On Detecting Cloud Container Failures from Computing Utility Sequences

Yu-Shao Liu[†], Hsu-Chao Lai[†], Jiun-Long Huang[‡], August F.Y. Chao[*]
[†]Inst. of Computer Science and Engineering, National Yang Ming Chiao Tung University,
[‡]Inst. of Data Science and Engineering, National Yang Ming Chiao Tung University,
[*]Taiwan Web Service Corporation,
Hsinchu, Taiwan
Email: ysl@cs.nctu.edu.tw, hsuchao.cs05g@nctu.edu.tw, jlhuang@cs.nctu.edu.tw, august.chao@twsc.io

***Abstract*—As the popularity of cloud platforms and container grows rapidly, managing clouds has become an important issue. For example, failed containers on cloud platforms would trigger automatic restart mechanism. However, the failed containers caused by user error are not fixable by restart, and may lead to the loop between failure and restart. Therefore, the looping failure will harm the overall performance of cloud. In this paper, we propose to identify possible container failures, where the utility behavior of containers (e.g., CPU usage, GPU usage, I/O throughput, etc) are factored in, in a machine learning approach. We propose a light-weight neural network EEGNet-SE to support fast inference in real-time. In addition, EEGNet-SE is able to distinguish dynamic relations between each utility for different tasks. We conduct a real cloud container dataset from Taiwan Cloud Computing (TWCC) platform. Experimental results manifest that EEGNet-SE boosts the performance and efficiency simultaneously, and outperforms the other state-of-the-art methods in terms of accuracy.**

***Index Terms*—Cloud Container Failure Detection, Neural Network, Multivariate Time-series Classification**

I. Introduction

Thanks to the rapid growth of network bandwidth and computation hardware, cloud platforms have been one of the most important technique in recent years, evident by Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), etc. These cloud platforms usually provide Container Compute Service (CCS), which is a containerized working environment supporting various applications (e.g., online music streaming, on-demand videos, machine learning models, Internet-of-Things, etc), such that users can easily create, remove, or cluster containers based on their demands. The annual revenues of AWS, and GCP have reached 13.5 and 13.1 billion USD respectively in 2020[1].

Managing CCS turns out to be important and essential for cloud providers. To be more specific, software failures causing container termination could be roughly divided into two categories: the administrative termination and the process failure [1]. The administrative termination resulting from internal networking issues in the clusters could be recovered by the automatic restarting mechanism fast. However, the process failures are caused by user error (e.g., wrong configurations or requesting too much memory), and they may keep looping between termination and restart until they are manually fixed. As a result, one small user process failure could result in severe performance drops of the whole platform, which potentially brings great damage to both the cloud providers and the users.

Technically speaking, cloud providers are able to directly access those troubled containers with authority of system administrator for interventions. However, it is irrational to trespass due to the user privacy issue [2]. In this regard, to manage each container indirectly, a promising solution is to collect the utility metrics (e.g., CPU usage, memory usage, or network I/O throughput) of each container instance for further analysis. Note that it is also impractical to ask human experts to identify failures based on those utilities from thousands of container runtime instances. Although several applications (e.g., Prometheus[2]) support utility monitoring, they do not facilitate automatic failure detection and discrimination. In this consequence, there is an urgent need of developing intelligent solutions for failure detection on cloud platforms.

In this paper, we formulate the failure detection problem, namely *Cloud Container Failure Detection* (CCFD). Given the Cloud Container Utility Sequences (CCUSeq), such as memory usage, CPU utilities, and I/O throughput, of a container as time-series data, the goal of CCFD is to early predict if the container is going to fail. The research challenges of CCFD are two-fold. 1) *Short decision time*. In order to make immediate failure alert in real-time scenario, the detection model should not be inefficient. One possible solution is to reduce the number of learning parameters, but it may also sacrifice the performance. Therefore, how to strike a good balance between performance and efficiency should be carefully examined. 2) *Complicated relations between utilities*. The shape of each CCUSeq of one container is not always identical to each other. For example, while training a deep model with GPUs, the virtual memory usage is stable but the GPU usage keeps oscillating. It is probably because the data loaded to the memory does not require further I/O but the GPUs are busy only if they are in a training epoch. How to extract useful information from heterogeneous shapes is an issue, which is

[1]https://techcrunch.com/2021/02/02/google-cloud-lost-5-6b-in-2020/

[2]Prometheus: https://prometheus.io/

also challenging in similar multivariate time-series prediction.

To tackle CCFD, we design a neural network-based model, namely EEGNet-SE, to efficiently and effectively profile each container based on their behavior, i.e., the CCUSeq. Specifically, for the first challenge, we notice that EEGNet [3] is a good fit since it is a light-weight Convolution Neural Network (CNN) model by employing depthwise separable convolution in handling multi-channel biosignals. However, EEGNet assumes that the relation between each channel is static or fixed, which may limit the performance due to the impractical assumption. Therefore, we improve it by concatenating Squeeze-and-Excitation blocks (SE blocks) [4], which exploit channel attention after the convolution layer in order to recalibrate the channel relations dynamically, where the second challenge is addressed. To evaluate the proposed EEGNet-SE, we conduct the experiments with a real cloud container dataset from Taiwan Cloud Computing (TWCC) platform. EEGNet-SE outperforms state-of-the-art multivariate time-series classification models in terms of testing accuracy.

II. Related Works

Current machine learning techniques used in cloud platforms focus on resource allocation [5], privacy protection [6], [7], and virtual machine provisioning [8]. The important failure detection problem has not drawn machine learning research community's attention yet.

CCFD is a type of multivariate time-series classification problem [9]. Multilayer Perceptron (MLP) [10] is the basic model that stacks multiple fully connected layers and activation function to learn temporal non-linear patterns. Fully Convolutional Neural Network (FCN) [10] conducts convolution layers and ends up with a global average pooling layer to reduce the parameters. Residual Network (ResNet) [11] was first designed for image classification, and further be used for time-series data by adding residual connections to FCN [10]. Encoder [12] introduces attention blocks to replace the global average pooling layer in FCN for flexible and learnable weights. Multi-scale Convolutional Neural Network (MCNN) conducs complicated data augmentation and skip sampling [13], which make it hard to train. t-LeNet is a time series-specific CNN [14], including two convolution layers and a fully connected layer with local max-pooling. Time Convolutional Neural Network (Time-CNN) [15] applies mean square error for loss instead of cross-entropy. Time Warping Invariant Echo State Network (TWIESN) [16] exploits recurrent structures and transformation at each timestamp, so the input space is transformed into a higher dimensional latent space for classification. However, the above methods do not dynamically distinguish different weights of CCUSeq for each container, which may harm the prediction performance.

III. Proposed Method

In this section, we first formally define the Cloud Container Failure Detection (CCFD) problem in Section III-A, and introduce the proposed EEGNet-SE in Section III-B.

A. Problem Formulation

Let $C = \{c_1, c_2, \cdots, c_N\}$ denote a set of N containers and c_n denotes the n-th container. For container c_n, its multivariate Cloud Container Utility Sequence (CCUSeq) data is denoted as $X^n = (X_1^n, X_2^n, \cdots, X_M^n)$, where X_m^k is the CCUSeq of a specific usage m (e.g., virtual memory). X_m^n is further defined as $X_m^n = \langle x_1^{n,m}, x_2^{n,m}, \cdots, x_T^{n,m} \rangle$, where $x_t^{n,m}$ is the measured utility value (e.g., megabytes of virtual memory) in real values at timestamp t and T is the predefined length of the observation time window. The whole dataset $D = \{(X^1, y^1), (X^2, y^2), \cdots, (X^N, y^N)\}$ is a collection of pair (X^n, y^n), where y^n denotes the class (i.e., label) of container c^n. For a dataset containing K classes, if X^n belongs to the k-th class ($1 \leq k \leq K$), we adopt the one-hot encoding technique where y^n is a vector of length K and the value of the k-th element is equal to 1 and 0 otherwise. Consequently, given the above dataset D, the goal of Cloud Container Failure Detection (CCFD) is to train a classifier to map those multivariate CCUSeq X^n to a probability distribution over the class variable y^n.

B. EEGNet-SE

With regard to classifying container behavior in real-time, we propose EEGNet-SE, which is a light-weighted neural network to profile each container based on their computing behavior, i.e., CCUSeqs. As illustrated in Fig. 1, after extracting time and frequency domain features from raw CCUSeqs, we employ EEGNet [3] as our foundation due to its Depthwise Separable Convolution [17], [18], which is a separated but paired convolution layers that approximates original convolution layer. To support dynamic channel attention among multiple utilities, we further improve the EEGNet by concatenating Squeeze-and-Excitation blocks [4]. Afterwards, two-layer fully-connected layer and a softmax layer are sequentially concatenated for classification, and we minimize the cross-entropy loss function as the objective:

$$\min \sum_{c=1}^{C} -t_c log(s_c),$$

where C is the number of classes. t_c and s_c denote the ground truth and the predicted score of class c by EEGNet-SE, respectively. The details of EEGNet and Squeeze-and-Excitation blocks are introduced as follows.

EEGNet. EEGNet is widely deployed in brain computer interface domain for classification of electroencephalography (EEG) and other biosignals. EEGNet adopts a pair of of 1-dimensional convolution layers to extract time domain and frequency domain features from input signals (CCUSeqs in our case) respectively. Afterwards, a *depthwise separable convolution layer* (DepConv) [17], [18] is concatenated to learn the latent representations of the whole signal spectrum. DepConv consists of two sequential steps. i) Depthwise layer: independently convolute each input channel (i.e., CCUSeq) with a depthwise filter. ii) Pointwise layer: implement channel-wise dot product for convolution across different CCUSeqs.

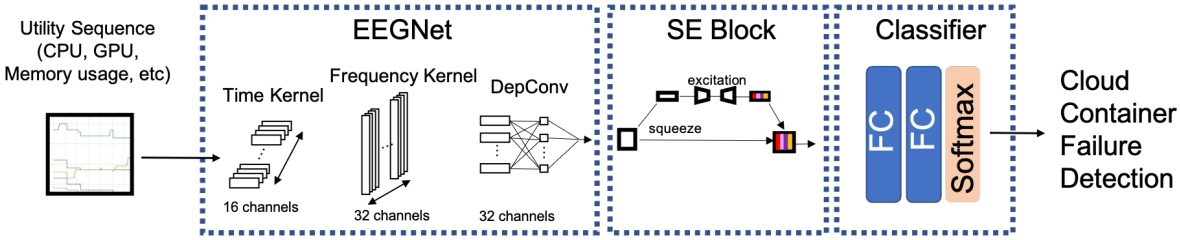

Fig. 1. Illustration of EEGNet-SE architecture (FC denotes the fully-connected layers).

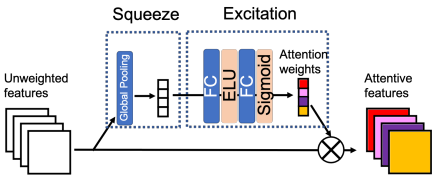

Fig. 2. Illustration of Squeeze-and-Excitation block.

Compared to the widely-used 2-dimensional convolution layer in image processing, using 1-dimensional convolution layer as EEGNet not only saves the amount of parameters to improve training efficiency[3], but also enhances interpretability.

Squeeze-and-Excitation Block (SE blocks). Although the DepConv layer supports channel-wise convolution to reveal the relationships between each CCUSeq, DepConv simply assumes that each channel has equal importance (i.e., weight). Nevertheless, those utilities play different roles in different application tasks in practice. For example, CPU-bounded and GPU-bounded applications should have different weights on CPU and GPU utilities, respectively. Directly applying EEGNet on CCFD hence may not be the best fit. Hence, we make the first attempt to integrate SE blocks with EEGNet in order to facilitate dynamic attention weights across different utilities.

As illustrated in the upper route of Fig. 2, SE blocks consists of a *Squeeze* operation and an *Excitation* operation [4]. The *Squeeze* operation is a global average pooling layer along the channel dimension. The *Excitation* operation is a two-layered fully-connected feed forward network. The first fully-connected layer reduces the channel dimension with a reduction ratio γ, which is a hyperparameter, and activated by a ELU function [19]. Experimental results in Section IV-B show ELU outperforms other state-of-the-art activation function. The second fully-connected layer increases the dimension with the sigmoid activation. The channel attention weights are represented as an M-dimensional vector after computed by SE blocks. The channel attention weights then channelwisely weight the original input features, as shown in the lower route in Fig. 2. With SE blocks, the weights of utilities are computed based on their features, and hence those important utilities can be emphasized via greater attention weights.

[3]The computing complexity of DepConv is $O(k\dot{c}+c^2)$, which is asymptotically lower than that of conventional 2D-convolution layers $O(k\dot{c}^2)$, where c and k are the number of channels and convolution kernels respectively [18].

In summary, we add an SE block after the DepConv layer of EEGNet, such that important utilities under different conditions can be extracted and emphasized in a machine learning approach. Note that SE blocks have very few parameters (roughly 1% of EEGNet). Therefore, adding SE-blocks does not slow down the efficiency of EEGNet-SE but significant improves performance in our experiments in Section IV-B.

IV. Experiments

A. Experiment Setting

To evaluate the performance of the proposed EEGNet-SE, we collect real data from Taiwan Computing Cloud (TWCC)[4]. TWCC is a computation-oriented cloud platform built on the infrastructure of a supercomputer with over 2000 NVIDIA Tesla V100 GPUs. The CCS of TWCC supports a containerized GPU working environment with NVIDIA-optimized deep learning frameworks such as TensorFlow, PyTorch, MXNet, etc. We collect the CCUSeqs spanning from June 29th to June 30th as the training data (13,151 records of 1,282 containers), and those sequences in the first hour of July 1st are the testing data (561 records of 561 containers). Each container has 6 CCUSeqs. The length of the observation time window T is set to 60 slots, and each slot is 60 seconds long. The goal is to classify the reason of container termination of each container from 4 different classes: *ContainerCannotRun*, *Completed*, *Error*, and *OOMError*. To alleviate the lack of data comprehensibility, we follow [13], [15], [20] to apply data augmentation technique for improving classification performance. Specifically, we implement the Discriminative Guided Warping with shapeDTW (DGW-sD) [21] to remain more original features and to solve the window warping issue.

For the baseline models, we compare with state-of-the-art deep learning models designed for the time series classification problem [9], including CNN-based methods: MLP, FCN, Residual Network, Encoder, MCNN, Time-CNN and an RNN-based method: TWIESN. We repeat the experiments 10 times and present the average results to reduce stochastic noise.

B. Experimenal Results on TWCC Dataset

Table I compares the accuracy of each method on TWCC dataset. EEGNet-SE outperforms every baseline model either with or without data augmentation, manifesting that the lightweight EEGNet-SE does not sacrifice but even boosts the

[4]Taiwan Computing Cloud: https://www.twcc.ai/

TABLE I
TESTING ACCURACY OF TWCC DATASET

Methods	Accuracy (w/o Augmentation)	Accuracy (w/ Augmentation)	Parameters
MLP	0.426	0.426	686,504
FCN	0.366	0.387	270,340
ResNet	0.356	0.392	507,396
Encoder	0.828	0.830	3,202,052
MCNN	0.426	0.426	1,368,324
t-LeNet	0.474	0.509	63,179
Time-CNN	0.511	0.535	970
TWIESN	0.799	0.801	-
LSTM	0.593	0.595	13,828
EEGNet	0.794	0.840	98,836
EEGNet-SE*	**0.834**	**0.853**	99,908

TABLE II
ACCURACY OF DIFFERENT CONFIGURATIONS ON TWCC DATASET

Activation	Train		Test	
	mean (%)	**stdev**	**mean (%)**	**stdev**
Hswish	0.840	0.58	0.630	12.7
ELU	0.824	0.35	0.794	6.42
ReLU6	0.849	0.72	0.640	14.54
LeakyReLU	0.841	0.58	0.579	11.54
SE-Hswish	0.863	0.69	0.618	14.47
SE-ELU	**0.866**	**0.39**	**0.834**	**1.86**
SE-ReLU6	0.862	1.12	0.535	10.17
SE-LeakyReLU	0.855	1.05	0.522	9.20

performance. It is because that we apply the SE blocks to dynamically adjust the weights of each CCUSeq, proved by the improvement compare to EEGNet. As expected, the performances of almost all the methods are improved with data augmentation. Encoder is the only competitive baseline, it may owing to its batch normalization layers. However, EEGNet-SE has 3 times less parameters than Encoder, leading to less training time and faster inference ability.

Table II compares the mean and standard deviation of accuracy of employing different activation functions in EEGNet-SE in training and testing phases, respectively. State-of-the-art activation functions, including Hardswish [22], ELU (α=0.1), ReLU6, LeakyReLU (α=0.01), are compared. The upper group omit SE blocks and the activation functions are only implemented after each convolution layer of EEGNet. SE-ELU outperforms the other setting, manifesting that both SE blocks and the ELU function are useful in recognizing important utility features in CCFD. On the other hand, the widely-used ReLU6 has worse performance because it is likely to cause gradient vanishing or so-called the dead ReLU problem in DepConv layers, as reported in [17].

V. CONCLUSION

In this paper, we identify the Cloud Container Failure Detection problem, and propose efficient EEGNet-SE to predict the container failures based on their utility sequences. EEGNet-SE employs depthwise convolution layers for approximation to reduce the number of parameters, and concatenate the Squeeze-and-Excitation blocks to dynamically attend important utilities. Experimental results on TWCC dataset manifest that EEGNet-SE outperforms the other deep models in terms of accuracy.

ACKNOWLEDGMENT

This work is supported in part by MOST under grants 109-2221-E-009-118-MY3, 110-2222-E-009-005-MY2, and 109-2218-E-009-015.

REFERENCES

[1] L. Abdollahi Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, "Deploying microservice based applications with kubernetes: Experiments and lessons learned," in *IEEE International Conference on Cloud Computing*, 2018.

[2] M. Cinque, R. Della Corte, and A. Pecchia, "Microservices monitoring with event logs and black box execution tracing," *IEEE Transactions on Services Computing*, 2019.

[3] V. J. Lawhern, A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, and B. J. Lance, "Eegnet: a compact convolutional neural network for eeg-based brain–computer interfaces," *Journal of Neural Engineering*, 2018.

[4] G. S. J. Hu, L. Shen, "Squeeze-and-excitation networks," in *CVPR*, 2018.

[5] B. Du, C. Wu, and Z. Huang, "Learning resource allocation and pricing for cloud profit maximization," in *AAAI*, 2019.

[6] J. Wang, J. Zhang, W. Bao, X. Zhu, B. Cao, and P. S. Yu, "Not just privacy: Improving performance of private deep learning in mobile cloud," in *KDD*, 2018.

[7] R. Peterson, A. S. da Silva, A. Carvalho, C. Fetzer, A. Martin, and I. Blanquer, "Vallum-med: Protecting medical data in cloud environments," in *CIKM*, 2020.

[8] C. Luo, B. Qiao, X. Chen, P. Zhao, R. Yao, H. Zhang, W. Wu, A. Zhou, and Q. Lin, "Intelligent virtual machine provisioning in cloud computing," in *IJCAI*, 2020.

[9] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. Muller, "Deep learning for time series classification: a review," *Data Mining and Knowledge Discovery*, 2019.

[10] Z. Wang, W. Yan, and T. Oates, "Time series classification from scratch with deep neural networks: A strong baseline," in *IJCNN*, 2017.

[11] K. He, X. Zhang, and S. R. J. Sunn, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[12] J. Serrà, S. Pascual, and A. Karatzoglou, "Towards a universal neural network encoder for time series." in *International Conference of the Catalan Association for Artificial Intelligence (CCIA)*, 2018.

[13] Z. Cui, W. Chen, and Y. Chen, "Multi-scale convolutional neural networks for time series classification," *arXiv preprint arXiv:1603.06995*, 2016.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[15] B. Zhao, H. Lu, S. Chen, J. Liu, and D. Wu, "Convolutional neural networks for time series classification," *Journal of Systems Engineering and Electronics*, 2017.

[16] P. Tanisaro and G. Heidemann, "Time series classification using time warping invariant echo state networks," in *IEEE ICMLA*, 2016.

[17] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *CVPR*, 2017.

[18] L. Kaiser, A. Gomez, and F. Chollet, "Depthwise separable convolutions for neural machine translation," in *ICLR*, 2018.

[19] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[20] B. K. Iwana and S. Uchida, "An empirical survey of data augmentation for time series classification with neural networks," *arXiv preprint arXiv:2007.15951*, 2020.

[21] B. K. Iwana and S. Uchida, "Time series data augmentation for neural networks by time warping with a discriminative teacher," in *ICPR*, 2021.

[22] A. Howard, M. Sandler, B. Chen, W. Wang, L. Chen, M. Tan, G. Chu, V. Vasudevan, Y. Zhu, R. Pang, H. Adam, and Q. Le, "Searching for mobilenetv3," in *ICCV*, 2019.