ICIC Express Letters
Volume **16**, Number **11**, November **2022**
ICIC International ©2022 ISSN 1881-803X
pp. 1185–1191

DISTRIBUTED PROCESSING OF DEEP LEARNING INFERENCE MODELS FOR MALICIOUS URL DETECTION

Hyojong Moon[1], Siwoon Son[2] and Yang-Sae Moon[1,*]

[1]Department of Computer Science
Kangwon National University
1 Gangwondaehakgil, Chuncheon-si, Gangwon-do 24341, Korea
moonhyojong@kangwon.ac.kr; *Corresponding author: ysmoon@kangwon.ac.kr

[2]CybreBrain Section
Future and Basic Technology Research Division
Electronics and Telecommunications Research Institute (ETRI)
218 Gajeong-ro, Yuseong-gu, Daejeon 34129, Korea
siwoon.son@etri.re.kr

Received December 2021; accepted March 2022

Abstract. *Stacking used for improving the accuracy of deep learning models incurs a long inference time due to its high complexity, and it further increases the latency significantly in data stream environments. In particular, the long latency is a severe problem in detecting malicious URLs since the real-time detection is a critical requirement. In this paper, we propose a distributed processing technique that efficiently processes a stacking-based inference model of detecting malicious URLs in data stream environments. Distributed algorithms may vary greatly in processing performance depending on their configurations, so we propose four different configurations: Independent Stacking, Sequential Stacking, Semi-Sequential Stacking, and Stepwise-Independent Stacking. We then evaluate the four configurations by comparing the latency and resource usage occurring in processing URL streams. Experimental results show that Stepwise-Independent Stacking, which has the property of both independent and sequential executions, is the most efficient configuration by providing the shortest latency.*
Keywords: Distributed computing, Deep learning, Stacking, Malicious URL detection

1. Introduction.
Recently, the ensemble technique [1] is widely used to improve the accuracy of deep learning models [2, 3]. Representative ensemble techniques include bagging, boosting, and stacking. Among them, we focus on stacking that retrains a new model using the result values predicted by multiple models as input features. However, the stacking-based model has the high computational complexity due to the use of multiple models, which greatly increases the inference time. This long inference time also causes long latency in data stream environments.

Since the data stream occurs quickly and continuously, the latency increases as time passes. In order to solve this problem, we use real-time distributed processing systems, and representative ones include Apache Storm [4], Spark [5], and Flink [5]. However, since a stacking-based model takes a long time to get the inference result, it will still incur the long latency even in the distributed processing system.

In this paper, we address how to efficiently process a stacking-based inference model of detecting malicious URLs in a data stream environment. In particular, we propose four distributed processing configurations using Apache Storm to efficiently handle a complex stacking model. The proposed configurations are Independent Stacking (*IS*), Sequential Stacking (*SS*), Semi-Sequential Stacking (*SSS*), and Stepwise-Independent Stacking (*SIS*).

DOI: 10.24507/icicel.16.11.1185

According to the experimental results on latency and resource usage, the SIS configuration shows the best performance by executing bolts, Storm's components, independently and at the same time sequentially.

Contributions of the paper can be summarized as follows. First, we present four different stacking models, each of which can detect malicious URLs in a distributed way. Second, we propose how to implement the stacking models in Apache Storm using its components of spouts and bolts. Third, through various actual experiments, we show that Stepwise-Independent Stacking is the best stacking model in detecting malicious URLs in the distributed environment.

The rest of the paper is organized as follows. Section 2 describes the related work on Apache Storm and stacking techniques. Section 3 presents the proposed models for distributed processing of malicious URL detection. Section 4 shows the results of experimental evaluation. Finally, Section 5 concludes the paper.

2. **Related Work.** Apache Storm [4] is an open source distributed framework developed by Twitter. A topology defines a series of input-processing-output tasks required for handling data streams in Storm [6]. This topology consists of a number of spouts and bolts. The spouts receive data, convert it into a tuple, which is the data type used by Storm, and deliver it to the bolts. The bolts receive tuples from the spouts or the previous bolts, and process their tasks. They then transmit the results to the next bolts, or send the results to the output or storage device [4, 6].

Stacking [1] first trains different types of base models, and then retrains a new stacking model using predicted values of the base models as input features. Unlike other ensemble techniques, it combines different types of independent but complete models. However, since the stacking-based model combines multiple independent models, its inference complexity is significantly higher than that of a single model. This high complexity causes a long inference time, and accordingly, incurs a long latency in a data stream environment. In this paper, we first propose a stacking model that performs malicious URL detection. We then present and evaluate four different configurations for distributed processing of this stacking model.

3. **Distributed Processing of Malicious URL Detection.** Figure 1 shows the structure of the proposed malicious URL detection model exploiting the stacking technique. In this stacking model, we use three base models: CNN (Convolutional Neural Network) [7], LSTM (Long Short-Term Memory) [8], and GRU (Gated Recurrence Unit) [9] for malicious URL detection. As shown in the figure, we train the stacking model composed of the Fully-Connected layer (*FC layer*) by using the predicted values of the three base models of CNN, LSTM, and GRU.

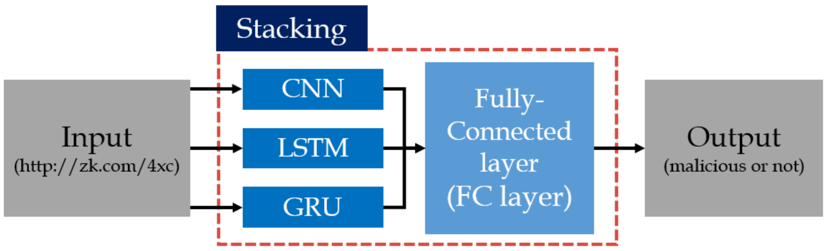


FIGURE 1. Stacking model of malicious URL detection

Figure 2 shows the detailed structures of CNN, LSTM, and GRU used for stacking. First, the CNN model [7] uses 1DCNN to classify URLs, which are one-dimensional sequence data. It consists of feature extraction and classification steps as shown in Figure 2(a). Second, the LSTM model [8] consists of LSTM, Dropout, and FC layers as

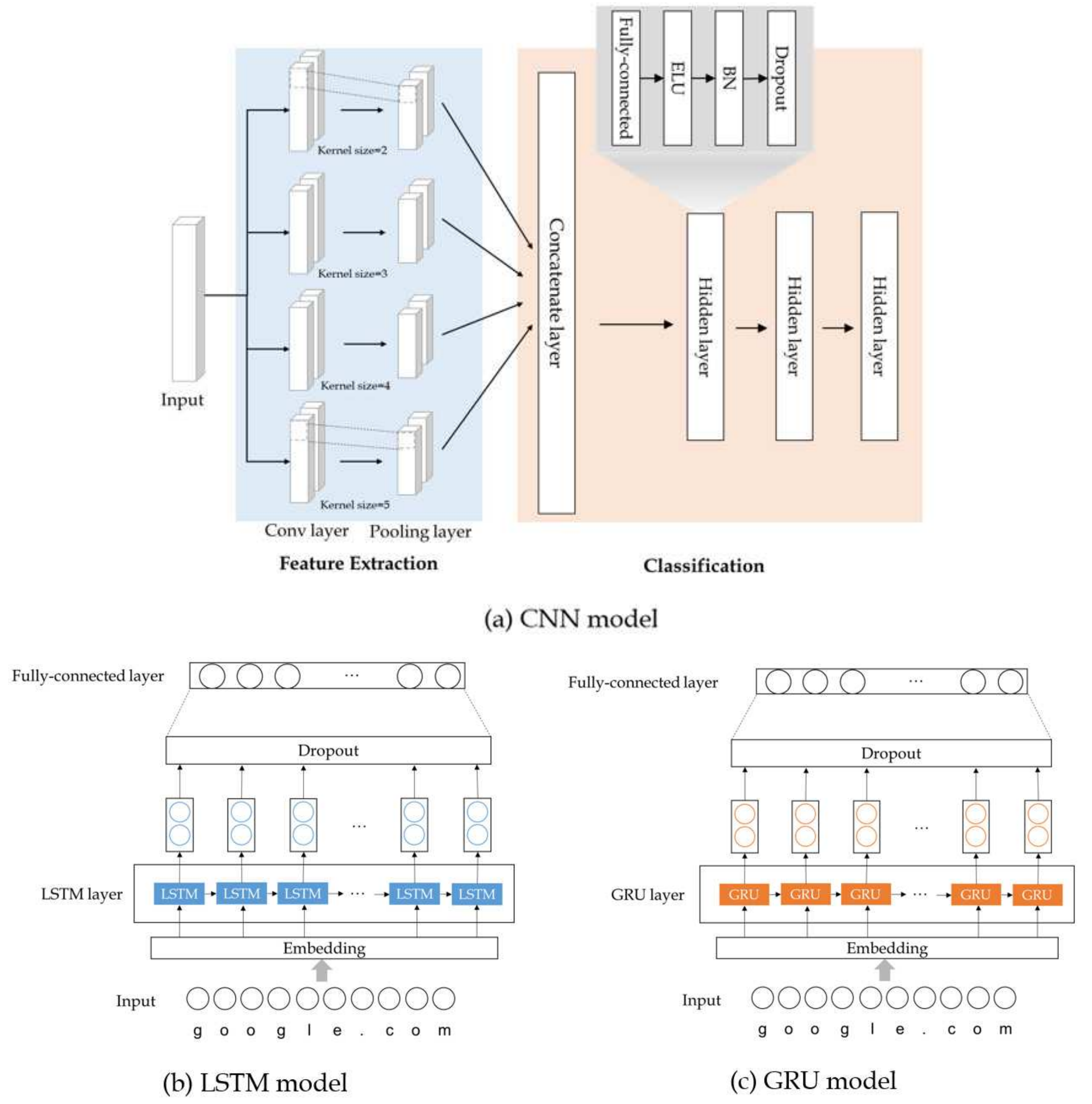


FIGURE 2. Base models used in malicious URL detection

shown in Figure 2(b). We use 128 as the output dimension of the LSTM layer and 0.5 as the dropout ratio to prevent overfitting as in [8]. Third, the GRU model [9] has less computational amount than LSTM, and is composed of GRU, Dropout, and FC layers. Figure 2(c) shows the GRU model, which is very similar to the LSTM model, where we use 128 and 0.5 as the training parameters same as in the LSTM model. The stacking model integrates and retrains the three predicted values output from these CNN, LSTM, and GRU models. The FC layer in the stacking model of Figure 1 combines three models and uses ReLU as an activation function.

We propose four different configurations to process the stacking model in a distributed manner. IS (*Independent Stacking*) separates the base models in Apache Storm and runs them independently to reduce the latency required to classify URL streams. Figure 3(a) shows the internal configuration of IS. First, URLs are inputs through spout, and the spout transmits the URL data to CNN, LSTM, and GRU bolts. Next, each inference bolt runs its base model independently, and transmits the result to the next FC layer bolt. Finally, the FC layer bolt performs inference again on the three prediction values, and provides the final result through the output bolt. However, each inference time required

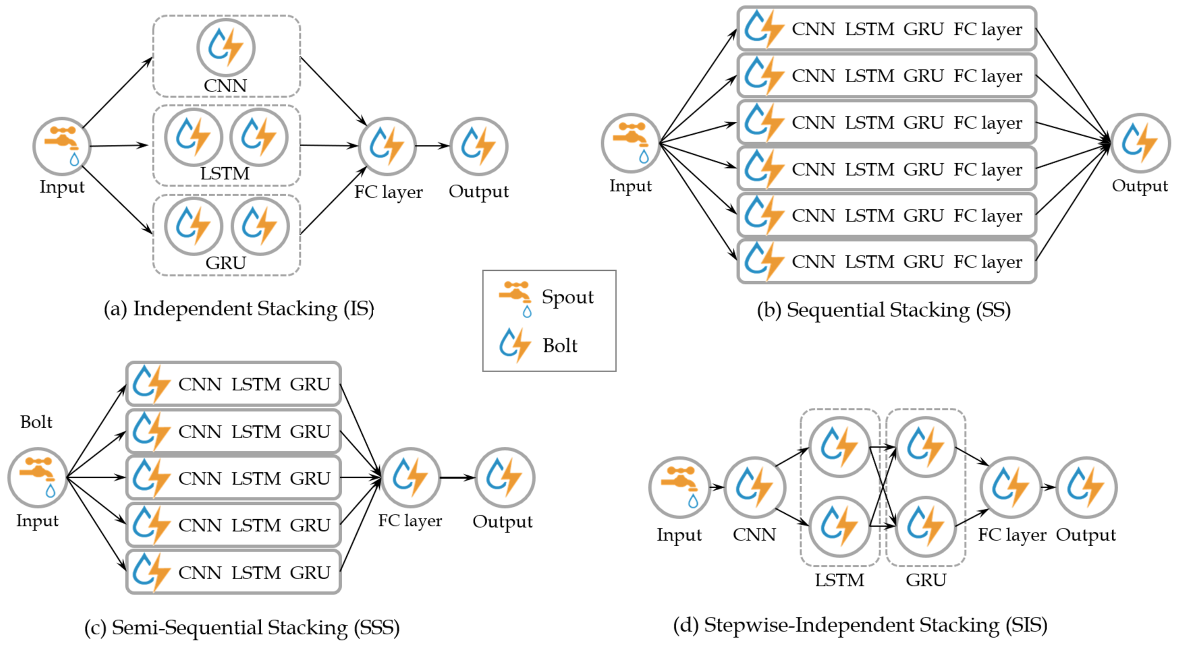


FIGURE 3. Configurations of the proposed stacking models

for CNN, LSTM, and GRU bolts is different, so the time for the predicted values to arrive at the FC layer bolt is also different. Thus, we need an additional operation to map each prediction value to the same URL data. That is, the FC layer bolt needs to wait while mapping the predicted values, and proceeds inference after completing the mapping.

SS (*Sequential Stacking*) tries to reduce network communication by sequentially executing all inference models in one bolt. Instead, by increasing the parallelism of bolts, it reduces the latency to classify the URL stream. Figure 3(b) shows this sequential configuration of SS. First, the input spout gets URL data and transmits it to the stacking bolt. Next, the stacking bolt predicts the value by executing all base models as well as the FC layer sequentially. Due to the sequential executions, each model needs to wait until the predecessor model completes its inference. Finally, the output bolt provides the prediction result.

SSS (*Semi-Sequential Stacking*) separates the FC layer from the base models to reduce the waiting time of SS. Also, in order to reduce the latency, it increases the parallelism of the bolts that execute the base models. Figure 3(c) shows the configuration of SSS. First, the spout gets and transmits URL data to the bolt of handling base models. Next, the model bolt executes three base models sequentially, and transmits the three predicted values to the FC layer bolt. Finally, the FC layer bolt infers the final result, and the output bolt returns the result.

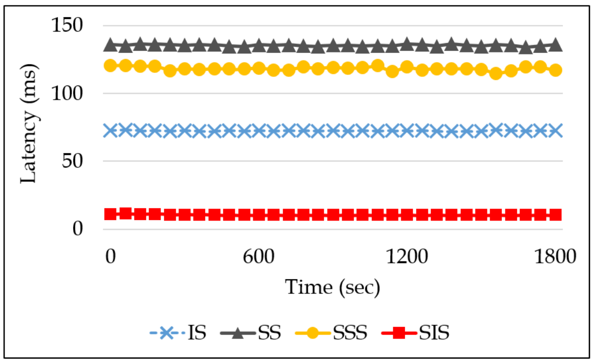
SIS (*Stepwise-Independent Stacking*) runs all base models independently like in IS, but it differs from IS in transmitting the predicted values of each model in sequence. This approach aims at achieving both independent processing of multiple models and sequential processing of data mappings. That is, SIS runs base models simultaneously like IS, and at the same time, eliminates the time-consuming task of data mappings unlike IS. Figure 3(d) shows this independent and sequential configuration of SIS. First, the input spout gets URL data and transmits it to the next CNN bolt. Second, the CNN bolt processes the URL data and transmits <CNN's predicted result, URL data> to the LSTM bolt. Third, the LSTM bolt processes the URL data and transmits <CNN's & LSTM's predicted results, URL data> to the GRU bolt. Fourth, the GRU bolt processes URL data again and transmits three predicted results to the FC layer bolt. Finally, the FC layer bolt obtains the final result, and the output bolt provides the result.

4. Experimental Evaluation.

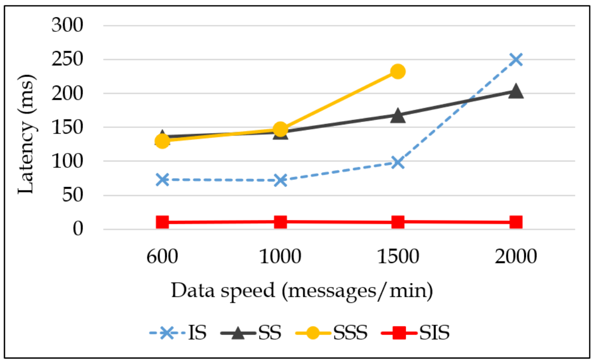
In the experiment, we compare the latency and resource usage for the proposed four configurations. The hardware platform is a distributed cluster consisting of one Intel Xeon E5-2630V3 2.4GHz 8 Core server as a master node and eight Intel Xeon E5-2630V3 2.4GHz 6 Core servers as slave nodes. Each node is equipped with 32GB RAM and 256GB SSD, and uses CentOS as the operating system. The spout generates URL data continuously and transmits it as input data.

4.1. Evaluation on the data generation speed.

First, we compare the latency required to classify the URL stream at the fixed data generation speed. All four configurations use one spout for input, one bolt for output, and six bolts for the inference model. The spout generates and transmits the data evenly at a rate of 600 URLs per minute. Figure 4(a) shows a graph comparing the latency to infer each URL in four configurations. As shown in the graph, SIS shows the shortest latency all the time. This is because in the case of SIS, each bolt operates independently, and the results are processed sequentially and immediately. Next, IS shows the second shortest latency due to the waiting time required for the mapping. SS and SSS have very low performance because three or four models are executed sequentially within the same bolt. We note that SSS reduces the latency compared to SS by separating the FC layer from the base models.



(a) 600 URLs/min

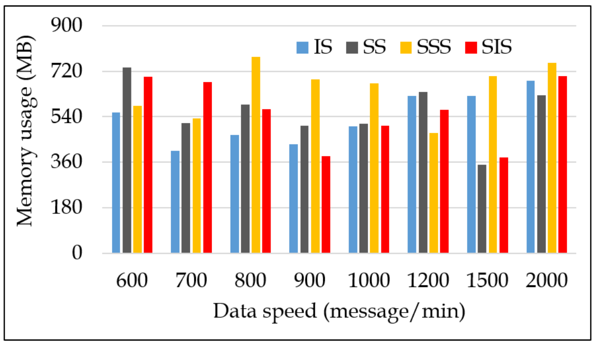


(b) 600~2000 URLs/min

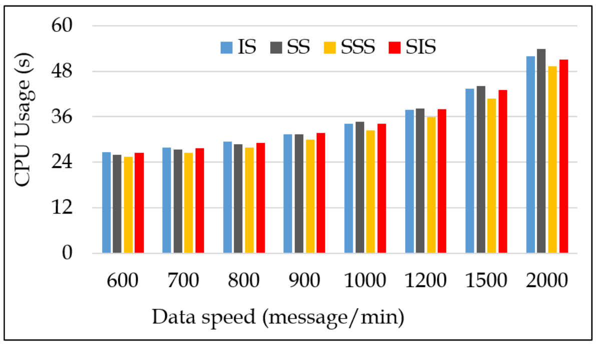
FIGURE 4. Latency comparison by the data generation speed

Next, we compare the latency and resource usage at each generation speed while increasing the speed of generating URL data. We generate the data at a rate of 600 to 2000 URLs per minute. For all four configurations, we use one input spout, one output bolt, and six inference bolts as in Figure 4(a). Figure 4(b) shows the comparison graph of the latency at each generation speed. Like Figure 4(a), SIS shows the shortest latency at all generation speeds. This is because SIS's independent and sequential execution mechanism makes predictions very fast, even when data generation speeds up. Next, IS shows the second best performance, but the latency tends to increase as the generation speed increases due to the data mapping operation. In particular, from the generation speed of 1500 per minute, IS's performance rapidly deteriorates, and the latency becomes even longer than that of SS. This is because IS solves the long latency problem of sequential execution with independent execution of base bolts, but it incurs a severe load on LSTM and GRU bolts due to insufficient parallelism. SS and SSS show relatively long latency compared to SIS and IS, and SSS becomes worse than SS as generation speed increases.

Figure 5 shows the resource usage in each configuration. First, the memory usage of Figure 5(a) differs every time in the four configurations as the generation speed increases, which seems to depend on the operating system situation and actually shows no meaningful difference. The CPU usage of Figure 5(b) shows a trend of increasing gradually in all configurations because the faster the generation speed, the greater the amount of data to be processed. Among them, SS and SSS show low CPU usage in the relatively



(a) Memory usage



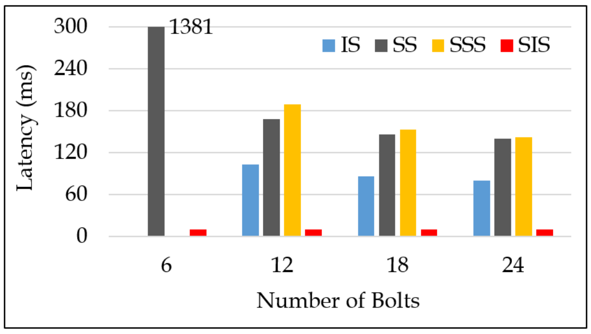
(b) CPU usage

FIGURE 5. Resource usage comparison by the data generation speed

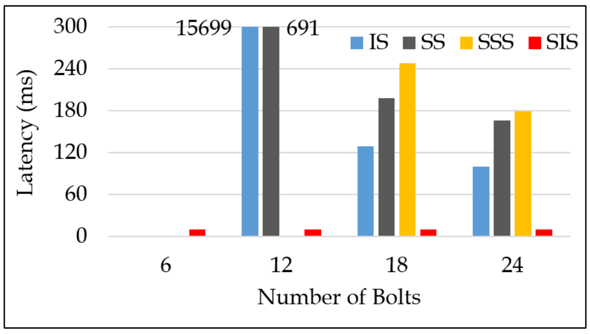
low generation speed. However, since SS processes all models within one bolt, the faster the generation speed, the longer the latency. Next, IS that needs mapping and SIS that needs to duplicate tuples show relatively high CPU usage. In the figure, SSS shows the lowest CPU usage, since it does not require additional operations such as mapping or duplicating tuples. However, not all configurations show significant differences in actual usage numbers.

4.2. **Evaluation on the parallelism.** In this experiment, we increase the number of bolts for each proposed configuration and compare the latency for different data generation speeds. We change the number of bolts to 6, 12, 18, and 24, and increase the parallelism by assigning the more bolts to the complex models that take a relatively long processing time.

Figure 6 compares the latency as the parallelism increases. First, Figure 6(a) shows the latency for the generation speed of 3000 per minute. As shown in the figure, SIS shows the shortest latency even at high parallelism due to its independent execution and sequential processing structure. IS still has a longer latency compared to SIS. This is because, even if we increase the number of bolts, it still consumes a relatively long waiting time for the mapping operation. SS and SSS also improve the performance as the parallelism increases, but they cannot overcome SIS or IS due to the structure of sequentially executing multiple models in one bolt. Figure 6(b) shows the latency at the generation speed of 6000 per minute, and we cannot measure the rest other configurations than SIS when the number of bolts is six. As the parallelism increases, the overall latency gradually decreases, and finally, the trend becomes similar to that of Figure 6(a). As a result, the more inference bolts we use, the shorter latency all four configurations have, but SIS with the best model structure shows the shortest latency in all cases.



(a) 3000 URLs/min



(b) 6000 URLs/min

FIGURE 6. Latency comparison by the parallelism increase

5. **Conclusions.** The stacking-based malicious URL detection model is much more complex than a single inference model, so it takes long time to infer data. Thus, if we simply apply this stacking technique to a data stream environment, it would incur a critical problem of long latency. To operate the stacking-based inference model of malicious URL detection efficiently in a data stream environment, in this paper we proposed to use distributed configurations using Apache Storm. We presented four stacking configurations: IS, SS, SSS, and SIS, and evaluated their latency and resource usage through various experiments. Experimental results showed that SIS, which executed Storm bolts both independently and sequentially, was much superior to other configurations by providing the shortest latency. As the future research work, we first investigate to use bagging and boosting models in addition to stacking models, and we also try to exploit Apache Spark as a distributed processing system in addition to Apache Storm.

Acknowledgment. This work was partly supported by Institute of Information & communications Technology Planning & evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2020-0-00077, Core Technology Development for Intelligently Searching and Utilizing Big Data based on DataMap) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1A2C1085311).

REFERENCES

[1] T. G. Dietterich, Ensemble methods in machine learning, *Proc. of Int'l Workshop on Multiple Classifier Systems*, Berlin, Germany, pp.1-15, 2000.

[2] J. Namgung, S. Son and Y.-S. Moon, Efficient deep learning models for DGA domain detection, *Security and Communication Networks*, vol.2021, Article ID: 8887881, 2021.

[3] S. Omer and R. Lior, Ensemble learning: A survey, *WIREs Data Mining and Knowledge Discovery*, vol.8, no.4, p.1249, 2018.

[4] R. Evans, Apache Storm, a hands on tutorial, *Proc. of 2015 IEEE Int'l Conf. on Cloud Engineering*, Tempe, Arizona, p.2, 2015.

[5] S. Chintapalli et al., Benchmarking streaming computation engines: Storm, Flink and Spark streaming, *Proc. of the IEEE Int'l Conf. on Parallel and Distributed Processing Symposium Workshops*, Chicago, IL, pp.1789-1792, 2016.

[6] S. Yang, S. Son, M.-J. Choi and Y.-S. Moon, Performance improvement of Apache Storm using InfiniBand RDMA, *Journal of Supercomputing*, vol.75, no.10, pp.6804-6830, 2019.

[7] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou and C. Wang, Machine learning and deep learning methods for cybersecurity, *IEEE Access*, vol.6, pp.35365-35381, 2018.

[8] S. Akarsh, S. Sriram, P. Poornachandran, V. K. Menon and K. P. Soman, Deep learning framework for domain generation algorithms prediction using long short-term memory, *Proc. of the 5th Int'l Conf. on Advanced Computing & Communication Systems (ICACCS)*, Coimbatore, India, pp.666-671, 2019.

[9] J. Namgung, S. Son and Y.-S. Moon, GRU-based deep learning algorithm for DGA classification, *Proc. of Korea Computer Congress 2020*, pp.937-939, 2020 (in Korean).