\partial C_t} \frac{\partial C_t}{\partial i_t} \frac{\partial i_t}{\partial W_i} = \sum_{t=0}^{T} \left[\delta C_t \odot \widetilde{C_t} \odot i_t \odot (1-i_t)\right] h_t^T \\
\frac{\partial L}{\partial W_o} &= \sum_{t=0}^{T} \frac{\partial L}{\partial h_t} \frac{\partial h_t}{\partial o_t} \frac{\partial o_t}{\partial W_i} = \sum_{t=0}^{T} [\delta h_t \odot \tanh(C_t) \odot o_t \odot (1-o_t)] h_t^T \\
\frac{\partial L}{\partial W_C} &= \sum_{t=0}^{T} \frac{\partial L}{\partial C_t} \frac{\partial C_t}{\partial \widetilde{C_t}} \frac{\partial \widetilde{C_t}}{\partial W_C} = \sum_{t=0}^{T} \left[\delta C_t \odot i_t \odot \left(1 - \left(\widetilde{C_t}\right)^2\right)\right] h_t^T
\end{aligned}$$

The parameters of the output layer contain α in the PReLU formula and θ in the sigmoid function. The parameter α is updated through gradient descent with momentum:

$$\Delta\alpha^t = \mu \Delta\alpha^{t-1} + \epsilon \frac{\partial L}{\partial \alpha}$$

where the momentum $\Delta\alpha^{t-1}$ is the change in α at the last gradient descent step t-1, μ is the factor for the momentum, and ϵ is the learning rate of the system. The parameter of the last layer's sigmoid function is updated with simple gradient descent:

$$\Delta\theta = \epsilon \frac{\partial L}{\partial \theta}$$

The loss function is the Huber loss [44] with L2 regularization:

$$L\left(y^{(i)}, \widetilde{y^{(i)}}\right) = \begin{cases} \frac{1}{2}\left(\widetilde{y^{(i)}} - y^{(i)}\right)^2 & \text{for} \left|\widetilde{y^{(i)}} - y^{(i)}\right| \leq \delta \\ \delta\left|\widetilde{y^{(i)}} - y^{(i)}\right| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} + ||\lambda||^2$$

where $y^{(i)}$ is the ground truth of the i^{th} sample, $\widetilde{y^{(i)}}$ is the prediction value of the i^{th} sample, and λ represents the parameters of the entire model. Huber loss is a smoother form of squared error loss, which is less sensitive to the outlier samples than squared error loss.

The parameter estimation method is mini-batch gradient descent, and the Adam optimizer [45] is selected to help the model to converge. The gradient update at each time step is calculated as follows in the Adam optimizer [45]:

$$\begin{aligned} g_t &= \nabla_\theta J(\theta_{t-1}) \\ m_t &= \beta_1 m_{t-1} + (1-\beta_1)g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)g_t^2 \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\ \theta_t &= \theta_{t-1} - \eta * \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} \end{aligned}$$

where η is the learning rate, β_1 and β_2 are the learning rate decay factors, and ϵ is a tiny number to avoid the divisor ever equaling 0. In the Adam optimizer, the moving average of the gradient and squared gradient are calculated as m_t and v_t. $\hat{m}_t$ and $\hat{v}_t$ are the bias correction of m_t and v_t because the moving average tends to have a large bias in the first few steps.

4 Experiments

In this section, experiments are carried out to demonstrate the effectiveness of our approach. First, in Datasets and Preprocessing and Parameter Setting, the preparation of the experiments will be introduced, which includes detailed statistics, descriptions of datasets, and the parameters of our model. Then, in Evaluation Metrics and Baseline Methods, the evaluation metrics and other workload prediction models are introduced. Next, the contrasting experimental results of our model and baseline methods on the datasets are discussed in Comparison of the Experimental Results and Discussion. A few more discussions about the model's intrinsic structures are presented in Discussion of History Window Length and Prediction Sequence Length and Trade-offs of the Deep Model and Attention Mechanism. Finally, a discussion about scrolling prediction is provided in Discussion of Scrolling Prediction.

4.1 Datasets and preprocessing

In this paper, two datasets collected from real cloud environments are used to evaluate the performance and verify the effectiveness of our method, Alibaba cluster-trace-v2018[1] and Dinda[2].

Alibaba cluster-trace-v2018 is provided by the Alibaba Open Cluster Trace Program and is the new version that contains the traces of approximately 4000 machines in a period of 8 days. Each machine in the cluster provides both long-running applications and batch workloads. The workload change through time of one host is shown in Fig. 6.

The Dinda workload dataset is collected by Carnegie Mellon University, whose traces were collected from late August 1997 to March 1998 on roughly the same group of machines. The Dinda dataset consists of four types of workload, which refer to four different runtime scenarios, the descriptions and statistics of which are shown in Table 1.

Before putting the data into our model, it is preprocessed through several stages. First, normal values in both datasets are scaled to a range from 0.1 to 0.9 by the minimum-maximum scaler:

$$x_i = LWL + \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}(UPL - LWL)$$

where $x_{\min}$ and $x_{\max}$ refer to the minimum value and maximum value of the dataset. LWL and UPL are the lower and higher limits of the target range, which are set to 0.1 and 0.9, respectively.

Second, abnormal values are replaced with specified values. For the Alibaba dataset, whose abnormal values are 101 and − 1, the substitution is set to 0 and 1, respectively, and represents machine failures caused by physical reasons and workload overflow.

4.2 Parameter setting

The hyper parameters of our approach are presented in Table 2.

The hyper parameters are determined in multiple ways. First, the three hyper parameters concerning the architecture, history window length, and dimension of the hidden state in the encoder network and decoder network are selected via grid search. The grid search of the history window length is conducted among $p \in \{12,18,24,30,36,42,48\}$, and the dimension of the hidden state in the encoder and decoder is searched among {16,32,64,128}, while the prediction step is fixed to 12. The length of the prediction sequence, i.e., the prediction step, is a variable whose impact on the accuracy will be discussed in

[1]https://github.com/alibaba/clusterdata
[2]http://www.cs.cmu.edu/~pdinda/LoadTraces/

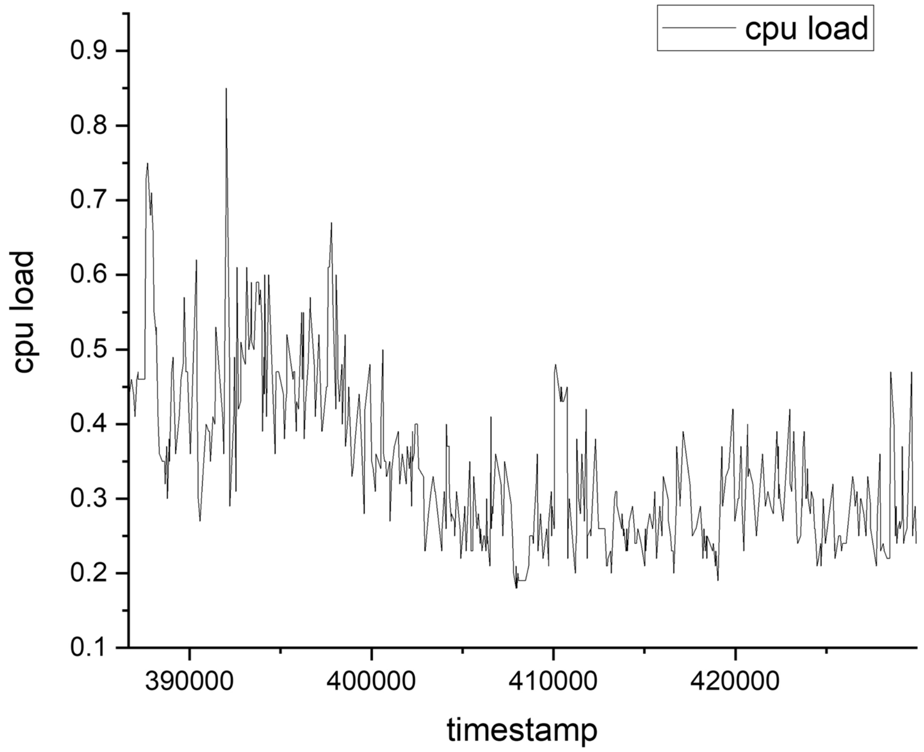

Fig. 6 The workload of one host in Alibaba cluster-trace-v2018. CPU load: The CPU workload of one host that changes from 0 to 1. The horizontal Axis is the timestamp in the dataset

Discussion of History Window Length and Prediction Sequence Length.

The other hyper parameters concerning model training are set according to prior works and some tuning. Batch size is set to the limit of the experiment device, where 128 is the max size for which the server does not go out-of-memory. The factor for momentum is set to 0.8 according to [43], and δ in the Huber loss function is set to 1.35 according to the distribution of outliers. The three factors in the Adam optimizer, i.e., the initial learning rate η and two factors for moving average β_1 and β_2, are set following [13, 45, 46].

4.3 Evaluation metrics

To evaluate the effectiveness of the workload prediction approaches, three metrics are considered, which are the mean absolute error (MAE), root mean squared error (RMSE), and mean absolute percentage error (MAPE). These metrics are computed as follows:

Table 1 Description of the Dinda workload dataset

Name	Description	Traces	Mean	Stddev
Axp0	A heavily loaded, highly variable interactive machine on the PSC cluster.	1,296,000	1	0.54
Axp7	A more lightly loaded batch machine on the PSC cluster that has interesting epochal behavior	1,123,200	0.12	0.14
Sahara	A moderately loaded, big memory compute server in the CMCL	345,600	0.22	0.33
Themis	A moderately loaded desktop machine.	345,600	0.49	0.5

Table 2 Hyper parameters of the LSTM encoder-decoder network with attention

Hyper parameter	Value
History window length	18
Dimension of hidden state in encoder	64
Dimension of hidden state in decoder	64
Batch size	128
Factor for momentum μ	0.8
δ in Huber loss function	1.35
Initial learning rate η	0.001
Factor for moving average β_1	0.9
Factor for moving average β_2	0.999

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n} |\widetilde{y^{(i)}} - y^{(i)}|$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\widetilde{y^{(i)}} - y^{(i)}\right)^2}$$

$$\mathrm{MAPE} = \frac{1}{n}\sum_{i=0}^{n} \left|\frac{\widetilde{y^{(i)}} - y^{(i)}}{y^{(i)}}\right| \times 100\%$$

Among the three metrics, MAE and RMSE are scale-dependent, and MAPE is scale-independent, which denotes the Manhattan distance, Euclid distance, and deviation proportion between the ground truth value and the prediction value. For each metric, the performance of a model is better when the metric gets a lower value.

In addition to the three metrics for regression tasks, root mean segment squared error (RMSSE) is also used to evaluate the models. RMSSE is a traditional metric for quantifying the prediction performance and was put forward in [8]. Because the actual workload is hard to predict in the past, RMSSE evaluates the error of the average workload between the ground truth value and the prediction. RMSSE is computed as follows:

$$\mathrm{RMSSE} = \sqrt{\frac{1}{s}\sum_{i=1}^{n} s_i (l_i - L_i)^2}$$

where $s_i = b\cdot 2^{i-1}$, $s = \sum_{i=1}^{n} s_i$, b is the basic window length, and s_i is the separate segment; l_i and L_i denote the prediction value and the ground truth value, respectively; and n is the number of segments.

4.4 Baseline methods

To verify the effectiveness of our approach, several baseline methods are selected for comparison.

ARIMA: The auto-regressive integrated moving average model (ARIMA) [6] is a traditional statistical model for time series data prediction. First, the model analyzes the time series data and transforms the non-stationary time series to stationary time series data through the k-order difference method, which is the key procedure of ARIMA. Then, according to the auto-correlation function and the partial autocorrelation function of the stationary time series, the order p for the lags of the auto-regressive model and the order q for the lags of the moving average model are determined. Finally, the least-squares method is applied to parameter estimation.

PSR + EA-GMDH: The phase space reconstruction (PSR) method combines the group method of data handling based on evolutionary algorithm (EA-GMDH) [47], a model that contains two stages of works. First, the model reconstructs the workload into multidimensional phase space. Then, the result is fed into the EA-GMDH network, where an evolutionary algorithm is responsible for adjusting the parameters of the model and finally outputting the prediction sequence.

LSTM: The basic long short-term memory network [12] is a recurrent neural network that uses LSTM as the computing unit. Unlike the encoder-decoder architecture, basic LSTM network outputs the prediction workload of the next time step when fed the current workload value.

GRUED: The gated recurrent unit (GRU) encoder-decoder model [13] is an RNN-based encoder-decoder network similar to ours. The differences are that our model uses LSTM rather than GRU and that our model is equipped with attention module.

4.5 Comparison of the experimental results and discussion

The performances of our approach and baseline methods for the Alibaba dataset are shown in Table 3, and the results for the Dinda dataset are shown in Fig. 7, where both results are the average value of five experiments. All of the models are used to make a 12-step prediction.

From Table 3, we can see that three RNN-based methods, basic LSTM, GRUED, and our model, are all better than the two non-RNN methods on both the Alibaba traces and Dinda dataset. Though the ARIMA model has a perfect theoretical basis, we find it hard to actually transform the historical workload sequence into its stationary form, which prevents the error from getting lower. The problem of PSR + EA-GMDH is that the model cannot make use of the long-term historical workload efficiently, so the model fails to achieve an accurate prediction when the prediction step is 12 steps long.

Among the three RNN-based methods, two models with an encoder-decoder architecture, GRUED, and our approach, score better than basic LSTM, which suggests the effectiveness of the encoder-decoder architecture. It is because the encoder network can not only extract the hidden features of the context but also extract that of the overall sequence, and the decoder network can select

Table 3 Workload prediction result of baseline methods and our approach on Alibaba cluster-trace-2018

Model	Alibaba×10^{-2}			
	MAE	RMSE	MAPE	RMSSE
ARIMA	7.016	8.337	33.126%	8.272
PSR + EA-GMDH	7.021	8.346	33.174%	8.311
LSTM	5.756	6.903	28.031%	6.014
GRUED	4.371	5.211	24.362%	4.432
Our model	3.520	4.134	19.529%	3.815

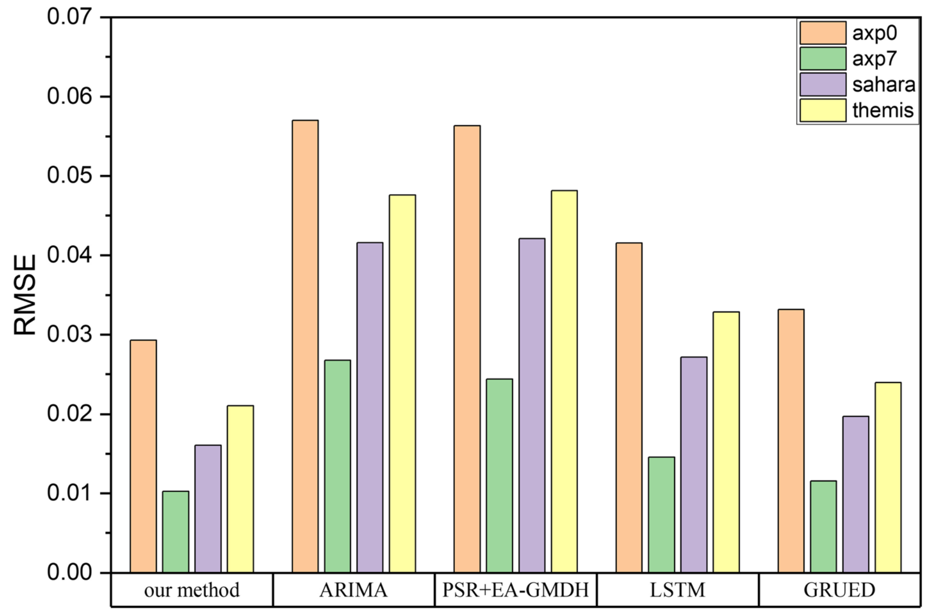

Fig. 7 Workload prediction RMSE of four types of clusters in the Dinda dataset. axp0: The orange bar, which is listed at the 1st position from left to right in each method section, is the mean RMSE of five experimental results on the axp0 trace. axp7: The green bar, which is listed at the 2nd position from left to right in each method section, is the mean RMSE of five experimental results on the axp7 trace. sahara: The purple bar, which is listed at the 3rd position from left to right in each method section, is the mean RMSE of five experimental results on the sahara trace. themis: The purple bar, which is listed at the 4th position from left to right in each method section, is the mean RMSE of five experimental results on the themis trace

the hidden features when outputting the prediction value. Our model outperforms GRUED because the attention mechanism enhances the decoder network, i.e., the decoder with attention, can evaluate which historical workload impacts the current computing step most, which is suitable for the batch workloads.

To study how the actual error at each position increases in the prediction sequence, we calculate the RMSE at each position in the 12-step prediction sequence of our model, GRUED, and LSTM, which is presented in Fig. 8. Additionally, to give an intuitive view of the error change, in Fig. 9, we put the ground truth and the prediction value together and present the prediction curves of the three methods.

In Fig. 8, there are three polylines, which represent the RMSE changes of three RNN-based methods at each position in the 12-step prediction. The RMSE values of all three methods tend to increase as the position in the prediction sequence becomes larger. Our model has the lowest error rate at each position in the prediction among the three RNN-based methods. Moreover, our method also has the slowest error growth rate. From the 1st position to the 12th position, the RMSE of our model increases by approximately 0.52×10^{-2}, while GRUED and LSTM increase by approximately 0.66×10^{-2} and 0.87×10^{-2}, respectively. This result proves that the attention mechanism is effective in mitigating error amplification in the long-term prediction.

In the three subfigures in Fig. 9, the ground truth workload and the predicted workload are put together, which are the black polyline and red polyline, respectively. In Fig. 9a, our model, the red polyline, is close to the black one, and most directions of change are predicted correctly. In Fig. 9b, for GRUED, the deviation of the predicted workload is slightly larger than that of our model, and a few directions of change are wrong. In Fig. 9c, LSTM, the red polyline, is almost not following the black polyline. There are several predictions with significant deviations, and the prediction of the direction of change is also unsatisfactory. The two models with the encoder-decoder architecture, ours, and GRUED have lower deviations than LSTM, which indicates that the encoder-decoder architecture is effective in reducing deviations. It demonstrates the effectiveness of the attention mechanism in that our model has fewer errors when predicting the direction of change compared to GRUED.

To investigate the impact of the prediction sequence length on the prediction accuracy, we fix the history window length to 18, and the prediction length varies from 4 to 24 with a step of 4. Because the ARIMA model and PSR + EA-GMDH model are weak in processing the long historical workload sequence, the experiment only compares the three RNN-based models. Figure 9 shows the overall RMSE, the mean RMSE of the entire prediction sequence, of our model, and the other two RNN-based models as the prediction sequence

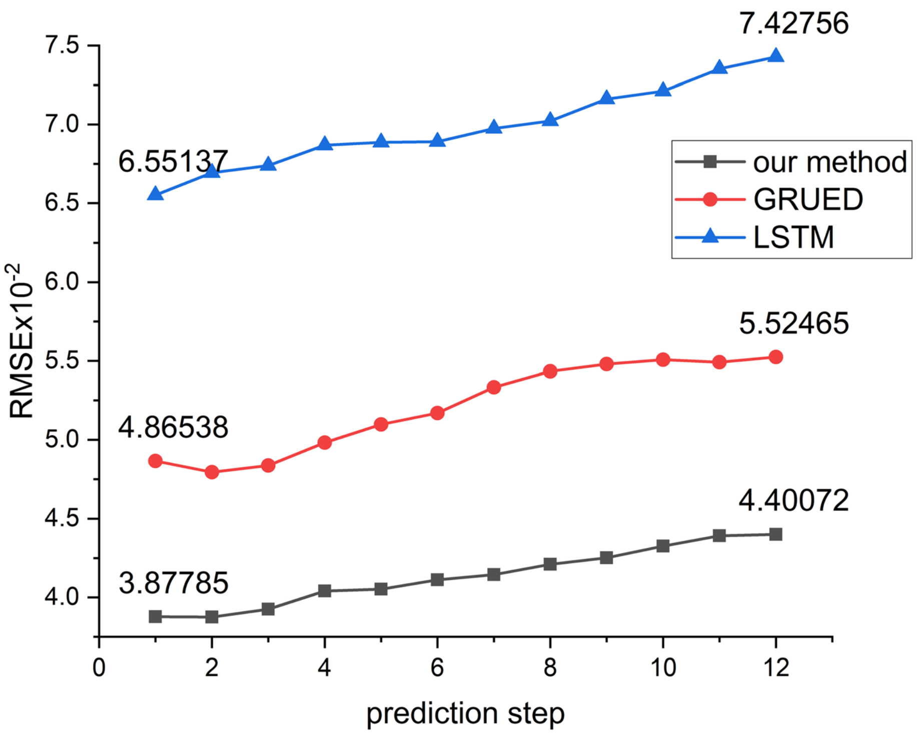

Fig. 8 RMSE of three RNN-based methods at each position in a 12-step prediction sequence. Our method: The black line with square symbols is the RMSE curve of our method at each position in a 12-step prediction sequence. RMSE is calculated over the entire validation set. GRUED: The red line with dot symbols is the RMSE curve of GRUED at each position in a 12-step prediction sequence. RMSE is calculated over the entire validation set. LSTM: The blue line with triangle symbols is the RMSE curve of LSTM at each position in a 12-step prediction sequence. RMSE is calculated over the entire validation set

grows longer from 4 to 24 with an interval of 4, where 24 is 1.5 times the history window length.

From Fig. 10, it is obvious that our method has a flatter growth curve than the other two RNN-based models. Both GRUED and LSTM have the growth elbow at a length of 12, and our method begins to show a clear growth trend at 16. Moreover, the growth trend of our approach is slower than those of the GRUED and LSTM. The RMSE values increase between 4-step prediction and 24-step prediction of our model is $0.58{\times}10^{-2}$, while that of GRUED and LSTM is $0.98{\times}10^{-2}$ and $1.11{\times}10^{-2}$, respectively.

An explanation of the phenomena in Figs. 8, 9, and 10 is that, when the prediction step gets longer, prediction error increases with each step, and the current step prediction amplifies the previous error. When the decoder with attention is decoding, the attention mechanism gives each workload in the historical sequence a weight to help decoding; thus, each prediction value sticks to the history, and the error may not be amplified very

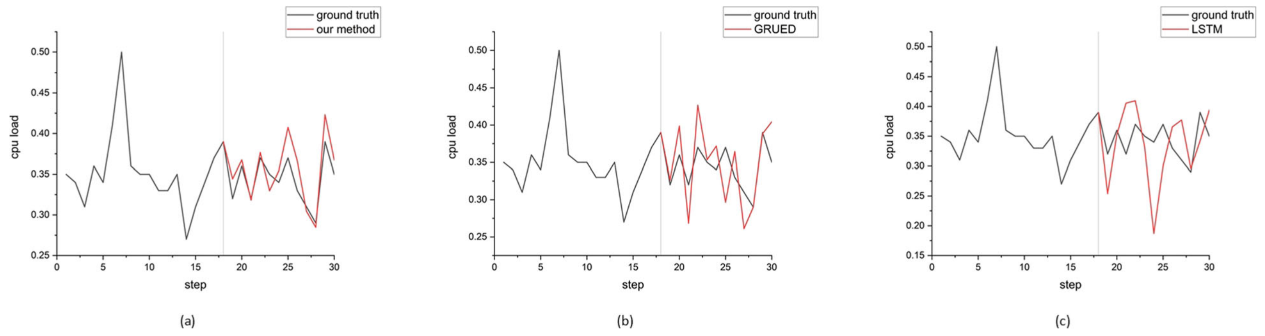

Fig. 9 Workload prediction results on a period of traces over the Alibaba dataset. **a** Our model. The red line in sub-figure (**a**). The segment between 19 and 30 is the predicted workload value from our method. **b** GRUED. The red line in sub-figure (**b**). The segment between 19 and 30 is the predicted workload value from GRUED**. c** LSTM. The red line in sub-figure (**c**). The segment between 19 and 30 is the predicted workload value from LSTM

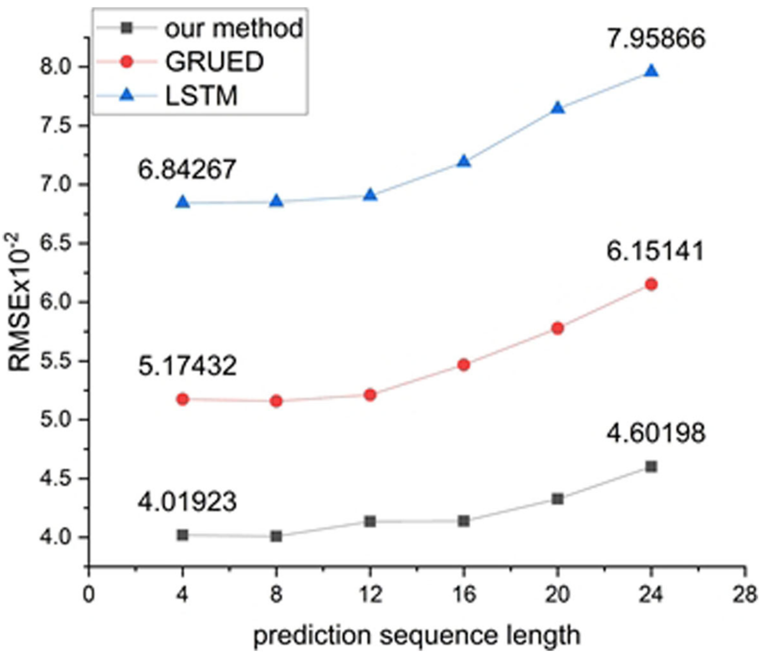

Fig. 10 Overall RMSE of three RNN-based models with different prediction sequence lengths on the Alibaba dataset. Our method: The black line with square symbols is the RMSE when our method is conducted with different prediction sequence lengths. RMSE is calculated over the entire validation set. GRUED: The red line with dot symbols is the RMSE when LSTM is conducted with different prediction sequence lengths. RMSE is calculated over the entire validation set. LSTM: The blue line with triangle symbols is the RMSE when LSTM is conducted with different prediction sequence lengths. RMSE is calculated over the entire validation set

quickly. However, when the prediction step is too long, the prediction value is less relevant to history, and the attention mechanism will fail to maintain the error rate.

4.6 Discussion of history window length and prediction sequence length

In Parameter Setting, we introduced how the parameters of our model are selected in the contrasting experiments. When the prediction sequence length is 12, the historical sequence length is searched among $p \in \{12,18, 24,30,36,42,48\}$ and the history window length with the best performance is 18. The best history window length being 18 does not mean that a longer history window length will increase the error, but a history window length of 18 is long enough in predicting a 12-step future workload. In terms of machine learning theories, a longer historical sequence leads to higher overall model complexity, and then it is easier for the model to get overfitting. Table 4 shows the RMSE of different history window lengths for the training set and the validation set when the prediction sequence length is fixed to 12.

In Table 4, the training set RMSE gradually declines and finally converges at approximately 1.753×10^{-2}; however, the minimum validation set RMSE appears between the 18th step and the 30th step, and the RMSE begins to grow after the 30th step. These statistics show that the well-fitting range of the 12-step prediction model is between 18 and 30, and then the model is overfitting.

To study the correlation of the prediction sequence length and history window length when the model is well-fitted, more cases are explored in Table 5, where a history window length is a well-fitting length when its relative RMSE ratio over the best RMSE is less than 1%. The grid search is conducted between 1 and 2 times the prediction sequence length with interval 2.

Table 4 RMSE of different history window lengths for the training set and validation set when the prediction sequence length is 12

History window length	Training$\times10^{-2}$	Validation$\times10^{-2}$	Relative ratio over the best
12	2.165	4.308	4.21%
18	1.819	*4.134*	0
24	1.786	4.137	0.07%
30	1.792	4.135	0.02%
36	1.753	4.141	0.16%
42	1.757	4.164	0.73%
48	1.751	4.187	1.28%

The RMSE on validation set in italic is the best result

Table 5 The best-fitting history window length of different prediction sequence lengths; minimum well-fitting is the history window length that is the minimum in the well-fitting cases

Prediction sequence length	Best-fitting	Best-fitting RMSE×10^{-2}	Minimum well-fitting
12	18	4.134	18
16	28	4.121	22
20	32	4.179	28
24	40	4.231	34

From Table 5, compared to the RMSEs in Fig. 10, we can see that, when predicting the same steps of the future workload, the model with a longer historical window has a lower error rate. Alternately, the values in the minimum well-fitting column in Table 5 are quite near to 1.5 times the prediction sequence length, which means that predicting multistep workload in the future with our approach only needs a 1.5-times-long historical sequence.

4.7 Trade-offs of the deep model and attention mechanism

In this section, we will discuss the trade-offs of the deep model and attention mechanism. Table 6 shows the workload prediction error of the four models, the single-layer model without attention, our proposed attention-based single-layer model, the deep model without attention, and the attention-based deep model, where the deep model consists of a three-layer LSTM encoder and three-layer LSTM decoder.

The deep models reduce the root mean squared error by 9.3% and 12% in the Alibaba trace dataset, respectively, compared to the single-layer model with and without attention module, which proves the predictive power of the deep model. However, in the experimental result of the Dinda themis dataset, a less complicated workload trace than Alibaba, the performances of the deep model and single-layer model are almost the same. Such phenomenon indicates that the complexity of the deep model exceeds the complexity of the trace prediction task in the conventional distributed cluster. When comparing the models with and without attention, we can find that the attention-based single-layer model and deep model have 17% and 16.3% less error, respectively, for the Alibaba trace dataset than the model without attention, which proves the effectiveness of the attention mechanism in predicting the workload of clusters running both long-term applications and batch workloads. The four models have almost the same prediction accuracy in the Dinda themis dataset, which means that the attention mechanism does not improve the predictive power of the traditional distributed cluster and does not have a negative impact on the prediction.

Table 6 Workload prediction RMSE of four models

Model	Alibaba×10^{-2}	Dinda themis×10^{-2}
Single w/o attention	4.972	2.217
Single	4.134	2.105
Deep w/o attention	4.375	2.073
Deep	3.752	2.081

Though the deep model has proved itself, the extra cost of carrying out a deep model requires discussion. In a three-layer deep model, the calculation is roughly three times that of a single-layer model. Because the recurrent neural network is hard to parallelize and the deep model has difficulty computing each layer in parallel, the deep model takes approximately three times as much time as a single-layer one. Furthermore, a deep model has many more parameters than a single-layer model, which requires more epochs to converge. The convergence curves of the four models are presented in Fig. 11.

From Fig. 11, it is obvious that the models without attention converge faster. The single-layer model without attention converges 1 epoch earlier than the model with attention, and the deep model without attention converges 3 epochs earlier than its attentive peer. Another phenomenon is that the three-layer network requires twice the training of the single-layer network. The single-layer network with attention converges at approximately the 12th epoch during the training, and the three-layer network with attention converges at the 25th epoch, where the three-layer network requires twice the training of the single-layer network. Furthermore, more local minimums, where, in Fig. 11, the adjacent RMSE change over the epoch is tiny, are encountered during convergence of the three layers than for the single-layer network.

4.8 Discussion of scrolling prediction

From the result in Comparison of the Experimental Results and Discussion, we can see that the RMSE increases when the prediction step grows. To reveal the increase of the error rate, scroll prediction is proposed. In scroll prediction, a long prediction sequence is divided into several short sequences, and each sequence is predicted in order, with the former sequence added to the historical sequence. For example, if the prediction sequence is y_1, y_2, ..., y_{12}, we divide it into two sequences, y_1, y_2, ..., y_6 and y_7, y_8, ..., y_{12}. Given the historical sequence x_1, x_2, ..., x_{18}, the first sequence predicted

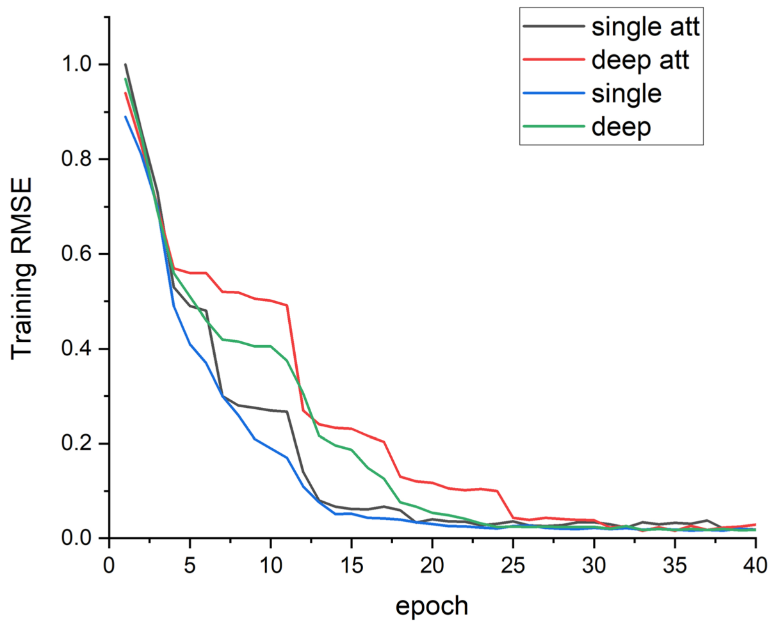

Fig. 11 Loss change in the process of convergence. single att: The black line is the loss curve of the single-layer LSTM encoder-decoder network with attention. deep att: The red line is the loss curve of the three-layer LSTM encoder-decoder network with attention. single: The blue line is the loss curve of the single-layer LSTM encoder-decoder network without attention. deep: The green line is the loss curve of the three-layer LSTM encoder-decoder network without attention

is $\widetilde{y}_1, \widetilde{y}_2, \ldots, \widetilde{y}_6$. Then, the first sequence is added to the historical sequence as $x_7, x_8, \ldots, x_{18}, \widetilde{y}_1, \widetilde{y}_2, \ldots, \widetilde{y}_6$, and the second sequence is predicted with the new historical sequence. Table 7 shows the experiment result of scroll prediction, where 24-step prediction is carried out with 18-step history, and the 36-step result is predicted with a history window length setting of 36.

The experimental result shown in Table 7 demonstrates the effectiveness of the scroll prediction method. Compared with the RMSE of the long 24-step prediction with no scrolling, the scroll prediction with 16 steps and 8 steps decreases the error rate by approximately 1.2%.

Table 7 RMSE of scroll prediction with different prediction modes, where Origin-a means the result of a-step prediction is carried out with no scrolling and Scroll-b&c[&d] means the result is predicted by cutting the long sequence into small sequences with b-step and c-step or b-step, c-step, and d-step

Prediction mode	Overall RMSE×10^{-2}
Origrin-24	4.602
Scroll-16&8	4.549
Scroll-12&12	4.415
Scroll-8&8&8	4.421
Origin-36	5.617
Scroll-24&12	5.453
Scroll-12&12&12	5.231

The scroll prediction with two 16-step and three 8-step predictions reduces the error rate by approximately 4%. The error reduction is more obvious in the longer-term prediction. The scroll with 24-step and 12 step prediction and with three sequence 12-step prediction have approximately 3% and 6.9% lower RMSE values, respectively.

To get a more intuitive view of improvements of the scroll prediction method, we put the prediction workload curve and the ground truth workload together in a graph, as shown in Fig. 12.

From Fig. 12, we can see that, in the first 12-step prediction segment, the prediction workload curves of all three modes are close to the ground truth workload. The trend extends to the next segment in (b) Scroll-24&12 and (a) Scroll-12&12&12, and (c) Origin mode starts to loss accuracy. In the last 12-step segment, it is obvious that (a) Scroll-12&12&12 outperforms the other two prediction modes.

5 Conclusion and future work

In this paper, we propose a novel approach for workload prediction. The LSTM encoder is used to extract the hidden features of the historical sequence and predict the workload. Then, the attention mechanism is applied to the decoder network to enhance the

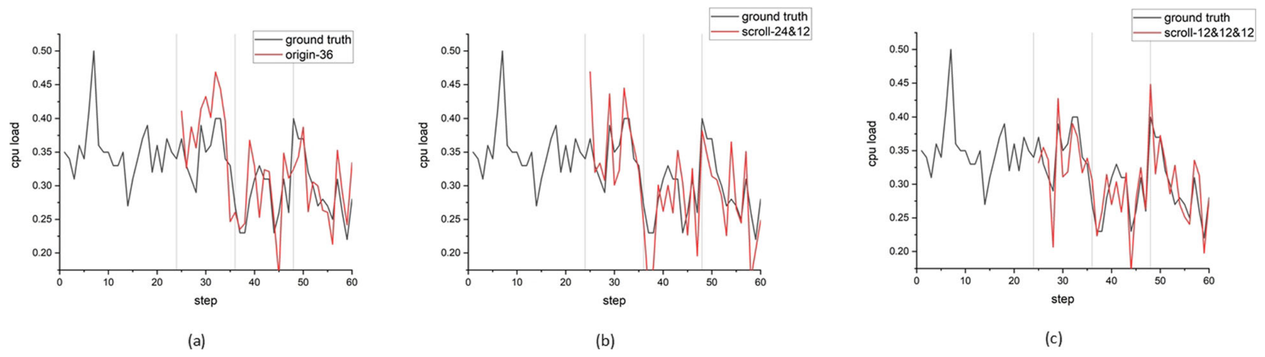

Fig. 12 Workload prediction results with/without the scroll prediction method. **a** Origin model for 36-step prediction. origin-36: The red line in sub-figure (**a**). The origin-36 line shows the predicted values of the origin model for 36-step prediction not using scroll prediction. scroll-24&12: The red line in sub-figure. **b** Scroll on two sequences with 24-step and 12-step prediction, respectively. **b** The scroll-24&12 line shows the predicted values obtained via scroll prediction by splitting the prediction sequence with a length of 12 into two sequences with lengths of 24 and 12. **c** Scroll on three sequences with 12-step prediction. ground truth: The black line in all three sub figures. Ground truth is the real work-load in the dataset. Scroll-12&12&12: The red line in sub-figure (**c**). The scroll-12&12&12 line shows the predicted values obtained via scroll prediction by splitting the prediction se-quence with a length of 12 into three sequences with lengths of 12

model's batch workload prediction ability. The proposed model has been evaluated in both a traditional distributed cluster environment and mixed cloud environment, and the experimental results demonstrate that our model achieves state-of-the-art performance. Moreover, we also propose a scroll prediction method to reduce the error occurring during long-term prediction, which splits a long-term prediction task into several small tasks. This approach can be used to relieve the problem of superimposed errors so that they may be amplified

In future work, we will study the use of batch DAGs to support the model for batch workload predictions, through which we would like to see more effective task scheduling. Moreover, the accuracy of the long-term forecast will be quantitatively verified by using probabilistic model checking considering the factors of nondeterminism and time constraints.

Abbreviations

AR: Auto-regressive model; ARIMA: Auto-regressive integrated moving average model; GRUED: Gated recurrent unit encoder-decoder network; k-NN: k nearest neighbor model; LSTM: Long short-term memory; MA: Moving average model; MAE: Mean absolute error; MAPE: Mean absolute percentage error; PReLU: Parametric rectifier linear unit; PSR + EA-GMDH: Phase space reconstruction method combines group method of data handling based on evolutionary algorithm; RMSE: Root mean squared error; RMSSE: Root mean segment squared error; RNN: Recurrent neural network

Acknowledgements

We would like to thank the authors of the literature cited in this paper for contributing useful ideas to this study.

Authors' contributions

Zhang came up with the initial concept of predicting workload with attention-based LSTM encoder-decoder network. Zhu and Zhang both designed the system prototype and implemented the experiments. They wrote the majority of the paper. Gao participated in the system design process, provided feedback, and proposed the scroll prediction method. Chen supported the experimental design process and provided great help in writing. All authors read and approved the final manuscript.

Funding

This work is supported by the National Key Research and Development Plan of China under Grant No. 2017YFD0400101, the Natural Science Foundation of China under Grant No. 61902236, and the Natural Science Foundation of Shanghai under Grant No. 16ZR1411200.

Availability of data and materials

The datasets of all of our measurements analyzed in this study are available in the following repositories:
1. https://github.com/alibaba/clusterdata
2. http://www.cs.cmu.edu/~pdinda/LoadTraces/

Competing interest

The authors declare that they have no competing interests.

Author details

[1]School of Computer Engineering and Science, Shanghai University, Shanghai 200444, China. [2]Shanghai Film Academy, Shanghai University, Shanghai 200072, China. [3]Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai, China. [4]Computing Center, Shanghai University, Shanghai 200444, China. [5]Shanghai ShangDa HaiRun Information System Ltd, Shanghai 200444, China.

Received: 7 August 2019 Accepted: 15 November 2019
Published online: 17 December 2019

References

1. Josep AD, Katz RA, Konwinski A, Lee G, Patterson D, Rabkin A. A view of cloud computing. Commun ACM. 2010;53.
2. Q. Zhang, L. Cheng, R. Boutaba, Cloud computing: state-of-the-art and research challenges. J Internet Serv Appl. **1**, 7–18 (2010)
3. Rajan K, Kakadia D, Curino C, Krishnan S. PerfOrator: eloquent performance models for resource optimization. In: Proceedings of the Seventh ACM Symposium on Cloud Computing. ACM; 2016. p. 415–27.
4. Lianyong Qi, Jiguo Yu, Zhili Zhou. An invocation cost optimization method for web services in cloud environment. Scientific Programming, Volume 2017, Article ID 4358536, 9 pages, 2017.
5. L. Yang, I.T. Foster, J.M. Schopf, in *international parallel and distributed processing symposium*. Homeostatic and tendency-based CPU load predictions (2003), p. 42
6. P.A. Dinda, Design, implementation, and performance of an extensible toolkit for resource prediction in distributed systems. IEEE Trans Parallel Distrib Syst **17**, 160–173 (2006)

7. Lianyong Qi, Xiaolong Xu, Wanchun Dou, Jiguo Yu, Zhili Zhou, Xuyun Zhang. Time-aware IoE service recommendation on sparse data, Mobile Information Systems, Volume 2016, Article ID 4397061, 12 pages, 2016.
8. 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. IEEE Computer Society Press; 2012. p. 21.
9. F. Benhammadi, Z. Gessoum, A. Mokhtari, CPU load prediction using neuro-fuzzy and Bayesian inferences. Neurocomputing **74**, 1606–1616 (2011)
10. Liang J, Cao J, Wang J, Xu Y. Long-term CPU load prediction. In: 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing. IEEE; 2011. p. 23–6.
11. Q. Yang, Y. Zhou, Y. Yu, J. Yuan, X. Xing, S. Du, Multi-step-ahead host load prediction using autoencoder and echo state networks in cloud computing. J Supercomput **71**, 3037–3053 (2015)
12. B. Song, Y. Yu, Y. Zhou, Z. Wang, S. Du, Host load prediction with long short-term memory in cloud computing. J Supercomput **74**, 6554–6568 (2018)
13. 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). IEEE. p. 186–91.
14. Di S, Kondo D, Cappello F. Characterizing cloud applications on a Google data center. In: 2013 42nd International Conference on Parallel Processing. IEEE; 2013. p. 468–73.
15. A.K. Mishra, J.L. Hellerstein, W. Cirne, C.R. Das, Towards characterizing cloud backend workloads: insights from Google compute clusters. ACM SIGMETRICS Perform Eval Rev. **37**, 34–41 (2010)
16. Chung J, Gulcehre C, Cho K, Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv Prepr arXiv14123555. 2014.
17. Sutskever I, Vinyals O, Le Q V. Sequence to sequence learning with neural networks. In: Advances in neural information processing systems. 2014. p. 3104–12.
18. Akioka S, Muraoka Y. Extended forecast of CPU and network load on computational grid. In: IEEE International Symposium on Cluster Computing and the Grid, 2004. CCGrid 2004. IEEE; 2004. p. 765–72.
19. L. Qi, R. Wang, S. Li, Q. He, X. Xu, C. Hu, Time-aware distributed service recommendation with privacy-preservation. Inform Sci **480**, 354–364 (2019)
20. Yang D, Cao J, Yu C, Xiao J. A multi-step-ahead CPU load prediction approach in distributed system. In: 2012 Second International Conference on Cloud and Green Computing. IEEE; 2012. p. 206–13.
21. L. Qi, P. Dai, J. Yu, Z. Zhou, Y. Xu, "Time-Location-Frequency"-aware Internet of things service selection based on historical records. Int J Distributed Sensor Netw **13**(1), 1–9 (2017)
22. M. Han, J. Xi, S. Xu, F.-L. Yin, Prediction of chaotic time series based on the recurrent predictor neural network. IEEE Trans Signal Process **52**, 3409–3416 (2004)
23. Wu Y, Yuan Y, Yang G, Zheng W. Load prediction using hybrid model for computational grid. In: 2007 8th IEEE/ACM International Conference on Grid Computing. IEEE; 2007. p. 235–42.
24. P. Singh, P. Gupta, K. Jyoti, Tasm: technocrat arima and svr model for workload prediction of web applications in cloud. Cluster Comput **22**, 619–633 (2019)
25. Dabrowski C, Hunt F. Using Markov chain analysis to study dynamic behaviour in large-scale grid systems. In: Proceedings of the Seventh Australasian Symposium on Grid Computing and e-Research-Volume 99. Australian Computer Society, Inc.; 2009. p. 29–40.
26. Huang J, Li C, Yu J. Resource prediction based on double exponential smoothing in cloud computing. In: 2012 2nd International Conference on Consumer Electronics, Communications and Networks (CECNet). IEEE; 2012. p. 2056–60.
27. N.J. Kansal, I. Chana, Energy-aware virtual machine migration for cloud computing-a firefly optimization approach. J Grid Comput. **14**, 327–345 (2016)
28. Wang X, Huang S, Fu S, Kavi K. Characterizing workload of web applications on virtualized servers. In: Workshop on Big Data Benchmarks, Performance Optimization, and Emerging Hardware. Springer; 2014. p. 98–108.
29. J. Cao, J. Fu, M. Li, J. Chen, CPU load prediction for cloud environment based on a dynamic ensemble model. Softw Pract Exp. **44**, 793–804 (2014)
30. R.N. Calheiros, E. Masoumi, R. Ranjan, R. Buyya, Workload prediction using ARIMA model and its impact on cloud applications' QoS. IEEE Trans Cloud Comput **3**, 449–458 (2014)
31. J. Kumar, A.K. Singh, Workload prediction in cloud using artificial neural network and adaptive differential evolution. Futur Gener Comput Syst **81**, 41–52 (2018)
32. B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, T. Wood, Agile dynamic provisioning of multi-tier internet applications. ACM Trans Auton Adapt Syst **3**, 1 (2008)
33. Lu C, Ye K, Xu G, Xu C-Z, Bai T. Imbalance in the cloud: an analysis on alibaba cluster trace. In: 2017 IEEE International Conference on Big Data (Big Data). IEEE; 2017. p. 2884–92.
34. Abdul-Rahman OA, Aida K. Towards understanding the usage behavior of Google cloud users: the mice and elephants phenomenon. In: 2014 IEEE 6th International Conference on Cloud Computing Technology and Science. IEEE; 2014. p. 272–7.
35. Verma A, Pedrosa L, Korupolu M, Oppenheimer D, Tune E, Wilkes J. Large-scale cluster management at Google with Borg. In: Proceedings of the Tenth European Conference on Computer Systems. ACM; 2015. p. 18.
36. Z. Huang, J. Peng, H. Lian, J. Guo, W. Qiu, Deep recurrent model for server load and performance prediction in data center. Complexity **2017** (2017)
37. J. Kumar, R. Goomer, A.K. Singh, Long short term memory recurrent neural network (lstm-rnn) based workload forecasting model for cloud datacenters. Procedia Comput Sci **125**, 676–682 (2018)
38. Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult. IEEE Trans Neural Netw **5**, 157–166 (1994)
39. S. Hochreiter, J. Schmidhuber, Long short-term memory. Neural Comput. **9**, 1735–1780 (1997)
40. Bahdanau D, Cho K, Bengio Y. Neural machine translation by jointly learning to align and translate. arXiv Prepr arXiv14090473. 2014.
41. Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv Prepr arXiv14061078. 2014.
42. Luong M-T, Pham H, Manning CD. Effective approaches to attention-based neural machine translation. arXiv Prepr arXiv150804025. 2015.
43. He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. 2015. p. 1026–34.
44. Huber PJ. Robust estimation of a location parameter. In: Breakthroughs in statistics. Springer; 1992. p. 492–518.
45. Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv Prepr arXiv14126980. 2014.
46. K. Cho, A. Courville, Y. Bengio, Describing multimedia content using attention-based encoder-decoder networks. IEEE Trans Multimed **17**, 1875–1886 (2015)
47. Q. Yang, C. Peng, H. Zhao, Y. Yu, Y. Zhou, Z. Wang, et al., A new method based on PSR and EA-GMDH for host load prediction in cloud computing system. J Supercomput. **68**, 1402–1417 (2014)

Publisher's Note