The Examples of How to Design Distributed GML systems

Driven by Industrial-purpose Application

Ethan Zhu

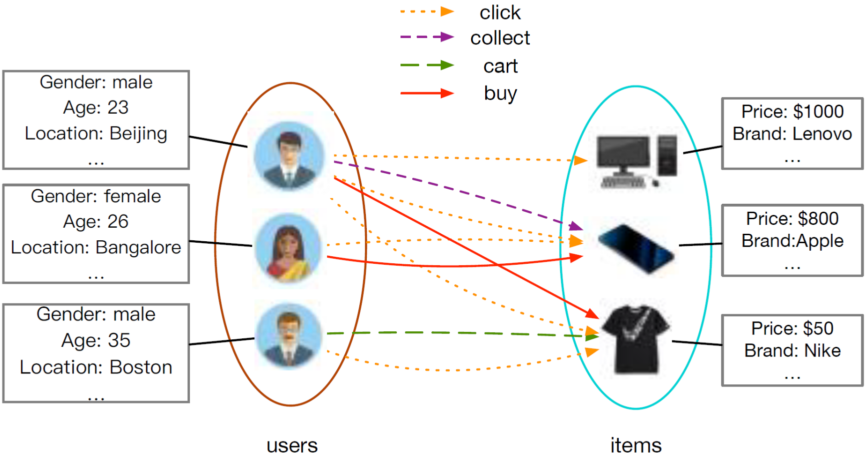
Department of Computer Science and Technology
Nanjing University

Apr, 1, 2021

Table of Contents

Table of Contents

Attributed Heterogeneous Graph



We are faced with

- Large scale graph datasets
  - ✓ Billion of vertices and billion of edges
  - ✓ Multiple types of vertices and edges
  - ✓ Nodes and edges has high demension features
- Converting the graph data into a low dimensional space
  - ✓ Keeping both the structural and property information to the maximum due to data dependency
- Scale-up and Scale-out
  - ✓ Limited stand-alone resources → (scale-up e.g. Amazon X-large)
  - ✓ Communication overhead is high in distributed environment → (scale-out)
- ...

We are faced with

- Large scale graph datasets
  - ✓ Billion of vertices and billion of edges
  - ✓ Multiple types of vertices and edges
  - ✓ Nodes and edges has high demension features
- Converting the graph data into a low dimensional space
  - ✓ Keeping both the structural and property information to the maximum due to data dependency
- Scale-up and Scale-out
  - ✓ Limited stand-alone resources → (scale-up e.g. Amazon X-large)
  - ✓ Communication overhead is high in distributed environment → (scale-out)
- ...

Designing and implementation GML systems···

Problem Definition

- Given an input graph G, which is a simple graph or an AHG, and a predefined number d on the dimension of embedding where $d \ll |V|$, the embedding problem is to convert the graph G into the d-dimensional space such that the graph property is preserved as much as possible.
- GNN is a special kind of graph embedding method, which learns the embedding results by applying graph neural networks.
- We concentrate on the vertex-level embedding.
- The output of the GNN is an embedding vector and will be fed into the downstream machine learning tasks, such as classification, link prediction and etc.

Table of Contents

We need to solve that

- How to elegantly integrate the heterogeneous information to be an unified embedding result
  - ✓ How to keep both the structural and property information to the maximum
- How to ensure scalability and fault tolerance
- How to improve the time and space efficiencies of GNN on large-scale graphs
- How to design efficient incremental GNN methods on dynamic graphs
- ...

Table of Contents

Overview of Papers

AliGraph: A Comprehensive Graph Neural Network Platform

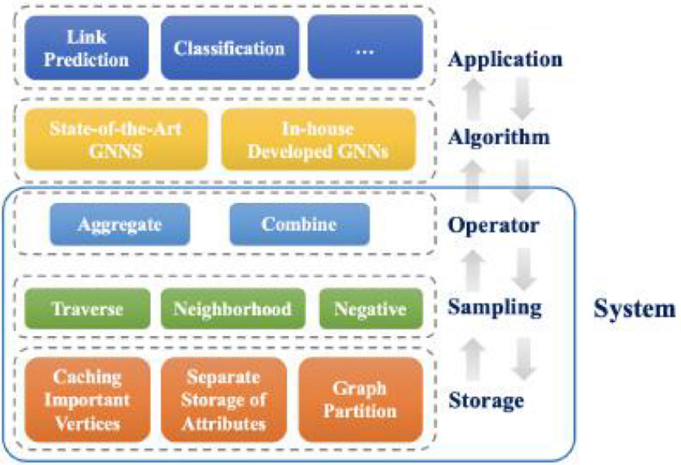
Rong Zhu, Kun Zhao, Hongxia Yang*, Wei Lin, Chang Zhou, Baole Ai, Yong Li, Jingren Zhou
Alibaba Group
{red.zr, kun.zhao, yang.yhx, weilin.lw, ericzhou.zc, jiufeng.ly, jingren.zhou}@alibaba-inc.com

AGL: A Scalable System for Industrial-purpose Graph Machine Learning

Dalong Zhang, Xin Huang, Ziqi Liu, Jun Zhou, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Lin Wang, Zhiqiang Zhang, Yuan Qi

Ant Financial Services Group, Hangzhou, China
{dalong.zdl, huangxi.hx, ziqiliu, jun.zhoujun, zhiyang.hzhy, xianzheng.sxz, zhibang.zg, fred.wl,,lingyao.zzq, yuan.qi}@antfin.com

Solution 1: Aligraph [1]



[1] AliGraph: A Comprehensive Graph Neural Network Platform https://dl.acm.org/doi/pdf/10.14778/3352063.3352127

The property of different methods

Category	Method	Heterogeneous Node	Heterogeneous Edge	Attributed	Dynamic	Large-Scale
Classic Graph Embedding	DeepWalk	✗	✗	✗	✗	✗
	Node2Vec	✗	✗	✗	✗	✗
	LINE	✗	✗	✗	✗	✗
	NetMF	✗	✗	✗	✗	✗
	TADW	✗	✗	✓	✗	✗
	LANE	✗	✗	✓	✗	✗
	ASNE	✗	✗	✓	✗	✗
	DANE	✗	✗	✓	✗	✗
	ANRL	✗	✗	✓	✗	✗
	PTE	✗	✓	✗	✗	✗
	Methpath2Vec	✗	✓	✗	✗	✗
	HERec	✗	✓	✗	✗	✗
	HNE	✗	✗	✗	✗	✗
	PMNE	✗	✓	✓	✗	✗
	MVE	✗	✓	✓	✗	✗
	MNE	✗	✓	✓	✗	✗
	Mvn2Vec	✗	✓	✓	✗	✗
GNN	Structural2Vec	✗	✗	✓	✗	✗
	GCN	✗	✗	✓	✗	✗
	FastGCN	✗	✗	✓	✗	✗
	AS-GCN	✗	✗	✓	✗	✗
	GraphSAGE	✗	✗	✓	✗	✗
	HEP	✓	✓	✓	✗	✗
	AHEP	✓	✓	✓	✗	✓
	GATNE	✓	✓	✓	✗	✓
	Mixture GNN	✓	✓	✓	✗	✗
	Hierarchical GNN	✓	✓	✓	✗	✗
	Bayesian GNN	✗	✓	✓	✗	✗
	Evolving GNN	✗	✓	✓	✓	✗

GNN framework

Algorithm 1: GNN Framework

Input: network $\mathcal{G}$, embedding dimension $d \in \mathbb{N}$, a vertex feature $\mathbf{x}_v$ for each vertex $v \in \mathcal{V}$ and the maximum hops of neighbors $k_{max} \in \mathbb{N}$.

Output: embedding result $\mathbf{h}_v$ of each vertex $v \in \mathcal{V}$

```
h_v^(0) ← x_v
for k ← 1 to k_max do
    for each vertex v ∈ V do
        S_v ← SAMPLE(Nb(v))
        h'_v ← AGGREGATE(h_u^(k-1), ∀u ∈ S)
        h_v^(k) ← COMBINE(h_v^(k-1), h'_v)
    normalize all embedding vectors h_v^(k) for all v ∈ V
h_v ← h_v^(k_max) for all v ∈ V return h_v as the embedding result for all v ∈ V
```

$\mathbf{h}_v^{(0)} \leftarrow \mathbf{x}_v$

for $k \leftarrow 1$ *to* k_{max} **do**

- **for** *each vertex* $v \in \mathcal{V}$ **do**
  - $S_v \leftarrow \textsc{Sample}(Nb(v))$
  - $\mathbf{h}'_v \leftarrow \textsc{Aggregate}(\mathbf{h}_u^{(k-1)}, \forall u \in S)$
  - $\mathbf{h}_v^{(k)} \leftarrow \textsc{Combine}(\mathbf{h}_v^{(k-1)}, \mathbf{h}'_v)$
- normalize all embedding vectors $\mathbf{h}_v^{(k)}$ for all $v \in \mathcal{V}$

$\mathbf{h}_v \leftarrow \mathbf{h}_v^{(k_{max})}$ for all $v \in \mathcal{V}$ return $\mathbf{h}_v$ as the embedding result for all $v \in \mathcal{V}$

GNN framework

Algorithm 1: GNN Framework

Input: network $\mathcal{G}$, embedding dimension $d \in \mathbb{N}$, a vertex feature $\mathbf{x}_v$ for each vertex $v \in \mathcal{V}$ and the maximum hops of neighbors $k_{max} \in \mathbb{N}$.

Output: embedding result $\mathbf{h}_v$ of each vertex $v \in \mathcal{V}$

$\mathbf{h}_v^{(0)} \leftarrow \mathbf{x}_v$

for $k \leftarrow 1$ *to* k_{max} **do**

  for *each vertex* $v \in \mathcal{V}$ **do**

    $S_v \leftarrow \textsc{Sample}(Nb(v))$

    $\mathbf{h}'_v \leftarrow \textsc{Aggregate}(\mathbf{h}_u^{(k-1)}, \forall u \in S)$

    $\mathbf{h}_v^{(k)} \leftarrow \textsc{Combine}(\mathbf{h}_v^{(k-1)}, \mathbf{h}'_v)$

  normalize all embedding vectors $\mathbf{h}_v^{(k)}$ for all $v \in \mathcal{V}$

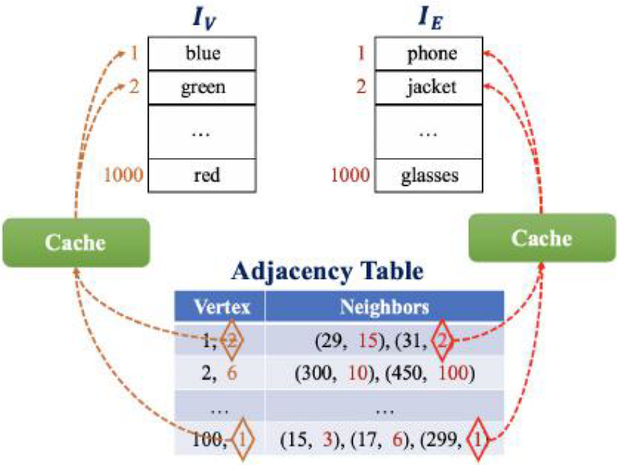
$\mathbf{h}_v \leftarrow \mathbf{h}_v^{(k_{max})}$ for all $v \in \mathcal{V}$ return $\mathbf{h}_v$ as the embedding result for all $v \in \mathcal{V}$

- Using the k-hop neighbors to keep structure and properties information

System optimization technique: Storage

- AliGraph platform is build on a distributed environment
  - ✓ Graph partition to minimize the number of crossing edges whose endpoints are in different workers.
    - The whole graph is divided and separately stored in different workers.
    - The goal of graph partition is to minimize the number of crossing edges whose endpoints are in different workers.
    - Aligraph includes four built-in state-of-the-art graph partition algorithms.
    - Users can choose the best partition strategy based on their own needs.
  - ✓ Separate storage of attributes
    - It is similar to normal form.
    - It increases the access time for retrieving the attributes.
  - ✓ Caching Neighbors of Important Vertices to reduce the communication cost
    - The trade-off exists between time and space.
    - It exists a user-specified threshold.

Separate storage of attributes



Partition and Caching

Algorithm 2: Partition and Caching

Input: graph $\mathcal{G}$, partition number p, cache depth h, threshold $\tau_1, \tau_2, \ldots, \tau_h$
Output: p subgraphs

Initialize p graph servers
for *each edge* $e = (u, v) \in \mathcal{E}$ **do**
  $j = \textsc{Assign}(u)$
  Send edge e to the j-th partition
for *each vertex* $v \in V$ **do**
  for $k \leftarrow 1$ *to* h **do**
    Compute $D_i^{(k)}(v)$ and $D_o^{(k)}(v)$
    if $\boxed{\frac{D_i^{(k)}(v)}{D_o^{(k)}(v)} \geq \tau_k}$ **then**
      Cache the 1 to k-hop out-neighbors of v on each partition where v exists

Partition and Caching

Algorithm 2: Partition and Caching

Input: graph $\mathcal{G}$, partition number p, cache depth h, threshold $\tau_1, \tau_2, \ldots, \tau_h$
Output: p subgraphs

```
Initialize p graph servers
for each edge e = (u, v) ∈ ℰ do
    j = ASSIGN(u)
    Send edge e to the j-th partition
for each vertex v ∈ V do
    for k ← 1 to h do
        Compute D_i^(k)(v) and D_o^(k)(v)
        if D_i^(k)(v) / D_o^(k)(v) ≥ τ_k then
            Cache the 1 to k-hop out-neighbors of v on each partition where
              v exists
```

Sampling

- TRAVERSE: is used to sampling a batch of vertices or edges from the whole partitioned subgraphs.
- NEIGHBORHOOD: will generate the context for a vertex. The context of this vertex may be one or multi hop neighbors, which are used to encode this vertex.
- NEGATIVE: is used to generate negative samples to accelerate the convergence of the training process.

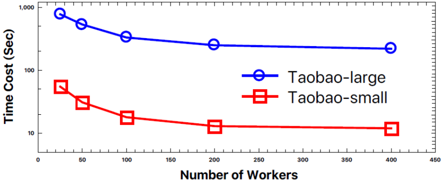
Operator

- Aggregate
  - ✓ Convolution operation
  - ✓ Elementwise mean, max-pooling neural network and long short-term memory (LSTMs)
- Combine
  - ✓ Concat ...

System Evaluation

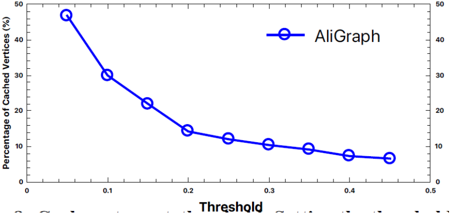
Dataset	# user vertices	# item vertices	# user-item edges	# item-item edges	# attributes of user	# attributes of item
Taobao-small	147,970,118	9,017,903	442,068,516	224,129,155	27	32
Taobao-large	483,214,916	9,683,310	6,587,662,098	231,085,487	27	32

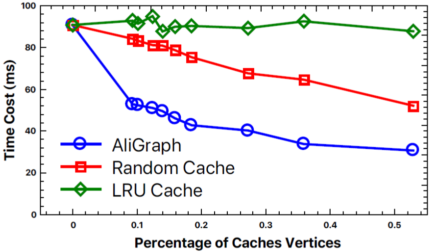
- Graph Build
  - ✓ The graph building time explicitly decreases w.r.t. the number of workers
  - ✓ AliGraph can build large-scale graphs in minutes, e.g. 5 minutes for Taobao-large.



System Evaluation

- Effects of Caching Neighbors





System Evaluation

- Effects of Sampling

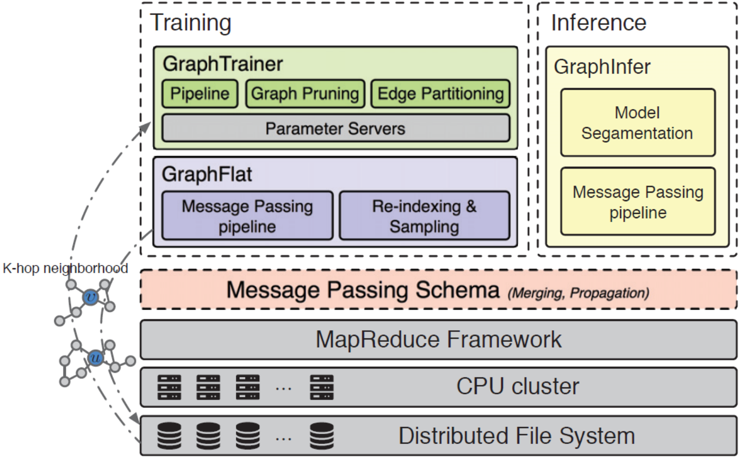
Dataset	Setting		Time (ms)		
	# of workers	Cache Rate	TRAVERSE	NEIGHBORHOOD	NEGATIVE
Taobao-small	25	18.46%	2.59	45.31	6.22
Taobao-large	100	17.68%	2.62	52.53	7.52

- Sampling methods are very efficient which finish between a few milliseconds to no more than 60ms.
- Sampling methods are efficient and scalable.

Conclusions

- In-memory processing systems using rpc comunication with the different workers.
- Using k-hop neighbors to keep structure and properties information.
- Design a series of optimization strategy.

Solution 2: AGL [2]



[2] AGL: A Scalable System for Industrial-purpose Graph Machine Learning http://www.vldb.org/pvldb/vol13/p3125-zhang.pdf

How to keep structure and properties information

- K-hop Neighborhood: The K-hop neighborhood w.r.t. a target node v, denoted as G_v^K , is defined as the induced attributed subgraph of G whose node set is $V_v^K = \{v\} \cup \{u : d(v,u) \leq k\}$, where $d(v,u)$ denotes the length of the shortest path from u to v.

How to keep structure and properties information

- K-hop Neighborhood: The K-hop neighborhood w.r.t. a target node v, denoted as G_v^K , is defined as the induced attributed subgraph of G whose node set is $V_v^K = \{v\} \cup \{u : d(v,u) \le k\}$, where $d(v,u)$ denotes the length of the shortest path from u to v.
- This is different from an Aligraph.

Theorem 1. Let $\mathcal{G}_v^K$ be the K-hop neighborhood w.r.t. the target node v, then $\mathcal{G}_v^K$ contains the **sufficient and necessary information** for a K layers GNN model, which follows the paradigm of Equation 1, to generate the embedding of node v.

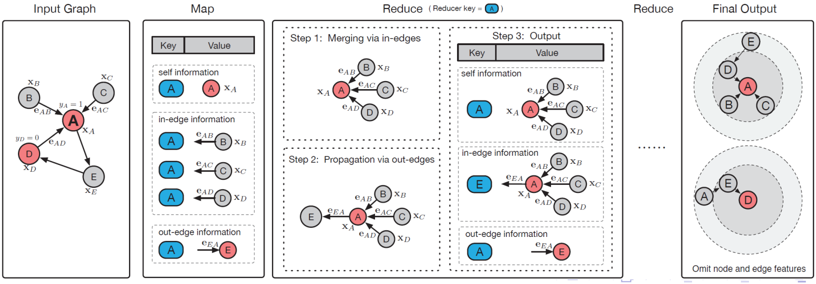
First, the 0^{th} layer embedding is directly assigned by the raw feature, i.e., $\mathbf{h}_v^{(0)} = \mathbf{x}_v$, which is also the 0-hop neighborhood. And then, from Equation 1, it's easy to find that the output embedding of v in each subsequent layer is generated only based on the embedding of the 1-hop in-edge neighbors w.r.t. v from the previous layer. Therefore, by applying mathematical induction, it's easy to prove Theorem 1. Moreover, we can extend the theorem to a batch of nodes. That is, the intersection of the K-hop neighborhoods w.r.t. a batch of nodes provides the sufficient and necessary information for a K layers GNN model to generate all the node embeddings in the batch. This simple theorem implies that in a K layers GNN model the target node's embedding at the K^{th} layer only depends on its K-hop neighborhood, rather than the entire graph.

GraphFlat

- GraphFlat is to generate independent K-hop neighborhoods.
- It is based on message passing.
- Those tiny K-hop neighborhoods are flattened to a protobuf strings and stored on a distributed file system.
- The k-hop neighborhood is self-contained, so we can load one or a batch of them rather than the entire graph into memory, and do the training similar to any other traditional learning methods

GraphFlat

- GraphFlat is to generate independent K-hop neighborhoods.
- It is based on message passing.
- Those tiny K-hop neighborhoods are flattened to a protobuf strings and stored on a distributed file system.
- The k-hop neighborhood is self-contained, so we can load one or a batch of them rather than the entire graph into memory, and do the training similar to any other traditional learning methods



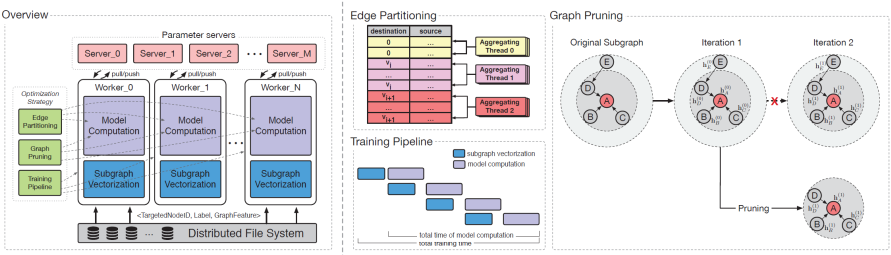
Hub node

- In network science, a hub is a node with a number of links that greatly exceeds the average.
- In the Reduce phase of GraphFlat, reducers that process such "hub" nodes could be much slower than others thus damage the load balances of GraphFlat.
- The huge K-hop neighborhoods w.r.t. those "hub" nodes may cause the Out Of Memory (OOM) problem.

System optimization technique: Sampling and Indexing

- Re-indexing: update shuffle keys using suffixes when exceeds a pre-defined threshold.
- Sampling framework: reduce the scale of the K-hop neighborhoods, especially for those "hub" nodes.
- Inverted indexing: replace the reindexed shuffle key with the original shuffle key.

GraphTrainer



- Training workers of GraphTrainer is independent of each other.
- Training workflow
  - ✓ subgraph vectorization
    - ∗ Adjacency matrix, A_α
    - ∗ Node feature matrix, X_α
    - ∗ Edge feature matrix, E_α
  - ✓ model computation
    - ∗ forward and backward

Datasets

Indices	Cora	PPI	Ogbn	UUG
#Nodes	2,708	56,944	2,449,029	6.23×10^9
#Edges	5,429	818,716	61,859,140	3.38×10^{11}
#feature	1,433	50	100	656
#Classes	7	121	47	2
#Train	140	44,906	196,615	1.2×10^8
#Valid	500	6,514	39,323	5×10^6
#Test	1,000	5,524	2,213,091	1.5×10^7
#Layers	2	3	3	2
Embedding	16	64	256	8
#Epochs	200	200	20	10

System Evaluation

- Effectiveness

Datasets	Methods	Base	PyG	DGL	AliGraph	AGL
Cora (Accuracy)	GCN	0.813	0.818	0.811	0.802	0.811
	GAT	0.830	0.831	0.828	0.823	0.830
PPI (micro-F1)	GraphSage	0.598	0.632	0.636	–	0.635
	GAT	0.973	0.983	0.976	–	0.977
Ogbn (Accuracy)	GCN	0.757	–	–	0.723	0.744
	GraphSage	0.780	–	–	0.745	0.775
UUG (AUC)	GCN	–	–	–	–	0.681
	GraphSage	–	–	–	–	0.708
	GAT	–	–	–	–	0.867

System Evaluation

- Efficiency

	GCN			GraphSAGE			GAT		
	1-layer	2-layer	3-layer	1-layer	2-layer	3-layer	1-layer	2-layer	3-layer
PyG	3.49	6.43	9.62	4.47	6.98	10.15	44.29	65.32	85.21
DGL	1.09	1.35	1.62	1.14	1.39	1.64	16.14	21.47	26.03
AGL_{base}	0.48	2.75	4.10	0.46	2.47	3.94	4.75	25.72	36.86
$\text{AGL}_{+\text{pruning}}$	0.48	1.93	3.23	0.46	1.67	2.99	4.75	13.88	20.01
$\text{AGL}_{+\text{partition}}$	0.42	1.22	1.60	0.34	0.97	1.39	4.63	22.65	33.45
$\text{AGL}_{+\text{pruning\&partition}}$	**0.42**	**1.13**	**1.52**	**0.34**	**0.88**	**1.35**	**4.63**	**13.73**	**18.63**

图: Time cost(s) per epoch on PPI in Standalone mode

Conclusions

- Construction of K-hop neighborhood, an information-complete subgraph for each node
- AGL provides the scalability, fault tolerance using mature infrastructures such as MapReduce.

Table of Contents

Key Insight

How to keep the structure and properties information···

Table of Contents

Future Work

- The k-hop neighberhoods design is very enlightening. I see if I can further think about it here, or use this solution idea in reverse to optimize the system.

Table of Contents

References I

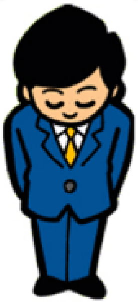
- Rong Zhu, Kun Zhao, Hongxia Yang, Wei Lin, Chang Zhou, Baole Ai, Yong Li, and Jingren Zhou. 2019. AliGraph: a comprehensive graph neural network platform. Proc. VLDB Endow. 12, 12 (August 2019), 2094–2105. DOI:https://doi.org/10.14778/3352063.3352127
- Dalong Zhang, Xin Huang, Ziqi Liu, Jun Zhou, Zhiyang Hu, Xianzheng Song, Zhibang Ge, Lin Wang, Zhiqiang Zhang, and Yuan Qi. 2020. AGL: a scalable system for industrial-purpose graph machine learning. Proc. VLDB Endow. 13, 12 (August 2020), 3125–3137. DOI:https://doi.org/10.14778/3415478.3415539
- Liang, S., Wang, Y., Liu, C., He, L., Huawei, L. I., Xu, D., Li, X. (2020). EnGN: A High-Throughput and Energy-Efficient Accelerator for Large Graph Neural Networks. IEEE Transactions on Computers.

References II

Yan, M., Deng, L., Hu, X., Liang, L., Feng, Y., Ye, X., ... Xie, Y. (2020, February). Hygcn: A gcn accelerator with hybrid architecture. In 2020 IEEE International Symposium on High Performance Computer Architecture (HPCA) (pp. 15-29). IEEE.

Table of Contents

Thank you!
Welcome for any questions!



Jun-peng Zhu
Department of Computer Science and Technology
Nanjing University