IEEE INFOCOM 2022 - IEEE Conference on Computer Communications | 978-1-6654-5822-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INFOCOM48880.2022.9796778

PreGAN: Preemptive Migration Prediction Network for Proactive Fault-Tolerant Edge Computing

Shreshth Tuli*, Giuliano Casale*, Nicholas R. Jennings*†
*Imperial College London
†Loughborough University
{s.tuli20, g.casale}@imperial.ac.uk, n.r.jennings@lboro.ac.uk

***Abstract*—Building a fault-tolerant edge system that can quickly react to node overloads or failures is challenging due to the unreliability of edge devices and the strict service deadlines of modern applications. Moreover, unnecessary task migrations can stress the system network, giving rise to the need for a smart and parsimonious failure recovery scheme. Prior approaches often fail to adapt to highly volatile workloads or accurately detect and diagnose faults for optimal remediation. There is thus a need for a robust and proactive fault-tolerance mechanism to meet service level objectives. In this work, we propose PreGAN, a composite AI model using a Generative Adversarial Network (GAN) to predict preemptive migration decisions for proactive fault-tolerance in containerized edge deployments. PreGAN uses co-simulations in tandem with a GAN to learn a few-shot anomaly classifier and proactively predict migration decisions for reliable computing. Extensive experiments on a Raspberry-Pi based edge environment show that PreGAN can outperform state-of-the-art baseline methods in fault-detection, diagnosis and classification, thus achieving high quality of service. PreGAN accomplishes this by 5.1% more accurate fault detection, higher diagnosis scores and 23.8% lower overheads compared to the best method among the considered baselines.**

***Index Terms*—Fault Tolerance; Preemptive Migrations; Edge Computing; Generative Adversarial Networks.**

I. Introduction

Edge computing is the processing of data close to the source where it is produced to optimize the service performance of such systems. This paradigm is closely integrated with the sensors and actuators in the Internet of Things (IoT) framework [1]. With edge computing becoming ubiquitous, it is essential to ensure that the edge nodes themselves do not become a point-of-failure for the running applications, and robust countermeasures are in place to incorporate network or node overloads/failures. Modern application demands of low latency task execution and resource constraints of the edge devices further exacerbate the problem [2]. The increasing volumes of the data requiring immediate processing and the resource constraints at the edge are pushing the compute resources to their limits, giving rise to a high chance of resource contention and node downtimes [3], [4]. This leads to resource unavailability and Service Level Objective (SLO) violations that can lead to significant financial losses [5]. Thus, it is critical to develop a fault-tolerance mechanism for edge computing to maintain low latency and high reliability.

Challenges. The problem of developing a robust fault-tolerance framework is challenging. The first stage of solving this is to be able to proactively predict faults before they occur and diagnose the root-cause issues to be able to run appropriate remediation steps. These steps should be carried out in near real-time for such systems to be effective [6]. Moreover, for such systems to be within the strict specifications of modern industrial demands, they need to be able to resolve diverse kinds of system or network related faults [7]. This may entail establishing the type of fault at the time of its prediction for a more informed recovery decision. The volatile nature of the workloads and resources increases the difficulty in the prediction of faults and their types by several fold [8]. Further, to avoid overlooking any fault that may cause significant adverse effects later and to avoid the overhead of false-positive predictions, such prediction models need to be extremely accurate. To do this, some existing methods leverage machine learning models due to their highly accuracy [9], [10]. However, in such cases, the annotated log traces available to brokers is limited, leading to restricted size of the labelled fault classification data. This makes it hard to train supervised machine learning models for fault detection and classification.

Existing solutions. Over the past few years, several fault-tolerance approaches have been proposed to enhance the service reliability of edge or cloud platforms [3], [11], [12]. A popular method is to provide node redundancy and network contingencies to avoid the crippling downtime of an edge element. However, with the increasing number of edge devices, having redundancy for each node is not feasible, considering the energy and cost implications of such a deployment [13]. Another way is to replicate the running instance of a task on a separate node [14]. This is not ideal for resource-constrained edge devices as it makes them more susceptible to resource contention and faults [15]. Yet another mechanism is to periodically save the execution state of the running tasks by checkpointing their corresponding containers. Containers provide a virtualization layer that allows the running applications to be independent of the underlying hardware facilitating efficient task restoration in the event of node failures [16]. When a node failure occurs, the system can transfer and resume the task on a different device, commonly referred to as “preemptive migration” [17]. However, when predicting the faults and deciding the restoration node in advance, checkpointing the containers periodically could lead to excessive stress on the system and the network. Instead, we could checkpoint only those containers that need to be restored. This saves the excess overhead of periodically checkpointing all running tasks. In recent years, many different methods have been proposed to decide the appropriate migration decisions to enhance service reliability. These range from Particle Swarm Optimization (PSO) [3], Integer Linear Programming [11], [18] and Bayesian Classification [9] to using Neural Network and clustering algorithms like k-means [10]. However, such methods are often not generalizable or struggle to adapt in volatile settings quickly, as discussed later in Section II. Thus, we propose a novel fault-tolerance model and demonstrate its efficacy against state-of-the-art baselines [9], [10].

Background and new insights. To be able to predict an appropriate preemptive migration decision, it is crucial to accurately detect, diagnose and classify faults in an online fashion. One way to achieve this is to create a deep generator model that predicts such a decision. Generative Adversarial Networks (GANs) have been shown to be very successful in this as they can reduce prediction errors and provide us with a robust anomaly prediction framework [19]. To successfully train a GAN, we need to efficiently model the temporal trends and the cross-correlations of performance and resource utilization metrics among different edge nodes. Recent developments in graph machine learning allow both of these to be done together efficiently [20], [21]. To minimize the total computational complexity of each layer of the deep neural network and make decision prediction time-efficient, we can use self-attention operations [22]. However, just having a GAN is insufficient to incorporate the fault types in the prediction framework. To be able to execute classification with limited data and allow quick adaptability, inspired from few-shot learning, we can extend prototype networks that generate an embedding vector for each class [23]. Moreover, to test whether the preemptive migration decision would improve the Quality of Service (QoS), co-simulation techniques can be used [24]. Due to the lack of integration interfaces, such advances have not been explored in the scope of edge reliability. This work leverages these concepts with necessary adaptations, to predict preemptive migration decisions for fault-tolerant edge computing.

Our Contributions. In this work, we propose **PreGAN**: **Pre**emptive Migration Prediction **GAN**. PreGAN uses graph attention and recurrent neural networks for feature extraction, a prototype network for classification and generates an input embedding for the GAN. The GAN-based prediction model enables robust training and time-efficient inference. We perform extensive empirical experiments on real-life edge testbed to compare and analyze PreGAN against the state-of-the-art methods. Our experiments show that PreGAN performs best in terms of QoS metrics, reducing the energy consumption, response time and SLO violations by up to 8, 5 and 12 percent, respectively. PreGAN achieves this by 5.1% more accurate fault detection, 10.7% lower task migrations and 23.8% lower overhead than the most accurate baseline.

The rest of the paper is organized as follows. Section II overviews related work. Section III outlines the PreGAN methodology for model training, preemptive migration prediction and execution. A performance evaluation of the proposed method is developed in Section IV. Finally, Section V concludes the paper and presents future directions.

II. Related Work

Fault-tolerance deals with developing systems that have an ability to withstand failures and faults in the workloads and efficiently manage resources to maintain optimum QoS. Most methods can be divided into two categories: reactive and proactive. The former take action after observing a system fault by typically checkpointing, replicating or resubmitting tasks affected by the fault [25]. The latter aim to avoid expensive fault recovery by predicting failures in advance and taking appropriate steps for remediation by preemptive migration or fault-aware scheduling [25]. As reactive schemes often lead to poor QoS in highly dynamic setups [3], we focus on proactive methods in this work.

Most contemporary state-of-the-art techniques employ some form of specialized algorithms or machine learning models. Dynamic Fault Tolerant Migration (DFTM) [11] is a recent method that uses an integer linear programming (ILP) formulation to analyze workload traffic, select the tasks running on the hosts that should be migrated and the target hosts for restoration. Methods like Multi-stage Coflow Scheduling (MCS) [18] also consider task co-dependencies to optimize QoS. Another ILP based method proposes a preference based fault management technique [26] that tries to balance between the QoS improvement and the migration cost using a multi-objective formulation. Such methods do not scale for real-time operations, making them unsuitable for mission critical edge applications. SFS [27] is a failover strategy that selects only those tasks that are about to violate their deadlines to reduce migration overheads. However, modeling the SLO deadlines does not consider other QoS parameters while optimizing the selection of the target hosts. This makes it often perform poorly in heterogeneous setups.

The Proactive Coordinated Fault Tolerance (PCFT) [3] method uses Particle Swarm Optimization (PSO) to reduce the overall transmission overhead, network consumption and total execution time for a set of tasks. This method first predicts faults in the running host machines by anticipating resource deterioration and uses PSO to find target hosts for preemptive migration decision. This approach mainly focuses on reducing transmission overheads in distributed cloud setups, but often fails to improve the I/O performance of the compute nodes [11]. The Energy-efficient Checkpointing and Load Balancing (ECLB) [9] technique uses Bayesian methods and neural networks to classify host machines into three categories: overloaded, underloaded and normal execution. This classification is used to decide appropriate task migrations to reduce the number of overloaded hosts. However, this model only considers computational overloads and does not consider other fault types. CSAVM [28] uses another evolutionary search scheme to take live migration decisions for the task queues. The method is used to optimize the power consumption of a compute setup by preventing unnecessary migrations. DDQP [29] uses double deep Q-networks to place services on network nodes. However, such reinforcement learning schemes are known to be slow to adapt in volatile settings [24].

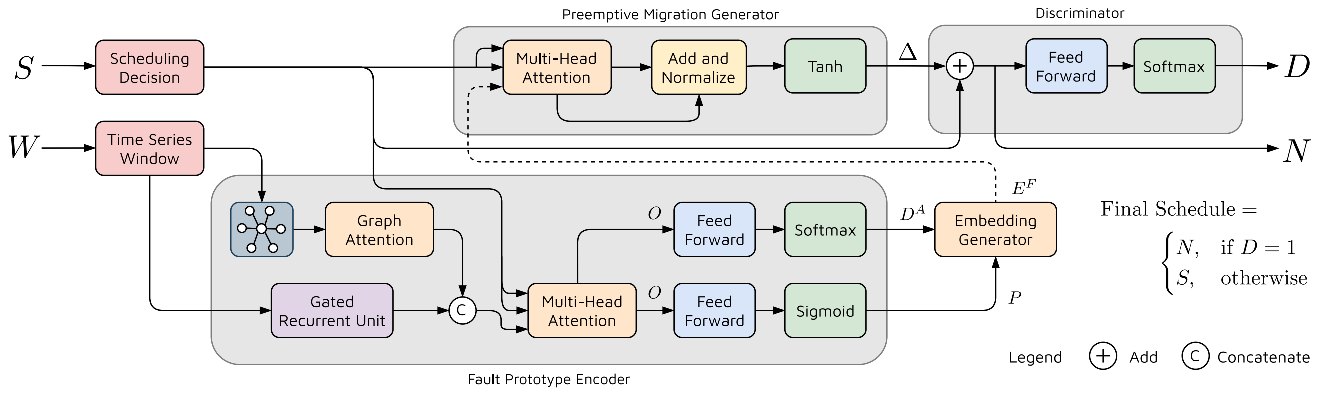


Figure 2: The PreGAN Model.

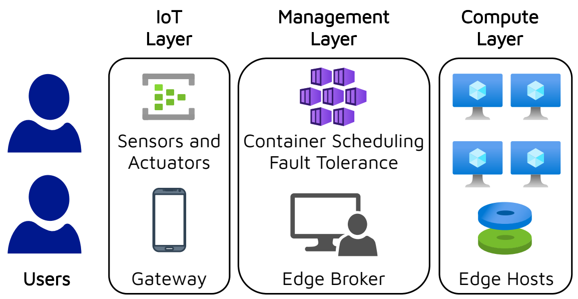


Figure 1: System Model.

A recent work, CMODLB [10], uses a clustering-based multiple objective dynamic load balancing technique to avoid resource contentions in cloud computing nodes. This method uses k-means to cluster nodes and identify overloaded hosts, PSO to select tasks and deep learning and fuzzy-logic optimization to select target hosts. However, slow PSO optimization leads to high recovery times that is often not helpful in highly dynamic systems [24]. In our experiments, we compare PreGAN against the state-of-the-art baselines DFTM, ECLB, PCFT and CMODLB.

III. Methodology

A. System Model and Problem Formulation

In this work, we target a standard heterogeneous edge computing environment where all nodes are in the same Local Area Network (LAN); see Figure 1 for an overview. Tasks are container instances generated from the sensors and actuators in the IoT layer and communicated to the compute nodes via gateway devices. The edge broker takes all scheduling, management and preemptive migration decisions.

We assume that there are a m number of host machines in the fog resource layer and denote them as $H = \{h_1, \dots, h_m\}$. We assume the paradigm of discrete time control, where we take scheduling and migration decisions at periodic intervals [2], [30], [31]. We assume a scheduler is already present in the broker. The t-th interval is denoted as I_t and the scheduling decision and host characteristics at this interval are denoted as S_t and $x_t \in \mathbb{R}^{m \times n}$ (each host has n features). The time-series $\{x_0, \dots, x_{t-1}\}$ is denoted as $\mathcal{T}_t$. We divide the problem of fault-tolerance into two sub-problems, defined as:

Fault Prediction: For an input multi-variate time-series $\{x_0, \dots, x_{t-1}\}$, we need to predict fault labels for each host in I_t as $y_t \in \{0, 1, \dots, c\}^m$. Here, $y_{t,i}$ denotes the output for h_i; $y_{t,i} = 0$ means no fault and $y_{t,i} = j$ means fault of class j among the user-specified c classes $j \in \{1, \dots, c\}$.

Preemptive Migration Decision: For an input scheduling decision S_t (set of task and host pairs) and a solution of the prediction problem y_t, we wish to predict a migration decision Δ_t. To model the dependence of a data point x_t at a timestamp t, we consider the sliding window of length k, denoted as

$$W_t = \{x_{t-k}, \dots, x_{t-1}\},$$

as is done in prior work [32]. Thus, in lieu of using just x_t, we use W_t as it allows us to capture the local context. For model robustness, we normalize the input windows. For the sake of simplicity and without loss of generality, we drop the t index whenever this is not ambiguous and use x, S, W, Δ.

B. PreGAN Model

Figure 2 provides an overview of the PreGAN model. The multi-variate time-series has now become the host characteristics x. We form a graph using the schedule S, such that there is an edge from h_i to h_j if there is a task migration from h_i to h_j in S. The n characteristics of each host are then used to populate the feature vectors of the nodes in the graph. We then use a graph attention network (GAT) with Gated Recurrent Units (GRUs). GAT allows us to extract the multi-host feature correlations with the migration information encoded as graph edges. GRU allows us to extract temporal trends in time-series log data for prediction of faults in the next interval. This, in conjunction with the migration decision from the underlying scheduler, is used to detect the hosts with a high chance of faults and predict the class prototypes for each fault. The detection and classification results are combined to create an embedding for the generator network that predicts the preemptive migration decision to amend the input schedule. The decision is forwarded to a discriminator network to compare against the original scheduling decision.

We now describe the complete pipeline in detail. The input time-series window W is first converted to a tensor of size $k \times m \times n$. The scheduling decision for each task is converted to a one-hot decision vector of size m. For p active tasks in the system, these one-hot vectors are stacked to form a matrix of size $p \times n$. We divide our model in two parts: (1) Fault Prototype Encoder (FPE) that aims to solve the first sub-problem and (2) Preemptive Migration GAN for the second sub-problem.

Fault Prototype Encoder. To infer over all hosts in the graph, we create a new global node connected to each host node [33]. This gives the graph attention operation as

$$O_1 = \text{sigmoid}\left(\frac{1}{n}\sum_{i\in\{1,\dots,m\}} \theta_{GAT} W_i\right), \tag{1}$$

where θ_{GAT} is the weight matrix for the GAT network. The time-series is also passed through a GRU, with $\bar{O}_2$ as the output of the previous interval

$$O_2 = \text{GRU}(W, \bar{O}_2). \tag{2}$$

For any three input tensors Q, K and V, we define Multi-Head Self Attention [22] as passing it through h (number of heads) feed-forward layers to get Q_i, K_i and V_i for $i \in \{1,\dots,h\}$, and then applying attention as

$$\begin{aligned} \text{MultiHeadAtt}(Q,K,V) &= \text{Concat}(H_1,\dots,H_h) \\ H_i &= \text{Attention}(Q_i,K_i,V_i). \end{aligned} \tag{3}$$

Here $\text{Attention}(Q_i,K_i,V_i)$ is the scaled-dot product attention operation [22]. We apply this on the GRU and GAT outputs (with task SLO requirements and host characteristics) to obtain

$$O = \text{MultiHeadAtt}(S, S, [O_1; O_2]). \tag{4}$$

The late-fusion of the two outputs allows the downstream models to exploit them independently [33]. The output O is a collection of attention based encodings for each host. We pass this encoding through two decoders for each host h_i,

$$\begin{aligned} D_i^A &= \text{softmax}(\text{FeedForward}(O_i)), \\ P_i &= \text{sigmoid}(\text{FeedForward}(O_i)), \end{aligned} \tag{5}$$

where $D_i^A \in \mathbb{R}^2$ denotes the fault prediction. The model predicts a fault in h_i if $D_i^A[1] \geq D_i^A[0]$. $P_i \in \mathbb{R}^E$ is a prototype embedding for each host corresponding to the fault class, irrespective of whether a fault is detected or not. This factored prediction for each host allows our model to be agnostic to the number of hosts in the system. We define the embedding for host h_i as

$$E_i^F = \begin{cases} P_i, & \text{if } D_i^A[1] \geq D_i^A[0] \\ [0]_{1\times E}, & \text{otherwise.} \end{cases} \tag{6}$$

We stack all host embeddings to generate E^F. This auto-regressive style of using fault predictions allows us to generate a representation of the fault prediction that only consists of class prototypes for the hosts where a fault is detected.

Preemptive Migration GAN. The embedding output from the FPE is passed through the Generator

$$\Delta = \tanh(\text{LayerNorm}(S + \text{MultiHeadAtt}(S, S, E^F))), \tag{7}$$

where the $\Delta \in \mathbb{R}^{p\times n}$ denotes the preemptive migration decision. This is added to the original scheduling decision to get $N = S + \Delta$. The final decision for each task p is calculated as $\text{argmax}(N_p)$. The output of the generator is passed to a discriminator decoder

$$D = \text{softmax}(\text{FeedForward}(N)), \tag{8}$$

where $D \in \mathbb{R}^2$ denotes predicted likelihood scores for using S and N as schedules. The final scheduling decision of PreGAN is defined as

$$\text{Final Schedule} = \begin{cases} N, & \text{if } D[1] \geq D[0] \\ S, & \text{otherwise.} \end{cases} \tag{9}$$

Summarizing, the FPE detects faults in hosts and generates class prototype embeddings for each host. These embeddings are stacked (E^F) and sent to a generator network that outputs a preemptive migration vector (Δ). The discriminator then provides a likelihood score to the new (N) and original (S) scheduling decisions. The final output is then the decision with the highest likelihood score.

Algorithm 1 The FPE training algorithm

```
Require:
    Fault Prototype Encoder $E$
    Dataset used for training $\{W_t, S_t, \hat{y}_t\}_{t=1}^T$
    Step size $\alpha$, Evolutionary hyperparameter $\epsilon$
    Iteration limit $L$
1:  Initialize weights in $E$. Set $l \leftarrow 0$
2:  Randomly initialize class prototypes $\{P_0^C, \dots, P_c^C\}$
3:  do
4:      for($t = 1$ to $T$)
5:          $D^A, P \leftarrow E(S_t, W_t)$
6:          $L_1 = \sum_{i=1}^m \big(\mathbb{1}(\hat{y}_{t,i} = 0)\log(D^A[0])$
                $+\mathbb{1}(\hat{y}_{t,i} > 0)\log(D^A[1])\big)$
7:          $L_2 = \sum_{i=1}^m \mathbb{1}(\hat{y}_{t,i} > 0)\big(\|P_i - P_{\hat{y}_{t,i}}^C\|_2$
                $-(\sum_{j\neq\hat{y}_{t,i}}\|P_i - P_j^C\|_2)\big)$
8:          for($i = 1$ to $m$)
9:              if($\|P_i - P_{\hat{y}_{t,i}}^C\|_2 = \min_j \|P_i - P_j^C\|_2$)
10:                 $P_{\hat{y}_{t,i}}^C \leftarrow (1-\alpha)\cdot P_{\hat{y}_{t,i}}^C + \alpha \cdot P_i$
11:         Update weights of $E$ using $L_1, L_2$
12:         $\alpha \leftarrow (1-\epsilon)\cdot\alpha$
13:         $l \leftarrow l + 1$
14: while $l < L$
```

C. Offline FPE Training

We now describe the training process for the FPE to detect and classify faults, summarized in Algorithm 1. To train the encoder, we first collect a dataset of input windows, scheduling decisions and fault class labels $\{W_t, S_t, \hat{y}_t\}_{t=1}^T$ by running the system without any preemptive migration. Here, $\hat{y}_{t,i}$ is the class label for h_i, with 0 indicating no fault. Now, the FPE encoder (denoted as E) generates the outputs D^A and P from the inputs W_t and S_t (line 5 in Alg. 1). We use the cross-entropy loss for fault detection (line 6 in Alg. 1). Here,

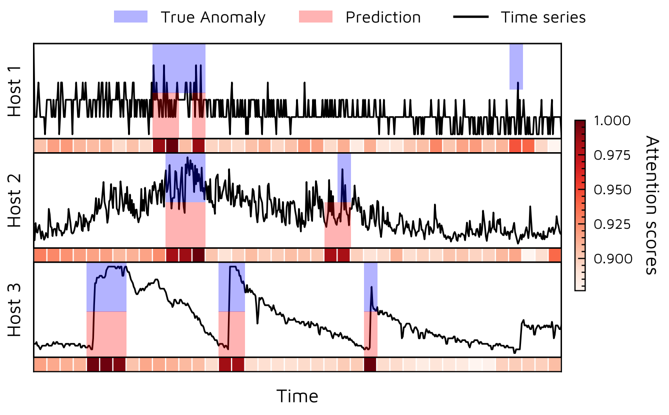


Figure 3: Visualization of Attention scores.

$\hat{y}_{t,i}$ is used to generate the ground-truth labels and the loss is summed over all hosts:

$$L_1 = \sum_{i=1}^{m} \Big(\mathbb{1}(\hat{y}_{t,i} = 0) \log(D^A[0]) + \mathbb{1}(\hat{y}_{t,i} > 0) \log(D^A[1]) \Big). \tag{10}$$

For fault-classification, we use a triplet loss. We first define class prototype vectors for each fault class $\{P_0^C, \dots, P_c^C\}$, that are randomly initialized from $[0,1]^E$. Just like the means in a k-means clustering approach, these vectors denote the centroids of the embeddings of their respective classes [23]. As is common in prototypical networks used in few-shot learning, the new embedding generated by the model belongs to the class which has the least Euclidean distance from the class prototypes. The triplet loss aims at reducing the distance of the output embedding from the true class prototype and increasing it from the false class ones (line 7 in Alg. 1). Thus, for all those hosts with a true fault, the loss includes the L2-norm between the output embedding and the true class prototype and negative L2-norm between the output and false class prototypes:

$$L_2 = \sum_{i=1}^{m} \mathbb{1}(\hat{y}_{t,i} > 0) \Big(\|P_i - P^C_{\hat{y}_{t,i}}\|_2 - \sum_{j \neq \hat{y}_{t,i}} \|P_i - P^C_j\|_2 \Big). \tag{11}$$

Another step in the training process is to update the class prototypes. If an output embedding belongs to the correct class, we can update the prototype of that class as

$$P^C_{\hat{y}_{t,i}} \leftarrow (1-\alpha) \cdot P^C_{\hat{y}_{t,i}} + \alpha \cdot P_i, \tag{12}$$

where α is the step-size. We exponentially decay our step-size to ensure that the model learns class prototypes quickly initially and converges to a stable prototype set (line 12 in Alg. 1). The triplet loss in conjunction with the prototype updates allow the prototypes to be distant from one another, making each prototype a characteristic identifier for that class.

Visualization of Attention Scores. Figure 3 visualizes the attention scores for the FPE encoder of the PreGAN model. The model is trained on a dataset collected from a real-setup (details in Section IV). We show the time-series, the average attention weights for each window (averaged over multiple heads and shown by the red heatmap) for the CPU utilization of the first 3 hosts in the system. It is apparent that the attention scores are highly correlated with the peaks in the data. This allows the model to specifically detect faults in each host individually. As shown in Figure 3, the faults are detected in those hosts for which the attention scores are high.

Algorithm 2 The GAN training algorithm

```
Require:
     Pretrained FPE Encoder $E$
     Generator and Discriminator networks $Gen$, $Disc$
     Co-Simulator Sim
 1: Initialize weights in $Gen$, $Disc$.
 2: for($t = 1$ to $T$)
 3:     $D^A, P \leftarrow E(S_t, W_t)$
 4:     Get $E^F$ by (6)
 5:     $\Delta = Gen(S_t, E^F)$ ▷ Generative preemptive migration
 6:     $N = S_t + \Delta$          ▷ Form new scheduling decision
 7:     $D = Disc(S_t, N)$         ▷ Compare the two decisions
 8:     Calculate $L_D$ using (13)
 9:     Update weights of $Disc$ using $L_D$
10:     Calculate $L_G$ using (14)
11:     Update weights of $Gen$ using $L_G$
```

D. Online GAN Training

We now describe how we use a pre-trained FPE encoder and a co-simulation engine to train the GAN model, summarized in Algorithm 2. Co-simulation is a technique in edge computing that allows a framework to run multiple event-based simulations with different parameters like scheduling decisions to optimize the QoS of the system [24]. As event-based co-simulators can provide us QoS estimates quickly, they allow us to avoid running decisions on the physical setup, saving time and execution costs. Thus, unlike a traditional GAN model, we train our discriminator in a self-supervised manner using a co-simulator.

Our GAN training runs in an online fashion instead of using a pre-collected dataset. For an input pair (S, W), we run the PreGAN model to obtain N and S. We run the co-simulator with the two scheduling decisions S and N and obtain the QoS scores. These scores may be calculated using a combination of metrics like energy consumption, response time and SLO violations [31], [34]. We denote the co-simulated QoS scores by $\mathrm{Sim}(S)$ and $\mathrm{Sim}(N)$. To make the likelihood score output of the discriminator correspond to the co-simulated scores, we use the binary cross-entropy loss as:

$$L_D = \frac{1}{2} \sum_{i=1}^{m} \Big(\mathbb{1}(\mathrm{Sim}(N) \geq \mathrm{Sim}(S)) \Big(\log(D[1]) + \log(1 - D[0]) \Big) + \mathbb{1}(\mathrm{Sim}(N) < \mathrm{Sim}(S)) \Big(\log(D[0]) + \log(1 - D[1]) \Big) \Big), \tag{13}$$

where the loss does not propagate to the generator (fixing N). This pushes the discriminator to give a higher likelihood score

Algorithm 3 The PreGAN testing algorithm

```
Require:
    Pretrained models $E$, $Gen$, $Disc$
 1: for($t = 1$ to $T$)
 2:     $D^A, P \leftarrow E(S_t, W_t)$
 3:     Get $E^F$ by (6)
 4:     $\Delta = Gen(S_t, E^F)$ ▷ Generative preemptive migration
 5:     $N = S_t + \Delta$ ▷ Form new scheduling decision
 6:     $D = Disc(S_t, N)$ ▷ Compare the two decisions
 7:     Execute final schedule obtained from (9)
 8:     if($D[1] \geq D[0]$)
 9:         return $N$ ▷ Return new decision if higher score
10:     return $S$
```

to the decision with a higher simulated QoS score. To train the generator, we use the adversarial loss

$$L_G = \frac{1}{2} \sum_{i=1}^{m} \Big(\log(D[1]) + \log(1 - D[0]) \Big), \tag{14}$$

where the loss propagates to the generator with the discriminator weights kept fixed. This is equivalent to Eq. (13) but with $\mathrm{Sim}(N) \geq \mathrm{Sim}(S)$, pushing the generator to output a migration decision Δ which gives a schedule N better than the original schedule S. To do this, the generator gets fault class labels from the FPE encoder.

This style of training a discriminator model has two benefits: it allows us to (1) run our model inference without the co-simulation at test time, eliminating the computationally expensive generation of simulated traces, (2) train our preemptive migration generator by backpropagating the adversarial loss to the generator network.

E. Online Inference

We now describe the inference procedure using the trained PreGAN model (summarized in Algorithm 3). For any input time-series window and scheduling decision W, S, the PreGAN model outputs D, N. Based on the likelihood scores of the discriminator D, PreGAN predicts if the preemptive migration decision would improve QoS or not. If it does, *i.e.*, $D[1] \geq D[0]$, N is executed, otherwise the original decision S is executed (lines 8-10 in Alg. 3).

Overall, the FPE in PreGAN allows us proactively predict faults and guide the GAN model to generate a preemptive migration decision over the schedule generated by a policy oblivious to future system faults. The discriminator of the GAN then performs a cost-benefit analysis over the original scheduling decision to avoid excessive migration overheads. This allows PreGAN to optimize scheduling decisions and cut down execution costs.

IV. Experiments

We compare the PreGAN method with the state-of-the-art baselines: DFTM [11], ECLB [9], PCFT [3] and CMODLB [10] (more details in Section II). Other heuristic based approaches have been omitted for brevity as these baselines outperform those in all the metrics we consider in our experiments. We use hyperparameters of the baseline models as presented in their respective papers. We train all deep learning models using the PyTorch-1.8.0 [35] library[1].

A. Evaluation Setup

Our evaluation testbed is a cluster with 16 Raspberry Pi 4B nodes.Our cluster contains 8 nodes with 4-GB RAM and another 8 but with 8-GB RAM. The power consumption models are taken from the commonly-used SPEC benchmarks repository [36]. We run all experiments for 100 scheduling intervals, with each interval being 300 seconds long, giving a total experiment time of 8 hours 20 minutes. We average over 5 runs and use diverse workload types to ensure statistical significance.

For our workloads, we consider the *DeFog* applications: Yolo, PocketSphinx and Aeneas [37]. DeFog is a fog computing benchmark suite that consists of various real-world application instances. The three specific applications used in our experiments were considered due to their heterogeneous resource requirements and volatile characteristics. At the beginning of every scheduling interval, we create $Poisson(\lambda)$ new workloads, sampled uniformly from the three applications. Poisson distribution is a natural choice for a bag-of-tasks workload model, common in edge environments [2], [31], [38].

B. Evaluation Metrics

For fault detection, we consider detection accuracy, precision, recall and F1 scores. For a test datapoint $\{W_t, S_t, \hat{y}_t\}$ with the PreGAN outputs D and N, the predicted and ground-truth labels are obtained as

$$\bigwedge_{i=1}^{m} \mathbb{1}(D[1] > D[0]) \text{ and } \bigwedge_{i=1}^{m} \mathbb{1}(\hat{y}_{t,i} > 0)$$

respectively. This ground-truth label is 1 when any of the edge hosts has a fault and 0 otherwise.

For fault diagnosis, we consider two commonly used metrics [39]: (1) HitRate@100% is the measure of how many ground truth dimensions have been included in the top candidates predicted by the model [40], (2) Normalized Discounted Cumulative Gain (NDCG@100%) [41]. For fault classification, we consider the classification accuracy. For test time, we also consider an "improvement ratio", which for an execution of T intervals is calculated as:

$$\text{Improvement Ratio} = \frac{1}{T} \sum_{t=1}^{T} \mathbb{1}(D[1] > D[0]). \tag{15}$$

This metric denotes the ratio of the times the model is able to predict a better scheduling decision than the original. This gives us an indication of the how well the preemptive migration prediction performs compared to the original scheduling decision. Together with the detection accuracy this gives us a complete picture on how well the model performs compared

[1]All model training and experiments were performed on a system with configuration: Intel i7-10700K CPU, 64GB RAM, Nvidia RTX 3080 and Windows 10 Pro OS. This was also the broker node in our setup.

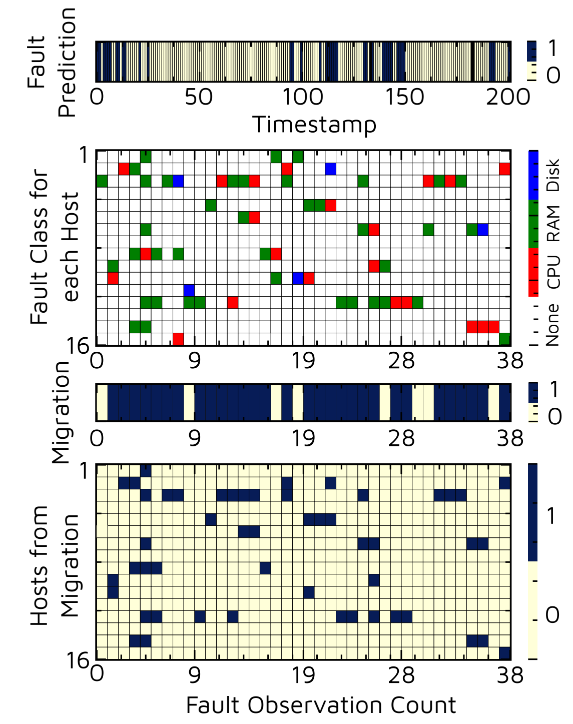


Figure 4: Visualization of fault prediction, classification and migration decisions with execution intervals.

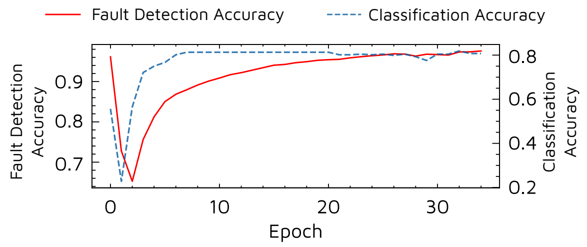


Figure 5: Training the Fault Prototype Encoder.

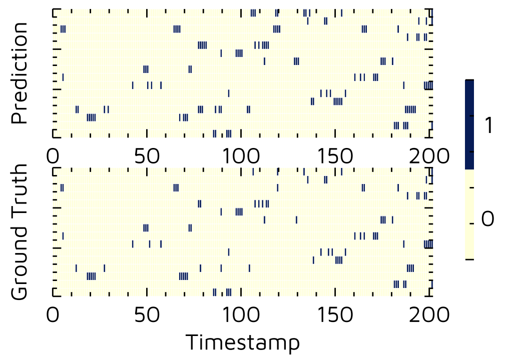
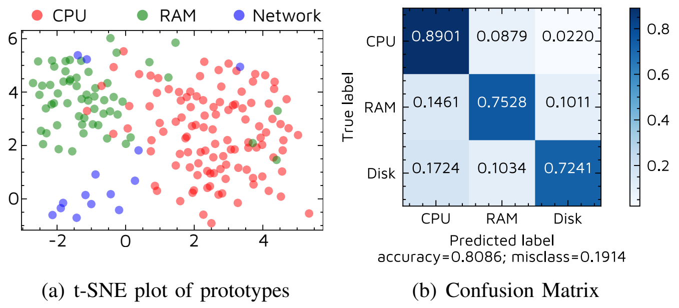


Figure 6: Fault Prediction with Ground Truth labels.



True label \ Predicted label	CPU	RAM	Disk
CPU	0.8901	0.0879	0.0220
RAM	0.1461	0.7528	0.1011
Disk	0.1724	0.1034	0.7241

(a) t-SNE plot of prototypes (b) Confusion Matrix

Figure 7: Fault Classification on the test set.

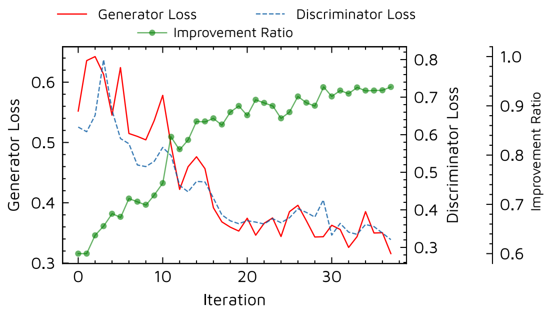


Figure 8: Training of the GAN network.

to the case where there is no preemptive migration. To explain these metrics with a simple visualization example on 200 intervals, consider Figure 9(f). The top heatmap denotes the intervals at which the model predicted a fault (fault denoted with black and yellow otherwise), specifically 38 intervals out of 200. The second heatmap shows the fault class prediction for each host (if any). The third heatmap shows the intervals in which the likelihood score of N was higher than S (30 out of 38) and the fourth heatmap shows the hosts from which there was a task migrated. Here, the improvement ratio is $30/38 = 0.7895$. Also, we observe a strong correlation between the fault class predictions and the host selected for task migration, thanks to the factored prediction in PreGAN.

We also consider standard QoS metrics including energy consumption, response time, fraction of SLO violations, migration count and resource utilization.

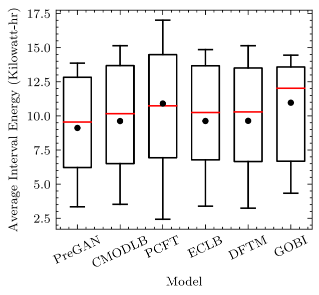
C. Implementation and Training

To conduct our tests, we use the COSCO framework that supports container orchestration in distributed edge clusters [24]. COSCO is at present the only framework that allows the generation of QoS scores using co-simulated traces. For a fair comparison, we use a common underlying scheduling policy, GOBI [24], that generates scheduling decisions (S_t) by optimizing QoS scores using a deep neural network based surrogate model.

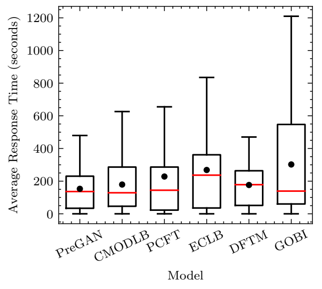
To generate the ground-truth fault labels and classes, we use the Anomaly Detection Engine for Linux Logs (ADE) tool [42]. The output anomaly classes for the tool are: CPU over-utilization (CPUO), Abnormal disk utilization (ADU), Memory leak (MEL), Abnormal memory allocation (AMA) and Network overload (NOL). CPUO occurs when the CPU utilization exceed 90%. When the disk controller of a system throttles read/write operations, an ADU fault occurs. MEL occurs when there is incremental allocation of 1 MB memory every 3 seconds. When RAM I/O utilization exceed 90%, it causes an AMA fault. NOL fault occurs when network buffer overloads and temporary disk space needs to be allocated for file transfers. All faults events are raised when the above conditions hold for at least 60 seconds. For simplicity, the CPUO anomaly is classified as a CPU fault, NOL/ADU are clubbed together as a Network fault (as network buffer

Table I: Performance scores of PreGAN and the baseline methods with standard deviation. The best scores are shown in bold.

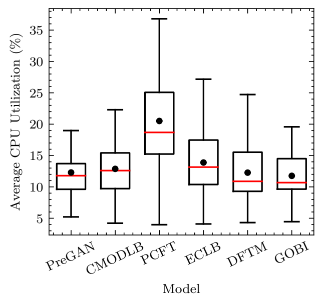
Method	Detection				Diagnosis		Overhead Ratio	Improvement Ratio
	Accuracy	Precision	Recall	F1 Score	HR@100	NDCG@100		
DFTM	0.8731 ±0.0234	0.7713 ±0.0823	0.8427 ±0.0199	0.8054 ±0.0872	0.5129 ±0.0212	0.4673 ±0.0019	**0.0413 ±0.0021**	0.3783 ±0.1001
ECLB	0.9413 ±0.0172	0.7812 ±0.0711	0.8918 ±0.0203	0.8329 ±0.0901	0.4913 ±0.0010	0.5239 ±0.0024	0.1028 ±0.0009	0.5912 ±0.0341
PCFT	0.8913 ±0.0108	0.8029 ±0.0692	**0.9018 ±0.0165**	0.8495 ±0.0312	0.5982 ±0.0094	0.5671 ±0.0020	0.0913 ±0.0014	0.6824 ±0.0473
CMODLB	0.9128 ±0.0112	0.8158 ±0.0343	0.9013 ±0.0091	0.8605 ±0.0284	**0.6309 ±0.0025**	0.5432 ±0.0031	0.2123 ±0.0003	0.7283 ±0.0065
PreGAN	**0.9635 ±0.00921**	**0.8723 ±0.0221**	**0.9018 ±0.0121**	**0.8868 ±0.0629**	0.6232 ±0.0069	**0.5898 ±0.0080**	0.1617 ±0.0017	**0.7605 ±0.0060**



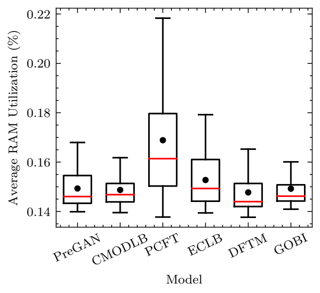
(a) Energy Consumption



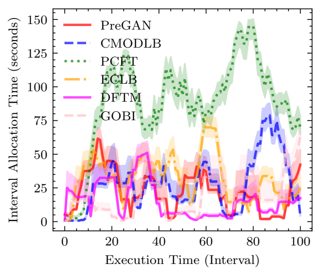
(b) Response Time



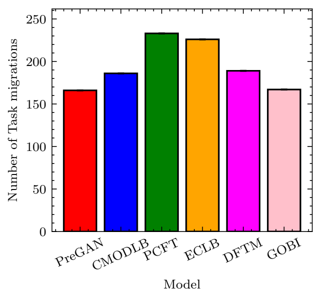
(c) CPU Utilization



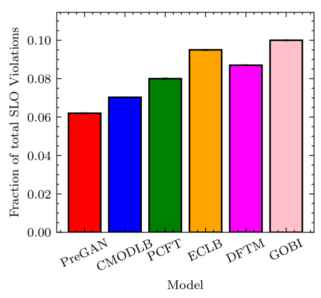
(d) RAM Utilization



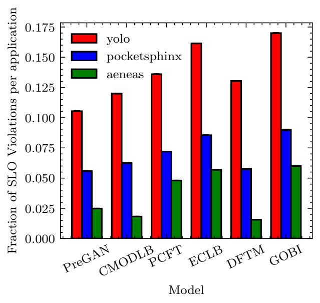
(e) Migration Time vs Interval



(f) Migration Counts



(g) Fraction of SLO Violations



(h) SLO Violations (per application)

Figure 9: Comparison of QoS parameters of PreGAN against baselines.

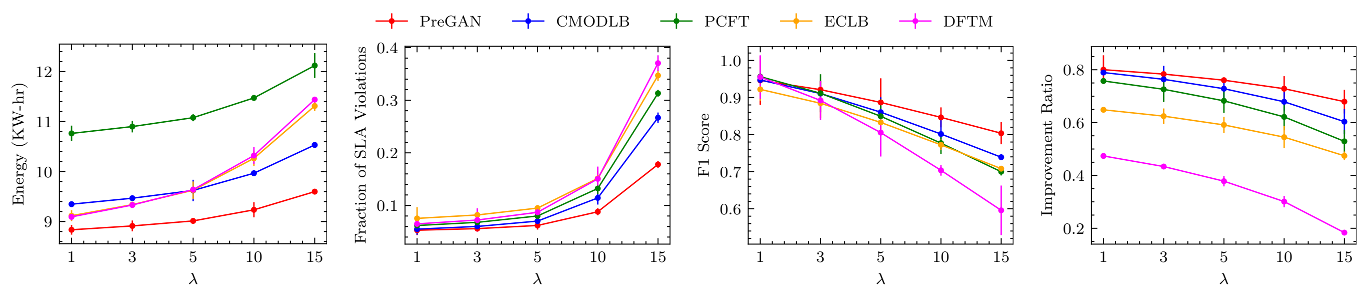
overloads almost always result in disk faults) and MEL/AMA are clubbed together as a RAM fault (due to high correlation between these two). Thus, we consider 3 fault types/classes.

To train the FPE and GAN models, we use the AdamW optimizer [43]. We train the FPE encoder and GAN using a traces of lengths 1000 and 1200 scheduling intervals respectively. These numbers were obtained using the early-stopping convergence criterion. The QoS score obtained from the co-simulations was a convex combination of the normalized energy consumption and SLO violations [24]. The hyperparameter values used in our experiments for PreGAN were obtained using grid search. We use the prototype embedding size of 8, windows size of 5, initial value of α as 0.9 and decay rate ϵ as 0.05.

Figure 5 shows how the FPE encoder converges in offline training using 80% of the dataset collected from traces of 1000 intervals (rest as test set). As the model is trained, its fault detection and classification accuracies improve. Figure 6 shows the predicted and ground-truth fault labels on the test set, with hosts on the y-axis and intervals on the x-axis. Figure 7(a) shows a t-SNE plot of the class prototypes predicted on the test set. Clearly, the prototype embeddings of the same class are clustered together, demonstrating that the model is able to distinguish among the different fault types. Classification confusion matrix on the test set can be seen in Figure 7(b). The prototype embeddings generation allows the model to generalize and accurately classify even previously unseen time-series inputs into one of the three classes. After training the FPE, we train the GAN. Figure 8 shows the discriminator (L_D) and generator (L_G) loss values with the the Improvement Ratio for the first 40 iterations, *i.e.*, the intervals with faults in the GAN training dataset.

D. Results

We now describe the results corresponding to 100 interval runs on the testbed using the DeFog workloads. The ground-truth labels are obtained offline, after the complete execution to compare the detection and diagnosis performance of the models. Table I shows the detection and diagnosis metrics with the improvement ratios. It also shows the overhead ratio in terms of the time it takes to run the preemptive migration model relative to the scheduling model. The overhead ratio is highest for CMODLB due to the several sequential steps in its model like k-means clustering, ANN inference, and PSO. On average, the accuracy, precision, recall and the F1 score of the PreGAN model are the highest. This is due to



(a) Energy Consumption (b) Fraction of SLO Violations (c) F1 Score (d) Improvement Ratio

Figure 10: Sensitivity Analysis for all models with λ (parameter of the discrete Poisson distribution).

the late-fusion of the embeddings obtained by the feature (GAT) and temporal trend (GRU) extraction. This allows the PreGAN model to not only exploit the correlations across host devices, but also sudden deviations from the expected resource utilization characteristics across time. Models like DFTM and ECLB use a basic sinusoidal thermal trend for all models and do not consider host heterogeneity. CMODLB uses k-means clustering to identify a common utilization threshold, irrespective of the running workloads or the host capacities. This gives higher fault detection scores for the PreGAN model. For fault-diagnosis, the CMODLB method has the highest HitRate (0.6309) with the PreGAN being very close (0.6232). The NDCG score of the PreGAN model is the highest. This is due to the factored fault prediction in the PreGAN model. Finally, the improvement ratio of the PreGAN model is the highest, *i.e.*, 0.7605 improving from 0.7283 of the CMODLB model by 4.42%.

Figure 9 compares the QoS scores of all models, including the vanilla GOBI approach without any preemptive migrations. The PreGAN model has the lowest average energy consumption of 9.0121 KW-hr, with DFTM being next at 9.4255 KW-hr. This is due to the relatively lower average CPU and RAM utilizations (Figures 9(c) and 9(d)). Figure 9(b) shows the average response times with response time being defined as the time between the creation of a task from an IoT sensor and the gateway receiving the response. Among the baselines, the DFTM approach has the lowest response time as it uses minimum-migration-time heuristic [44] for task migrations (Fig. 9(e)). PreGAN avoids unnecessary migrations to prevent avoidable use of network resources, improving overall system reliability (Fig. 9(f)). Even with higher response times, CMODLB and PreGAN achieve low SLO violation rates as their migration decisions are SLO aware. Figure 9(h) shows the SLO violation rates for each application. For all models, Yolo has the highest violation rate due to its compute heavy utilization characteristics. The migration decisions in PreGAN use the fault class labels to decide the target hosts in the migration decisions. The fine-grained classification in PreGAN allows it to migrate CPU intensive tasks from a compute constrained node to a node with low CPU utilization, even if the target node is running at capacity on RAM and Disk. Compared to the binary classification in CMODLB, this gives PreGAN more choices for the target host, allowing lower overheads in migration and more balanced resource utilization.

E. *Sensitivity Analysis*

Figure 10 shows the variation of the energy consumption, SLO violations, F1 score and improvement ratio with the λ parameter in our Poisson distribution used to model the workloads. We vary λ from 1 to 15 ($\lambda = 15$ constantly gives $> 90\%$ CPU utilization for all hosts). Under a higher λ more tasks are produced, making the fault prediction harder. This is apparent from the drop in the F1 scores, leading to higher SLO violations. Even the energy consumption increases due to the increase in the average CPU utilization of the system. Overall, PreGAN shows the least relative drop in F1 scores and improvement ratio as we increase λ giving the least SLO violations even in workload heavy executions.

V. Conclusions

We have presented a preemptive migration prediction model (PreGAN) that can detect, diagnose and classify faults in edge computing environments. PreGAN uses GAT and GRU for feature extraction with a Multi-Head-Attention and Prototype prediction decoder to detect and classify faults. PreGAN leverages a generator model to utilize the anomaly class prototypes to output a delta scheduling decision (migrations) to rectify the faults and improve QoS. The discriminator model with co-simulations allow PreGAN to decide between the original and modified scheduling decisions. Moreover, PreGAN's discriminator aids generator training using adversarial loss and does not require it to run co-simulations at test time. This allows PreGAN to have high detection and classification accuracies that aid efficient fault recovery for optimal QoS. Specifically, PreGAN achieves an improvement of 8%, 5% and 12% for energy consumption, response times and SLO violations respectively. It is also able to correctly identify faults, giving an average F1 score of 0.8868, higher than the state-of-the-art models. PreGAN is able to achieve this with 23.8% lower overheads compared to the baseline method with the highest F1 score. This makes PreGAN an ideal choice for reliable edge computing with time-critical applications.

As part of future work, we propose to extend PreGAN to utilize unsupervised models to work in settings without labeled data [45]. Further, the current model assumes a master-slave design and we plan to explore extensions to enable deploying PreGAN in serverless platforms with streaming tasks [46].

The PreGAN code is available at `https://github.com/imperial-qore/PreGAN`.

References

[1] S. S. Gill, S. Tuli *et al.*, "Transformative effects of IoT, Blockchain and Artificial Intelligence on cloud computing: Evolution, vision, trends and open challenges," *Internet of Things*, vol. 8, pp. 100–118, 2019.

[2] S. Tuli, S. Ilager *et al.*, "Dynamic Scheduling for Stochastic Edge-Cloud Computing Environments using A3C learning and Residual Recurrent Neural Networks," *IEEE Transactions on Mobile Computing*, 2020.

[3] J. Liu, S. Wang *et al.*, "Using proactive fault-tolerance approach to enhance cloud service reliability," *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 1191–1202, 2016.

[4] A. V. Dastjerdi and R. Buyya, "Fog computing: Helping the internet of things realize its potential," *Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[5] B. Nicoletti, *Cloud computing in financial services*. Springer, 2013.

[6] D. Park, S. Kim *et al.*, "LiReD: A light-weight real-time fault detection system for edge computing using LSTM recurrent neural networks," *Sensors*, vol. 18, no. 7, p. 2110, 2018.

[7] S. Malik and F. Huet, "Adaptive fault tolerance in real time cloud computing," in *2011 IEEE World Congress on services*. IEEE, 2011, pp. 280–287.

[8] S. Ristov, T. Fahringer *et al.*, "Resilient techniques against disruptions of volatile cloud resources," in *Guide to Disaster-Resilient Communication Networks*. Springer, 2020, pp. 379–400.

[9] A. Sharif, M. Nickray *et al.*, "Fault-tolerant with load balancing scheduling in a fog-based IoT application," *IET Communications*, vol. 14, no. 16, pp. 2646–2657, 2020.

[10] S. Negi, M. M. S. Rauthan *et al.*, "CMODLB: an efficient load balancing approach in cloud computing environment," *The Journal of Supercomputing*, pp. 1–53, 2021.

[11] V. Sivagami and K. Easwarakumar, "An improved dynamic fault tolerant management algorithm during vm migration in cloud data center," *Future Generation Computer Systems*, vol. 98, pp. 35–43, 2019.

[12] P. Kumari and P. Kaur, "A survey of fault tolerance in cloud computing," *Journal of King Saud University-Computer and Information Sciences*, 2018.

[13] M. Zhou, R. Zhang *et al.*, "Security and privacy in cloud computing: A survey," in *2010 Sixth International Conference on Semantics, Knowledge and Grids*. IEEE, 2010, pp. 105–112.

[14] Z. Zheng, T. C. Zhou *et al.*, "Component ranking for fault-tolerant cloud applications," *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 540–550, 2011.

[15] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–37, 2019.

[16] W. Z. Khan, E. Ahmed *et al.*, "Edge computing: A survey," *Future Generation Computer Systems*, vol. 97, pp. 219–235, 2019.

[17] C. Engelmann, G. R. Vallee *et al.*, "Proactive fault tolerance using preemptive migration," in *2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2009, pp. 252–257.

[18] B. Tian, C. Tian *et al.*, "Scheduling coflows of multi-stage jobs to minimize the total weighted job completion time," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 864–872.

[19] D. Li, D. Chen *et al.*, "MAD-GAN: Multivariate anomaly detection for time series data with generative adversarial networks," in *International Conference on Artificial Neural Networks*. Springer, 2019, pp. 703–716.

[20] P. Veličković, G. Cucurull *et al.*, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[21] J. Chung, C. Gulcehre *et al.*, "Empirical evaluation of gated recurrent neural networks on sequence modeling," in *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[22] A. Vaswani, N. Shazeer *et al.*, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 6000–6010.

[23] J. Snell, K. Swersky *et al.*, "Prototypical networks for few-shot learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 4080–4090.

[24] S. Tuli, S. R. Poojara *et al.*, "COSCO: Container Orchestration Using Co-Simulation and Gradient Based Optimization for Fog Computing Environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 101–116, 2022.

[25] S. M. Ataallah, S. M. Nassar *et al.*, "Fault tolerance in cloud computing-survey," in *2015 11th International computer engineering conference (ICENCO)*. IEEE, 2015, pp. 241–245.

[26] B. Ray, A. Saha *et al.*, "Proactive fault-tolerance technique to enhance reliability of cloud service in cloud federation environment," *IEEE Transactions on Cloud Computing*, 2020.

[27] B. Mohammed, M. Kiran *et al.*, "Failover strategy for fault tolerance in cloud computing environment," *Software: Practice and Experience*, vol. 47, no. 9, pp. 1243–1274, 2017.

[28] A. Satpathy, S. K. Addya *et al.*, "Crow search based virtual machine placement strategy in cloud data centers with live migration," *Computers & Electrical Engineering*, vol. 69, pp. 334–350, 2018.

[29] L. Wang, W. Mao *et al.*, "DDQP: A double deep Q-learning approach to online fault-tolerant SFC placement," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 118–132, 2021.

[30] S. Tuli, R. Mahmud *et al.*, "Fogbus: A blockchain-based lightweight framework for edge and fog computing," *Journal of Systems and Software*, 2019.

[31] D. Basu, X. Wang *et al.*, "Learn-as-you-go with megh: Efficient live migration of virtual machines," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 8, pp. 1786–1801, 2019.

[32] J. Audibert, P. Michiardi *et al.*, "USAD: UnSupervised Anomaly Detection on Multivariate Time Series," in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 3395–3404.

[33] S. Tuli, R. Bansal *et al.*, "TANGO: Commonsense Generalization in Predicting Tool Interactions for Mobile Manipulators," *International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.

[34] S. Tuli, S. S. Gill *et al.*, "HUNTER: AI based holistic resource management for sustainable cloud computing," *Journal of Systems and Software*, pp. 111–124, 2021.

[35] A. Paszke, S. Gross *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Advances in Neural Information Processing Systems*, vol. 32, pp. 8026–8037, 2019.

[36] Standard Performance Evaluation Corporation. Spec power consumption models. [Online]. Available: https://www.spec.org/cloud_iaas2018/results/

[37] J. McChesney, N. Wang *et al.*, "DeFog: fog computing benchmarks," in *The 4th ACM/IEEE Symposium on Edge Computing*, 2019, pp. 47–58.

[38] Y. Mao, J. Zhang *et al.*, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[39] H. Zhao, Y. Wang *et al.*, "Multivariate time-series anomaly detection via graph attention network," *International Conference on Data Mining*, 2020.

[40] Y. Su, Y. Zhao *et al.*, "Robust anomaly detection for multivariate time series through stochastic recurrent neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2828–2837.

[41] K. Järvelin and J. Kekäläinen, "Cumulated gain-based evaluation of ir techniques," *ACM Transactions on Information Systems (TOIS)*, vol. 20, no. 4, pp. 422–446, 2002.

[42] Open Mainframe Project. Anomaly Detection Engine for Linux Logs (ADE). [Online]. Available: https://www.openmainframeproject.org/projects/anomaly-detection-engine-for-linux-logs-ade

[43] N. Saleh and M. Mashaly, "A dynamic simulation environment for container-based cloud data centers using containercloudsim," in *2019 Ninth International Conference on Intelligent Computing and Information Systems (ICICIS)*. IEEE, 2019, pp. 332–336.

[44] A. Beloglazov and R. Buyya, "Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[45] Z. He, P. Chen *et al.*, "A spatiotemporal deep learning approach for unsupervised anomaly detection in cloud systems," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[46] G. Casale, M. Artač *et al.*, "Radon: rational decomposition and orchestration for serverless computing," *SICS Software-Intensive Cyber-Physical Systems*, vol. 35, no. 1, pp. 77–87, 2020.