This artifact describes the key contributions of the paper followed by the required steps to reproduce the experimental results presented in the paper.

I. Artifact Identification

A. Abstract

Graph Neural Networks (GNNs) are powerful models for learning over graphs. To speed up training on very large, real-world graphs (billion scale edges) several distributed frameworks have been developed. A fundamental step in every distributed GNN framework is graph partitioning. On several benchmarks, we observe that these partitions have heterogeneous data distributions which affect model convergence and performance. They also suffer from class imbalance and out-of-distribution problems, resulting in lower performance than centralized implementations.

We holistically address these challenges, by developing entropy-aware partitioning algorithms that minimize total entropy and/or balance the entropies of graph partitions. We observe that by minimizing the average entropy of the partitions, the micro average F1 score (accuracy) can be improved. Similarly, by minimizing the variance of the entropies of the partitions and implementing a class-balanced sampler with Focal Loss, the macro average F1 score can be improved. We divide the training into a synchronous, model generalization phase, followed by an asynchronous, personalization phase that adapts each compute host's models to their local data distributions. This boosts all performance metrics and also speeds up the training process significantly.

We have implemented our algorithms on the DistDGL framework where we achieved a 4% improvement on average in micro-F1 scores and 11.6% improvement on average in the macro-F1 scores of 5 large graph benchmarks compared to the standard baselines.

B. System and Data Model

To evaluate our system, we need to partition the given input graph (using DGL graph data format), and subsequently perform distributed training on multiple compute hosts. To partition a graph into N partitions, we require a single compute host. However, to perform distributed training, we require cluster of N compute hosts. For each compute host, we assign a partitioned graph along with a set of train, validation and test nodes. We distributed these partitions to the compute hosts via a network file system (NFS). For distributed training on N compute hosts, we configure one compute host as a server and the remaining hosts as clients. All the clients access the common storage via NFS.

II. Artifact Dependencies and Requirements

1) Hardware Configuration

To evaluate the artifcats on the smaller datasets, we require 4 compute hosts with 32 GB memory and NFS. To evaluate them on larger graphs (i.e., OGB-Papers), we require 16 machines with 128GB RAM with NFS. For smaller datasets we performed our experiments on AWS using 32GB RAM r6a.xlarge machine using 4 instances. We configured the instances with NFS. For larger datasets we have used a shared commodity cluster, using up to 128 GB memory on each compute host.

Note that for ease of artifact evaluation, we can provide the AWS container with the required environment for smaller datasets. However, our system can also configured by provisioning and setting up the AWS instances with the following instructions.

A. Server Configuration:

Create an Ec2 instance with the following settings.

```
name: pc_node
os: ubuntu 22.04
instance type: r6a.xlarge
default vpc
subnet: ap-south 1c
existing security grp: launch-wizard-1

In security grp select: nfs-server
Instance type: r6a.4xlarge
```

While creating the instance, it is required to enter the following script in the textbox corresponding to the additional details. These instructions will be execute at the time of launching the machine.

```
Common for server and client
#!/bin/bash
//Assuming home directory as /home/ubuntu
cd /home/ubuntu
export HOME=/home/ubuntu
wget \
https://repo.continuum.io/miniconda/\
Miniconda3-latest-Linux-x86_64.sh \
-O /home/ubuntu/miniconda.sh
chmod +x miniconda.sh
bash miniconda.sh -b \
-p /home/ubuntu/miniconda3
source miniconda3/bin/activate
conda create -y -n envforgnn python=3.9
conda activate envforgnn
conda install -y pytorch==1.9.0 \
torchvision==0.10.0 \
torchaudio==0.9.0 \
cpuonly -c pytorch
conda install -y -c dglteam dgl=0.9
conda install -y pandas
conda install -y scikit-learn
conda install -y matplotlib-base
conda install -c conda-forge ogb
export DGLBACKEND=pytorch

sudo apt-get install -y nfs-kernel-server
mkdir -p /home/ubuntu/workspace
sudo -- bash -c 'echo \
"/home/ubuntu/workspace \
```

```
172.31.16./20 \
(rw,no_root_squash,sync,no_subtree_check)" \
>> /etc/exports'
sudo systemctl restart nfs-kernel-server
```

Once the server is configured, it required to be launched before the clients can be configured.

B. Client Configuration

Create 3 Ec2 instances each with the following configuration.

```
name: pc_node
os: ubuntu 22.04
instance type: r6a.xlarge
default vpc
subnet: ap-south 1c
existing Security grp: launch-wizard-1
Storage: 8 gb ssd
```

Similar to server configuration, it is required to enter the following script in the textbox corresponding to the additional details. These instructions will be execute at the time of launching the machine.

```
#!/bin/bash
cd /home/ubuntu
export HOME=/home/ubuntu
wget \
https://repo.continuum.io/miniconda/\
Miniconda3-latest-Linux-x86_64.sh \
-O /home/ubuntu/miniconda.sh
chmod +x miniconda.sh
bash miniconda.sh -b \
-p /home/ubuntu/miniconda3
source miniconda3/bin/activate
conda create -y -n envforgnn python=3.9
conda activate envforgnn
conda install -y pytorch==1.9.0 \
torchvision==0.10.0 \
torchaudio==0.9.0 \
cpuonly -c pytorch
conda install -y -c dglteam dgl=0.9
conda install -y pandas
conda install -y scikit-learn
conda install -y matplotlib-base
conda install -c conda-forge ogb
export DGLBACKEND=pytorch

sudo apt-get install -y nfs-common
mkdir -p /home/ubuntu/workspace
sudo mount -t nfs \
<put nfs server private ipv4>:\
/home/ubuntu/workspace \
/home/ubuntu/workspace
mount -a
```

Once the server and the clients are launched, they need to be configured with passwordless authentication using the instructions given at https://linuxize.com/post/how-to-setup-passwordless-ssh-login/

The Data Model consists of mainly the graphs. The graphs have been stored in the DGL graph data format. In this format, a folder is made for a set of partitions of a graph. There is JSON file detailing the node and edge splits for each partition, and then there are respective folders for each partition which store node and edge features in binary format.

III. Artifact Installation

The artifacts can be downloaded from the following git hub repository and can be set up using the instructions provided in the *ReadME*. https://anonymous.4open.science/r/EAT-DistGNN-398C/README.md The repository contains the source code for partitioning and performing distributed training. The partitioning code contains three partitioning strategies described in the paper, i.e., METIS, Edge Weighted (EW), Entropy Balanced (EB). The training code consists of all our distributed training algorithms: 1) Class Balanced Sampler (CBS), 2) Generalized-Personalized (GP) model, along with scripts to run them with required hyper-parameters.

Through the artifacts, we expect to reproduce all the major results listed in the paper, however, a statistical variability can be expected. Moreover, graph partitioning and training time will depend on the machine configurations.

IV. Reproducibility of Experiments

The experimental workflow has the following steps: (1) graph partitioning, (2) distributed training, and (3) post processing to generate the plot graphs and tables listed in the paper. To ease the workflow for evaluators we automated all the above steps for each graph. The *README* present in the repository shows the instructions to execute the automated script on each dataset.

The times required for completing all experiments listed in the paper, for each of the input graph are given below. Note that these are the times are upper bounds on the hardware configuration used in the paper.

1) Flickr - 1 hour
2) Yelp - 4 hours
3) OGBN-Products - 6 hours
4) Reddit - 5 hours
5) OGB-Papers - 20 hours

Running the automated script file for each respected graph will generate the following tables and figures reported in the paper.

1) Table for average entropy, std. dev in entropies and time to partition, across various graph partitioning algorithms.
2) Table for comparing performance metrics of various algorithms for different graph datasets
3) Table for performance analysis of OGB-Papers dataset using METIS partitioning

4) Table for wall clock times (in sec) across all the 4 partitions of each partitioning scheme for various graph datasets
5) Figure for the convergence curves for training loss, validation micro and macro-F1 scores for Flickr, Reddit, Yelp and OGBN-Products using various partitioning schemes
6) Figure for Average training speeds across all partitions of default partitioning scheme for various graph datasets.