2017 IEEE 24th International Conference on Web Services

Online Reliability Prediction via Long Short Term Memory for Service-Oriented Systems

Hongbing Wang* Zhengping Yang* Qi Yu†
*School of Computer Science and Engineering and Key Laboratory of Computer Network and Information Integration
Southeast University, NanJing 210096, China, 211189
hbw@seu.edu.cn, zpyang@seu.edu.cn
†College of Computing and Information Sciences, Rochester Institute of Technology, 102 Lomb Memorial Drive
Rochester, NY 14623-5608, USA
qi.yu@rit.edu

***Abstract*—A service-oriented System of System (SoS) integrates component services into a value-added and more complex system to satisfy the complex requirements of users. Due to a dynamic running environment, online reliability prediction for the loosely coupled component systems that ensures the runtime quality poses a major challenge and attracts growing attention. To guarantee the stable and continuous operation of systems, we propose a online reliability time series prediction method basing on long short term memory (LSTM), which is a modified Recurrent Neural Networks trained with historical reliability time series to predict the reliability of component systems in the near future. We conduct a series of experiments on a dataset composed of real web services and compare with other competitive approaches. Experimental results have demonstrated the effectiveness of our approach.**

***Keywords*-Online Reliability Prediction, Time Series, Service-Oriented Computing, System of Systems**

I. Introduction

Recently, System of Systems (SoS) has attracted significant attention with the growing demand for large-scale software systems, which satisfy users' complex requirements. It creates a new system by integrating existing component systems into a value-added and more complex one [1], [2]. Service-Oriented Architecture (SOA) has been widely adopted to build service-oriented SoS [3]. A service-oriented SoS put a strong emphasis on interaction, collaboration and communication between the various components of the system. Attentions have been placed in monitoring, detection and protection of the whole system because of the loosely coupled relationship among individual systems. Furthermore, a service-oriented SoS usually operates in the highly dynamic environment. As a result, the changes in individual systems may cause a cascading effect, and finally force the whole system to shut down [4]. Consequently, how to ensure the runtime Quality of Service (QoS) for a service-oriented SoS has become a challenge and attracted more and more attention nowadays.

To guarantee the stable and continuous operation of systems, Proactive Fault Management (PFM) which can improve the reliability of systems was proposed [5]. Research on self-*properties, e.g., configuration, healing, optimization, or protection, has been an important direction for quality assurance of individual systems [6]. Specifically, to achieve PFM, the component systems should be selected optimally when integrating a service-oriented SoS [7]. It presents new challenges to the existing reliability prediction approaches. In order to support self- optimization, the reliability prediction must be conducted in a nearly real-time (or online) fashion, which can forecast the reliability in the near future.

The duration time of each invoked component system may vary from one to another as different users act differently and the communication links may be unstable. In addition, the reliability may fluctuate when a component is under invocation [7]. In order to address the challenges above, the online reliability prediction approach should manage the changes of variable prediction period. In most cases, when a predetermined prediction period is long enough, we can divide it into multiple intervals and predict the reliability of those time periods, referred to as, the reliability time series. In other words, the main goal of online reliability prediction for service-oriented SoS is to predict the time series of the future period of time, based on current system state and historical records.

Existing online failure prediction approaches are designed for traditional systems that are not as complex as a service-oriented SoS that usually runs in a highly dynamic running environment. Failure Tracking based approaches heavily rely on statistical characteristics of historical errors in time distribution, and neglect the failure mechanisms [5], [8], [9]. Furthermore, approaches, like Symptom Monitoring, Detected Error Reporting, Undetected Error Auditing need to analyze the internal parameters of the system (or log files), which are difficult to obtain for a service-oriented SoS. These various kinds of online prediction methods, such as conditional probability based Bayesian forecast method, nonparametric prediction method, curve fitting method, the semi-Markov model, SVM model, component interaction

978-1-5386-0752-7/17 $31.00 © 2017 IEEE
DOI 10.1109/ICWS.2017.19

graph model, and collaborative filtering technology [10], [11], [12], are only partially suitable for online reliability prediction of a service-oriented SoS system, because the reliability varies with time. There is no distinct regularity in the changes of reliability time series, which makes these existing methods insufficient for a service-oriented SoS.

To deal with the challenges above, we propose an online reliability time series prediction method basing on LSTM, which is a modified Recurrent Neural Networks (RNNs). RNNs have advantages in dealing with sequential patterns. It is inspired by the cyclical connectivity of neurons in the brain and uses iterative function loops to store information. They can learn what to store and what to forget which makes them process context information flexibly and recognize the relations and features of sequences more accurately. A RNN takes previous information into consider, and trains with the information of the previous one time point. It is hard to learn the relevance of reliability between time points because the reliability waves with time. However, the adjacent two points may still possess relatively strong correlation as the changes won't appear and then disappear in very short period of time. It will take some time to influence the reliability of component systems.

However, a RNN suffers from long-term dependencies and the vanishing gradient problem if the length of sequence grows [13]. To deal with these two problems, Long Short Term Memory (LSTM) was proposed [14]. There are many other modified RNN methods which solve the two problems more or less, such as Bidirectional RNNs, Deep (Bidirectional) RNNs, and GRU (Gated Recurrent Unit Recurrent Neural Networks). Bidirectional RNNs take sequences in future into consideration, which is not necessary in our problem; Deep (Bidirectional) RNNs have multiple levels in each step, which is not efficient; GRU(Gated Recurrent Unit Recurrent Neural Networks) is similar to LSTM in structure and performs closely to each other in general [15]. As a result, we choose to adopt LSTM in this work and other methods may be experimented in future.

In summary, the major contributions of this paper are summarized as below:

1) We build upon and extend our prior work [7] by developing a new method for online reliability time series prediction.
2) We present the online reliability time series prediction method basing on LSTM .
3) We conduct a series of experiments to verify effectiveness of our method and compare it with other competitive approaches.

The rest of this article is organized as follows. Section 2 gives an overview of related research. Our method is elaborated in Section 3 . Section 4 presents the experimental results. Finally, Section 5 concludes our present work and outlines some future research directions.

II. Related work

In this section, we given an overview of existing works that are most relevant to the proposed approach, including reliability predication for service-oriented systems and online prediction methodologies.

A service-oriented SoS is achieved via service computing technologies [3]. Thus, we regard it as an implementation of a service-oriented system. A service-oriented system usually runs in a dynamic environment and SoS adds further complexity to it. Fault tolerance has been proposed in order to tackle a dynamic running environment, so that the runtime quality can be guaranteed [4]. To ensure the stable execution of service-oriented systems, reliability prediction has become more and more important. A number of works have been conducted in recent years.

Yu et al. take the user difference into consideration and then predict the QoS of services [16], [17]. In [18], Clustering (CLUS)-based reliability prediction is proposed to estimate the reliability of atomic service invocations according to the historic invocations. The overall system reliability can be calculated by integrating the results of component services by considering the compositional structures [19]. Influenced by the approaches above, we propose an online reliability prediction method for service-oriented systems. It predicates the reliability of running services that is in favor of the runtime quality. In fact, online reliability prediction has been widely researched in recent years. *Online Failure Prediction* aims to predict whether a failure may occur in the near future or not. Existing online failure prediction methods can be classified into three categories [5].

1) *Failure Tracking* based approaches aim to predict the upcoming failures using the occurrence and types of previous failure. Most works are based on probability distributions [10], [20]. Another approach is to use statistical methods (or correlation analysis) to analyze the correlation between different failures [21].
2) *Symptom Monitoring* based approaches detect the state of the system so as to predict the failure that will happen in the future. There are four most frequently used methods, namely Function Approximation, Classifiers, System Models, and Time Series Analyses [22], [23], [11].
3) *Detected Error Reporting* analyzes the reports collected via some logging mechanism and predicts the upcoming failures. Typical methods include rule-based methods, pattern recognition methods such as semi-Markov chain, and heuristic algorithms [24], [25], [26].

The traditional online failure prediction methods are designed for non-network systems. In contrast, a service-oriented SoS is loosely coupled and runs in the dynamic environment which makes the traditional methods insufficient. In this paper, we will employ *Long Short Term*

Memory (LSTM), a Recurrent Neural Network, to predict the online reliability time series for the component systems of a service-oriented SoS.

III. Time Series Prediction

A. Problem Formulation

In this section, we formally define the prediction problem. To ensure the stable operation of a software system, reliability has attracted wide attention as one of the most important measures to quantify software quality [27]. It can be measured in different forms, such as Mean Time To Failure (MTTF), Mean Time Between Failures (MTBF), hazard rate (the system survives till time t), etc. Since the components of a SoS usually operates in a highly dynamic environment, there is no regularity of failure occurrence. This means approaches like MTTF, MTBF and hazard rate are not suitable for the problems mentioned in our paper. We measure the reliability based on the failure rate that is calculated by counting the numbers of failure invocations over the total invocations during a fixed time period. We use the metric proposed in [7] to calculate the reliability of the components in a SoS.

Definition 1 (**Reliability**). *Assume that there is a component system S and a client node C. When C invokes S, the reliability of S is calculated according to the failure rate. Let u represent a time period. We formally define the reliability of S at time t_u evaluated by C as:*

$$r(u) = e^{-\gamma \times len(u)}, \tag{1}$$

where $len(u)$ represents the length of time period u, $\gamma = {}^{n_f}\!/_{N}$ is the failure-rate of equal length time-interval client-side invocation evaluations for S, N represents the number of total invocation times, and n_f denotes the number of invocations with failure responses or performance anomalies, which deviate from the user (or application system) requirement.

Based on the definition of the reliability in one time period, we give the definition of reliability time series below.

Definition 2 (**Reliability Time Series**). *Let Δt_p represent the prediction period. We divide it into n equal intervals, i.e., $u_1, u_2, \cdots, u_n$. The corresponding online reliability time series to be predicted in Δt_p, the future time, is defined as r_p which is vector describing the reliability values in multiple time points, i.e.,*

$$r_p = (r_{u_1}, r_{u_2}, \cdots, r_{u_n}), \tag{2}$$

where n represents the number of time points, and r_{u_i} represents the i-th time point's reliability value, and $i \in [1, n]$.

B. Foundation of the Prediction Models

Recurrent neural networks have the form of a chain of repeating modules of a neural network. In standard RNNs, this repeating modules will have a very simple structure, such as a single $tanh$ layer (see Figure 1). LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for a long period of time is practically their default behavior, not something they struggle to learn. LSTMs also have a chain like structure, but the repeating modules have a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way (see Figure 2).

The architecture of a simple LSTM is shown in Figure 3. A complex LSTM network may have multiple memory cells. The standard RNNs cannot handle the sequences as shown in Figure 3. This is because the input information decreases exponentially and has little contribution to later outputs, also known as the vanishing gradient problem.

The LSTM architecture consists of a set of recurrently connected subnets, known as memory blocks. Each block contains one or more self-connected memory cells and three multiplicative units: the input, output, and forget gates, which provide continuous analogues of write, read and reset operations for the cells [14].The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.The cell state works like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to flow along it unchanged. The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures, called gates.

Figure 2 provides an illustration of a LSTM memory block with a single cell, where i, f, o and c are input gate, forget gate, output gate, and cell activation vectors. The multiplication gates enable the LSTM memory cells to store and access information over long periods of time, thus reduce the vanishing gradient problem. These three gates control the cell state after collecting activations from inside and outside the block. The sigmoid layer outputs numbers between zero and one and describes how much of each component should be let through. A value of zero means "let nothing through", while a value of one means "let everything through". The function of the gates will be enumerated and briefly described below.

C. LSTM-based Prediction Approach

Suppose that we have the record of historical reliability time series $x = (x_1, \cdots, x_T)$. The later time points can be seen as the future reliability results to the earlier time points, so we have the expected reliability $y = (y_1, \cdots, y_T)$. In this section, we will describe our model, i.e., the LSTM-based online reliability time series prediction. The training process will be explained in detail.

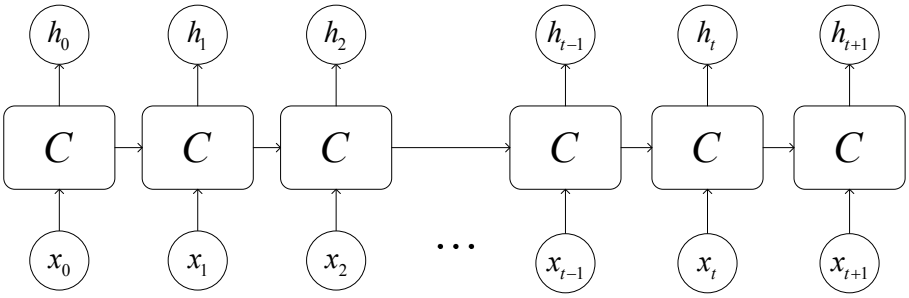

Figure 3. An LSTM network.

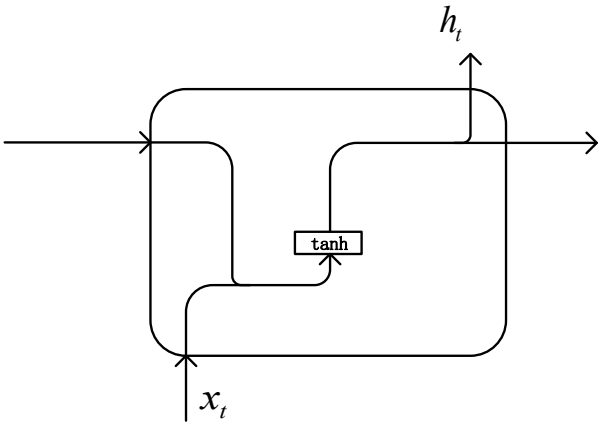

Figure 1. A module of standard RNNs

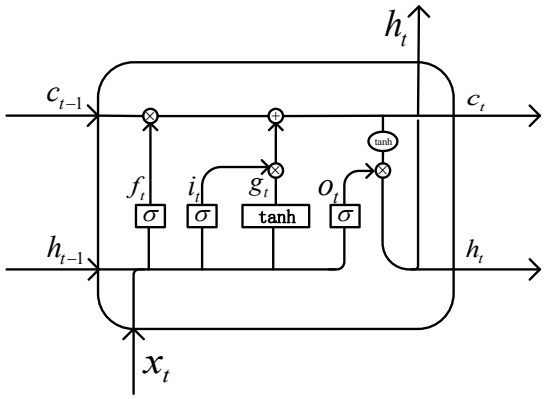

Figure 2. A cell for Long Short-term Memory.

First, the forget gate decides what the information to throw away from the cell states. We take x_t and h_{t-1} into consideration:

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right) \tag{3}$$

where W terms denote weight matrices (e.g. W_{xi} is the input-hidden weight matrix), the b terms denote bias vectors (e.g. b_i is hidden bias vector), and σ is the logistic sigmoid function.The forget gate outputs a number between 0 and 1 for each number in the cell state c_{t-1}. One implies that we will keep the cell state c_{t-1} while zero represents we won't keep it. The next step is to decide what information to store in the cell state. This contains two parts. In the first one part, the input gate will decide which state to be updated by:

$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + b_i\right) \tag{4}$$

Next, a tanh layer creates a vector of new candidate values, g_t, which could be added to the state. In the next step, we'll combine these two to create an update to the state.

$$g_t = \tanh\left(W_{xg}x_t + W_{hg}h_{t-1} + b_g\right) \tag{5}$$

Then we can update the old cell state c_{t-1} to a new cell state c_t. The results of previous steps have already decided what to do. We multiply the old state by f_t to forget the things we want do forget earlier. We add the $i_t g_t$ because it is the new new candidate values, scaled by how much we decide to update each state value.

$$c_t = f_t c_{t-1} + i_t g_t \tag{6}$$

Then, we decide what we want to output. We should run through a sigmoid layer first. And then we put the cell state through tanh (to push the values to be between 0 and 1). Afterwards, we multiply it by the output of the sigmoid gate, so that we only output the parts we like.

$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + b_o\right) \tag{7}$$

$$h_t = o_t \tanh\left(c_t\right) \tag{8}$$

We have $x_t^c = [x_t, h_{t-1}]$. So parts of the equations above can be written as follows:

$$f_t = \sigma\left(W_f x_t^c + b_f\right) \tag{9}$$

$$i_t = \sigma\left(W_i x_t^c + b_i\right) \tag{10}$$

$$g_t = \tanh\left(W_g x_t^c + b_g\right) \tag{11}$$

$$o_t = \sigma\left(W_o x_t^c + b_o\right) \tag{12}$$

There we end the forward pass, it is time for reverse conduction. We use h_t and y_t, namely the prediction result and the true value at time t to compute the loss l_t. Our goal is to minimize l_t.

$$l_t = f(h_t, y_t) = \|h_t - y_t\|^2 \tag{13}$$

$$L = \sum_{1}^{T} l_t \tag{14}$$

Our ultimate goal in this case is to use gradient descent to minimize the loss L over an entire sequence of length T. Furthermore, we need to adjust the parameters w, where w is a scalar parameter of the model (e.g., it may be an entry in the matrix W_g).

$$\frac{\mathrm{d}L}{\mathrm{d}w} = \sum_{t=1}^{T}\sum_{j=1}^{M} \frac{\mathrm{d}L}{\mathrm{d}h_t(j)} \frac{\mathrm{d}h_t(j)}{\mathrm{d}w} \tag{15}$$

where $h_t(i)$ is the result corresponding to the $i-th$ memory cell's hidden output and M is the total number of memory cells. For notational convenience, we introduce the variable L_t, which represents the cumulative loss from time t.

$$L_t = \sum_{a=t}^{T} l_a \tag{16}$$

Since the network propagates forward, so we have:

$$\frac{\mathrm{d}L}{\mathrm{d}h_t(j)} = \sum_{a=1}^{T} \frac{\mathrm{d}l_a}{\mathrm{d}h_t(j)} = \sum_{a=t}^{T} \frac{\mathrm{d}l_a}{\mathrm{d}h_t(j)} = \frac{\mathrm{d}L_t}{\mathrm{d}h_t(j)} \tag{17}$$

Consequently,

$$\frac{\mathrm{d}L}{\mathrm{d}w} = \sum_{t=1}^{T}\sum_{j=1}^{M} \frac{\mathrm{d}L_t}{\mathrm{d}h_t(j)} \frac{\mathrm{d}h_t(j)}{\mathrm{d}w} \tag{18}$$

Afterwards, we can adjust the weight matrices W and the bias vectors b. Keep training the LSTM network with forward pass and reverse conduction until loss L converges.

D. Solution

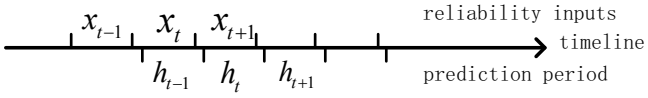

Figure 4. Reliability time series.

After the network is trained successfully, we can utilize the trained network to make predictions. Specifically, it is similar to the forward pass of training. In our model, the relation with historical reliability time series x_t and the prediction results h_t is shown in Figure 4. In our paper, we aim to prediction the reliability time series in near future δt_p period. It contains N time series and the length of each time period is u. To predict r_p in section III-A mentioned, We can take the historical reliability records in T adjacent time periods into the model, then we have the prediction result $h = (h_1, \cdots, h_T)$, where h_T will be the reliability of the component in u_1 time period, e.g., $r_{u_1} = h_T$. Afterwards, let $x_0 = x_1$, $x_1 = x_2$,...,$x_T = r_{u_1}$, and predict again to get the new h_T. Let$r_{u_2} = h_T$, and then update the historical training set x_t with r_{u_2}. Continue to predict by taking the historical reliability time series into the model and repeat these steps above until we have all r_p estimated.

We present the complete process of our LSTM-based online reliability time series prediction in Algorithm 1.

Input: The set of historical reliability time series for training, x_t, $t \in [1, T]$;
The set of real reliability prediction time series for training set, y_t, $t \in [1, T]$;
The number of reliability time series to predict in the near future, N;
Output: The reliability prediction result in future, r_p;
1: Training the LSTM network and computing the reliability time series h_t according to the weight matrices W, the bias vectors b and the input training set x_t, the historical reliability time series;
2: Computing the loss L between the prediction result h_t and the real result y_t;
3: Adjusting the weight matrices W and the bias vectors b to more proper values;
4: **for all** $i \in [1, N]$ **do**
5: Taking the historical T reliability time series to predict, and let the reliability in future $r_{u_i} = h_T$;
6: **for all** $t \in [1, T-1]$ **do**
7: $x_t = x_{t+1}$;
8: **end for**
9: $x_T = r_{u_i}$, updating the historical training set with the new prediction result.
10: **end for**
11: **return** r_p.

Algorithm 1: LSTM-based online reliability time series prediction.

IV. Experimental Evaluation

In this section, we present an experimental evaluation for our approach. As stated in Section I, owing to the similar structure with SOA, the components of service-oriented SoS can be seen as the web services. Therefore, we conduct the experiments using Web services. In our paper, we use the dataset in [7], include 100 representative Web services from different domains and locations. It contains the parameters of service state in 24 hours, which are then divide it to 4320 time series.

A. Methods for Comparison

We will compare with some existing methods to justify the effectiveness of our approach mentioned in [5].

- **Average Value of Historical Reliability (*AVHR*)**: The reliability prediction is based on collaborative filtering approaches. It uses the prediction on multiple clients together to determine the final result. We use the average historical prediction values as *AVHR* for comparison.
- **Regression (*Reg*)**: This method is trained using the historical reliability time series via regression analysis. It then estimates the reliability time series in future. We choose the least square fitting function.
- **Bayes' Rules (*BR*)**: This method is based on Bayes' Rules. It collects the statistics on the conditional probability of motifs conversion and finds the nearest motif to the observed up-to-date reliability time series. The corresponding prediction motif will be the prediction result.
- **Online Reliability Prediction(*ROP*)**: This method was proposed in our previous work [7]. It uses Probabilistic Graphical Models to handle the reliability time series prediction. Specifically, it discovers motifs from historical time series, and label time series using the most similar motifs. Then it constructs conditional probability table to express the causal relationships of reliability time series and to predict the reliability time series in future.

B. Evaluation Metrics

We employ the widely adopted MAE (Mean Absolute Error) and RMSE (Root Mean Square Error) to evaluate the prediction accuracy of different prediction approaches.

$$MAE = \frac{\sum_{n=1}^{N} |p_n - o_n|}{N} \tag{19}$$

$$RMSE = \sqrt{\frac{\sum_{n=1}^{N} (p_n - o_n)^2}{N}} \tag{20}$$

where N represents the total number of predictions, p_n is the prediction reliability result and o_n is the observed reliability value. Smaller values of MAE and RMSE indicate better prediction accuracy.

C. Impact of Training Set Scale

To study how the size of the training set effects the prediction accuracy, we conduct a set of experiments on 1500, 2000, 2500, 3000, 3500, 4000, 4500 reliability time series respectively. Furthermore, we utilize *LSTM*, *AVHR*, *BR*, *Reg* to predict separately, where the number of motifs in *BR* is set as 25. The results are shown in Figure 5.

We can see from Figure 5 that all the four methods' MAE and RMSE metric values decrease as the scale of training set become larger. That is to say, larger training set scale brings better prediction results. However, our LSTM-based prediction approach shows no advantage with other methods at the very beginning. When the number of time series for training increases, its accuracy become higher, even overtake the other methods around 3000 time series taken into training. What's more, the increasing trend slows down when the size of training set reaches 3500 in the graph. In sum, if the training set scale is large enough, our method will perform better than the other four methods. Therefore, the size of training set should be set in a appropriate range. Training beyond this range indicates increased time cost in calculating which may not lead to further accuracy increase.

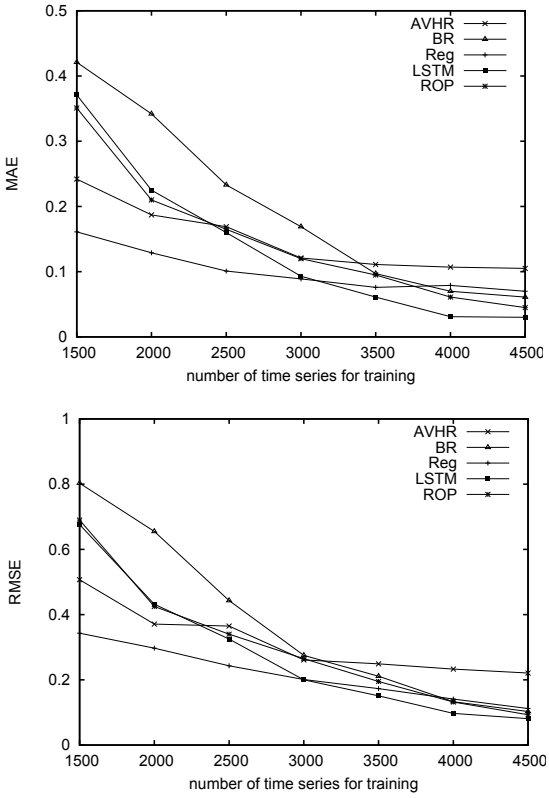

Figure 5. Prediction accuracy as impacted by the training set scale.

D. Impact of Prediction Period

As mentioned in section I, the invocation time of component services is not fixed. It depends on the user's behavior. In our paper, we consider the length for each time series is aphotic. Consequently, the prediction period δt_p may contain several time series which will influence the accuracy of prediction results. So we conduct a set of experiments when δt_p is 10s, 20s, 30s, 40s, 50s, 60s, respectively, to observe the impact of prediction period. The experiment trains 4500 time series. The results are shown in Figure 6.

As can be seen from Figure 6, in addition to the AVHR method, as the length of prediction period increases, the

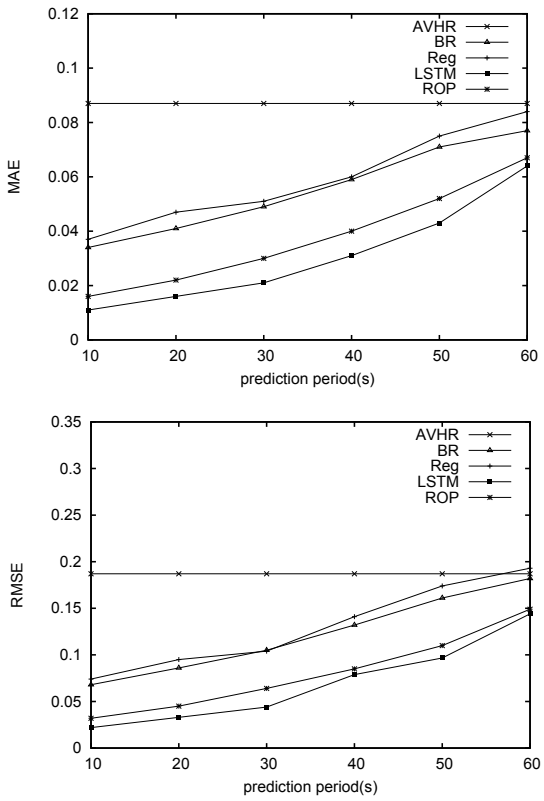

Figure 6. Prediction accuracy as impacted by the prediction period.

prediction accuracy of all the three other method increases. Besides, although LSTM increase sharply and the others rise in a slow trend, LSTM-based prediction method performs better comparing with the other approaches for the MAE and RMSE values of LSTM are always the smallest in our experiments. The BR and Reg method may behavior better when the prediction period δt_p become longer, but for short-term prediction problem, the LSTM method perform fits better.

E. Discussion

From the experiments above, the effectiveness of LSTM-based online reliability time series prediction approach has been demonstrated.

We conduct two sets of experiments to observe the impact of training set scales and prediction periods respectively. We compare our method with other three methods that are in common use. The LSTM-based prediction method shows its advantage in short-term prediction problem. Also, when we train with sufficient historical reliability time series, the LSTM-based approach will forecast with a higher accuracy.

V. Conclusion and Future Work

In this paper, we present an online reliability time series prediction method based on LSTM. We conduct a series of experiments to verify the effectiveness of our method and compare it with other competitive approaches. For future work, we intend to improve our work in two-fold. First, in our experiments, we evaluate the effectiveness of our approach based on a dataset with parameter changes of Web services in 24 hours. This time period is relatively short and may not show the overall information of Web services. Consequently, we plan to conduct our experiments on a larger and more comprehensive dataset to further demonstrate the effectiveness of our approach. Second, there may be many other service parameters that have impacts on reliability, such as throughout and response time. However, we only take reliability into consideration, which may not be adequate to ensure the runtime quality. As a result, we will update the model that considers the impacts of these factors, simultaneously. Last, one limitation of our LSTM-based approach is that we predict by iteration when the prediction period contain multiple time series. This may affect the prediction accuracy for longer term prediction problems. Therefore, it is necessary to further optimize the structure of our LSTM-based approach.

Acknowledgment

This work was partially supported by NSFC Projects (Nos. 61672152, 61232007, 61532013), Collaborative Innovation Centers of Novel Software Technology and Industrialization and Wireless Communications Technology.

References

[1] M. Jamshidi, *System of systems engineering: innovations for the twenty-first century*. John Wiley & Sons, 2011, vol. 58.

[2] A. Sousa-Poza, S. Kovacic, and C. Keating, "System of systems engineering: an emerging multidiscipline," *International Journal of System of Systems Engineering*, vol. 1, no. 1-2, pp. 1–17, 2008.

[3] K. J. Rothenhaus, J. B. Michael, and M. T. Shing, "Architectural patterns and auto-fusion process for automated multisensor fusion in soa system-of-systems," *IEEE Systems Journal*, vol. 3, no. 3, pp. 304–316, 2009.

[4] M. N. Huhns and M. P. Singh, "Service-oriented computing: key concepts and principles," *Internet Computing IEEE*, vol. 9, no. 1, pp. 75–81, 2005.

[5] F. Salfner, M. Lenk, and M. Malek, "A survey of online failure prediction methods," *Acm Computing Surveys*, vol. 42, no. 3, p. 10, 2010.

[6] O. Babaoglu, M. Jelasity, A. Montresor, C. Fetzer, S. Leonardi, A. V. Moorsel, and M. V. Steen, "Self-star properties in complex information systems," *Lecture Notes in Computer Science*, 2005.

[7] H. Wang, L. Wang, Q. Yu, Z. Zheng, M. Lyu, and A. Bouguettaya, "Online reliability prediction via motifs-based dynamic bayesian networks for service-oriented systems," *IEEE Transactions on Software Engineering*, 2016, doi:10.1109/TSE.2016.2615615.

[8] Z. Xue, X. Dong, S. Ma, and W. Dong, "A survey on failure prediction of large-scale server clusters," in *Eighth Acis International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/distributed Computing*, 2007, pp. 733–738.

[9] Q. Guan, Z. Zhang, and S. Fu, "Ensemble of bayesian predictors and decision trees for proactive failure management in cloud computing systems," *Journal of Communications*, vol. 7, no. 1, pp. 52–61, 2012.

[10] A. Csenki, "Bayes predictive analysis of a fundamental software reliability model," *IEEE Transactions on Reliability*, vol. 39, no. 2, pp. 177–183, 1990.

[11] G. A. Hoffmann, K. S. Trivedi, and M. Malek, "A best practice guide to resource forecasting for computing systems," *IEEE Transactions on Reliability*, vol. 56, no. 4, pp. 615–628, 2008.

[12] S. Malefaki, S. Trevezas, and N. Limnios, "An em and a stochastic version of the em algorithm for nonparametric hidden semi-markov models," *Communication in Statistics-Simulation and Computation*, vol. 39, no. 2, pp. 240–261, 2009.

[13] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 5, pp. 157–166, 1994.

[14] A. Graves, *Long Short-Term Memory*. Springer Berlin Heidelberg, 2012.

[15] J. Chung, C. Gulcehre, K. H. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *Eprint Arxiv*, 2014.

[16] Q. Yu, "Qos-aware service selection via collaborative qos evaluation," *World Wide Web*, vol. 17, no. 1, pp. 33–57, 2014.

[17] Z. Zheng and M. R. Lyu, "Personalized reliability prediction of web services," *Acm Transactions on Software Engineering and Methodology*, vol. 22, no. 2, pp. 3–4, 2013.

[18] M. Silic, G. Delac, and S. Srbljic, "Prediction of atomic web services reliability based on k-means clustering," in *Joint Meeting on Foundations of Software Engineering*, 2013, pp. 70–80.

[19] Z. Zheng and M. R. Lyu, "Collaborative reliability prediction of service-oriented systems," in *ACM/IEEE International Conference on Software Engineering*, 2010, pp. 35–44.

[20] J. D. Pfefferman and B. Cernuschi-Frias, "A nonparametric nonstationary procedure for failure prediction," *IEEE Transactions on Reliability*, vol. 51, no. 4, pp. 434–442, 2002.

[21] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R. Sahoo, "Bluegene/l failure analysis and prediction models," in *International Conference on Dependable Systems and Networks*, 2006, pp. 425–434.

[22] K. Vaidyanathan and K. S. Trivedi, "A measurement-based model for estimation of resource exhaustion in operational software systems," in *International Symposium on Software Reliability Engineering*, 1999, pp. 84–93.

[23] A. Andrzejak and L. Silva, "Deterministic models of software aging and optimal rejuvenation schedules," in *Ifip/ieee International Symposium on Integrated Network Management*, 2007, pp. 159–168.

[24] R. Vilalta, C. V. Apte, J. L. Hellerstein, and S. Ma, "Predictive algorithms in the management of computer systems," *Ibm Systems Journal*, vol. 41, no. 3, pp. 461–474, 2002.

[25] R. Durbin, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press,, 1998.

[26] D. Levy and R. Chillarege, "Early warning of failures through alarm analysis - a case study in telecom voice mail systems," in *International Symposium on Software Reliability Engineering*, 2003, pp. 271–280.

[27] I. Standard, "Ieee standard glossary of software engineering terminology," *IEEE Std*, vol. 42, no. 3, pp. 112 – 118, 1990.