Distributed Belief Propagation

Ayşe Irmak Erçevik , Mehmet Deniz Türkmen

TOBB University of Economics and Technology

Abstract. With the developments such as mobile banking and blockchain technology, money transfers have become very easy. This convenience also accelerated and increased the circulation of money among people. However, tracking and control of money movements has also become difficult. As a result, crimes such as fraud and money laundering found an field of activity. In this study, we propose to use a distributed storage and computation system in order to track money transfers instantly. In particular, we keep our transaction history in a distributed file system as a graph data structure. We try to detect illegal activities by using Graph Neural Networks (GNN) in distributed manner. We simulate incoming transactions as a mini batch of inference data. Finally, we compare our proposed parallel system with a linear system and show the temporal efficiency of our method.

1 Intoduction

Graph Neural Networks (GNN) have attracted much attention in recent years. It obtained promising results on a form of data that is more general and flexible than images or tables. In this work, we propose distributed Graph Convolutional Network (GCN) to detect fraudulent transactions on dynamic Bitcoin transaction graphs. We prefer Spark Framework for resource partitioning and synchronous task managing.By using, Ray methods in BigDL-ORCA Framework, we train and inference the model in a distributed manner.In addition, we analyze the execution time of the train and inference phases which is affected by the Mode of Spark Cluster, number of workers per node and resources assigned per node.

2 Related Work

2.1 Distributed Graph Processing

Graph data can be quite large in size. As a result, there has been a lot of work on the idea of keeping and processing graphs in a distributed manner.

Pregel is a framework aiming efficient processing of graphs. It is proposed by malewicz2010pregel and designed especially for optimization of iterative message passing algorithms. gonzalez2014graphx introduce GraphX, a framework inheriting Apache Spark's features and providing efficient storage and

computation on graph data. GraphX comes with built-in APIs for parallel computation of message passing algorithms as Pregel does. chen2019powerlyra propose PowerLyra, another graph-special framework. It differs from other by employing a degreesensitive partitioning approach to achieve a balanced distribution of especially skewed graphs. van2022gdll presents GDLL, a library for distributed training of graph neural networks. They define their architecture with 3 layers: distributed storage of graphs, distributed processing of graphs and efficient distributed training of GNNs. They take into account the number of layers in the GNN for partitioning to reduce data transfer overhead. wang2019deep introduces Deep Graph Library (DGL), a open-source library for distributed storage of Graphs and distributed training of GNNs.

2.2 Fraud Detection

The recent changes in the banking system have brought a risk of fraudulent actions along with easiness. Thus, many systems that are responsible for the conservation of money put effort into preventing such occurrences. Many research sharing the same motivation focus on the detection of fraudulent activities.

varmedja2019credit test supervised machine learning algorithms for fraud detection and compare them. carcillo2021combining approach the fraud detection problem as an anomaly detection problem and draw attention to supervised learning methods' insufficiency in extracting anomalies. Accordingly, they propose employing unsupervised learning methods for anomaly detection and, combining unsupervised and supervised methods. liu2020alleviating address the problematic sides of neighbor aggregation in GNNs that can cause fraudulent clues to vanish. They introduce GraphConsis, a context-aware model. wang2018privacy presents an architecture concerning customer privacy issues. The architecture provides a infrastructure where data is processed in a distributed manner for fraud detection and identity privacy is ensured.

3 Proposed Method

3.1 Dataset

In this study, we test our architecture on Elliptic bitcoin dataset [1] . The dataset constitutes a directed acyclic graph (DAG). In the graphs, nodes represent transactions and directed edges represent bitcoin flows between transactions. The graph contains 203769 nodes and 234355 edges. Each node is labeled with one of three category: licit, illicit (e.g., (scams, malware, terrorist organizations, ransomware, Ponzi schemes, etc.) and unknown. Only %2 of the transactions are illicit, %21 of transactions are licit and remaining part is unknown. No information is given on the other nodes, which are classified as "unknown". There are 166

[1] https://www.kaggle.com/datasets/ellipticco/elliptic-data-set

features for each node. Feature descriptions are not provided due to privacyrelated concerns.

3.2 Data Preprocessing

As mentioned, 77% of transaction data is unlabelled. Therefore, we exclude the unlabelled samples from train-test split. Secondly, we do not apply any feature selection strategy and utilize all 168 features in the data.

3.3 Used Technologies

Graph Convolutional Networks As the prediction model, we employ a graph convolutional network (GCN) proposed by kipf2016semi. In our configuration of model, there is prepossessing layer, which is simple a dense layer. This layer is followed by two consecutive graph convolutional layers with 32 hidden units. Output of convolutional layers are fed to another dense layer, which produces the output of the model.

Spark Cluster Local Standalone Mode We used Spark Cluster Modes (LocalMode / Standalone-Mode) to share the resources in the working environment in distributed methods. In Local-Mode, resource of a physical environment is shared by worker threads, while in Standalone-Mode, Multiple physical resources are shared among Worker Nodes and Master Node.To observe the relationship between the number of worker threads the amount of Core, We share the 16 GB12 Core RAM to the worker threads via Spark Cluster in LocalMode. Moreover, the execution time of training and inference process has been analyzed by running these pheses on Standalone-Mode, 8 GB-4 Core two Debian VM instances.

BigDL – ORCA Library for Distributed Training and Inference BigDL is a distributed deep learning framework for Apache Spark. In this work, we prefer to use ORCA library which is used for scaling out single node Python notebook across large clusters (to process distributed Big Data). The ORCA Context object encapsulates the Spark Context and Ray Context Objects. Hence,both the methods from Spark framework and Ray framework can be used in the Driver program. With the Spark Context object, we can execute our training and inference phases on worker nodes (Spark Standalone-Mode) or threads(Spark LocalMode) in distributed manner. In Training and Inference phases, we can call Ray-DistributedMethods over Ray-Context object.
Orca-estimator object encapsulates the necessary methods for training and inference of models created on different frameworks such as Tensorflow, Pytorch, Spark.In this way, framework independent, orca-estimator.fit() and orca-estimator.predict() methods can be used.

3.4 Implementation

In our study, Implementation can be examined under 2 phase: (1)Preparation of the working environment and resources, (2)Creation of the GCN model.

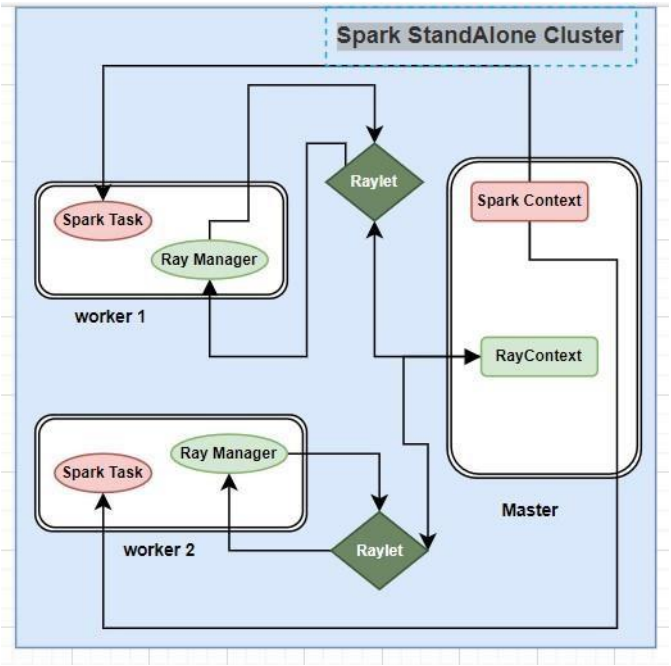

Fig.1. Ray in Spark Cluster

Preperation for BigDL-ORCA Framework: BigDL – ORCA Library requires Python 3.7 and Spark 2.4.6. At First, we created a conda environment and installed necessary libraries for BigDL – ORCA. However,the JVM version required for the creation of the Orca-Context object and Spark 2.4.6 was not compatible. For that reason,we install Spark 3.1.1 and BigDL – ORCA-3 (2021) and change Python version 3.7 to 3.8 to overcome other incompatibility problems. Since there is no up-to-date documentation for ORCA-3 (still contains version conflicts), execute train and inference phases over the Orca-Estimator object was challenging.

Preperation for Spark Cluster Local-Mode we create Orca-Context object, executed over Spark Local-Mode by using Pyspark 3.3.1 (an interface for Apache Spark in Python).

Preperation for Spark Cluster Standalone-Mode We created 3 Debain VM instances, 1 of which is Master Node and 2 of which is Worker Node.We installed Spark 3.3.1 in all nodes and connected to Master Node using the Pysparak interface. After that, we tried to distribute the source code, created for the training and inference steps, and training dataset to the worker Nodes through OrcaContext object.

Creation of the GCN model we install Tensorflow 2.9.2 for creating GCN Model which is both compatible with Spark 3.3.1 and ORCA-3 frameworks. We train the model with orca-estimator-from-tensorflow.fit() method. However, we can not use

orca-estimator-from-tensorlof.predict() method for our inference phase. We encountered **Py4JavaError** Error.

4 Experiments & Results

In this section, we present a set of experiments, where the utility of parallel GCN training is examined. We aim to reduce training time. For that purpose, we use BigDL-Orca API. We carry out a bunch of model trainings with different configurations through Orca, and test how they perform compared to bare-training (e.g., without a tool designed for distributed training).

4.1 With and Without BigDL-Orca

BigDL-Orca is a framework designed for distributed training on multiple nodes. However, before examining its distributed training performance, we want to observe how it performs only with one node. For that purpose, after setting the number of nodes to 1 in ORCA, we change only the number of CPU cores allocated for training. We do the same for the bare-training, where we do not make use of Orca, thanks to Tensorflow API allowing us to set the maximum number of CPU threads allocated.

Figure 2 shows the seconds needed for the training of GCN model for varying number of CPU cores. The first resutls to derive from the figure is that as the number of allocated CPU cores increases, the training phase becomes faster as expected. However, the performance improvement shown decays as we go to the right side for both lines. That means, the training phase cannot be parallelized more after some point. According to the figure, 6-8 cores are sufficient for a training with optimal speed on one node. Another thing the figure tells is that Orca running with one node offers no performance improvement, but a setback. This is a surprising result for us, but understandable when we consider the complex architecture behind Orca.

4.2 Distributed versus Non-Distributed Training

In this experiment, we test BigDL-Orca's distributed training performance. For this purpose, we fix the number of threads as 4 and only change the number of workers per node. The number of nodes is one again, since we do not have a cluster containing multiple machines. Hence, we hope to simulate distributed computing by running multiple workers on one node. For Orca, we have four results. We couldn't try for further configurations, because the machine we use is able to run maximum 4 workers per node. Keras API already runs on one executor, therefore it is represented by only one point in the figure. **Figure 3** shows that Orca performs worse dramatically with more executors, which is the

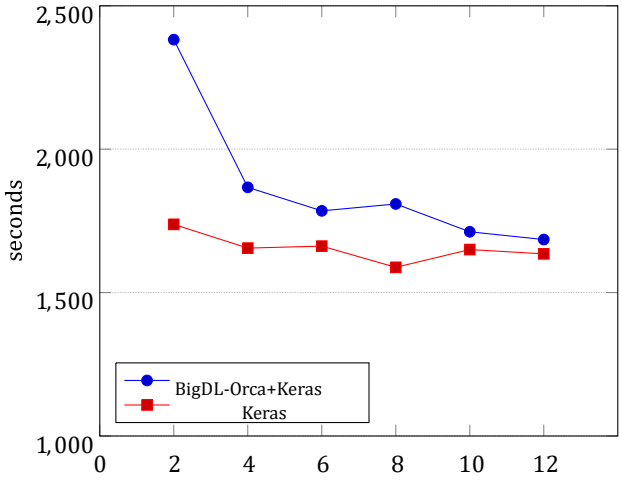

Fig.2. Training performance wrt. number of CPU cores

opposite what we expect from Orca. The probable reason behind this is that the computational sources are shared between the executors, which might cause a synchronization overhead.

5 Future Work

As mentioned, since our cluster consists of only one one, we couldn't test Orca's distributed training performance with multiple nodes. Instead, we used "localMode" and "Standalone-Mode" of the spark cluster manager. As a future work, we will extend our study by integrating additional machines into our cluster. We will employ Kubernetes as the cluster manager. Finally, we will integrate KAFKA into our system for handling streaming data.

6 Conclusion

In this study, we focused on how distributed computing can be utilized for fraud detection. In particular, we aimed to reduce training time of inference models. For that purpose, we used BigDL-Orca API with different configurations and observed the impacts of these. We concluded that, computation of a job in a distributed manner does not only depend on the availibility of computational sources, but also parallelizability of the job. Beside that, distributed computing may turn into a burden if it is not configured properly.

References

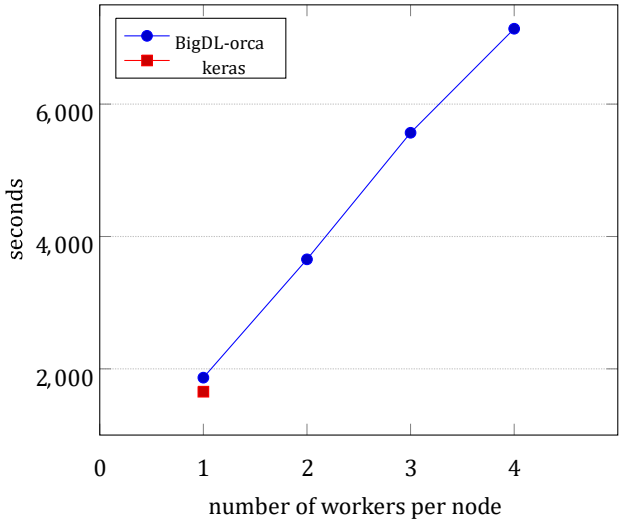

Fig.3. Training performance wrt. number of worker nodes

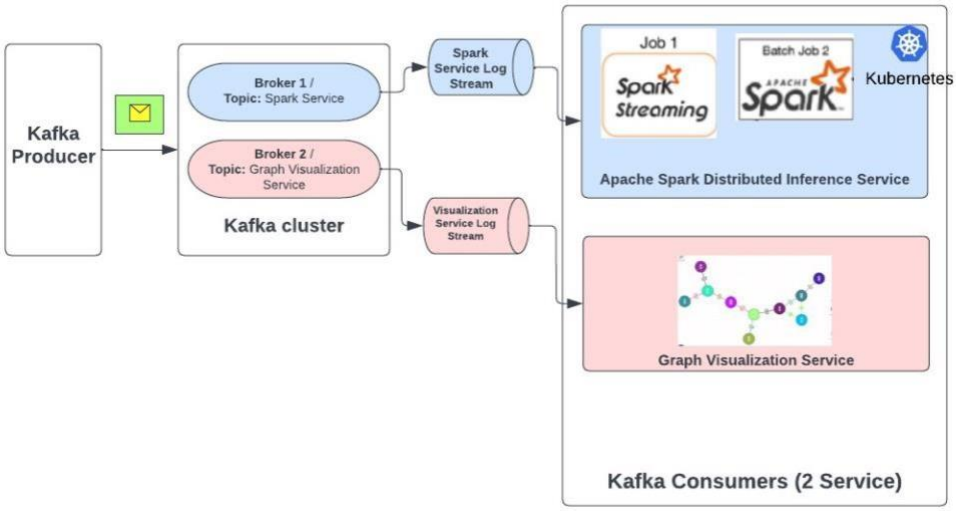

Fig.4. Future Work Architecture