J. Parallel Distrib. Comput. 119 (2018) 162–171

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

# A data-driven approach of performance evaluation for cache server groups in content delivery network

Ziyan Wu [a], Zhihui Lu [a,*], Wei Zhang [a], Jie Wu [a], Shalin Huang [b], Patrick C.K. Hung [c]

[a] School of Computer Science, Fudan University, Shanghai 200433, China
[b] Wangsu Science & Technology Co., Ltd., Shanghai, China
[c] Faculty of Business and IT, University of Ontario Institute of Technology, Canada

## HIGHLIGHTS

- We frame CDN performance evaluation problem as a sequence learning problem.
- We use representation learning by LSTM auto-encoder to extract useful features from CDN monitoring log data.
- We use a deep neural network to predict the reach rate of CDN service, and we compare our methods with state-of-arts methods which show ours is superior by empirical studies.

## ARTICLE INFO

*Article history:*
Received 31 January 2018
Received in revised form 4 April 2018
Accepted 16 April 2018
Available online 27 April 2018

*Keywords:*
Edge computing
Deep learning
Content delivery network
Sequence learning
Predictive analysis
High dimensional data

## ABSTRACT

In industry, Content Delivery Network (CDN) service providers are increasingly using data-driven mechanisms to build the performance models of the service-providing systems. Building a model to accurately describe the performance of the existing infrastructure is very crucial to make resource management decisions. Conventional approaches that use hand-tuned parameters or linear models have their drawbacks. Recently, data-driven paradigm has been shown to greatly outperform traditional methods in modeling complex systems. We design a data-driven approach to building a reasonable and feasible performance model for CDN cache server groups. We use deep LSTM auto-encoder to capture the temporal structures from the high-dimensional monitoring data, and use a deep neural network to predict the reach rate which is a client QoS measurement from the CDN service providers' perspective. The experimental results have shown that our model is able to outperform state-of-the-art models.



## 1. Introduction

There is a trend [7,11,13,18,20] that both academia and industry use data-driven methods to model complex networked systems. Traditional approaches typically use some simple heuristics. These methods have several drawbacks. They cannot accurately reflect the complex systems due to the lack of knowledge of the real-world environment. Driven by the opportunity to collect and analyze data (e.g., application quality measurement from end users), many recent proposals have demonstrated the promise of using deep learning to characterize and optimize networked systems. Drawing parallel from the success of deep-learning on pattern recognition, instead of using an empirical analytical model to describe the complex interaction of different features, we use deep learning methods and treat networked systems as a black-box.

Uploading all data or deploying all applications to a centralized cloud is infeasible because of the excessive latency and bandwidth limitation of the Internet. A promising approach to addressing centralized cloud bottleneck is edge computing. Edge computing pushes applications, data and computing power (services) away from centralized points to the logical extremes of a network. Edge computing replicates fragments of information across distributed networks of web servers, which may spread over a vast area. As a technological paradigm, edge computing is also referred to as mesh computing, peer-to-peer computing, autonomic (self-healing) computing, grid computing, and by other names implying non-centralized, nodeless availability [5]. CDN (content delivery network or content distribution network) is a typical representative of edge computing. A CDN is a globally distributed networked system deployed across the edge of Internet. Composed with geographically distributed cache servers, CDNs deliver cached content to customers worldwide based on their geographic locations. Extensively using cache servers, content delivery over CDN has

* Corresponding author.
*E-mail address:* lzh@fudan.edu.cn (Z. Lu).

https://doi.org/10.1016/j.jpdc.2018.04.010
0743-7315/

low latency, reliability, supports better quality of experience and security.

The CDN Service providers are increasingly using data-driven mechanisms to build the performance model of their service-providing systems. To build a model to accurately provide an understanding of the performance of the existing infrastructure such as the health of cache groups and network status, is very crucial to make resource management decisions including content placement, network traffic scheduling, and load balance of the CDN network. Modeling all available physical resources, we can maximize a resource utilization in terms of service quality, cost, profit, etc.

Generally, CDN providers do not have direct measurement from the clients (the logs from video players, web browser that can show the QoE of clients). Instead, they use the indirect measurement reach rate, the ratio of requests that meet the minimum standard, which is collected and calculated offline from the log of the HA proxy of CDN cache groups. In order to enable themselves make resource management decisions in real time, the CDN providers have to use the metrics that can be collected in the real time to infer the reach rate.

Cache server groups can be characterized as multi-dimensional, highly non-linear, time variant vectors. The metrics collected from members of the CDN cache server groups are sequence data that are measured every minute, which have hundreds of dimensions. The state-of-art methods are typically simple heuristics which are oversimplified and biased due to the human experience, or linear models, which cannot characterize the complex relationship between multiple metrics. Deep learning is a branch of machine learning based on a set of algorithms and attempt to model high-level abstractions in data by using artificial neural network architectures composed of multiple non-linear transformations [17]. They have a lot of successful applications in sequence modeling [15]. Compared to other machine learning techniques, a lot of work shows that it can detect complex relationships among features, and extract a hierarchical level of features from high-dimensional data, including monitoring data.

We frame our problem as a sequence learning problem, which consists of three stages: (1) feature engineering, (2) representation learning by LSTM auto-encoder to extract useful features, (3) a feed forward neural network black-box machine learning algorithm to output the predictions.

Our main contributions are listed below:

- We present a data-driven feasible approach to evaluating the performance of cache server groups in Content Delivery Network.
- We frame performance evaluation problem as a sequence learning problem.
- We use representation learning by LSTM auto-encoder to extract useful features from data.
- We compare our methods with state-of-arts methods to show that ours is superior by empirical studies.

The remainder of this paper is organized as follows. In Section 2, we first introduce the related concepts as our research background. In Section 3, we formulate our performance evaluation problem as a sequence learning problem and we also compare the baseline methods. In Section 4, we introduce our method of feature engineering to reduce the dimensionality for the high-dimensional data. We also introduce our reach rate prediction model based on LSTM auto-encoder. In Section 5, we show our experiment setting and demonstrate performance improvements of our methods over baseline models. Section 6 is discussion for related work. We provide concluding remarks in Section 7.

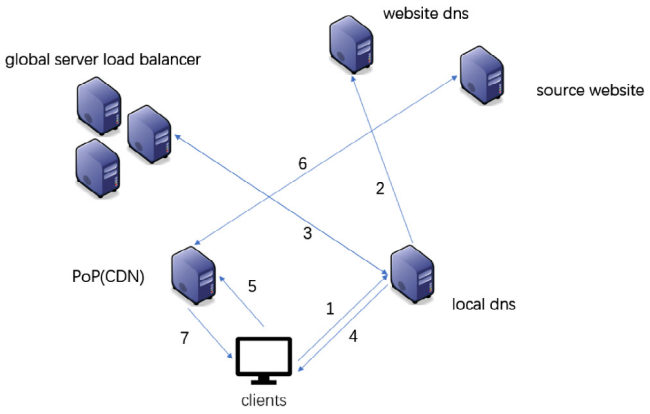



**Fig. 1.** The basic working procedure of CDN.

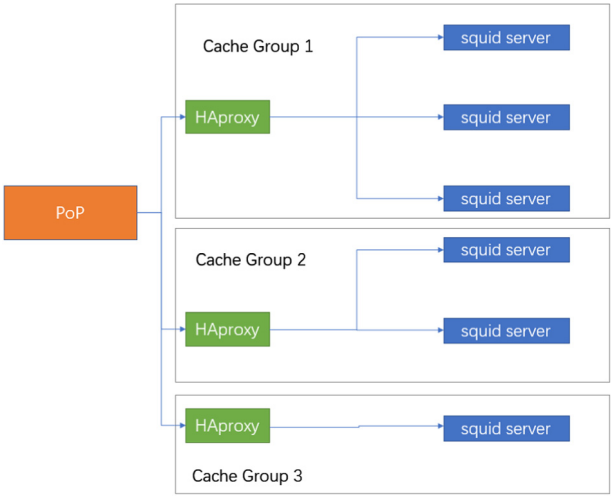



**Fig. 2.** The structure of cache server groups.

## 2. Background

A content delivery network or content distribution network (CDN) (Fig. 1) is a geographically distributed network of cache servers. CDN helps content provider to deliver web pages and other multimedia content to the clients, based on the locations of the clients and cache servers nearby the clients. The basic working procedure is as follows. Step 1: a client sends a request to local DNS. Step 2: the DNS finds the CNAME and redirects the request to the GSLB (global server load balance). Step 3: the local DNS server sends a request to the GSLB and GSLB returns the ip address of CDN servers based on the scheduling policy. Step 4: the local DNS returns the ip address to the clients. Step 5: clients request to fetch content from the selected PoP. Step 6: the cache server groups will pull the content from source website if the content does not exist locally. Step 7: content is sent to the clients.

A CDN cache server group (Fig. 2) is the basic resource scheduling unit for CDN. A CDN cache group is a load balanced cluster that consists of interconnected cache servers. A typical implementation consists of HAproxy and squid servers. HAproxy distributes the requests from clients to cache servers. HAproxy can be set to use different algorithms to maximize the utilization of every server. Round-robin algorithm distributes the load equally to each server in a homogeneous cluster. In a heterogeneous cluster, weighted round-robin algorithm is used. A weight was assigned to the server

**Table 1**
The feature list.

| Feature list | Meaning |
|---|---|
| Group 1 | |
| cpu1.usage | The utilization of cpu 1 |
| cpu2.usage | The utilization of cpu 2 |
| cpu3.usage | The utilization of cpu 3 |
| cpu4.usage | The utilization of cpu 4 |
| cpu5.usage | The utilization of cpu 5 |
| cpu6.usage | The utilization of cpu 6 |
| … | |
| mem_cached | The size of memory cached |
| mem_buffers_cache_free | The size of memory buffer |
| memory.swap | The size of memory swap |
| disk.used.sda1 | The size of disk in sda1 |
| disk.used.sda2 | The size of disk in sda2 |
| disk.used.sda3 | The size of disk in sda3 |
| … | … |
| channeltraffic_in | Channel traffic flowing in |
| channeltraffic_out | Channel traffic flowing out |
| ioutil_util_sda | IO utility of sda |
| ioutil_util_sdb | IO utility of sdb |
| ioutil_util_sdc | IO utility of sdc |
| … | … |
| iowait.wait | Unfinished I/O requests when cpu is idle |
| hitratio.port8101 | Hit ratio of port 8101 |
| hitratio.port81021 | Hit ratio of port 8101 |
| … | … |
| resptime.resp.resp8105 | Response time of 8105 |
| resptime.resp.resp80 | Response time of 80 |
| resptime.resp.resp8101 | Response time of 8101 |
| … | … |
| Group 2 | |
| aggregate.cpu | The sum of cpu usage of a single machine |
| aggregate.diskused | The sum of dist usage of a single machine |
| aggregated.ioutil | The sum of IO utility of a single machine |
| aggregate.resptime | The average of response time of a single machine |
| … | … |
| Group 3 | |
| all_machine.cpu | The sum of cpu usage of all machines |
| all_machine.diskused | The sum of disk usage of all machines |
| all_machine.ioutil | The sum of IO utility of all machines |
| … | … |

based on its processing capabilities. A heterogeneous structure of a cluster adds complexity to the feature engineering.

As CDN providers do not have direct QoS measurement from the clients (the logs from video players, web browsers that can show the QoE), they use the measurement reach rate. They calculated reach rate from the log of the HA proxy of CDN cache groups with a delay about three minutes.

The number of metrics we collected from the cache servers of cache group is different due to the different configuration, from 64 to 134. We listed the features we constructed in Table 1. There are about 312 features. They include CPU utilization, network utilization, disk utilization, memory utilization and so on. We organize the features into 3 groups. The features in the first group are from the raw data we directly collected from the caching servers. The features of the second group are what we construct based on statistics of the metrics of a single cache server. The features of the third group are what we construct based on the statistics of the metrics of the whole cache server groups.

## 3. Problem formulation and models comparison

We argue that performance evaluation should be formulated as a sequence learning problem. Since we can collect the caching servers' performance metrics and network metrics at time intervals of one minute, we can use a sequence model to describe the relationship between metrics collected from cache groups and reach rate.

There are three categories of sequence learning problem, which are many to many, one to many and many to one. Our goal is to model the relationship between a sequence of collected metrics and reach rate within a certain period of time, which is many to one. In general, we can use the following formulation to describe the prediction process.

Given a sequence of vectors, $\{x_n\} = \{x_\alpha \in R^d | \alpha \in N\}$, where $d$ is the number of the features, we use $\{x_n\}$ to represent the sequence and $x_t$ to describe a data point at time t with $d$ dimensions.

Given another target sequence, $\{y_n\} = \{y_\alpha \in R | \alpha \in N\}$, our goal is to find the relation between $\{x_n\}$ and $\{y_n\}$, which is

$$y_t = f(x_t, x_{t-1}, \ldots, x_{t-n+1})$$

where $n$ is the window size and $f$ is the mapping we want to learn from the data.

Many models can be used to approximate $f$ in sequence modeling. The most naive way is to use simple heuristics, which is to use an exponential moving average to linearly map each metrics to a score in a certain time interval. The parameters depend on the experience of the operators. This method is impractical: it can hardly generalize well and requires tedious repetitive tuning.

### 3.1. Linear models for sequence learning

There are some conventional approaches which use data to learn, such as the multiple linear regression model (MLR). The MLR method builds a model of a sequence that is composed of a linear part and a random noise part. The linear part models the linear relationship between the dependent variables and independent variables, and the random noise reflects the unpredictable randomness. Formally, the model for multiple linear regression, given n observations, is

$$y_t = c + \sum_{i=t-n+1}^{t} (\beta_i x_{t_i}) + \epsilon_t.$$

Furthermore, the linear part incorporates historical values of the sequence. $y$ represents the target we want to model. $c$ represents the constant parameter of the linear decomposition. $\beta$ represents the parameters to be computed and $\epsilon$ reflects the random noise part. The best-fitting line for the observed data is calculated by the least square method.

Linear models are easy to implement and have good interpretation and thus are widely used in many scenarios. However, linear models are shown not sufficient to describe some nonlinear behaviors of the complex network systems. In many cases, neural networks tend to outperform linear models. In our experiments, we observe a non-linear relationship between the metrics and reach rate.

### 3.2. Non-linear for sequence learning

Deep learning has many applications in sequence learning. Deep learning is a branch of machine learning based on a set of algorithms that attempts to model high-level abstractions in data by using artificial neural network (ANN) architectures composed of multiple non-linear transformations. Compared to other machine learning techniques, it can detect complex relationships among features, can extract a hierarchical level of features from raw data. Thus it can build a model more accurate with less time.

A generic three-layered neural network is illustrated in Fig. 3. In this study, the input matrix $I \in R^{M \times (N \times T+1)}$ where $M$ is the number of training examples and $N \times T + 1$ is the number of features (metrics that collected from the cache servers) concatenating the bias term. The input matrix is then multiplied by the model parameters matrix $W_1$ to produce the hidden state matrix $h$. The output of the

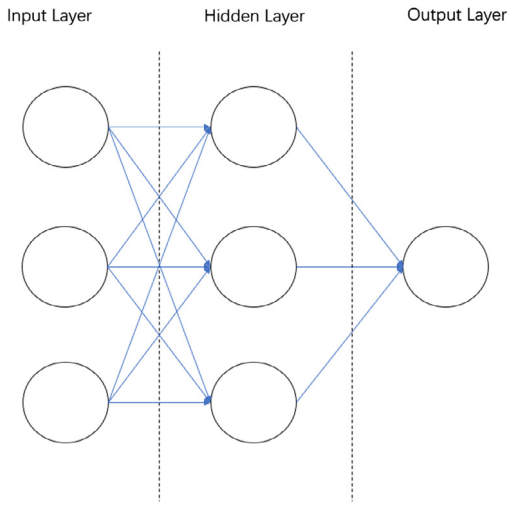



**Fig. 3.** A generic three-layered neural network.

first layer is transformed by an activation function. We can add more layers to the network. The size and number of hidden layers can be varied to model systems of varying complexity. The whole process can be imagined as the information propagating forward. We can formalize the forward propagation as follows:

### 3.2.1. Feed forward neural network

$$\begin{aligned}\mathbf{h}_1 &= \text{sigmoid}(\mathbf{W}_1 * \mathbf{I})\\ \mathbf{h}_t &= \text{sigmoid}(\mathbf{W}_{t-1} * \mathbf{h}_{t-1})\\ \mathbf{O} &= \text{sigmoid}(\mathbf{W} * \mathbf{h}_{lastlayer})\end{aligned} \tag{1}$$

where sigmoid is the activation function (see Fig. 4):

$$\text{sigmoid}(x) = \frac{1}{1+e^{-x}} \tag{2}$$

the loss function $J$ uses mean square error:

$$J = \frac{1}{n} \times \sum_{t=1}^{N} (y'_t - y_t)^2.$$

The process of training any neural network model can be broken down into 5 steps: (1) Randomly initialize the model parameters. (2) Forward propagate using the forward propagation algorithm. (3) Compute the cost function $J$. (4) Back propagate using the chain rule. (5) Use the gradients calculated and adjust weights to minimize the cost function $J$.

One major assumption for NNs is the independence of samples. For sequence learning problem, however, this assumption does not hold true, since the samples of our problem have a temporal relationship: the status of the system of the next timestep depends not only on the status in the current timestep but also on the previous timesteps of indefinite length. One solution is to use a sliding window to capture the sequential relationship between the samples. The performance of this method depends on the window size, which is not practical since the dependencies length is not a fixed value. RNN eliminates the need to find the size of the window [9].

### 3.2.2. RNN

RNNs are a class of neural networks that depend on the sequential nature of their inputs. Such inputs could be text, speech, time series, and anything else in which the occurrence of an element in

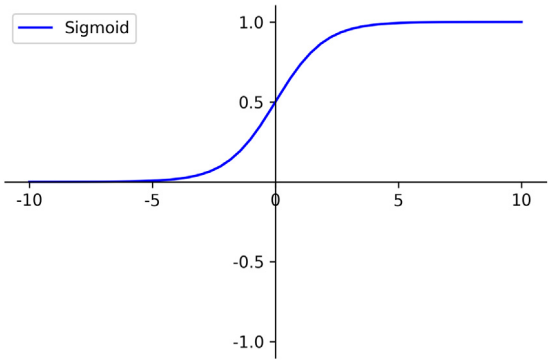



**Fig. 4.** Sigmoid function as activation function.

the sequence is dependent on the elements that appeared before it. The promise of recurrent neural networks is that the temporal dependence and contextual information in the input data can be learned [2,4].

RNNs process the input sequence one element at a time and maintain a hidden state vector which acts as a memory for past information. They learn to selectively retain relevant information allowing them to capture dependencies across several time steps. This allows them to utilize both current input and past information while making future predictions. All this is learned by the model automatically without much knowledge of the cycles or time dependencies in data. RNNs obviate the need for a fixed size time window and can also handle variable length sequences. Moreover, the number of states that can be represented by an NN is exponential in the number of nodes (see Fig. 5).

RNNs maintain a hidden vector $\mathbf{h}$, which is updated at time step $t$ as follows:

$$\mathbf{h}_t = \tanh(\mathbf{W} * \mathbf{h}_{t-1} + \mathbf{I} * x_t) \tag{3}$$

where tanh is the hyperbolic tangent function (see Fig. 8), $\mathbf{W}$ is the recurrent weight matrix and $I$ is an input wight matrix. The hidden state $\mathbf{h}_t$ and weight matrix $W'$ are then used to make a prediction

$$\mathbf{y}_t = f(\mathbf{W}' * \mathbf{h}_t) \tag{4}$$

where $f$ can be a fully connected layer that linearly maps the hidden state to an output. By using $\mathbf{h}$ as the input to another RNN, we can stack RNNs, creating deeper architectures

$$\mathbf{h}_t^l = \sigma(\mathbf{W} * \mathbf{h}_{t-1}^l + \mathbf{I} * \mathbf{h}_t^{l-1}). \tag{5}$$

The training of RNNs uses back-propagation through time (BPTT). RNNs have several drawbacks: 1. Training vanilla RNNs is known to be particularly difficult, with vanishing and exploding gradients being one possible explanation [2]. 2. RNNs are not capable of learning long-term dependencies. These issues are solved by introducing LSTM cell [10].

## 3.3. Models comparisons

In sum, non-linear models have better accuracy than linear models in non-linear problems. RNN-based methods such as RNN and our LSTM-based methods proposed in the next section can capture temporal structures in the data. Simple heuristics and MLR cannot describe the interdependency of the features of data by their natures. It is hard to decide the parameters of models by experience in simple heuristics methods. Non-linear models take more time to train compared to linear models, and RNN is hard to train because of the vanishing gradients problem. So we choose a LSTM-based solution for our problem. The detailed comparison is listed in Table 2.

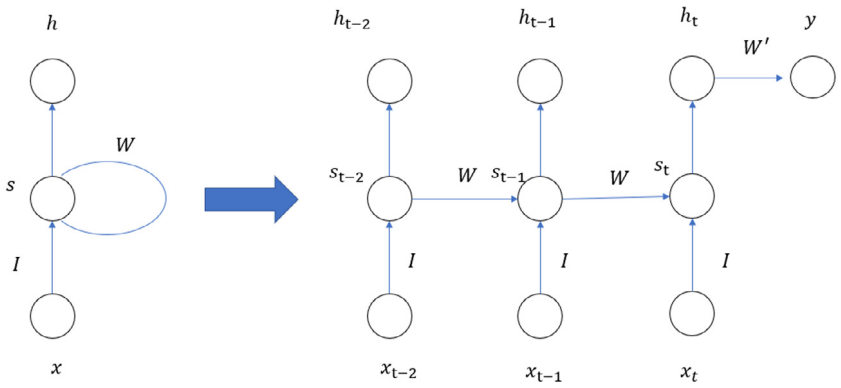



**Fig. 5.** The left shows the backloop structure of RNN and right shows that RNN can be thought as infinite deep layers neural network unfolded along the dimension of time.

**Table 2**
Models comparison.

| | Simple heuristics | MLR | NN | RNN | LSTM | LSTM auto-encoder |
|---|---|---|---|---|---|---|
| Accuracy | * | ** | *** | **** | ***** | ****** |
| Speed of predictions | *** | *** | ** | ** | ** | * |
| Tolerance to redundant features | * | ** | *** | *** | *** | ***** |
| Attributes independence | * | ** | *** | *** | *** | *** |
| Speed of training | – | ** | *** | *** | * | ** |
| Model parameters handling | *** | * | * | * | * | ** |
| Temporal structures | No | No | No | Yes | Yes | Yes |

## 4. Methods

### 4.1. Feature engineering

The feature engineering is the process after data-cleansing, in which we fill the missing data and reformat it. The purpose of this stage is two-fold: (1) find a unified equal-length vector representation of all of the cache groups. The metrics collected are in the granularity of caching servers which have different dimensionality. To make things more complex, a cache group may have different number of caching servers. (2) reduce the dimensionality, for it is very time-consuming to train the models when the number of dimensionality is too high.

Our feature engineering has three steps: 1. Data prepossessing which converts raw data into high-dimensional vectors. 2. Feature correlation analysis that characterizes the linear relationship between every pair of features. 3. Cluster analysis that selects the set of features (see Fig. 6).

As there are hundreds of features, there are many overlaps among the variables. We use correlation in statistics to group highly correlated variables and create composite features that can represent each group of variables. Correlation is an analysis of two or more observed or random variables to determine dependence between the variables. This dependence can be classified as the probability that changes in one variable affect the behavior of the second variable. Pearson's correlation defines this dependence in the interval between $-1$ and 1. Pearson's correlation for two given random variables $X$ and $Y$ are computed by dividing the covariance of both variables with the product of their standard deviations.

Generally, cases of high correlation compute to a value close to 1.0. High anticorrelation is associated with a value close to $-1.0$ and no correlation is assumed. If the value is around 0.0, the variables appear to be linearly independent.

After we calculate the correlation matrix for all the features, the correlations of these features are regarded as the distance. We aim to choose the representative features to represent the data. Then, we use DBSCAN [6] to cluster these features to eliminate closed ones. DBSCAN is a density-based clustering algorithm. Compared to other algorithms like $K$-means, we do not have to specify the number of features and can find any cluster of any shape. By defining the neighborhood parameter $\epsilon$, we specify how much extent we regard two features as close. If a feature is highly correlated with another one, we should add these features into the same cluster. When the clustering is done, we choose the representative features from the cluster. By clustering, we eliminate the redundant information that exist in the data and accelerate the process of training of models. We formulate the process in Algorithm 1.

**Algorithm 1** Feature Clustering and Selection

**Input:** All features list: $F$; Neighborhood parameter: $\epsilon$;
**Output:** Selected features list $F_S$;
1: Data prepossessing;
2: Initialize the correlation matrix: $M$;
3: **for** $i$ *in* $F$ **do**
4: **for** $j$ *in* $F$ **do**
5: $M_{ij} = \frac{E[(X-\overline{X})(Y-\overline{Y})]}{\sigma X \sigma Y}$
6:
7: $F_s = DBSCAN(M)$
8: **return** $F_s$

### 4.2. Prediction model design

In this section, we introduce how the architecture we use to predict reach rate of CDN cache group using the data output from the feature engineering stage. The key components of our model are LSTM, LSTM auto-encoder, and deep feed forward neural network.

#### 4.2.1. LSTM

LSTM, introduced in [10], addresses the problem of vanishing gradients by introducing a memory cell. Recently, LSTM has been successfully applied in time series prediction problems [19]. The inner working mechanism of LSTM cell (Fig. 7) is listed in the

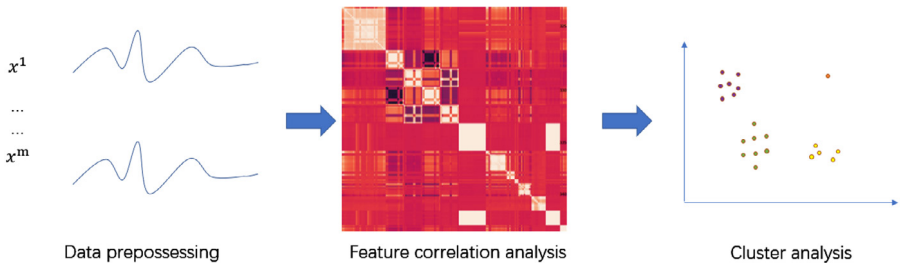



**Fig. 6.** Three steps of feature engineering.

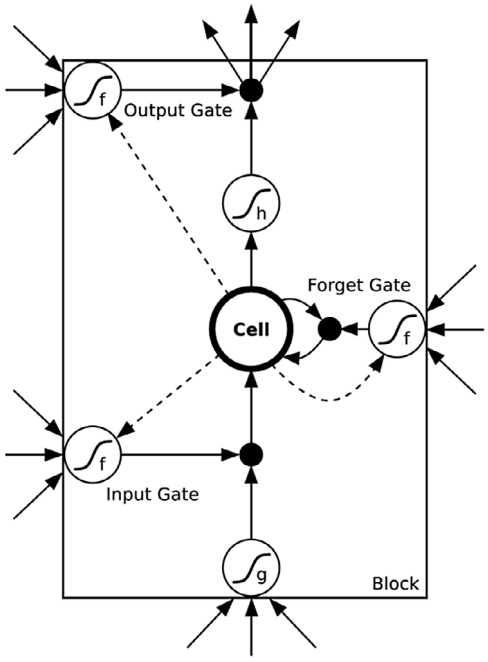



**Fig. 7.** LSTM cell [21].

following formula:

$$\begin{aligned}
\mathbf{g}_u &= \sigma(\mathbf{W}_u * \mathbf{h}_{t-1} + \mathbf{I}_u * x_t) \\
\mathbf{g}_f &= \sigma(\mathbf{W}_f * \mathbf{h}_{t-1} + \mathbf{I}_f * x_t) \\
\mathbf{g}_o &= \sigma(\mathbf{W}_o * \mathbf{h}_{t-1} + \mathbf{I}_o * x_t) \\
\mathbf{g}_c &= \tanh(\mathbf{W}_c * \mathbf{h}_{t-1} + \mathbf{I}_c * x_t) \text{ (see Formula 3 and Fig. 8)} \\
\mathbf{m}_t &= \mathbf{g}_f \odot \mathbf{g}_o + \mathbf{g}_u \odot \mathbf{g}_c \\
\mathbf{h}_t &= \tanh(\mathbf{g}_o \odot \mathbf{m}_{t-1})
\end{aligned} \tag{6}$$

here $\sigma$ is the logistic sigmoid function, $\mathbf{W}_u$, $\mathbf{W}_f$, $\mathbf{W}_o$, $\mathbf{W}_c$ are recurrent weight matrices and $\mathbf{I}_u$, $\mathbf{I}_f$, $\mathbf{I}_o$, $\mathbf{I}_c$ are projection matrices.

#### 4.2.2. LSTM auto-encoder

An LSTM auto-encoder (Fig. 9) contains the following: an encoding function, a decoding function, and a distance function between the amount of information loss between the compressed representation of your data and the decompressed representation (i.e. a "loss" function). The encoder and decoder will be chosen to be deep layered LSTM network. So the parameters of the encoding/decoding functions can be optimized to minimize the reconstruction loss, using Stochastic Gradient Descent.

Prior to fitting the data prediction model, we first conduct a pretraining step to fit an encoder that can extract useful and representative embeddings from time series. The goals are to ensure that

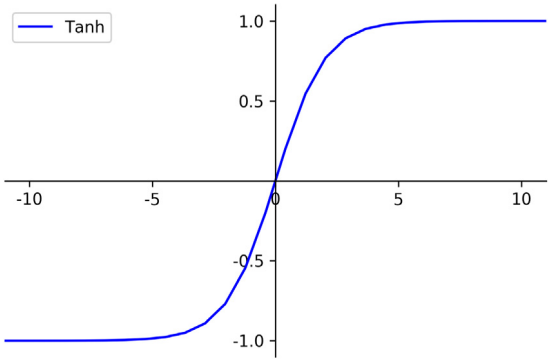



**Fig. 8.** Tanh as activation function.

(i) the learned embedding provides useful features for prediction and (ii) unusual input can be captured in the embedded space, which will get further propagated to the prediction network in the next step.

As illustrated in Fig. 10, given a multivariate time series of data, the encoder reads the vectors as input and transforms them as latent variables. During the pretraining, only the weights of LSTM auto-encoder are trained. The LSTM auto-encoder is trained to reconstruct the input on the output side.

In the second step, a fully connected feed forward network takes the latent variables which are the outputs of the encoder as input. We take the latent variables as training set and reach rate as the targets and use the gradient descent to train the network. We use 9/10 of the data as training set and 1/10 of the data as test set.

## 5. Evaluation

### 5.1. Experimental settings and dataset

Our implementation uses the Google opensource deep learning library, Tensorflow [25], version 1.2.0. We ran our experiments on a physical machine running an Ubuntu 16.04 operation system, intel i7-6700HQ, 16 GB memory, and GPU gtx1060.

In feature clustering and selection phase, we set the neighborhood parameter $\epsilon$ as 0.8. In the pre-training procedure, we use minibatch Stochastic Gradient Descent (SGD) together with the Adam optimizer to train the LSTM auto-encoder model. The size of the minibatch is 128. The weights can be learned by standard LSTM learning algorithm propagation through time with mean squared error as the objective function. In the training procedure, we use a 4 layer feed-forward neural network. We use the batch gradient descent training algorithm to train the neural network.

To test the performance, we select two cache groups with average request above 7000 per minute. The first cache server groups have 13 cache servers while the other has 10 cache servers.

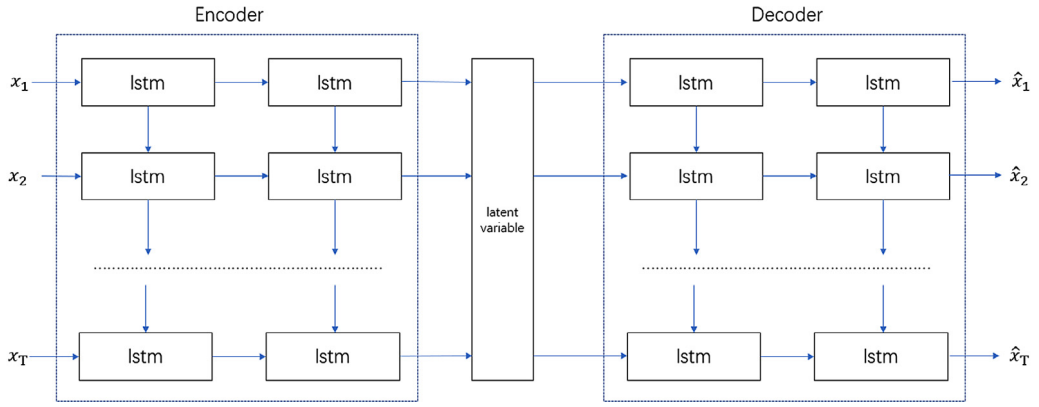



**Fig. 9.** LSTM auto-encoder model.

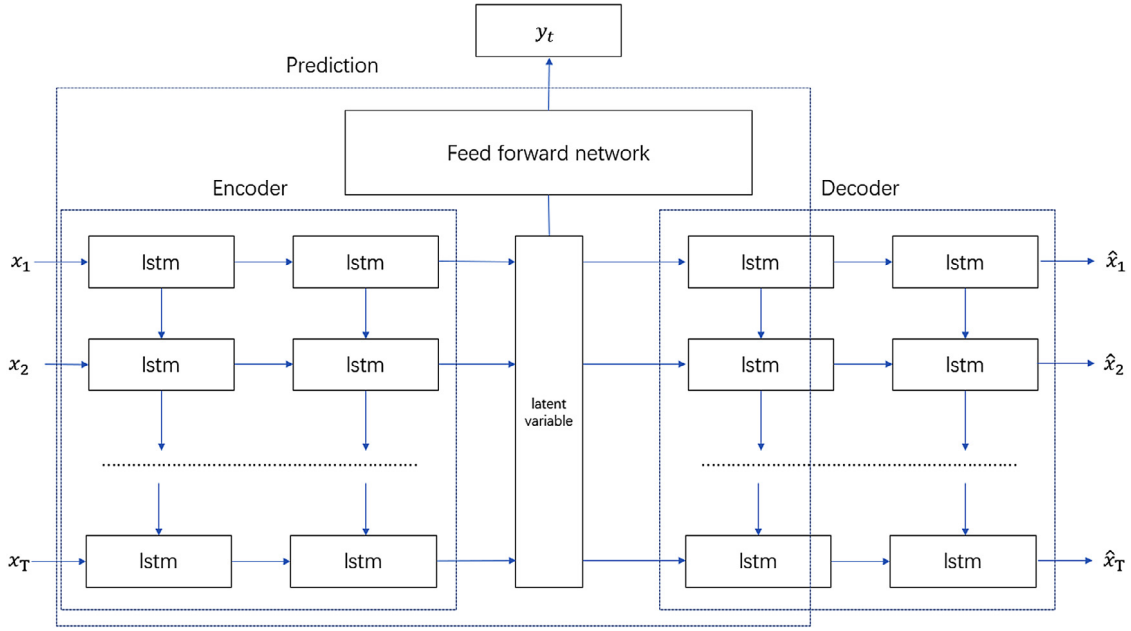



**Fig. 10.** LSTM auto-encoder with a deep feedforward network.

The frequency of the metrics and reach rate are both minute-by-minute. These data cover the period from January 7, 2018 to January 22, 2018.

We divide the entire dataset into two parts. The first part is the training set, and the second one is the test set. We use the training set to train the model and test set to test its generalization ability. The training set contains the data of 12 days. The test set contains the remaining 4 days.

### 5.2. Baseline

We compare our model with other baseline models which are listed as follows:

1. multiple linear regression model
2. feed-forward neural network
3. vanilla LSTM model
4. LSTM encoder–decoder with a feed-forward neural network.

### 5.3. Experimental results

We use mean absolute error (MAE) to measure the accuracy of our models. MAE is a scale-dependent measure. Specifically, assuming $y'$ is the target at time $t$, $y$ is the predicted value at time $t$ and we have $T$ timesteps.

$$MAE(y', y) = \frac{1}{T} \times \sum_{t=1}^{T} \left| y'_t - y_t \right|. \tag{7}$$

The performance comparison is listed in Table 3. From the experiment, we observe that those methods using neural networks are superior to the one that using linear models, for linear models cannot characterize the non-linear relationship between the features and targets. The RNN-based methods perform better than the feed forward neural network, for it contains the hidden state which captures the temporal information. Our method performs the best, for LSTM auto-encoder which can extract temporal structure better

**Table 3**
Performance comparison.

| Models | Training set | Test set |
|---|---|---|
| Multiple linear regression model | 7.16 | 8.46 |
| Feed-forward neural network with sliding windows | 2.41 | 2.50 |
| Vanilla LSTM model | 1.77 | 1.81 |
| LSTM auto-encoder with a feed-forward neural network | 1.50 | 1.73 |

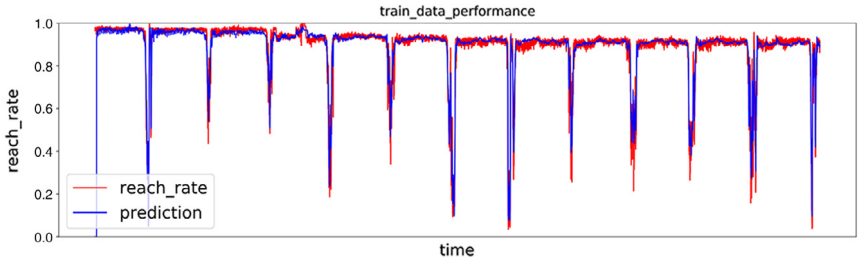



**Fig. 11.** The prediction result on training set.

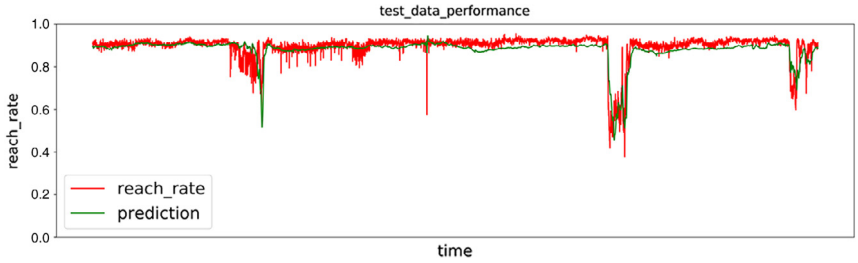



**Fig. 12.** The prediction result on test set.

from the high-dimensional data into a lower dimensional dense representation.

The effectiveness of our model is illustrated in Figs. 11 and 12. In Fig. 11, we can observe a periodic decline of reach rate. The decline of reach rate is an implication of performance deterioration caused by the high demands. The highest demand is often around 10 pm and corresponds with the minimum point of the curves. The experiments results show that our model can fit the training data well. Our model can detect the performance downfalls that we are particularly interested in. In the Fig. 12, it shows our model can generalize well because it can detect performance downfalls. It can be used in the production environment to predict the future performance deterioration to guide the resource management decisions.

## 6. Related work

There is extensive research regarding CDN, for a large portion of internet traffic was boosted by service provided by CDN. There are three categories: (1) long-term network planning, including optimized CDN design that relates to PoP selection and cache deployment [8,14,24], and (2) short-term, run-time cache management, including content replacement and prefetching strategies in the CDN network [1,3,16]. (3) CDN selection to optimize QoE for the client [12,13]. Our research falls in the second category.

CDN cache management is complex due to its structure, and large amounts of metrics data. When evaluating the complex system, the evaluation method can fall into two categories: model-driven method and data-driven method. In the model-driven method, the mathematical model characterizing the inner components of a system has to be explained explicitly. In [24], a model-driven method is used to solve the optimal cache-deployment problem. However, in the data-driven method, the data characterizing the behavior of the system instead of the analytical model is necessary. In [13], CDN selection decisions of QoE optimization decision are based on the past experience of decisions and decisions outcome. In [20], an ABR decision framework using deep reinforcement learning treats the network as a black box. In [27], a convolutional bi-directional LSTM network is used to predict the machine health by sensor data. In [26], the VM selection mechanism allows users to balance performance gains with cost reductions. From the characteristics of the above methods, the data-driven method, which takes the gathered data as the basis and is independent of the object's prior knowledge, is a more useful approach for performance evaluation, fault detection, and reliability evaluation.

In recent years, deep learning has gained success in different fields, including image recognition, speech recognition, and natural language understanding [17]. RNN [23] outperforms the traditional feed-forward neural network in modeling sequence data because its structure can characterize the temporal structure of input data by introducing hidden state. Training RNN is hard because of vanishing gradients problem; LSTM addresses these issues by introducing LSTM cell [10]. LSTM has been widely proven successful in sequence modeling. In [22], an attention-based LSTM outperforms traditional methods like ARIMA. In [28], a robust model using LSTM was used to predict the number of trips and do anomaly detection for Uber. Drawing parallel from the success of LSTM in other fields, we use LSTM-based solution to solve our problems.

## 7. Conclusion and future work

In this paper, we present a data-driven approach to evaluating the performance of cache server groups in Content Delivery Network. The LSTM auto-encoder can capture the long-term temporal information in the sequences. This paper shows that it is feasible

to apply state-of-the-art deep learning techniques to model networked systems that provide estimation for its performance. The empirical studies show that ours has outperformed the conventional methods.

There are hundreds of cache groups across China in our cooperated CDN operator. Although our method can be used to train on all kinds of cache groups of different structures, the trained model is for one specific cache group. A unified model for all kinds of cache groups is required. As the reach rate is an indirect measurement of QoS of clients, collecting data from client ends will provide useful insights. We also want to develop the online training methods for our models because we observe that the relationship between sensors and the reach rate changes over time. We shall try variants of RNN in the future such as GRU and SRU, which may have better performance or less training cost in our performance evaluation applications.

## Acknowledgments

The work of this paper is supported by National Natural Science Foundation of China under Grant No. 61728202-Research on Internet of Things Big Data Transmission and Processing Architecture based on Cloud-Fog Hybrid Computing Model Grant No. 61572137-Multiple Clouds based CDN as a Service Key Technology Research, and Shanghai 2016 Innovation Action Project under Grant 16DZ1100200-Data-trade-supporting Big Data Testbed.

## References

[1] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, K.K. Ramakrishnan, Optimal content placement for a large-scale VoD system, IEEE/ACM Trans. Netw. 24 (4) (2016) 2114–2127. http://dx.doi.org/10.1109/TNET.2015.2461599.

[2] Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neural Netw. 5 (2) (1994) 157–166. http://dx.doi.org/10.1109/72.279181. URL http://www.ncbi.nlm.nih.gov/pubmed/18267787, http://ieeexplore.ieee.org/document/279181/.

[3] S. Borst, V. Gupta, A. Walid, Distributed caching algorithms for content distribution networks, in: Proceedings - IEEE INFOCOM, http://dx.doi.org/10.1109/INFCOM.2010.5461964.

[4] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. URL http://www.statnlp.org/wp-content/uploads/2016/02/rnn.pdf.

[5] Edge computing. URL https://en.wikipedia.org/wiki/Edge_computing.

[6] M. Ester, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proc. 1996 Int. Conf. Knowledg Discovery and Data Mining, KDD '96, 1996, pp. 226–231.

[7] J. Gao, Machine Learning Applications for Data Center Optimization. URL https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42542.pdf.

[8] S. Hasan, S. Gorinsky, C. Dovrolis, R.K. Sitaraman, Trade-offs in optimizing the cache deployments of CDNs, in: Proceedings - IEEE INFOCOM, 2014, pp. 460–468. http://dx.doi.org/10.1109/INFOCOM.2014.6847969.

[9] M. Hermans, B. Schrauwen, Training and Analyzing Deep Recurrent Neural Networks, pp. 1–9.

[10] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neural Comput. 9 (8) (1997) 1735–1780. http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[11] J. Jiang, R. Das, G. Ananthanarayanan, P.A. Chou, V.N. Padmanabhan, V. Sekar, E. Dominique, M. Goliszewski, D. Kukoleca, R. Vafin, H. Zhang, VIA: Improving Internet Telephony Call Quality Using Predictive Relay Selection. http://dx.doi.org/10.1145/2934872.2934907. URL https://www.cs.cmu.edu/~junchenj/via.pdf.

[12] J. Jiang, V. Sekar, H. Milner, D. Shepherd, I. Stoica, H. Zhang, CFA: A Practical Prediction System for Video QoE Optimization. URL https://www.cs.cmu.edu/~junchenj/cfa.pdf.

[13] J. Jiang, S. Sun, V. Sekar, H. Zhang, Pytheas: Enabling data-driven quality of experience optimization using group-based exploration-exploitation, in: 14th USENAIX Symposium on NSDI.

[14] P. Krishnan, D. Raz, Y. Shavitt, The cache location problem, IEEE/ACM Trans. Netw. 8 (5) (2000) 568–582. http://dx.doi.org/10.1109/90.879344.

[15] M. Längkvist, L. Karlsson, A. Loutfi, A review of unsupervised feature learning and deep learning for time-series modeling. http://dx.doi.org/10.1016/j.patrec.2014.01.008. URL https://ac.els-cdn.com/S0167865514000221/1-s2.0-S0167865514000221-main.pdf?_tid=73b58696-de42-11e7-99f2-00000aab0f27&acdnat=1512976445_b3a701a33285bf903a86108fc3a2b956.

[16] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, S. Chouvardas, Placing dynamic content in caches with small population, in: Proceedings - IEEE INFOCOM 2016-July. arXiv:1601.03926, http://dx.doi.org/10.1109/INFOCOM.2016.7524380.

[17] Y. Lecun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444. http://dx.doi.org/10.1038/nature14539. arXiv: arXiv:1312.6184v5.

[18] T.A.O. Li, Data-Driven Techniques in Computing System Management, 50 (3).

[19] P. Malhotra, L. Vig, G. Shroff, P. Agarwal, Long Short Term Memory Networks for Anomaly Detection in Time Series. URL https://www.researchgate.net/profile/Pankaj_Malhotra3/publication/304782562_Long_Short_Term_Memory_Networks_for_Anomaly_Detection_in_Time_Series/links/5880506308ae71eb5dbfbd10/Long-Short-Term-Memory-Networks-for-Anomaly-Detection-in-Time-Series.pdf.

[20] H. Mao, R. Netravali, M. Alizadeh, Neural adaptive video streaming with pensieve, in: Mohammad Alizadeh MIT Computer Science and Artificial Intelligence Laboratory, 2017, pp. 197–210. http://dx.doi.org/10.1145/3098822.3098843.

[21] W. Mulder, S. Bethard, M.F. Moens, A Survey on the Application of Recurrent Neural Networks to Statistical Language Modeling, Academic Press Ltd., 2015, pp. 61–98.

[22] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, G.W. Cottrell, A Dual-Stage Attention-Based Recurrent Neural Network for Time Series Prediction. URL http://cseweb.ucsd.edu/~yaq007/DA-RNN.pdf.

[23] J. Schmidhuber, A local learning algorithm for dynamic feedforward and recurrent networks, Connect. Sci. 1 (4) (1989) 403–412. http://dx.doi.org/10.1080/09540098908915650. URL http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=B1927E88DDAA6F03912D61DD23B7DABE?doi=10.1.1.51.8813&rep=rep1&type=pdf%0Ahttp://www.tandfonline.com/doi/abs/10.1080/09540098908915650%5Cnhttps://www.tandfonline.com/doi/full/10.1080/09540098908915650%0Ah.

[24] G. Tang, K. Wu, R. Brunner, Rethinking CDN design with distributee time-varying traffic demands, in: IEEE INFOCOM 2017 - IEEE Conference on Computer Communications, IEEE, 2017, pp. 1–9. http://dx.doi.org/10.1109/INFOCOM20178057028. URL http://ieeexplore.ieee.org/document/8057028/.

[25] TensorFlow. URL https://www.tensorflow.org/.

[26] N.J. Yadwadkar, B. Hariharan, J.E. Gonzalez, B. Smith, R.H. Katz, Selecting the best VM across multiple public clouds, in: Proceedings of the 2017 Symposium on Cloud Computing - SoCC '17, ACM Press, New York, New York, USA, 2017, pp. 452–465. http://dx.doi.org/10.1145/3127479.3131614. URL http://dl.acm.org/citation.cfm?doid=3127479.3131614.

[27] R. Zhao, R. Yan, J. Wang, K. Mao, Learning to monitor machine health with convolutional Bi-directional LSTM networks, Sens. (Switzerland) 17 (2) (2017) 1–18. http://dx.doi.org/10.3390/s17020273.

[28] L. Zhu, N. Laptev, Deep and Confident Prediction for Time Series at Uber. http://dx.doi.org/10.1109/ICDMW.2017.19.

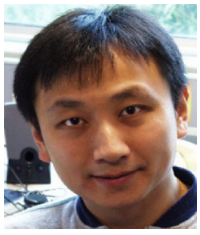
**Ziyan Wu** is a master student at School of Computer Science, Fudan University. His research interests are cloud computing, data-driven networking, content delivery network, edge computing, and big data process architecture.

**Zhihui Lu** is an Associate Professor in School of Computer Science, Fudan University. He received a Ph.D. computer science degree from Fudan University in 2004, and he is a member of the IEEE and China computer federation's service computing specialized committee. His research interests are cloud computing and service computing technology, big data architecture, edge computing, and content delivery network.

**Wei Zhang** is a master student at School of Computer Science, Fudan University. His research interests are data-driven optimization, cloud computing, and distributed system.

**Jie Wu** is a Professor at School of Computer Science, Fudan University. His research interests are internet technology, big data architecture, service computing, cloud computing, content delivery and P2P streaming technology, and he received a Ph.D. computer science degree from Fudan University in 2008.

**Shalin Huang** is R & D director of CDN department of Wangsu Science Technology Co., Ltd. She owns a lot of patents on CDN technology. Her research interests include content delivery network, cloud computing, edge computing, and big data.



**Patrick C.K. Hung** is a Professor at the Faculty of Business and Information Technology in University of Ontario Institute of Technology. Patrick has been working with Boeing Research and Technology in Seattle, Washington on aviation services-related research projects. He owns a U.S. patent on Mobile Network Dynamic Workflow Exception Handling System with Boeing. His research interests include services computing, cloud computing, big data, and content delivery network.