Task Failure Prediction in Cloud Data Centers Using Deep Learning

Jiechao Gao, *Student Member, IEEE*, Haoyu Wang, *Student Member, IEEE*, and Haiying Shen, *Senior Member, IEEE*

Abstract—A large-scale cloud data center needs to provide high service reliability and availability with low failure occurrence probability. However, current large-scale cloud data centers still face high failure rates due to many reasons such as hardware and software failures, which often result in task and job failures. Such failures can severely reduce the reliability of cloud services and also occupy huge amount of resources to recover the service from failures. Therefore, it is important to predict task or job failures before occurrence with high accuracy to avoid unexpected wastage. Many machine learning and deep learning based methods have been proposed for the task or job failure prediction by analyzing past system message logs and identifying the relationship between the data and the failures. In order to further improve the failure prediction accuracy of the previous machine learning and deep learning based methods, in this article, we propose a failure prediction algorithm based on multi-layer Bidirectional Long Short Term Memory (Bi-LSTM) to identify task and job failures in the cloud. The goal of Bi-LSTM failure prediction algorithm is to predict whether the tasks and jobs are failed or completed. The trace-driven experiments show that our algorithm outperforms other state-of-art prediction methods with 93 percent accuracy and 87 percent for task failure and job failures respectively.

Index Terms—Task failure, cloud data center, deep learning

1 Introduction

Nowadays, cloud computing service has been wildly used because it provides high reliability, resource saving, and also on-demand services. The cloud data centers include processors, memory units, disk drives, networking devices, and various types of sensors that support many applications (i.e., jobs) from users. Fig. 1 shows the structure of modern cloud data centers. The users can send requests such as store data and run applications to the cloud. Each cloud data center is composed with physical machines (PMs), and each PM can support a set of virtual machines (VMs). The tasks that are sent from users are processed in each VM. Such a large scale cloud data center can host hundreds of thousands of servers which often run tons of applications and receive work requests every second from users all over the world. A cloud data center with such heterogeneity and intensive workloads may sometimes be vulnerable to different types of failures (e.g., hardware, software, disk failures). The reasons of the failures are complex [1]. Fig. 2 shows the reasons of cloud failure. These types of failures can cause significant cost to users and cloud providers. Take software failures as an example [2], Ya-hoo Inc. and Microsoft's search engine, Bing, crashed for 20 mins in January 2015, which cost about $9000 per minute to reboot the system. In October 2013, Knight Capital's [3] cloud based automatic stock trading software went down for 45 min because of an error in trading algorithm, which costed $440 million to the company. Previous research [4], [5], [6], [7] found that hardware failure, especially disk failure, is a major contributing factor to the outages of cloud services. Because disk failure can not only lead to service unavailability but also result in permanent data loss. While a failure in a single disk might be rare, a system with thousands of disks will often experience failures from time to time. Various causes of failures in cloud computing are also discussed in several research papers [8], [9], [10] including software failure, hardware failure, services failure, power outage, natural disasters, network infrastructure, cybercrime, and human errors. These many different types of failures will lead to the application running failures. Thus, accurate prediction for the occurrence of application failures beforehand can improve the efficiency of recovering the failure and application running.

A job is comprised of one or more tasks, each of which is accompanied by a set of resource requirements. A job fails when one of its tasks fails. The previous works [5], [11], [12], [13], [14], [15], [16], [17], [18] use statistical and machine learning approaches such as Hidden Semi-markov Model (HSMM) and Support Vector Machine (SVM) to predict the task and job failures in cloud data centers. They use CPU usage and memory usage, unmapped page cache, mean disk I/O time and disk usage as inputs and the task failure or job failure as the output. However, HSMM and SVM assume that all their inputs are stationary and independent of each other which are not true in the cloud data centers. Thus, they cannot handle the sequence data or high dimensional data, in which data in time points or different features may be dependent on each other. In the cloud data

• *The authors are with the Department of Computer Science, University of Virginia, Charlottesville, VA 22904 USA. E-mail: {jg5ycn, hw8c, hs6ms}@virginia.edu.*

Manuscript received 14 Jan. 2020; revised 27 Apr. 2020; accepted 7 May 2020. Date of publication 11 May 2020; date of current version 15 June 2022. (Corresponding author: Haiying Shen.)
Digital Object Identifier no. 10.1109/TSC.2020.2993728

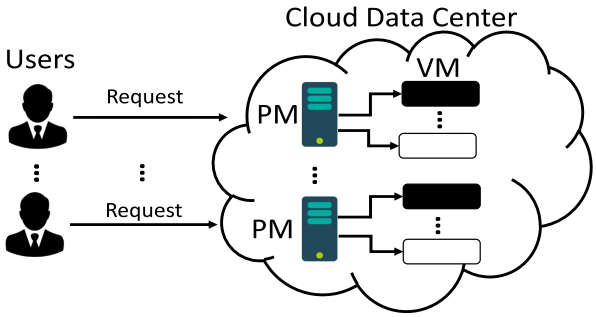


Fig. 1. Overview of cloud data center structure.

centers, the input features and noisy data are diverse in nature and have dependencies on the past events. Thus HSMM and SVM cannot handle the failure prediction in cloud data centers.

Meanwhile, some other researchers applied deep learning approaches such as Recurrent Neural Network (RNN) and LSTM for the failure prediction [19], [20], [21], [22], [23], [24]. They use CPU usage, memory usage, unmapped page cache, mean disk I/O time and disk usage as input and the task or job failure as output. However, the traditional RNN has a serious drawback for the data that has long-term dependencies [25], which means the past events can influence future events and a temporary memory of events that happened a while ago may be essential for producing a useful output action. Since the error signals are back-propagated through time, they may exhibit decay [26]. To overcome this issue, deep learning approaches such as LSTM can be used to better handle the failure prediction problem which can capture the long term dependencies. Some of the recent research papers [20], [21], [22] have already shown that LSTM performs better than HSMM, SVM, and RNN in terms of accuracy in task or job failure prediction in cloud data centers.

However, the LSTM based prediction methods still have a few shortcomings. First, the methods [20], [21], [22], [24] only consider CPU usage, memory usage, cache memory usage, mean disk I/O time, and disk usage as input features. More input features may further increase the prediction accuracy. Second, the LSTM based prediction model used single-layer LSTM construction which can not handle multiple input features well compared to multi-layer construction [26]. Third, in the cloud data center, input features like CPU usage and memory usage are highly related over time. For a given time for prediction, the LSTM based prediction model always sets higher weights on the data closer to the time, and lower weights on the data further away from the time, with the assumption that the data further away from the time always has lower impact to the prediction. However, such settings cannot accurately reflect the impact degree as the further data may still have higher impact on the failure (e.g., failures in long term jobs). The performance can be better if the weights of data items are determined based on the real data trace. Thus, a new prediction model is needed to build to implement failure prediction in the cloud data center in order to achieve better accuracy in prediction.

To overcome the shortcomings, in this paper, we build a failure prediction model based on multi-layer Bidirectional LSTM and name it as Bi-LSTM. First, Bi-LSTM has more input features than previous methods, which include task priority, task resubmissions, and scheduling delay. Second, Bi-LSTM has a multi-layer structure that can better handle multiple input features for higher accuracy. Multi-layer construction can narrow the number of parameters in the calculating functions but still with the same number of neurons which can reduce the calculation time [26]. Third, unlike LSTM [27] that simply uses one forward state, Bi-LSTM uses both forward and backward states, which more accurately determines the weights of both closer and further input features. We perform the trace-driven failure prediction study by using Google cluster trace and we compare the performance of Bi-LSTM with other state-of-art prediction methods.

Our contributions in this paper are as follows:

1) We present a deep learning based prediction model named Bi-LSTM for task and job failures prediction. We first input the data into forward state and backward state in order to adjust both the weight of both closer and further input features. We find that the further input features are essential to achieving high prediction accuracy.
2) We compare Bi-LSTM with other representative models including statistical models, machine learning models and deep learning models in terms of accuracy, F1 score, precision, and recall. The results show that our algorithm detects task failures and job failures with an accuracy of 93 and 87 percent respectively.
3) We also discuss that, in real world could data center, when a failure is detected, site reliability engineers (SREs) are required to prevent and fix the problem to achieve QoS. In general, SREs should take a time gap to locate and fix the failure. If the failure can be predicted before certain time gap, then we can achieve higher QoS in our system. However, prediction with certain time gap may also cause lower prediction accuracy. So we need to choose the balance between the time gap size and prediction accuracy. Our result shows that Bi-LSTM can achieve 90 percent prediction accuracy with 15 mins gap size.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 presents the trace study. Section 4 presents our proposed prediction method and experiment process. Section 5 presents the performance evaluation of our methods. Section 6 concludes the paper with remarks on our future work.

2 Related Work

We classify the previous related work into two parts: failure analysis in cloud data centers and failure prediction.

Failure Analysis in Cloud Data Centers. Ford *et al.* [28] studied the impact of correlated failures on availability of distributed storage systems for Google clusters. They found that although disk failures can result in permanent data loss, the major reason for most unavailability in the Google cloud storage system is transitory node failures. They also developed an availability model using Markov chains to

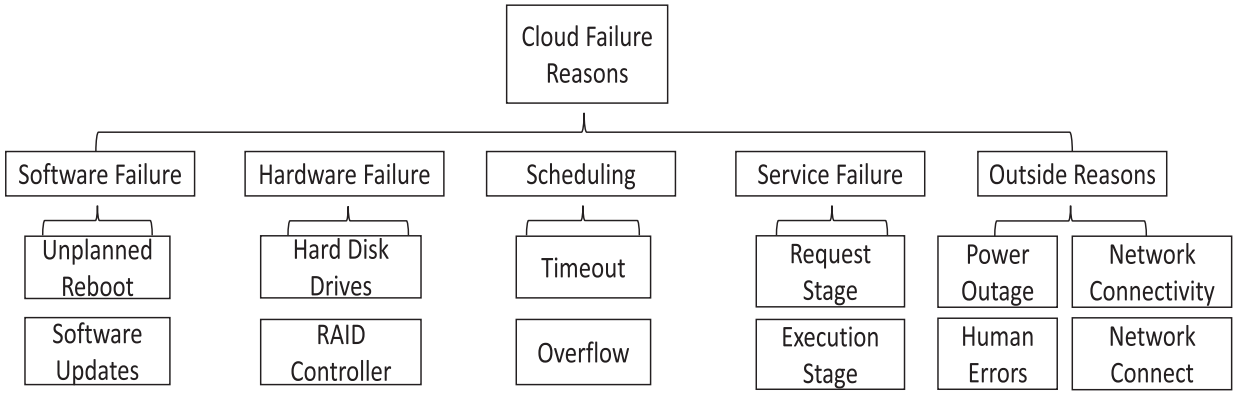


Fig. 2. Cloud failure reasons [1].

reason about past and future availability including the effects of different choices of data replication, data placement and system parameters to provide feedback and recommendations.

Padmanabhan *et al.* [29] considered failures from the perspective of web clients. They observe that the majority of failures occur during the TCP handshake as a result of end-to-end connectivity issues. They also find that web access failures are dominated by server-side issues. These findings highlight the importance of studying failures in data centers hosting web services.

Birke *et al.* [30] conducted a failure analysis on physical machines and virtual machines hosted on commercial data centers in IBM, using one-year-long data collected over 10K servers. Their analysis highlights the differences and similarities of PM and VM failure patterns. They found that VMs have lower failure rates than PMs, and show a surprising trend that, in contrast to PMs, increasing the computation intensity by VM unit does not increase failure rate.

Netmedic *et al.* [31] aimed to diagnose application failures in enterprise networks. By taking into account state of components that fail together (as opposed to grouping all components that fail together), it is able to limit the number of incorrect correlations between failures and components.

Failure Prediction Methods. We further classify the failure prediction methods to three categories: statistical approaches, machine learning approaches and deep learning approaches. For these approaches, they use input features such as CPU usage, memory usage, mean disk I/O time, and disk usage.

Statistical Approaches. Chalermarrewong *et al.* [5] proposed a framework including Autoregressive-Moving-Average Model (ARMA) and fault tree analysis for the system availability prediction within a cluster. If a fault is likely to happen, the framework alerts the cluster resource manager and the appropriate actions such as replacing a node with potential failure will be taken to prevent the loss of data or computing. Amin *et al.* [11] proposed a forecasting approach based on Autoregressive-Integrated-Moving-Average (ARIMA) and Generalized Autoregressive Conditional Heteroscedastic (GARCH) models to predict response time and time between failures in the web services. Zhao *et al.* [13] used HMM and Hidden Semi-markov Model to predict disk failures in cloud storage systems. They considered various features such as memory usage, disk usage and disk I/O time measured at consecutive time intervals for a disk drive as time series and modeled such time series to classify failed disks and good disks. However, statistical approaches such as HSMM assume that all their inputs are stationary and independent of each other which are not true in the cloud data centers. Thus, they cannot handle the sequence data or high dimensional data, in which data in time points or different features may be dependent on each other.

Machine Learning Approaches. Murray *et al.* [14] developed a new algorithm to predict the failures in hard drives called Multiple Instance Naive Bayes (mi-NB) which used naive Bayesian learning as its base classifier. They compared mi-NB with SVM and concluded that SVM is computationally expensive for this problem. Fronza *et al.* [15] introduced an approach for predicting software failures based on SVMs and Random Indexing (RI). They used RI to represent sequences of operations (i.e., user commands). Then, an SVM model classifies the representations as either failures or non-failures. Guan *et al.* [16] applied supervised learning based on Bayesian classifiers and decision tree classifiers to forecast future system failure occurrences in the cloud. Pitakrat *et al.* [17] proposed a hierarchical online failure prediction approach called Hora. Hora employed a combination of a failure propagation model and software system failure prediction techniques based on Bayesian networks. Zhang *et al.* [18] designed and implemented a new tool based on Random Forest (RF) called PreFix, for accurately predicting whether there will be a switch failure in the near future. However, machine learning approaches such as SVM have the same shortcomings just like statistical approaches, so they cannot handle the sequence data well in cloud data centers.

Deep Learning Approaches. Du *et al.* [21] presented DeepLog, which is used to predict task and job failures by using LSTM and density clustering approaches. Das *et al.* [22] provided a powerful technique to process High Performance Computing (HPC) logs using LSTM for efficient failure prediction. They used a three-phase deep learning approach to first train to recognize chains of log events leading to a failure, second re-train chain recognition of events augmented with expected lead times to failure, and third predict lead times during testing/inference deployment to predict which specific node fails in how many minutes. Xu *et al.* [19] proposed an RNN based model for predicting hard disk drive failure and giving health degrees, which treats the observed SMART features such as disk usage and disk I/O time as time-sequence data. Chen *et al.* [23] used RNN for predicting failures via various features and performance time series data in the Google cluster traces. Islam

et al. [24] performed a failure characterization study of the Google cluster workload trace and presented LSTM to predict the task failures. However, deep learning approaches such as RNN and LSTM also have several shortcomings. For RNN, it cannot handle the data with long-term dependency. For LSTM, it overcomes the drawback in RNN. But LSTM still has a drawback that it sets higher weights on the data closer to the time, and lower weights on the data further away from the time, with the assumption that the data further away from the time always has lower impact on the prediction.

To overcome all the above shortcomings, we propose a deep learning based failure prediction model: Bi-LSTM, for task and job failures prediction which can adjust the weights of both closer and further input features to achieve better prediction performance.

3 Trace Analysis

The Google cluster trace [32] starts at 19:00 EDT on Sunday May 1, 2011, and it records the resource utilization of CPU and memory usage of each task on the Google cluster of about 12.5k machines for 29 days. The trace contains 672,075 jobs and more than 48 million tasks in the 29 days. This trace is a randomly-picked 1 second sample of CPU and memory usage from within the associated 5-minute usage-reporting period for each task. Each job is composed of one to tens of thousands of tasks which are shown with information such as CPU usage, memory usage, cache memory usage, mean disk I/O time, disk usage, task priority, the number of task resubmissions, task scheduling delay and etc. The priorities are classified into five categories: lowest, low, middle, high and highest priority by task scheduler and ranging from 0 to 11 which is set by cloud service provider [33].

Termination status means whether certain task is finished or not. Each job and task has several possible termination statuses. These are: (1) evicted, (2) killed, (3) failed (due to an exception or abnormal condition), and (4) finished (successfully terminated). We found that around 42 percent of the jobs and 40 percent of the tasks are not finished. Within the unfinished jobs and tasks, 97.6 percent of the jobs are failed and 97.5 percent of the tasks are failed. This is because evicted and killed jobs and tasks are very rare termination status. Therefore, we focus mainly on finished and failed jobs and tasks in our work. Specifically, we consider two kinds of failures: job failure and task failure. The job failure means the job is descheduled due to task failures. Because in the trace, a job consists of at least one task, and each task is constrained by scheduling and resource usage limits. The task failure means the task is descheduled due to a task failure such as software bugs.

3.1 Data Preprocessing

In order to build a reliable prediction system with high accuracy performance, we should understand the trace data and extract the relevant features. The previous research on Google cluster trace [32], [33] indicate that in addition to the resource usage, the following features are also correlated with job and task failures. Therefore, our prediction model additionally uses these features to leverage the prediction performance.

1) *Task priority* decides whether a task is scheduled on a machine. The work in [33] indicates that that the highest and the lowest priority jobs experience a higher rate of failures than the middle priority batch jobs.
2) *Task resubmissions* means that the tasks can be resubmitted multiple times after abnormal terminations during the life cycle of a job. We observe that the number of task resubmissions for the failed jobs is much higher than the task resubmissions for the finished jobs. The ratios of jobs with tasks that execute more than once for failed and finished jobs are 35.8 and 0.9 percent respectively and about 75 percent of the jobs have tasks that are resubmitted at most four times which is the maximum resubmission time for certain tasks.
3) *Scheduling delay* means the waiting time for certain task. We found that tasks that cannot be finished have a significant scheduling delay compare to finished tasks.

We preprocess the data to fit our prediction model and achieve better performance in cloud failure prediction. In the Google cluster trace, the data tables of system and application metrics cover task resource usage measures and various attributes of the jobs, tasks, nodes and users in separate files. We join those tables to get the associated performance data of each job and task. For each task, we combine the user ID and job ID to tag it as a unique task ID, so that each task has its information with all features and it is with the task termination status as the classification target.

3.2 Input Feature Determination

In the training phase, the previous research [23] built the prediction model in cloud by using five classes of resource usage measures which are CPU usage, memory usage, cache memory usage, mean disk I/O time, and disk usage. In our training phase, we enlarge the set of input dimensions to improve the failure prediction accuracy. So we additionally add the following features to the inputs: task priority, the number of task resubmissions, and scheduling delay. We set all the measures of these features as a *feature vector* so we can put them together at any single time point as an input. For job failure prediction, we consider that a job is predicted to be failed when any of its tasks is predicted to fail. Thus, the input of job failure prediction is the input of each task failure prediction.

4 Main Design

Next, we will introduce how we use Bi-LSTM to predict task and job failures. There are two phases: training and testing. In the training phase, we input the time series data for each task one by one to the model. Each task is labeled by failed or finished. We adjust the parameters of the model according to the comparison between the label and predicted output. In the testing phase, for a given set of time series input data of one task, the model will calculate the failure possibility of one task from the end time of the input data to the task completion time.

The process of practical for failure prediction is as follows. In the cloud system, a centralized failure prediction

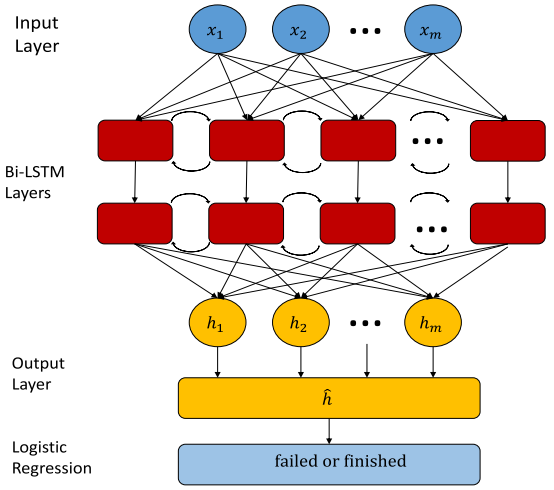


Fig. 3. Architecture of multi-layer Bidirectional LSTM (Bi-LSTM) [20].

application, with a trained failure prediction machine learning model, runs on one machine first collects and records all the resource usage information of all the machines such as CPU and memory usage in google cluster trace from the system log. Then the agent in Bi-LSTM prediction model will predict whether the tasks will be failed or finished based on the recorded data collected in the previous step. If a task is predicted to be failed, the cloud monitoring system will kill this task or job in advance and then restart it.

4.1 Model Architecture

Fig. 3 shows the architecture of Bi-LSTM model, which consists of one input layer, two Bi-LSTM layers, one output layer and the Logistic Regression (LR) layer to classify whether the tasks and jobs are failed or finished. As mentioned earlier, our Bi-LSTM based method is more advantageous than the LSTM based prediction methods in the following aspects. First, Bi-LSTM has more input features than previous methods, which include task priority, task resubmissions and scheduling delay. Second, Bi-LSTM has multi-layer structure which can better handle multiple input features for higher accuracy. Third, Bi-LSTM can determine the weights of data items based on their real impact to the failure rather than simply setting higher weights to data item closer to the given time for prediction than the data items further away from the time, which helps achieve higher prediction accuracy. Below we introduce the input layer and output layer.

4.1.1 Input Layer

In the training phase, the previous research [23] built up the prediction application in cloud by using five classes of resource usage measures which are mean CPU usage, mean memory usage, unmapped page cache, mean disk I/O, and mean disk usage. In our training phase, we try to enlarge the set of input dimensions to better address the failure prediction accuracy. So we add the following attributes as our inputs with the five classes we mentioned before, which are task priority, the number of task resubmissions, and scheduling delay. We set all the measures of these attributes as a vector so we can put them together at any time.

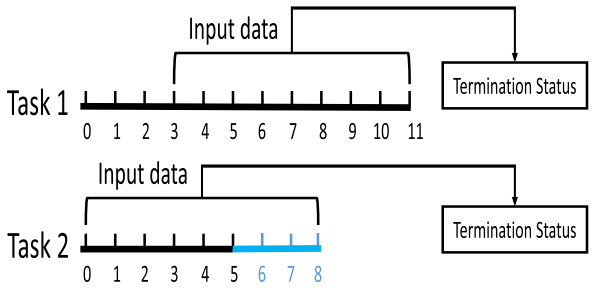


Fig. 4. An example of prediction procedure.

Given a dataset, we divide it into training set (e.g., 90 percent tasks) and testing set (e.g., 10 percent tasks). The training set is used to learn the Bi-LSTM model, and the testing set is used to predict whether a task will be failed or finished in the next time point. Each input is a feature vector at a time point. There are 100 inputs from time point $t-100$ to time point $t-1$. The output is whether the task is failed or finished at time point t. For each task, we got its data for the last 101 time points for training and testing. If the number of time points for a certain task is less than 100, we set the feature vector input as 0.

Fig. 4 shows a simple example of training and testing procedure. Each axis means the time sequence of one task. Take the testing procedure for instance, the black brackets represent the input data of the testing and the number of the data points in the brackets means the number of input data w. In this example, in task 1, the input data is the value of time = 3 to the value of time = 11, so the total number of input data is w = 9. The black part that is not in the bracket represents the data points that are not used in the testing process. In task 2, the input data is the value of time = 0 to the value of time = 8, so the total number of input data is w = 9 as well. However, task 2 is finished or stopped at time = 5. To ensure the input data is in the same format, we expand the input data in task 2 from time = 5 to time = 8. So the total number of input data can be w = 9 as well, and we set the value of time = 5 to time = 8 as 0. The blue part in the bracket represents that the task has less than 9 time points, so it is expanded by value = 0. Task 1 represents the tasks that have more data points than input data, and task 2 represents the tasks that have fewer data points than input data. The squares represent the predicted termination status, which represents the task is failed or finished.

4.1.2 Bi-LSTM Layer

Fig. 5 shows the structure of Bi-LSTM layer. The hidden states are divided into two parts: forward state and backward state. The basic idea is to present each sequence of inputs forwards and backwards to two separate hidden states to capture information, respectively. Then the two hidden states are concatenated to form the final outputs to the logistic regression functions. Unlike LSTM [27] that simply uses one forward state, Bi-LSTM uses both forward and backward states, which more accurately determines the weights of both closer and further input features. The forward state consists of the memory cells connecting adjacent neurons form cycles that are self-connections of a neuron to itself across time. The input of memory cells includes the

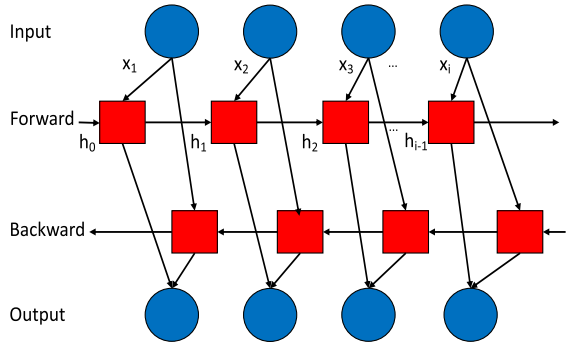


Fig. 5. Structure of Bi-LSTM layer [34].

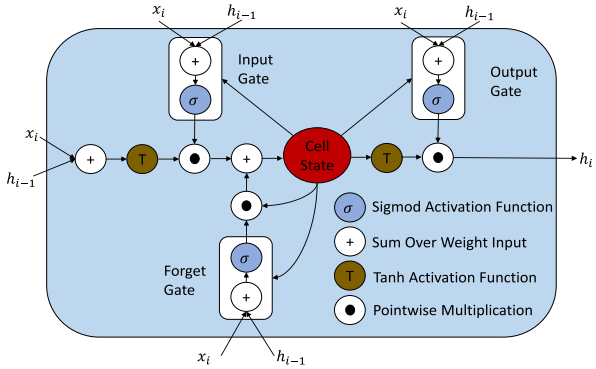


Fig. 6. Structure of memory cell in Bi-LSTM [35].

data x_i $(i = 1, 2, \ldots m)$ from the previous layer as well as the data n_{i-1} $(i = 1, 2, \ldots m)$ from themselves of the previous position. Different from the forward status, in backward state, n_{i-1} $(i = 1, 2, \ldots m)$ are from the next position. Therefore, the output of the Bi-LSTM layers can adjust the weights of data items from closer data point to the given time for prediction than the data items further away from the time. The structure of memory cell in Bi-LSTM is very suitable for processing data with sequential and time dependencies on multiple scales since the connections pass state information across time steps allowing previously processed data to affect subsequent data.

Fig. 6 shows the structure of memory cell in Bi-LSTM. To better understand the concept, we can observe from Fig. 6 that the Bi-LSTM layer provides a fine nonlinear control mechanism over what is put into the memory and removed from the memory. Each memory cell has three gates to protect and control its state, including the input gate, forget gate and output gate. The input gate determines which cell state should be updated. The forget gate decides what information should be overlooked. The output gate resolves which part of the cell state will be exported. The formulae of the memory cell can be shown as follows:

$$\begin{aligned} g_i &= \varphi(w_{gx}x_i + w_{gh}h_{i-1} + b_g) \\ n_i &= \sigma(w_{nx}x_i + w_{nh}h_{i-1} + b_n) \\ f_i &= \sigma(w_{fx}x_i + w_{fh}h_{i-1} + b_f) \\ o_i &= \sigma(w_{ox}x_i + w_{oh}h_{i-1} + b_o) \\ s_i &= g_i \odot n_i + s_{i-1} \odot f_i \\ h_i &= \varphi(s_i) \odot o_i, \end{aligned} \tag{1}$$

where $w_g x$, $w_n x$, $w_f x$, and $w_o x$ are weight coefficients of the input x_i of the memory cell.$w_g h$, $w_n h$, $w_f h$, and $w_o h$ are weight coefficients of the previous output $h_i - 1$ of the memory cell, respectively. b_g , b_n, b_f, and b_o are bias of the input node, the input gate, the forget gate, and the output gate. s_i and $s_i - 1$ are cell state values at time i and $i - 1$, respectively. As shown in Fig. 6, $\odot$ are the pointwise multiplications, σ are sigmoid functions and φ are the tanh functions. The gradients of the weight coefficients and biases are optimized by the stochastic gradient descent method.

4.1.3 Output Layer

In the output layer, from an input sequence $\{x_1, x_2 \ldots x_m\}$, the memory cells in the Bi-LSTM layers will produce a representation sequence $\{h_1, h_2 \ldots h_m\}$. We define the mean of these outputs as the mean pooling, which is expressed as follows:

$$\widehat{h} = \frac{1}{m}\sum_{i=1}^{m} h_i, \tag{2}$$

where h_i is the ith element in the representation sequence.

4.1.4 Logistic Regression Layer

In the logistic regression layer, the result from output layer is fed to the logistic regression functions whose target is the class label (i.e., failed or finished) associated with the input sequence. It means that the termination statuses generated in the Bi-LSTM model are produced based on the input features. We set up a threshold for the probability value of the failure to determine the termination statuses. Based on $\widehat{h}$, the logistic regression functions calculate the probability value of failure. If it is smaller than the threshold, it classifies as the termination statuses into failed, and otherwise, it classifies the termination statuses into finished. We use different thresholds in evaluation section to test the performance in out experiments.

Algorithm 1. Bi-LSTM Failure Prediction Algorithm

Input: Data features of each time point for each task and job
Output: Termination statuses of the tasks and jobs
select Bi-LSTM prediction model parameters
foreach *job* **do**
  foreach *task in job* **do**
    extract task features into vectors in the order of time series
    take the vectors as input to the Bi-LSTM prediction model
    predict the task termination status
  end
  calculate the number of failed tasks n
  if $n > 0$ **then**
    termination status of job is failed
  else
    termination status of job is finished
end

4.1.5 Algorithm Explanation

Algorithm 1 describes the Bi-LSTM failure prediction algorithm. We first extract task features as input data, then we send them into the Bi-LSTM prediction model to predict the task termination status. After we get all the termination status for all the tasks in each job, we can determine the termination

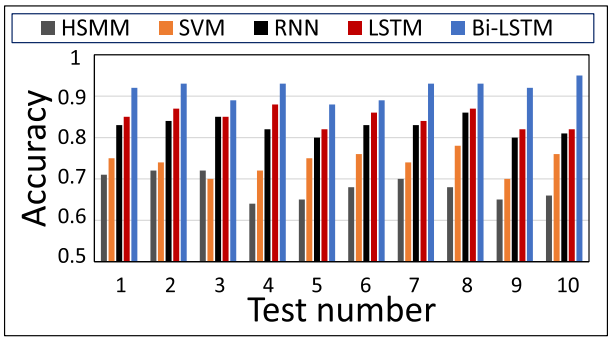


Fig. 7. Task failure prediction accuracy.

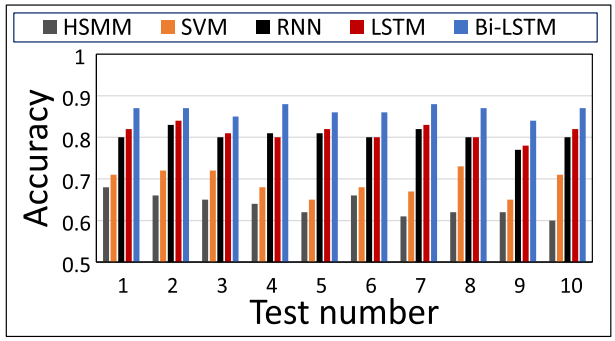


Fig. 9. Job failure prediction accuracy.

status of jobs. For job failure prediction, we consider that a job is predicted to be failed when any of its tasks is predicted to fail. So if one of the tasks that belong to certain job fails, the termination status of the job is failed, otherwise the job is finished.

5 Performance Evaluation

5.1 Experiment Setting

The experiments are deployed in our local server with a GTX 1080 GPU. The trace is originally stored in separated files of approximate size 200 GB, and the data features are represented by key-value pairs. We load these data into a MySQL database for ease of analysis. The prediction method is applied based on Tensorflow in Python. According to our experiment, to achieve better performance, we set 17 memory cells in each layer. We chose 555,555 tasks in total in our experiments, in which 50,000 tasks are for training and 55,555 tasks are for testing.

The batch size is set as 100 which means we use 100 tasks for one training step. We have 5,000 training steps for each epoch, and the number of training epochs was set to 1,000, which means we used the 500,000-task dataset 1,000 times for training. For each training and testing, we used the data of the first 100 time points of a task since many tasks are longer than 100 time points [36]. The threshold of probability value of the machine learning model is selected in the range of [0, 0.2, 0.4, 0.6, 0.8, 1] in our experiment respectively. The threshold we select for Figs. 7, 8, 9 and 10 is 0.8.

We compared our Bi-LSTM model with the prediction models used in previous researches, which are HSMM [13], SVM [15], RNN [23] and LSTM [24], to compare the prediction results under the same experiment settings. We use the same training dataset, validation dataset and test dataset to produce the experiment results. Like our method, all the comparison methods also do not consider the evicted and killed jobs because these kinds of jobs are rare. For the comparison methods, we directly used the open source code from [37], [38], [39], [40] and only changed the input and output formats.

1) *Hidden Semi-Markov Models [13]* HSMM is an extended model of HMM. Different from HMM, HSMM considers the state-resident probability distribution as explicit. It adds time components to the structure of the defined HMM and overcomes the limitations of HMM which is the prediction only depends on nearby state.
2) *Support Vector Machine [15]* SVM is also widely used in such failure prediction according to previous research. It has better performance when dealing with high-dimensional problem [41]. Also, it basically does not involve probability measures and laws of large numbers, so it is different from existing statistical methods. In essence, it avoids the traditional process from induction to deduction.
3) *Recurrent Neural Network [23]* RNN is recently used in this topic. RNN can not only deal with stationary input and output patterns but also with pattern sequences of arbitrary length [42] which means RNN can achieve better performance when handles the time series data.
4) *Long Short Term Memory (LSTM) [24]* LSTM is also recently used in this topic. It is a deep learning model which is developed from Recurrent Neural Network. Since the error-signals could exhibit exponential decay

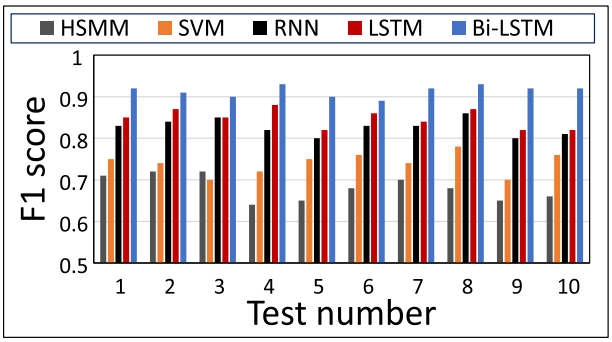


Fig. 8. Task failure prediction F1 score.

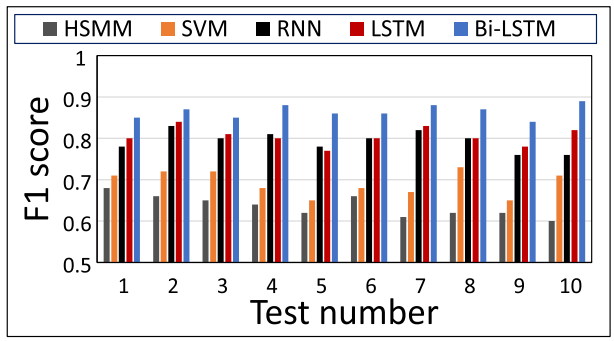


Fig. 10. Job failure prediction F1 score.

as they are back-propagated through time, which leads to long-term signal being effectively lost as they are overwhelmed by undecayed short term signals. LSTM can overcome the drawback for data with long-term dependencies also capable of modeling the temporal connections between hidden states.

5.2 Validation

We use K-fold cross validation to reduce the over-fitting that might happen in the modeling and make sure that there are no random factors affecting the prediction accuracy in order to evaluate the accuracy performance of mBi-LSTM and other comparison methods. Specifically, we divide the whole dataset into k subsets. There are k rounds of training and testing. In round i, the ith subsets is used as the testing set and the other $k-1$ subsets are put together to form a training set. Then the average error across all k trials is computed. In the previous researches [43], [44], [45], k is discussed to set as 10. So we choose k=10 in our model.

5.3 Metrics

To illustrate the performance of the our method, we use three metrics to determine the better results.

5.3.1 Accuracy and F1 Score

We determine the performance of different models by using accuracy and F1 score. The prediction accuracy is calculated by

$$A_n = 1 - \frac{|P_n - R_n|}{R_n}, \tag{3}$$

where A_n is the prediction accuracy of nth prediction, P_n is the predicted value of nth prediction and R_n is the real value in nth prediction. The Y axis value is prediction accuracy (A_n) for each method. The F1 score is calculated by

$$\begin{aligned} PPV &= TP/(TP+FP) \\ TPR &= TP/(TP+FN) \end{aligned} \tag{4}$$

$$\begin{aligned} F1 &= \frac{2 \cdot PPV \cdot TPR}{PPV + TPR} \\ &= \frac{2 \cdot TP}{2 \cdot TP + FP + FN}, \end{aligned} \tag{5}$$

where TP is true positive value, FP is false positive value, FN is false negative value. PPV is the positive predictive value, which is also known as $Precision$, TPR is the true positive rate, which is also known as $Recall$. The F1 score takes a balance that both precision and recall rates can reach the highest point at the same time.

5.3.2 Receiver Operating Characteristic (ROC)

A ROC curve is drawn by pairs of False Positive Rate (FPR) (X axis value) and True Positive Rate (TPR) (Y axis value). FPR is calculated by

$$FPR = FP/(FP+TN), \tag{6}$$

where TN is true negative value. For each prediction, it produces a pair of (FPR, TPR). The value is between range of [0, 1]. In our experiment, we calculate the average of FPR and TPR from the 10 validation tests to create and record the pair of (FPR, TPR) for each threshold of probability value of the machine learning model. To better show the performance of different model, we draw Receiver Operating Characteristic (ROC) to see the performance of different prediction methods. If the ROC curve of certain method can completely enclose the ROC curves of the others, which means the Area Under Curve (AUC) of certain method is larger. We are safe to say that the performance of certain method is better.

5.3.3 Time Cost Overhead

To maintain high Quality of Service (QoS) of applications, the failure should be detected in a short time. So not only the prediction accuracy should be considered, but also the time cost overhead. We use training time latency and testing time latency to show the time cost overhead of the prediction methods. The time cost overhead of training is calculated from the time when the first task inputs the untrained model to the time when the trained model is generated. The time cost overhead of testing is calculated from the time when all the input data is input into the trained model to the time when the prediction value generated.

5.4 Experimental Results

Now we evaluate the performance of our method and compare with previously proposed prediction methods [13], [15], [23], [24].

5.4.1 Performance Comparison

We first evaluate the task level failure prediction. At the task level, we classify the termination statuses of task submissions based on the attributes and performance data. In all the target classes, the status finish is considered as one class, and the status failed is considered as other class.

Fig. 7 shows the accuracy of each prediction method in each test among tasks. The result follows HSMM < SVM < RNN < LSTM < Bi-LSTM. For HSMM, as we discussed before, HSMM only depends on each state and its corresponding observation object. It cannot handle the sequence data in the whole trace or the high dimensional data well. SVM has better accuracy performance than HSMM because SVM can handle the data which has high dimension. However, the performance of SVM is worse than RNN and LSTM. The reason is that in data center, dataset such as Google cluster trace has a huge among of data. SVM only has better performance when the dataset is not so big [46]. For LSTM, it overcomes the drawback of RNN, which is for data with long-term dependencies. However, as we discuss before, LSTM cannot adjust the weight of further data point for a given time in the time series dataset. For Bi-LSTM, it has the forward and backward states which can more accurately determine the weights of data items that are closer and further to the given time in the prediction. We can observe that LSTM can achieve 85 percent of accuracy and Bi-LSTM can achieve 93 percent of accuracy which is higher than LSTM.

Fig. 8 shows the F1 score of each prediction method in each test among tasks. The results follow the same trend and order as in Fig. 7 due to the same reasons. The result shows that LSTM can achieve 84 percent of F1 score and

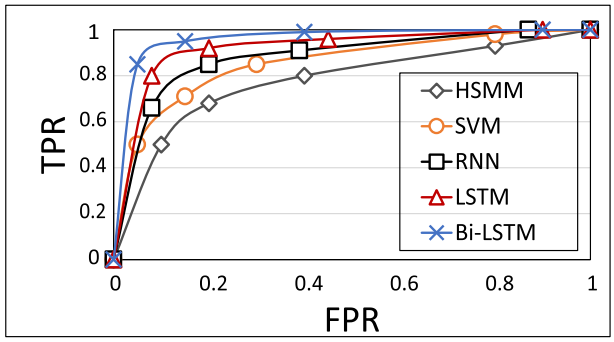


Fig. 11. ROC for different methods in different thresholds for tasks.

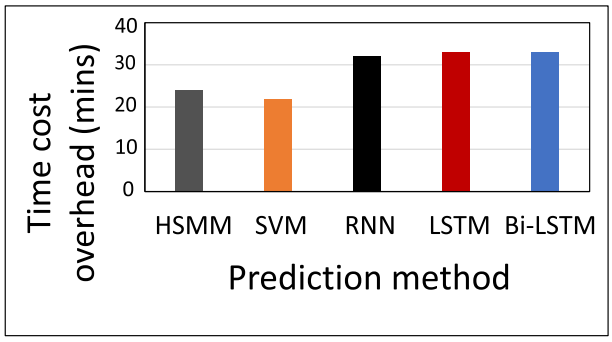


Fig. 13. Training process time cost overhead.

Bi-LSTM can achieve 92 percent of F1 score which is higher than LSTM.

Fig. 11 shows the ROC for different methods in different probability value thresholds among tasks. The result follows HSMM < SVM < RNN < LSTM < Bi-LSTM. We can observe that Bi-LSTM has the largest AUC, which means Bi-LSTM is the prediction method with the best and most stable performance. The low FPR can also indicate the proactive failure management based on prediction results become more effective compared with other methods.

At the job level, we classify the termination statuses of task submissions based on the attributes and performance data. In all the target classes, the status finish is considered as one class, and the status failed is considered as other class.

Figs. 9 and 10 show the accuracy and F1 score of each prediction method in each test among jobs. The results follow the same trend and order as in Figs. 7 and 8 because of the same reasons. We can observe that LSTM can achieve 81 percent of accuracy and 80 percent of F1 score. Bi-LSTM can achieve 87 percent of accuracy and 86 percent of F1 score which are higher than LSTM.

Fig. 12 shows the ROC for different methods in different probability value thresholds among tasks. The results follow the same trend and order as in Fig. 11 due to the same reasons.

Figs. 13 and 14 show the time cost overhead of training and testing process. We can observe that the time cost of LSTM and Bi-LSTM is similar. In this case, we can achieve higher prediction result without higher time cost, which is the key to maintain QoS of applications.

5.4.2 Performance Versus Input Size

In the large-scale cloud data centers, failures such as task and job failures become more and more common instead of occasional. For example, a task failure can cause the failure of the whole job and also make the job running time unpredictable. Failure-aware resource scheduling is therefore crucial for enhancing system availability and scalability. However, cloud schedulers still experience many failures because of unpredicted events such as unpredicted demands of services, software and hardware failures. The efficiency of the tasks and jobs scheduler requires a proactive response about changes in cloud environments. If we predict the failures in the cloud systems accurately, we can adjust scheduling decisions accordingly and reduce the amount of tasks and jobs scheduling failures.

From the prediction result, we can observe that Bi-LSTM can achieve both higher prediction accuracy and larger AUC than HSMM, SVM, RNN and LSTM at the same time. The time cost overhead indicates that Bi-LSTM can also maintain better performance with the same time cost. In our experiment, we set the input data point as 100. If we set more input data points in our experiment, the prediction result might be slightly better than the result we proposed now, but the time cost overhead will be higher too. On the other hand, if we set fewer input data points, the prediction accuracy will be lower but the time cost overhead will be lower. In the cloud failure prediction, both prediction accuracy and time cost is important to maintain QoS of applications. So we find the balance between the input data points and the time cost overhead to achieve better performance.

Figs. 15 and 16 show the accuracy of each prediction method among tasks and jobs with different input size. We choose the average accuracy of 10 test to draw these Figures. The results of accuracy follow the same trend and order as in Figs. 7 and 9 because of the same reasons. The results of accuracy with different input size follow input size=10 < input

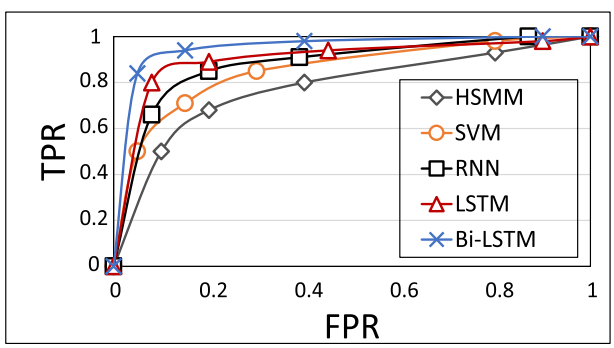


Fig. 12. ROC for different methods in different thresholds for jobs.

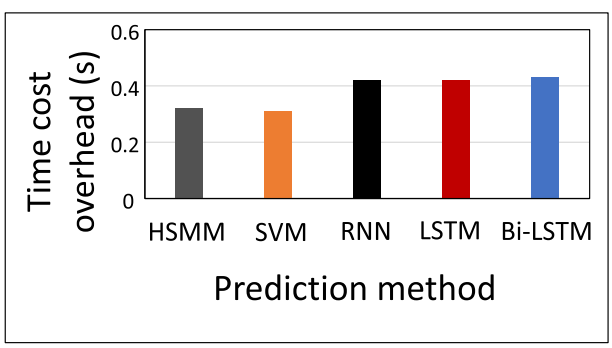


Fig. 14. Testing process time cost overhead.

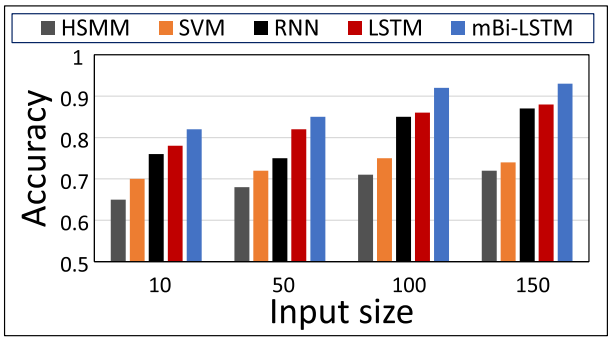


Fig. 15. Prediction accuracy with different task input size.

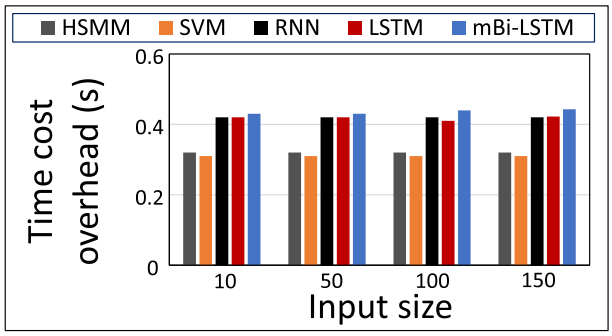


Fig. 18. Testing time cost overhead with different input size.

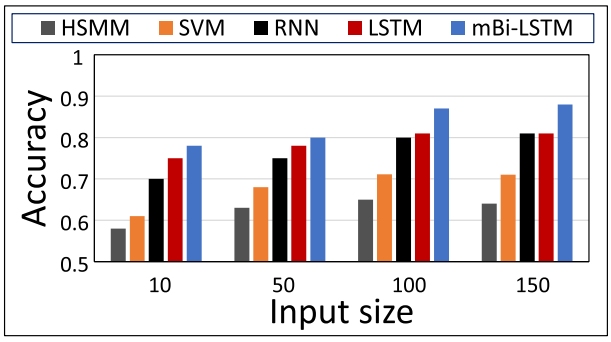


Fig. 16. Prediction accuracy with different job input size.

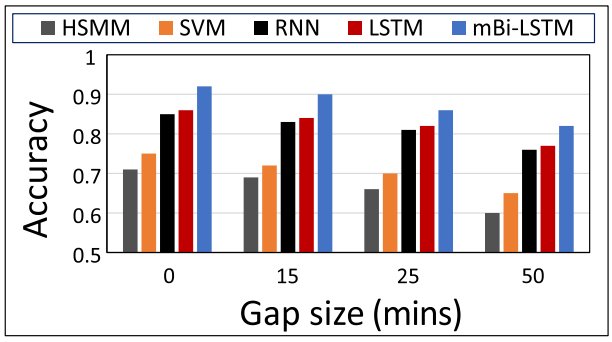


Fig. 19. Prediction accuracy with different gap size for tasks.

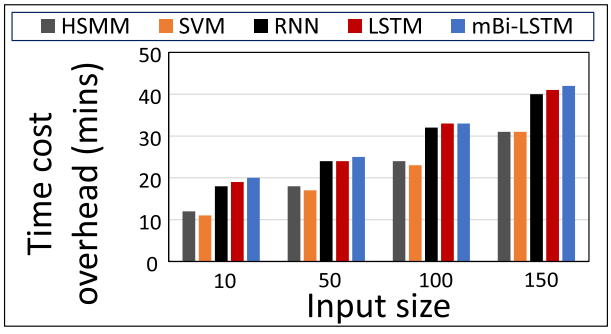


Fig. 17. Training time cost overhead with different input size.

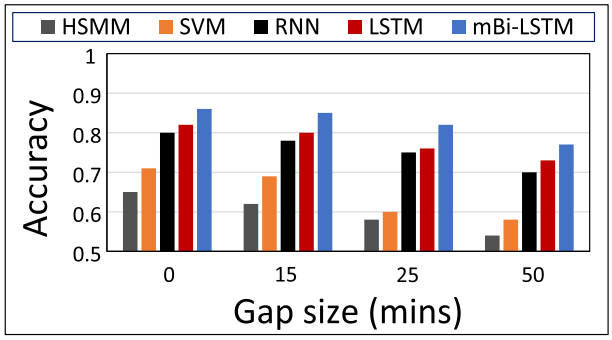


Fig. 20. Prediction accuracy with different gap size for jobs.

size=50 < input size=100≈input size=150. The reason is that, with more input data, the prediction model can be more accurate and trustworthy which reflect better prediction result. We also observe that the prediction accuracy with 100 input size and 150 input size is similar which reflects that the prediction model with 100 input size can already achieve better performance.

Since both prediction accuracy and time cost is important to maintain QoS of applications, we also compare the time cost overhead with different input sizes. The input size means the number of input data points. Figs. 17 and 18 show the time cost overhead of training and testing process for each method with different input sizes. The results of training process follow input size=10 < input size=50- < input size=100 < input size=150. The reason is that with more input data, it will take more time to train the model with all the data in the training dataset. The results of testing process follow input size=10≈input size=50≈input size=100≈input size=150. The reason is that the increasing of input data size cannot affect the time cost of testing process. So from the result above, we can observe that input size=100 can achieve similar prediction accuracy result with lower time cost.

5.4.3 *Performance Versus Gap Size*

In a real-world data center, when a failure is detected, site reliability engineers are required to prevent and fix the problem to achieve QoS. In general, SREs should take a time gap to locate and fix the failure. If the failure can be predicted before certain time gap, then we can achieve higher QoS in our system. However, prediction with certain time gap may also cause lower prediction accuracy. So we need to choose the balance between the time gap size and prediction accuracy.

Figs. 19 and 20 show the accuracy of each prediction method among tasks and jobs with different time gap size. We choose the average accuracy of 10 test to draw these Figures and the input size we used is 100. The results follow gap size=0 > gap size=10 > gap size=20 > input size=50. The

reason is that with the data nearer the prediction time, we can achieve higher accuracy. Since each data point collected in Google cluster trace is 5 mins and the typical responding time for SREs is 15 mins [47]. We can observe that Bi-LSTM can achieve 90 percent prediction accuracy with 15 mins gap size.

6 Conclusion

In cloud data centers, high service reliability and availability are crucial to application QoS. In this paper, we proposed a failure prediction model multi-layer Bidirectional LSTM (called Bi-LSTM). Bi-LSTM can more accurately predict the termination statuses of tasks and jobs using Google cluster trace compared with previous methods. In our method, we first input the data into forward state and backward state in order to adjust the weight of both closer and further input features. We then find that the further input features is essential to achieving high prediction accuracy. Second, in the experiments, we compare Bi-LSTM with other comparison methods including statistical, machine learning and deep learning based methods and evaluate the performance with three metrics: accuracy and F1 score, receiver operating characteristic and time cost overhead. The results show that we achieved 93 percent accuracy in task failure prediction and 87 percent accuracy in job failure prediction. We also achieved 92 percent F1 score in task failure prediction and 86 percent F1 score in job failure prediction. Our prediction method Bi-LSTM also has low FPR which can also indicate the proactive failure management based on prediction results become more effective. We also observe that the time cost overhead for Bi-LSTM is almost the same compared with RNN and LSTM, which means Bi-LSTM can achieve higher prediction performance with no further time cost.

In the future work, we will first try to implement our methods in real data center to examine the real-world performance and then work on how to use the failure prediction results from Bi-LSTM to build a failure recovery system in data center which can further improve the fault tolerance performance. We will also implement a job scheduling algorithm that can migrate the tasks and jobs to a suitable machine. The scheduler will be scalable and have the capability of dynamically adjusting its scheduling decisions based on the Bi-LSTM prediction results.

Acknowledgments

This research was supported in part by U.S. NSF Grants NSF-1827674, CCF-1822965, OAC-1724845, CNS-1733596 and ACI-1661378, and Microsoft Research Faculty Fellowship 8300751, and AWS Machine Learning Research Awards. The author would like to thank Liuwang Kang's help in this article. An early version of this work was presented in the Proceedings of Big Data 2019 [48].

References

[1] Overcoming the cause of data center outrages, 2019. Accessed: Apr. 2019. [Online]. Available: https://www.365datacenters.com/portfolio-items/overcoming-causes-data-center-outages/

[2] 2015. Accessed: Apr. 2019. [Online]. Available: https://techcrunch.com/2015/01/02/following-bing-coms-brief-outage-search-yah oo-com-goes-down-too/

[3] 2013. Accessed: Apr. 2019. [Online]. Available: https://nypost.com/2013/10/26/knight-capital-computer-meltdown-just-waiting-to-happen/

[4] M. Sedaghat, E. Wadbro, J. Wilkes, S. De Luna, O. Seleznjev, and E. Elmroth, "DieHard: Reliable scheduling to survive correlated failures in cloud data centers," in *Proc. 16th IEEE/ACM Int. Symp. Cluster Cloud Grid Comput.*, 2016, pp. 52–59.

[5] T. Chalermarrewong, T. Achalakul, and S. C See, "Failure prediction of data centers using time series and fault tree analysis," in *Proc. IEEE 18th Int. Conf. Parallel Distrib. Syst.*, 2012, pp. 794–799.

[6] Q. Xin, E. Miller, and S. Schwarz, "Evaluation of distributed recovery in large-scale storage systems," in *Proc. 13th IEEE Int. Symp. High Perform. Distrib. Comput.*, 2004, pp. 172–181.

[7] S. Mitra, M. Ra, and S. Bagchi, "Partial-parallel-repair (PPR): A distributed technique for repairing erasure coded storage," in *Proc. 11th Eur. Conf. Comput. Syst.*, 2016, pp. 1–16.

[8] Y. Sharma, B. Javadi, W. Si, and D. Sun, "Reliability and energy efficiency in cloud computing systems: Survey and taxonomy," *J. Netw. Comput. Appl.*, vol. 74, pp. 66–85, 2016.

[9] M. Gallet, N. Yigitbasi, B. Javadi, D. Kondo, A. Iosup, and D. Epema, "A model for space-correlated failures in large-scale distributed systems," in *Proc. Eur. Conf. Parallel Process.*, 2010, pp. 88–100.

[10] B. Schroeder and G. Gibson, "A large-scale study of failures in high-performance computing systems," *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 4, pp. 337–350, Fourthquarter 2010.

[11] A. Amin, A. Colman, and L. Grunske, "An approach to forecasting QoS attributes of web services based on ARIMA and garch models," in *Proc. IEEE 19th Int. Conf. Web Services*, 2012, pp. 74–81.

[12] R. Baldoni, L. Montanari, and M. Rizzuto, "On-line failure prediction in safety-critical systems," *Future Gener. Comput. Syst.*, vol. 45, pp. 123–132, 2015.

[13] Y. Zhao, X. Liu, S. Gan, and W. Zheng, "Predicting disk failures with HMM-and HSMM-based approaches," in *Proc. Ind. Conf. Data Mining*, 2010, pp. 390–404.

[14] J. Murray, G. Hughes, and K. Kreutz-Delgado, "Machine learning methods for predicting failures in hard drives: A multiple-instance application," *J. Mach. Learn. Res.*, vol. 6, pp. 783–816, 2005.

[15] I. Fronza, A. Sillitti, G. Succi, M. Terho, and J. Vlasenko, "Failure prediction based on log files using random indexing and support vector machines," *J. Syst. Softw.*, vol. 86, pp. 2–11, 2013.

[16] Q. Guan, Z. Zhang, and S. Fu, "Ensemble of Bayesian predictors and decision trees for proactive failure management in cloud computing systems," *J. Commun.*, vol. 7, pp. 52–61, 2012.

[17] T. Pitakrat, D. Okanović, A. van Hoorn, and L. Grunske, "Hora: Architecture-aware online failure prediction," *J. Syst. Softw.*, vol. 137, pp. 669–685, 2018.

[18] S. Zhang *et al.*, "PreFix: Switch failure prediction in datacenter networks," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, 2018, Art. no. 2.

[19] C. Xu, G. Wang, X. Liu, D. Guo, and T. Liu, "Health status assessment and failure prediction for hard drives with recurrent neural networks," *IEEE Trans. Comput.*, vol. 65, no. 11, pp. 3502–3508, Nov. 2016.

[20] Y. Cheng, H. Zhu, J. Wu, and X. Shao, "Machine health monitoring using adaptive kernel spectral clustering and deep long short-term memory recurrent neural networks," *IEEE Trans. Ind. Informat.*, vol. 15, no. 2, pp. 987–997, Feb. 2019.

[21] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1285–1298.

[22] A. Das, F. Mueller, C. Siegel, and A. Vishnu, "Desh: Deep learning for system health prediction of lead times to failure in HPC," in *Proc. 27th Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2018, pp. 40–51.

[23] X. Chen, C. Lu, and K. Pattabiraman, "Failure prediction of jobs in compute clouds: A Google cluster case study," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops*, 2014, pp. 341–346.

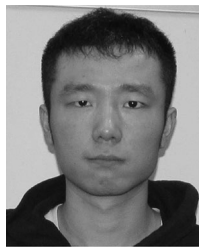
[24] T. Islam and D. Manivannan, "Predicting application failure in cloud: A machine learning approach," in *Proc. IEEE Int. Conf. Cogn. Comput.*, 2017, pp. 24–31.

[25] P. Angeline, G. Saunders, and J. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 54–65, Jan. 1994.

[26] F. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with LSTM," *Neural Comput.*, vol. 12, pp. 2451–2471, 2000.

[27] S. Wang, X. Wang, S. Wang, and D. Wang, "Bi-directional long short-term memory method based on attention mechanism and rolling update for short-term load forecasting," *Int. J. Electr. Power Energy Syst.*, vol. 109, pp. 470–479, 2019.
[28] D. Ford *et al.*, "Availability in globally distributed storage systems," in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, 2010, pp. 61–74.
[29] V. Padmanabhan, S. Ramabhadran, S. Agarwal, and J. Padhye, "A study of end-to-end web access failures," in *Proc. ACM CoNEXT Conf.*, 2006, pp. 1–13.
[30] R. Birke, I. Giurgiu, L. Chen, D. Wiesmann, and T. Engbersen, "Failure analysis of virtual and physical machines: Patterns, causes and characteristics," in *Proc. 44th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, 2014, pp. 1–12.
[31] S. Kandula, R. Mahajan, P. Verkaik, S. Agarwal, J. Padhye, and P. Bahl, "Detailed diagnosis in enterprise networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 243–254, 2010.
[32] C. Reiss, A. Tumanov, G. Ganger, R. Katz, and M. Kozuch, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. 3rd ACM Symp. Cloud Comput.*, 2012, pp. 1–13.
[33] X. Chen, C. Lu, and K. Pattabiraman, "Failure analysis of jobs in compute clouds: A Google cluster case study," in *Proc. IEEE 25th Int. Symp. Softw. Rel. Eng.*, 2014, pp. 167–177.
[34] P. Zhou *et al.*, "Attention-based bidirectional long short-term memory networks for relation classification," in *Proc. 54th Annu. Meeting Assoc. Comput. Linguistics*, 2016, pp. 207–212.
[35] Z. Huang, W. Xu, and K Yu, "Bidirectional LSTM-CRF models for sequence tagging," 2015, *arXiv:1508.01991*.
[36] J. Kumar, R. Goomer, and A. Singh, "Long short term memory recurrent neural network (LSTM-RNN) based workload forecasting model for cloud datacenters," *Procedia Comput. Sci.*, vol. 125, pp. 676–682, 2018.
[37] Hidden semi markov model, 2015. Accessed: Apr. 2020. [Online]. Available: https://github.com/takumayagi/pyhsmm
[38] Support vector machine, 2019. Accessed: Apr. 2020. [Online]. Available: https://github.com/ShankyTiwari/Support-Vector-Machine-algorithm-in-python
[39] Recurrent neural network, 2018. Accessed: Apr. 2020. [Online]. Available:https://github.com/jiegzhan/time-series-forecasting-rnn-tensorflow
[40] Long short term memory, 2019. Accessed: Apr. 2020. [Online]. Available: https://github.com/jaungiers/LSTM-Neural-Network-for-Time-Series-Prediction
[41] T. Joachims, "Making large-scale SVM learning practical," Tech. Rep., 1998.
[42] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," 2015, *arXiv:1506.00019*.
[43] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. 14th Int. Joint Conf. Artif. Intell.*, 1995, pp. 1137–1143.
[44] G. Golub, M. Heath, and G. Wahba, "Generalized cross-validation as a method for choosing a good ridge parameter," *Technometrics*, vol. 21, pp. 215–223, 1979.
[45] S. Arlot and A. Celisse, "A survey of cross-validation procedures for model selection," *Statist. Surv.*, vol. 4, pp. 40–79, 2010.
[46] J. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Process. Lett.*, vol. 9, pp. 293–300, 1999.
[47] Q. Zhang *et al.*, "Deepview: Virtual disk failure diagnosis and pattern detection for azure," in *Proc. 15th Symp. Netw. Syst. Des. Implementation*, 2018, pp. 519–532.
[48] J. Gao, H. Wang, and H. Shen, "Task failure prediction in cloud data centers using deep learning," in *Proc. Int. Conf, Big Data*, 2019, pp. 1111–1116.

Jiechao Gao (Student Member, IEEE) received the BS degree from Jilin University, China, in 2016, and the MS degree from Columbia University, New York, in 2018. He is currently working toward the PhD degree with the Department of Computer Science, University of Virginia, Charlottesville, Virginia. His research interests include distributed networks, cloud computing, machine learning algorithms and applications.



Haoyu Wang (Student Member, IEEE) received the BS degree from the University of Science & Technology of China, China, and the MS degree from Columbia University, New York. He is currently working toward the PhD degree with the Department of Computer Science, University of Virginia, Charlottesville, Virginia. His research interests include data center, cloud and distributed networks.

Haiying Shen (Senior Member, IEEE) received the BS degree in computer science and engineering from Tongji University, China, in 2000, and the MS and PhD degrees in computer engineering from Wayne State University, Detroit, Michigan, in 2004 and 2006, respectively. She is currently an associate professor with the Department of Computer Science, University of Virginia, Charlottesville, Virginia. Her research interests include distributed computer systems, cloud computing, big data and cyber-physical systems. She is a Microsoft Faculty fellow of 2010, and a senior member of the ACM.