EdgeMLSim: A Framework for Modeling and Simulation of Edge Computing and Evaluation of Fault-tolerant Machine Learning

***Abstract*—Billions of connected devices and entities are expecting to share a tremendous amount of data every day. Various machine learning (ML) systems designed for the edge computing paradigm have been proposed to extract insights from the data. Understanding and quantifying the performance, fault-tolerance, and ML metrics (e.g., prediction accuracy) of these systems under varying load, system size, computation resource, and communication constraint is an immense challenge. To streamline the process, this paper presents EdgeMLSim: a generalized and extensible simulation framework that enables seamless modeling, simulation, and experimentation of fault-tolerant edge-based ML algorithms.**

***Index Terms*—Machine Learning, Edge Computing, Simulation, Fault-tolerance**

I. Introduction

To handle the exponential growth of connected devices and the data they collect and generate, numerous frameworks have been proposed to train machine learning (ML) model on top of edge or wireless devices, e.g., mobile phones, sensor networks, smart transportation, and Industrial Internet-of-Things (IIoT), etc [10]. Several ML algorithms from both academia and industry are specifically designed for the edge computing architectures, including edge learning framework [5], eSGD [9], fog learning [4], and federated learning on mobile phones at Google [1], etc. This emerging trend was perfectly summarized by Nic Lane in the MobiCom20 keynote [6]. He concluded the keynote with a big prediction: "*Devices (will) replace Data Centers as core of ML.*"

One reason that we are still far from Nic's ambitious prediction is that we, as a community, still lack fundamental understanding of existing systems, especially the fault-tolerance aspect. To address this issue, we propose EdgeMLSim, which allows practitioners to quickly understand the performance, ML metrics (e.g., accuracy, precision, F1-score, etc.), fault-tolerance, and bottlenecks of different ML systems in a straightforward way. More importantly, our tool allows research teams to compare the developing system with state-of-the-art systems fairly and comprehensively, and supports reproduction and replication of research efforts.

As a first step, we focus on the distributed deep learning framework that is integrated with the three-layer edge computing architecture and the parameter server ML training, as depicted in Figure 1. Such an architecture is adopted in many prior works, e.g., [4], [5], [9], [10].

ML Preliminary: EdgeMLSim supports a widely adopted approach – distributed Stochastic Gradient Descent (SGD)

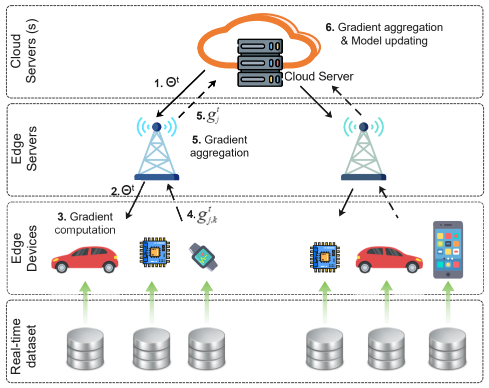


Fig. 1: Three-level Architecture of Edge Computing.

algorithms. Given a cost function Q, the SGD algorithm is designed to output an optimal parameter θ^* in a d-dimensional space such that the cost function is minimized. In the edge computing paradigm, each edge server j has a local cost function Q_j, which captures the characteristics of the **real-time location-specific data**. That is, each location may have a different set of data pattern (non-i.i.d. data distribution). The distributed SGD algorithm aims to minimize the overall cost function and compute θ^* such that the sum of all the cost functions is minimized:

$$\theta^* = \underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{j=1}^{n} Q_j(\theta) \tag{1}$$

In the three-level edge computing architecture (as depicted in Figure 1), both edge servers and cloud server need to perform the aggregation, but only the cloud server will update the parameter. In the usage scenario, each gradient computation is over a small batch of data samples; hence, the computation is very efficient, even on a small device.

For brevity of presentation, we assume a *synchronous* system, i.e., all the servers and edge devices proceed in synchronous rounds. EdgeMLSim supports asynchronous and partially asynchronous systems. Currently, it integrates distributed SGD algorithms with the three-level architecture, in which nodes execute the following steps in each round $t \geq 0$: (1) Cloud server sends parameter θ^t to all edge servers; (2) Edge server j forwards θ^t to nearby workers $w_{j,k}^t$; (3) Each worker $w_{j,k}^t$ randomly chooses a (small) data batch from its local dataset that are collected using on-device sensors from its current location, and computes a local estimate gradient $g_{j,k}^t$; (4) Worker then sends $g_{j,k}^t$ back

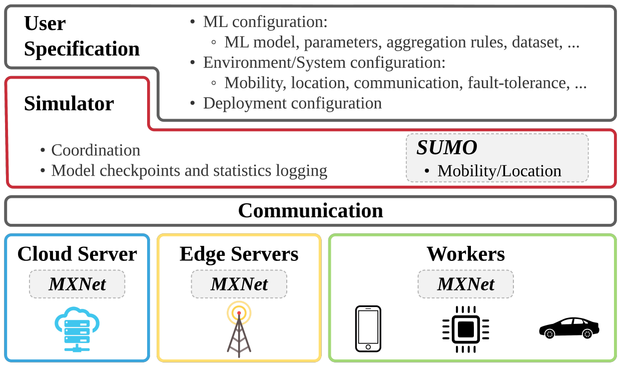

User Specification
• ML configuration:
◦ ML model, parameters, aggregation rules, dataset, ...
• Environment/System configuration:
◦ Mobility, location, communication, fault-tolerance, ...
• Deployment configuration
Simulator
• Coordination
• Model checkpoints and statistics logging
SUMO
• Mobility/Location
Communication
Cloud Server
MXNet
Edge Servers
MXNet
Workers
MXNet

Fig. 2: The architecture of EdgeMLSim.

to the edge server j; (5) Upon collecting enough gradients from nearby workers, edge server j computes and sends an aggregated gradient g_j^t to cloud server; and (6) Upon collecting aggregated gradients from all the edge servers, cloud server then updates the parameter using the gradient descent approach with step size η: $\theta^{t+1} = \theta^t - \eta \sum_{j=1}^{n} g_j^t$.

Main Contribution: We present EdgeMLSim, a light and flexible framework that allows users to quickly evaluate the design of edge-based ML systems under different configurations. Due to the popularity and wide adoption, we build EdgeMLSim around MXNet[1] and SUMO[2] – MXNet provides the deep learning framework designed for flexible prototyping and SUMO provides realistic discrete-time and trace-based simulation of mobility pattern of moving edge devices. Figure 2 presents the architecture of EdgeMLSim and its key components – gray boxes represent the integrated libraries. Development on top of MXNet allows our framework to distribute the training of most ML models developed for MXNet into the edge computing paradigm. Essentially, EdgeMLSim simplifies the experiment and simulation on a cluster with heterogeneous machines by providing (i) automatic cluster-wide allocation of ML training tasks; (ii) specification of the target edge computing system; and (iii) simulation of mobility, communication, and faults.

We stress that EdgeMLSim trains an ML model using MXNet on real machines with real datasets; hence, users can trust the reported performance and metrics. Our preliminary implementation will be released if this poster is accepted.

Related Work: Simulation is a mature technique to imitate operations and complicated interaction among multiple factors within a real-world system. Many simulation systems have been developed for various purposes, including NS-3 and OMNet++ for network protocols, CloudSim [3] for cloud systems, PeerSim [8] for peer-to-peer systems, iFogSim [2] for edge computing systems, etc. To our knowledge, none of the systems have natural support for ML tasks.

[1]mxnet.apache.org/
[2]Simulation of Urban MObility: www.eclipse.org/sumo/

II. Design of EdgeMLSim

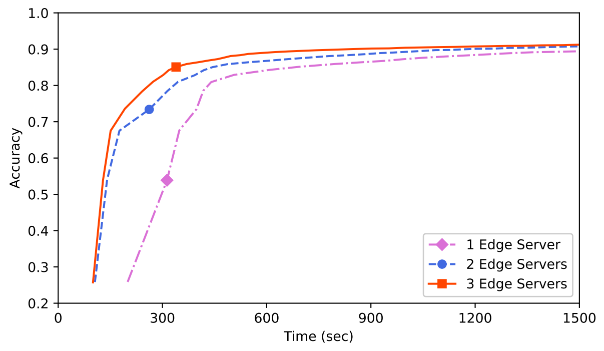
Due to its popularity in the ML literature, we develop EdgeMLSim using Python. EdgeMLSim has four main components, as described below and in Figure 2:

- *User Specification*: this component allows users to specify their edge-based ML system under evaluation. Users can specify, through configuration files, four key aspects: (i) *ML training*: ML model, batch size, learning rate, aggregation rules (including when to aggregate gradients, and how and where to aggregate them); (ii) *edge computing system*: number of edge servers/devices, location and mobility of edge devices, communication range, processing power; (iii) *faulty behavior*: crash/Byzantine, global/local fault models, customized Byzantine attacks and aggregation rules at different layers;[3] (iv) *simulation deployment*: standalone or cluster modes. In the first mode, all the simulation and training are conducted on a single machine, whereas EdgeMLSim can also be deployed to multiple machines to have a more realistic and parallel evaluation in the cluster mode.
- *Simulator*: The simulator acts as a coordinator of the simulation. It has five responsibilities: (i) loading training, validation, and testing datasets; (ii) using SUMO to generate mobility and location information which are used to generate the time-location data for each edge devices; (iii) coordinating computing nodes to perform the ML training tasks; (iv) simulating and managing real-time location-specific data; (v) collecting statistics and metrics, and logging model checkpoints.
- *Communication*: In the current version, we connect computing nodes and simulator with TCP channels. Computing nodes use channels to exchange gradients and parameters of the ML model. The simulator sends control messages via the channels as well. We plan to integrate some features from NS-3 and OMNet++ to model more realistic wireless communication, especially the ones among edge devices, in the future.
- *Computing Nodes*: following Figure 1, we have three types of computing nodes. Cloud server and edge servers use MXNet to update model, and aggregate gradients. Workers use MXNet to compute a local estimate gradient. Based on user's specification, some nodes may perform Byzantine attacks.

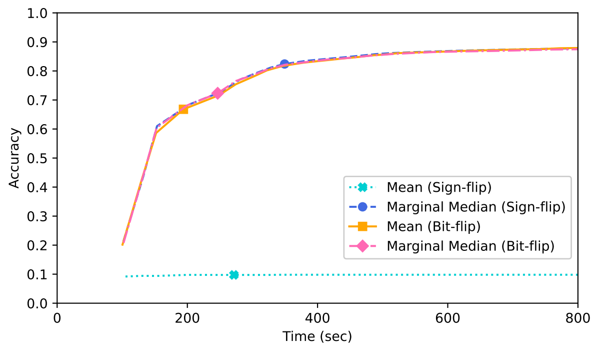
To simulate real-time dataset generated or collected at each edge device, we first randomly form data batches and then allocate to each edge server to model the scenario that data distribution in each region (or location) may have distinctive characteristics. The data batches can be configured with i.i.d. distribution (independent and identically distributed) and *disjoint local* datasets (or non-i.i.d. datasets).

In EdgeMLSim, we implement a clock at the Simulator to keep track of time spent on ML tasks and communication.

[3]EdgeMLSim is designed to be modular, so that experienced users can also implement desired Byzantine behaviors and ML algorithms in Python.



(a) Accuracy vs. #Edge Servers



(b) Accuracy vs. Fault-tolerance

Fig. 3: Results of Preliminary Evaluation

The clock starts ticking when an experiment starts and pauses when the Simulator is processing the auxiliary simulating tasks between epochs, e.g., shuffling data, logging model checkpoints, and recording metrics. We only quantify the time for nodes to perform training and communication.

III. Preliminary Evaluation

Our preliminary evaluation uses the MNIST handwritten digits [7], which contains 60k training examples, and a test set of 10k examples, for our experiments. We define our ML model to be a Sequential model with 3 dense layers, batch size 100, and learning rate 0.001. For mobility, we use the real-world map of Boston Common and traffic generated by SUMO. We pre-select locations of the edge servers, with communication range of 50 meters, which means that edge devices need to be within the range to exchange messages with edge servers. Data is distributed in the i.i.d. fashion.

All experiments are conducted on Google Cloud Platform (GCP) in order to simulate the case that tasks are performed in a distributed fashion (i.e., using a cluster mode). The VM specifications of the simulator and cloud server are e2-standard-2 (2 vCPUs, 8 GB memory). Edge servers and workers are e2-standard-8 (8 vCPUs, 32 GB memory) and e2-highcpu-32 (32 vCPUs, 32 GB memory), respectively. All VMs run Ubuntu 18.04. All the experiments have 1 cloud server, 1 to 3 edges servers (deployed on the same VM), and 100 workers which are evenly distributed into 2 groups (and deployed on 2 VMs). Depending on the mobility of edge devices, not all 100 workers perform the training at the same time. Also, each device uses at most 1 vCPU, which is a realistic assumption.

Our first experiment studies the impact of varying number of edge servers. We use Mean as the aggregation method and 10 gradients are accumulated before each aggregation in a synchronous round. We do not apply Byzantine faults for this experiment. We run each configuration 5 times and report the average in Figure 3a. Not surprisingly, increasing the number of edge servers helps the model converge faster and more accurately. More edge servers provide a broader coverage on the map, thus enabling more edge devices to perform training in parallel.

Our second experiment evaluates the fault-tolerance of two aggregation methods, Mean and Marginal Median [11], against two Byzantine faults – Bit-flip attack and Sign-flip attack. The other parameters are the same as the previous experiment except that we fix the number of edge servers to 3. Figure 3b presents the average of 5 runs. Both Mean and Marginal Median tolerate Bit-flip attack. Only Marginal Median tolerates Sign-flip attack, but Mean does not.

While the results are not surprising intuitively, they demonstrate EdgeMLSim's potential of evaluating realistic settings in edge-based ML systems. Key factors that are difficult to quantify in theory, but can be estimated using our tool include: (i) data distribution; (ii) mobility pattern and computation power of edge devices; (iii) communication range/loss; (iv) faults; and (v) location and computation power of edge servers, etc.

References

[1] Federated Learning: Collaborative Machine Learning without Centralized Training Data, Apr 2017.
[2] R. Buyya and S. N. Srirama. *Modeling and Simulation of Fog and Edge Computing Environments Using iFogSim Toolkit*. 2019.
[3] R. N. Calheiros et al. Cloudsim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw. Pract. Exper.*, 41(1):23–50, Jan. 2011.
[4] S. Hosseinalipour et al. From federated learning to fog learning: Towards large-scale distributed machine learning in heterogeneous wireless networks, 2020.
[5] Y. Huang et al. When deep learning meets edge computing. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*.
[6] N. Lane. Keynote: What is next for the efficient machine learning revolution? In *The 26th MobiCom*, 2020.
[7] Y. Lecun et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
[8] A. Montresor and M. Jelasity. Peersim: A scalable p2p simulator. In *2009 IEEE International Conference on Peer-to-Peer Computing*.
[9] Z. Tao and Q. Li. esgd: Communication efficient distributed deep learning on the edge. In *USENIX HotEdge 2018*.
[10] X. Wang et al. Convergence of edge computing and deep learning: A comprehensive survey. *IEEE Communications Surveys Tutorials*, 22(2):869–904, 2020.
[11] D. Yin, et al. Byzantine-robust distributed learning: Towards optimal statistical rates. In ICML 2018.