AliGraph: An Industrial Graph Neural Network Platform

Kun Zhao, Wencong Xiao, Baole Ai, Wenting Shen, Xiaolin Zhang, Yong Li, Wei Lin
{kun.zhao,wencong.xwc,baole.abl,wenting.swt,xiaolin.zxl,jiufeng.ly,weilin.lw}@alibaba-inc.com
Alibaba Group

Abstract

We introduce AliGraph, an industrial distributed computation system for graph neural network (GNN). It empowers end-to-end solutions for users to address their scenarios by taking graph into deep learning frameworks. As an independent and portable system, the interfaces of AliGraph can be integrated with any tensor engine that is used for expressing neural network models. By co-designing the flexible Gremlin-like interfaces for both graph query and sampling, users can customize data accessing pattern freely. Moreover, AliGraph also shows excellent performance and scalability. It allows pluggable operators to adapt to the fast development of GNN community and outperforms existing systems an order of magnitude in terms of graph building and sampling.

ACM Reference format:
Kun Zhao, Wencong Xiao, Baole Ai, Wenting Shen, Xiaolin Zhang, Yong Li, Wei Lin. 2019. AliGraph: An Industrial Graph Neural Network Platform. In *Proceedings of Workshop on AI Systems at SOSP 2019, Ontario, Canada, Oct 27, 2019 (AI Systems '19)*, 3 pages.

1 Introduction

Recent years, we have witnessed the burst of graph neural network (GNN) in many artificial intelligence fields, including social networks [3], recommendations [6], neural language understanding [2], etc. Graph is a natural data organization in industrial cases. The training samples of many deep learning tasks usually date from a graph, such as the model for a recommender system, the relationship between user and item forms a heterogeneous graph. Unfortunately, the value of graph data has not been fully exploited due to the system constraints. The raw graph data is heterogeneous, probably reaching a size with billions of vertices and tens of billions edges. Both vertices and edges have multiple attributes attached on. To perform deep neural network on such a graph, a lot of pre-processing (*e.g.*, data aligning), need to be carried out to fill up the deficiency of tensor based deep learning frameworks, such as TensorFlow [1] or PyTorch [4], consuming great computation power and storage space. Moreover, researchers/developers desire an integrated system to perform end-to-end training with loss function directly mapping to the raw data.

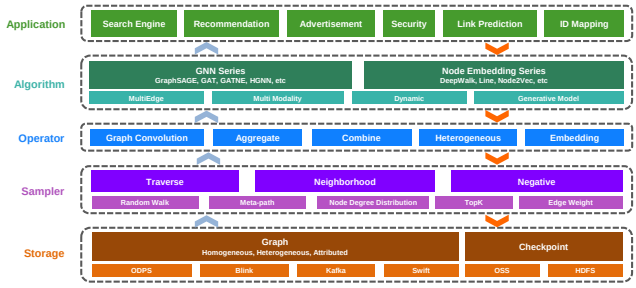

Figure 1. AliGraph overview.

We introduce AliGraph, an industrial GNN platform that aims for end-to-end solutions, to serve researchers and developers. AliGraph provides an integrated development environment that empowers the whole procedure from data storage to application models, which largely reduces the cost of GNN exploration. We abstract *SAMPLING*, *AGGREGATION*, and *COMBINATION* to describe a GNN model[7]. Figure 1 illustrates the architecture of AliGraph in five layers. We can generally divide it into a graph engine and a tensor engine. The graph engine takes charge of data preprocessing, graph building and sampling, providing APIs that simplify the distributed graph data access. The tensor engine, which is used to describe the neural network, has many alternatives. We just need to confirm that the data stream between the two engines is available.

In the next section, we describe three new design highlights of AliGraph, compared to our VLDB paper [7]:

- Co-design interface for graph query and sampling.
- Pluggable operators for GNN ecosystem.
- Distributed threading model for accelerating large-scale graph construction.

2 AliGraph

2.1 Unified Graph and Sampling API

Since the fast developing of GNN algorithms and complicated industrial requirement, the system API design should be flexible to support the newly proposed algorithms while compatible with the existing approaches. In AliGraph, we adopt

AI Systems '19, Oct 27, 2019, Ontario, Canada

ACM ISBN .

the interface design principle of Gremlin [5] and extend it to support flexible multi-hop sampling and customized sampler.

Figure 2 illustrates a typical two-hop sampling path in a recommendation scenario whose graph data consists of user and item vertices. Listing 1 demonstrates how to implement it with Gremlin like API. The sampler starts from user vertices and then samples a batch of 512 user vertices randomly. For each sampled user vertex, sampling for 10 neighbor edges from user to item with probability proportional to the edge weight (*EdgeWeight* sampler) is performed. Thereby, a total of 5120 item vertices are selected. The second-hop sampling considers the outer edge set of each selected item vertex and samples the top 5 (*TopK* sampler) edges from item to item sorted by edge weight. In total, 31232 vertices are sampled, including 512 user vertices and 30720 item vertices, together with 5120 user-item edges and 25600 item-item edges.

Note that, alongside with some built-in operators such as *EdgeWeight* and *TopK* sampler in this example, AliGraph also supports customized sampler for advanced users.

Figure 2. A two-hop sampling example.

```
from aligraph import *
g = Graph()
g.V("user").shuffle().batch(512)
 .outE("u2i").sample(10).by("EdgeWeight").inV()
 .outE("i2i").sample(5).by("TopK").inV()
```

Listing 1. A two-hop sampling example

2.2 Pluggable Operators

AliGraph is extensible to empower our users to customize samplers and other operators. Usually, when defining a sampler, developers have to consider not only the implementation of the sampler's functionality on a local server, but also how to partition the request and stitch the sub-responses for distributed sampling. Our framework simplifies the user efforts by hiding some system implementation details, such as RPC and message dispatching. First of all, the developer should define the schema of a sampler as illustrated in Listing 2, including parameters, inputs, and outputs.

```
import aligraph
aligraph.DefineOp(Name="MySampler")
        .Param(type="int", shape=[])
        .Input(type="int", shape=[-1])
        .Output(type="int", shape=[-1])
        .Output(type="float", shape=[-1])
```

Listing 2. Define the schema of a customized sampler

After that, the abstract functions ***Process()***, ***Map()***, and ***Reduce()*** need to be implemented by the developer. AliGraph will drive the sampler as described in Algorithm 1. Therefore, developers can use the sampler with the corresponding name as indicated in Listing 3.

```
g.V("user").shuffle().batch(512)
 .outE("u2i").sample(10).by("MySampler").inV()
```

Listing 3. An example of a user defined sampler

Algorithm 1 The Execution Framework of an Sampler

Given ***Context*** and ***Sampler***.
Context supplies interfaces to access *parameters*, *inputs*, and *outputs*. ***Process()*** is implemented in the ***Sampler***. It will access the data through ***Context*** and fill the response on local server.
The customized ***Map()*** and ***Reduce()*** will be implemented in the ***Context***. ***Map()*** will partition the context into small pieces indexed by server id, and ***Reduce()*** will stitch sub-responses into a whole one.

```
if Context.server_id = current_server_id then
    Sampler.Process(Context)
    return
else
    context_list = Context.Map()
end if
for all ctx ∈ context_list do
    Call(ctx.server_id, ctx)
end for

Context.Reduce(context_list)
```

2.3 Lock-Free Multi-thread Graph Building

Graph building is a blocking stage which must be ready before data accessing. For large-scale graph data, distributed graph construction can last for hours. The whole graph data needs to be partitioned into servers and network communication is required to exchange data among servers through RPC. The partition strategy varies from different cases.

To construct heterogeneous and attributed graphs from rawdata, three main phases are needed, including data reading and parsing, graph partitioning and dis-patching, and memory indexing. We propose a lock-free multi-thread method to reduce the time cost, which takes just several minutes to build a heterogeneous and attributed graph with billions of vertices and tens of billions edges. Figure 3 demonstrates data exchanges between two servers when building a graph.

More specially, we use multiple threads to read the raw graph data from file system independently. The raw data will be parsed and partitioned into pieces based on a given strategy. Some of them will be directed to the corresponding queues at the local server, and the others will be dispatched to other servers through RPC. Each processing thread is bonded to a data queue, consuming data and building graph structure in memory. The other servers who receive data through RPC will also put the data to the corresponding data queue in local, and the processing thread treats the received data in a consistent way of the local data for processing.

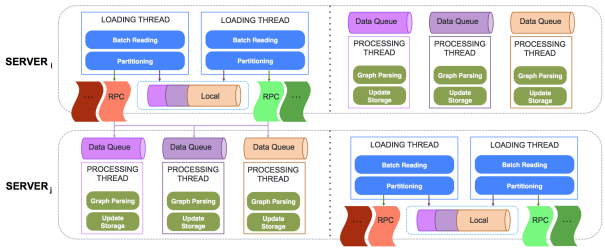

Figure 3. Lock-Free threading model when building graph.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 265–283.

[2] Takuo Hamaguchi, Hidekazu Oiwa, Masashi Shimbo, and Yuji Matsumoto. 2017. Knowledge transfer for out-of-knowledge-base entities: A graph neural network approach. *arXiv preprint arXiv:1706.05674* (2017).

[3] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.* 40, 3 (2017), 52–74. http://sites.computer.org/debull/A17sept/p52.pdf

[4] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. 2017. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. *PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration* 6 (2017).

[5] Marko A. Rodriguez. 2015. The Gremlin Graph Traversal Machine and Language. *CoRR* abs/1508.03843 (2015). arXiv:1508.03843 http://arxiv.org/abs/1508.03843

[6] Ziqi Wang, Yuwei Tan, and Ming Zhang. 2010. Graph-based recommendation on social networks. In *2010 12th International Asia-Pacific Web Conference*. IEEE, 116–122.

[7] Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. 2019. AliGraph: A Comprehensive Graph Neural Network Platform. *PVLDB* 12, 12 (2019), 2094–2105. http://www.vldb.org/pvldb/vol12/p2094-zhu.pdf