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*
[c] *Faculty of Business and ITUniversity of Ontario Institute of Technology, Canada*

Abstract

In industry, Content Delivery Network(CDN) Service providers are increasingly using data-driven mechanisms to build their performance models of their service-providing systems. To build 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 have been shown to greatly outperform traditional methods in modeling complex systems. We design a approach that using these techniques to build a 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-arts models.

Keywords: *edge computing, deep learning, content delivery networks, sequence learning, predictive analysis, high dimensional data*

1. Introduction

There is a trend [1] [2] [3] [4] [5] that using 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 real-word 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 recognizaition, instead of using 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[6]. 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 low latency, reliability, supports better quality of experience and security.

The CDN Service providers are increasingly using data-driven mechanisms to build their 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, 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 don't 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 which is collected and calculated from the log of the HA proxy of CDN cache

*Corresponding author
Email address: lzh@fudan.edu.cn (Zhihui Lu)

groups. The computation of reach rate is done offline . 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 characterised as multi-dimensio highly non-linear, time variant vectors. The metrics that 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 using simple heuristics which are oversimplifed 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 that attempts to model high-level abstractions in data by using artificial neural network architectures composed of multiple non-linear transformations [7]. They have a lot of successful applications in sequence modeling[8]. Compared to other machine learning techniques, a lot of work show that it can detect complex relationships among features, can extract hierarchical level of features from high-dimensional data, including monitoring data.

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

Our main contributions are listed below:

- 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 and show ours is superior by empirical studies.

The remain organization of this paper is as follows. In Section 2, we first introduce the related concept as our research background. In Section 3, we formulate our performance evaluation problem as a sequence learning problem and then compare the baseline methods. In Section 4, we introduce our method of feature engineering to reduce the dimensionality for the high-dimensional data and 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 and future work. We provide concluding remarks in Section 7.

2. Background

A content delivery network or content distribution network (CDN) (figure 1) is a geographically distributed network of cache servers. CDN helps content provider deliver web pages and other multimedia content to the clients,

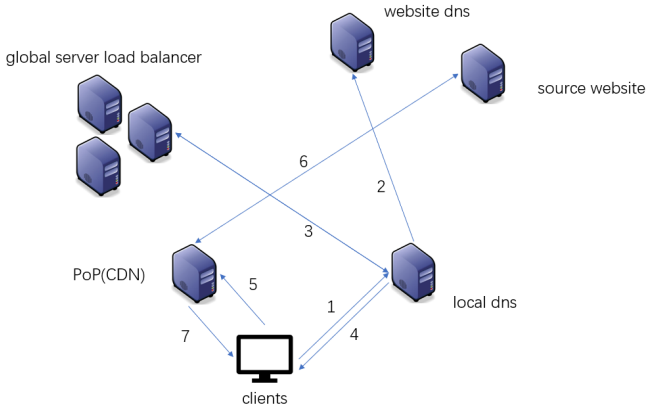

Figure 1: The basic working procedure of CDN

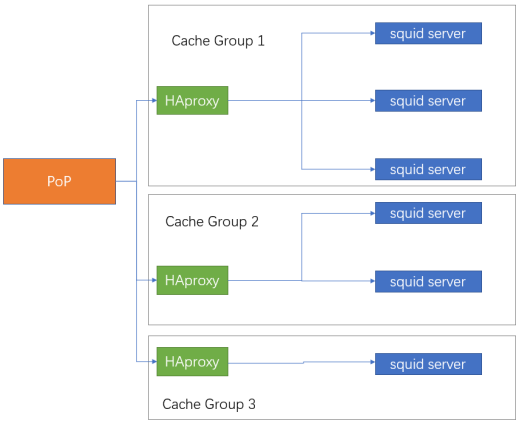

Figure 2: The structure of cache server groups

based on the locations of the clients and cache servers nearby the clients. The basic working procedure is as follows. Step 1: client sends a request to local DNS. Step 2: the DNS finds the CNAME and redirects the request to the GSLB (global sever 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 return 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 doesn't exist locally. Step 7: content is sent to the clients.

A CDN cache server group (figure 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 homogenous cluster. In a heterogeneous clus-

ter, weighted round-robin algorithm is used. A weight was assigned to the server 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), so they use the measurement reach rate . They calculated reach rate from 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 are 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. This is the lst of candidate input features from one cache server group. We organize the features into groups. The features in the first group 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 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 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 models 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 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.

Giving 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

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

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 use an exponential moving average to linearly map each metrics to a score in a certain time interval. The parameters are depending on the experience of the operators. This method is impractical: it can hardly generalize well and requires tedious repetitive tuning.

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 request when cpu is idle
hitratio.port8101	hit retio of port 8101
hitratio.port81021	hit retio of port 8101
...	...
load.total$_{l}oad$	total load of
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.CPU.max	the max CPU usage of a single machine
aggregate.resptime	the average of resone time of a single machine
...	...
group 3	
all_machine.cpu	the sum of cpu usage of all machines
all_machine.diskused	he sum of dist usage of all machines
all_machine.ioutil	he sum of IO utility of all machines
all_machine.CPU.max	he max CPU usage of all machine
all.resptime	the average of resone time of a single machine
...	...
group 4	
bandwidth available	the oacket download speed
response time	time that the packet arrives

Table 1: The feature list

3.1. Linear Models for Sequence Learning

There are some conventional approaches which use data to learn such as using 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, 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, β 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 much real works. 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. They have a lot of successful applications in sequence modeling. Compared to other machine learning techniques, it can detect complex relationships among features, can extract a hierarchical level of features from raw data. So it can build a model more accurate with less time. They are advantageous for modeling intricate systems because neural networks do not require the user to predefine the feature interactions in the model, which assumes relationships within the data. Instead, the neural network searches for patterns and interactions between features to automatically generate the best fit model.

3.2.1. Feed Forward Neural Network

A generic three-layered neural network is illustrated in Figure 2. In this study, the input matrix $I \in R^{M\times(N+1)}$ where M is the number of training examples and $N+1$ 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 first layer is transformed by an activation function. We

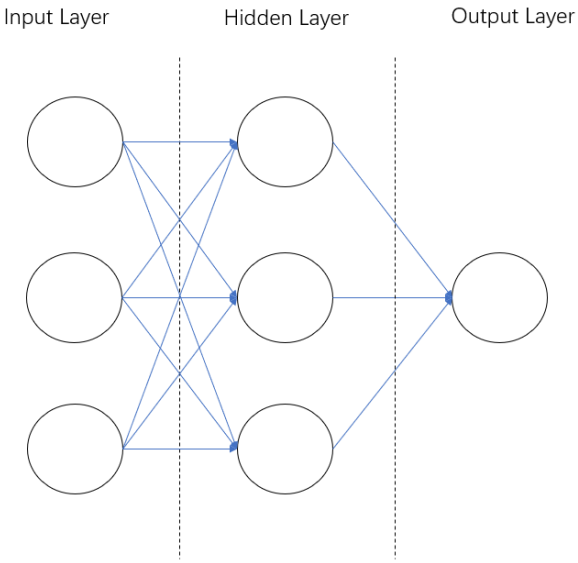

Figure 3: MLP structure

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:

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

where sigmoid is the activation function:

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

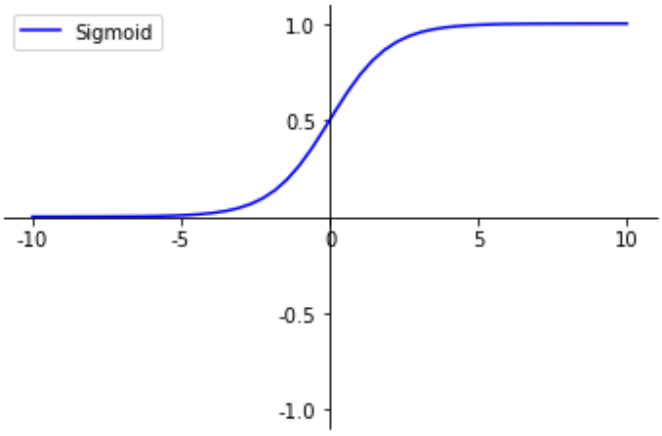

Figure 4: Sigmoid function as activation function

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 four steps: (1) Randomly initialize the model parameters, (2) forward propagation algorithm, (3) Compute the cost function J, (4) back propagate using the chain rule, (5) using the gradients calculated, 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 doesn't hold true, for the samples of our problem have a temporal relationship: the status of the system of the next timestep not only depends 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 isn't practical for the dependencies length which isn't 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 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[10] [11].

RNNs 3 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.

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

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

where tanh is the hyperbolic tangent function, $\mathbf{W}$ is the recurrent weight matrix and I is a input wight matrix. The hidden state $\mathbf{h}$ is then used to make a prediction

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

where f can be 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

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

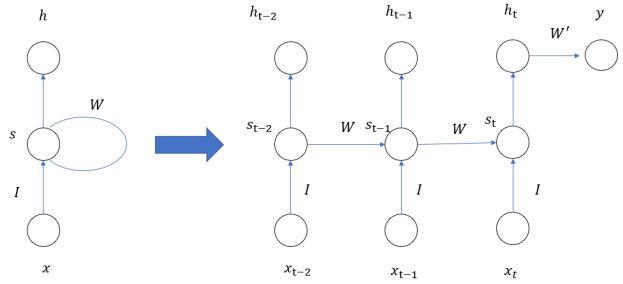

Figure 5: The left shows the backloop structure of RNN and right shows that RNN can be thought as infinite deep layers neural network unfolded along the dimension of time

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 [10]. 2. RNNs aren't capable of learning long-term dependencies. LSTM address these issues by introducing LSTM cell[12]

3.3. Models Comparisons

In sum, non-linear models have a better accuracy than linear models in non-linear problems. RNN-based methods such as RNN, LSTM and our methods proposed in next section can capture temporal structures in the data. Simple heuristics and MLR cannot decribe the interpendency of the features of data by their natures. It's hard to decide the paremeters of models by experience in simple huristcis 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.

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 propose of this stage is two-fold: (1) to 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. As showed in graph. To make things more complex, a cache group may have different number caching servers. As we only care about. (2)

Table 2: Models Comparison

	Simple heuristics	MLR	NN	RNN	LSTM	LSTM Auto-Encoder
Accuuracy	*	**	***	****	*****	******
Speed of Predictions	***	***	**	**	**	*
Tolerance To Redundant Features	*	**	***	***	***	*****
Attributes Interpendence	*	**	***	***	***	***
Speed of Traning	-	**	***	***	*	**
Model Paremeters Handling	***	*	*	*	*	**
Temporal Strucutres	no	no	no	yes	yes	yes

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.

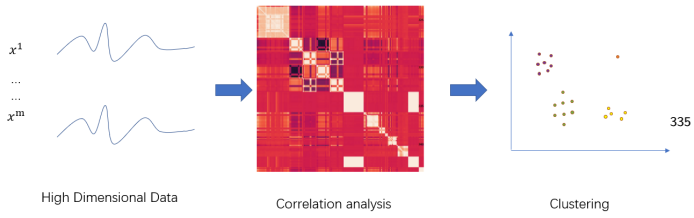

Figure 6: three steps of feature engineering

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 a dependence between the variables. This dependence can be classified as the probability that changes in one variable affect the behavior of the second variable. The Pearsons correlation defines this dependence in the interval between -1 and 1. Pearsons 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 distance. We want to choose the representative features to represent the data. Then, we use DBSCAN [13] to cluster these features to eliminate closed ones. DBSCAN is a density-based clustering algorithm. Compared to other algorithms like K-means, we don't have to specify the number of features and can find any cluster of any shape. By defining the neighborhood parameter ϵ, 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 elimate redundant information exsited in data. We formulate the process in Algorithm 1.

Algorithm 1 Feature Clustering and Selection

Input: All features list: F; Neighborhood parameter: ϵ;
Output: Selected features list F_S;

Data prepossessing;
Initialize the correlation matrix: M;
for i *in* F **do**
  for j *in* F **do**
    $M_{ij} = \frac{E[(X-\overline{X})(Y-\overline{Y})]}{\sigma X \sigma Y}$
$F_s = DBSCAN(M)$
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 foward neural network.

4.2.1. LSTM

LSTM, introduced in [12], addresses the problem of vanishing gradients by introducing a memory cell. [14] applied in time series. The inner working of LSTM(figure 7) are listed follows:

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

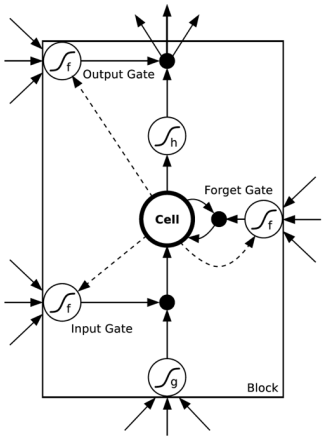

Figure 7: LSTM cell

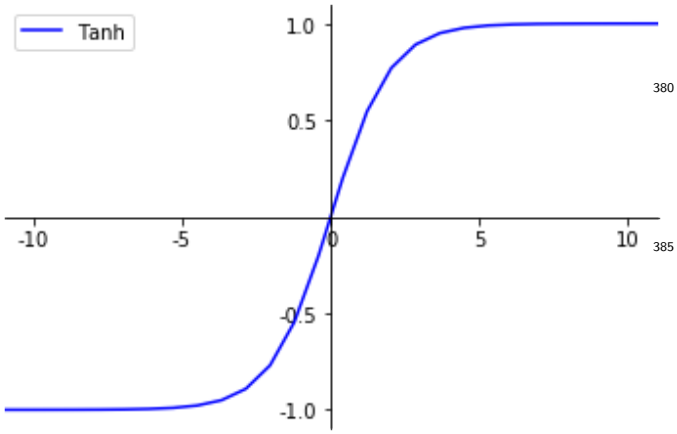

Figure 8: tanh as activation function

here σ 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 (figure 9) contains: 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 optimize to minimize the reconstruction loss, using Stochastic Gradient Descent.

Prior to fitting the data prediction model, we first conduct a pre-training step to fit an encoder that can extract useful and representative embeddings froma time series. The goals are to ensure that (i) the learned embedding provides useful features for prediction and (ii) unusual input can be captured in the embedded space, which will get

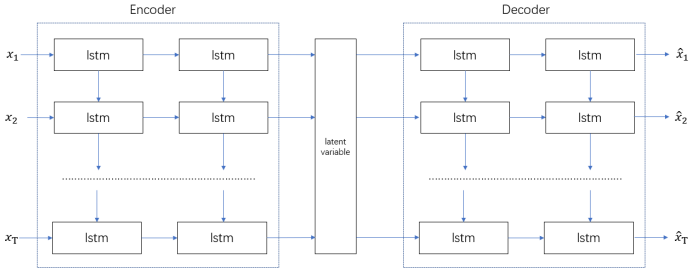

Figure 9: LSTM Auto-encoder Model

further propagated to the prediction network in the next step.

As illustrated in figure 10. Given a multivariate time series of data, the encoder reads the vectors as input and transform 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 foward network take 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[15], 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 th neighborhood parameter ϵ 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 throught time with mean squared error as the objective function. In the training procedure we use a 4 layer feed-foward 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 servers groups have 13 cache servers while the other has 10 cache servers. The metrics are raw data collected from the cache servers, including CPU utilization, network utilization, disk utilization, memory utilization of two cache group servers. The frequency of the metrics

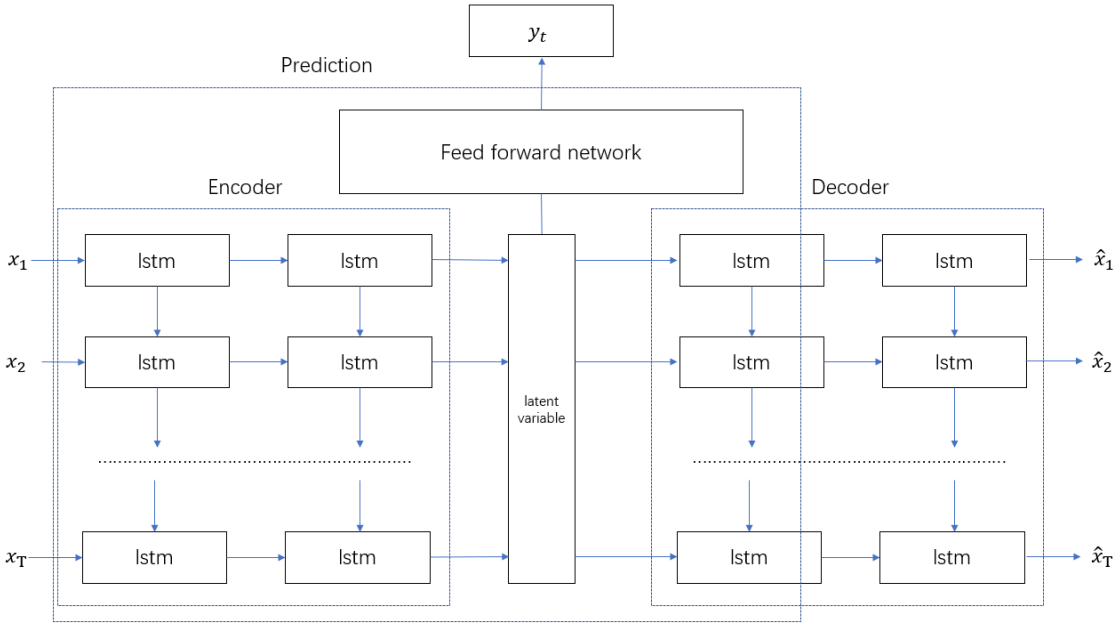

Figure 10: lstm auto-encoder with a deep feadfoward network

and reach rate are both minute-by-minute. This data covers the period from January 7, 2018 to January 22, 2018.

We divide the entire data set into two parts. The first part is the training set and second one is the test set. We use the training set to train the model and test set to test its generalization ability. The traning 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 model which are listed follow:

1. multiple linear regression model
2. feed-foward neural network
3. vanilla LSTM model
4. LSTM encoder-decoder with a feed-foward 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 and y is the predicted value at time t.

$$MAE(y^{'}, y) = \frac{1}{N} \times \left| y^{'}{}_t - y_t \right| \tag{7}$$

The performance comparison is listed in Table 2. From the experiment, we observe that those methods using neural networks are superior to the one that using linear model, for linear model cannot characterize the non-linear relationship between the features and targets. The recurrent network based methods perform better than the feed foward 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 from the high-dimensional data into a lower dimensional dense representation.

The result of our model is illustrated in the figure 11 and 12. In 12, we can observe a periodic decline in the picture. The bottom is often caused by the high demands. The highest demand is often around 10 p.m. Our model can fit the traning data very well. Our model can detect the performance downfall that we are particularly interested in. In the figure 12, it shows our model can generalize well in the cases it never sees.

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 category: (1) long-term network planning, including optimized CDN design that relates to PoP selection and cache deployment[16][17][18], and (2) short-term, run-time cache management, including content replacement and prefetching strategies in the CDN network[19][20][21]. (3) CDN selection to optimize QoE for the client[1][22]. Our research falls in the second categories.

CDN Cache management is complex due to its structure, large amounts of metrics data. When evaluating the complex system, the evaluation method can fall into two categories: model-driven method, data-driven method. In model-driven method, the mathematical model characterizing the inner components of a system has to be explained explicitly. In [18], a model-driven method is used to solve the optimal cache-deployment problem. However, in the data-driven method, the data characterizing the behavior

Table 3: Performance Comparison

Models	training set	test set
multiple linear regression model	7.16	8.46
feed-foward neural network with sliding windows	2.41	2.50
vanilla LSTM model	1.77	1.81
LSTM auto-encoder with a feed-foward neural network	1.72	1.87

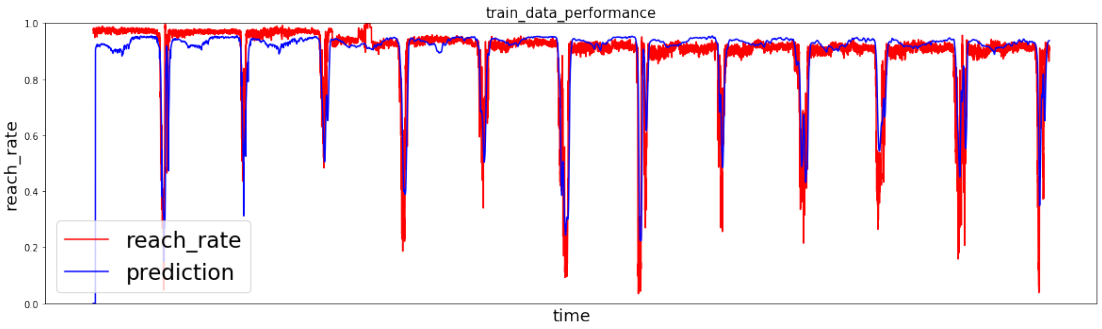

Figure 11: The prediction result on training set.

of the system instead of the analytical model is necessary. In [1], CDN selection decisions of QoE optimization decision is based on the past experience of decisions and decisions outcome. In [2], an ABR decision framework using deep reinforcement learning treats the network as a black box. In [23], a convolutional bi-directional LSTM network is used to predict the machine health by sensor data. in [24], 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 objects prior knowledge, is a more useful approach for performance evaluation, fault detection, and reliability evaluation.

In recent year, deep learning has gained success in different fields, including image recognization, speech recognization, natural language understanding[7]. RNN[25] outperform 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[12]. LSTM has been widely proven successful in sequence modeling. In [26], an attention-based lstm outperforms traditional methods like ARIMA. In [27], a robust model was used to predict the number of trips and do anomaly detection for Uber.

7. Conclusion and Future Work

In this paper, we present a data-driven approach to evaluate the performance of cache server groups. 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 in China. Although our method can be used to train on all kinds of cache group 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 clients 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.

8. Acknowledgment

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] J. Jiang, S. Sun, V. Sekar, H. Zhang, Pytheas: Enabling Data-Driven Quality of Experience Optimization Using Group-Based Exploration-Exploitation, 14th USENAIX Symposium on NSDI.

[2] H. Mao, R. Netravali, M. Alizadeh, Neural Adaptive Video Streaming with Pensieve, Mohammad Alizadeh MIT Computer Science and Artificial Intelligence Laboratory (2017) 197–210doi:10.1145/3098822.3098843.
URL http://dx.doi.org/10.1145/3098822.3098843

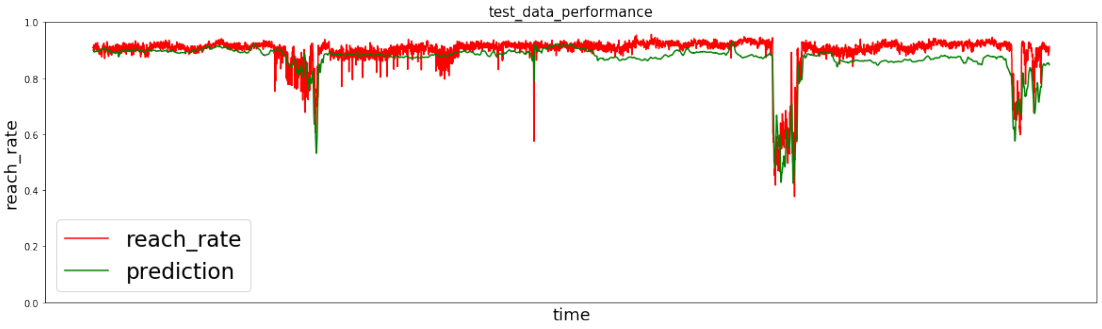

Figure 12: The prediction result on test set.

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

[4] 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 Selectiondoi:10.1145/2934872.2934907.
URL https://www.cs.cmu.edu/~junchenj/via.pdf

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

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

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

[8] M. Längkvist, L. Karlsson, A. Loutfi, A review of unsupervised feature learning and deep learning for time-series modelingdoi: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

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

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

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

[12] S. Hochreiter, J. Schmidhuber, Long Short-Term Memory, Neural Computation 9 (8) (1997) 1735–1780. doi:10.1162/neco.1997.9.8.1735.

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

[14] 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

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

[16] P. Krishnan, D. Raz, Y. Shavitt, The cache location problem, IEEE/ACM Transactions on Networking 8 (5) (2000) 568–582. doi:10.1109/90.879344.

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

[18] 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. doi:10.1109/INFOCOM.2017.8057028.
URL http://ieeexplore.ieee.org/document/8057028/

[19] S. Borst, V. Gupta, A. Walid, Distributed caching algorithms for content distribution networks, Proceedings - IEEE INFOCOMdoi:10.1109/INFCOM.2010.5461964.

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

[21] D. Applegate, A. Archer, V. Gopalakrishnan, S. Lee, K. K. Ramakrishnan, Optimal Content Placement for a Large-Scale VoD System, IEEE/ACM Transactions on Networking 24 (4) (2016) 2114–2127. doi:10.1109/TNET.2015.2461599.

[22] 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

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

[24] N. J. Yadwadkar, B. Hariharan, J. E. Gonzalez, B. Smith, R. H. Katz, Selecting the ¡i¿best¡/i¿ 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. doi:10.1145/3127479.3131614.
URL http://dl.acm.org/citation.cfm?doid=3127479.3131614

[25] J. Schmidhuber, A local learning algorithm for dynamic feedforward and recurrent networks, Connection Science 1 (4) (1989) 403–412. doi: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

[26] 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

[27] L. Zhu, N. Laptev, Deep and Confident Prediction for Time Series at Uberdoi:10.1109/ICDMW.2017.19.