



Efficient Data Loader for Fast Sampling-based GNN Training on Large Graphs ADSLAB

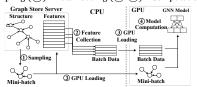


Youhui Bai¹, Cheng Li^{1,2}, Zhiqi Lin¹, Yufei Wu¹, Youshan Miao³, Yunxin Liu³, Yinlong Xu^{1,2} USTC1, Anhui Province Key Laboratory of High Performance Computing2, Microsoft Research3

BACKGROUND AND MOTIVATIONS

Sampling-based GNN Training Pipeline:

Sampling (1) – Data Loading (2-3) – Computation (4)

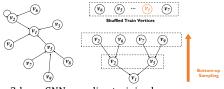


Challenges of Existing Training Pipeline:

- High data loading cost (up to 74% in end to end training)
- Redundant data loading per epoch (~4x graph data volume)
- Low GPU utilization (~8% memory consumption)
- Sequential execution of data loading and GNN computation

GNN Neighbor Sampling for Example:

Bottom-up sampling, top-down forwarding



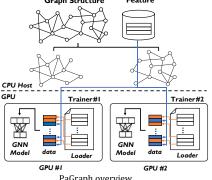
2-layer GNN sampling training leverages 2-neighbor neighbor sampling

We introduce PaGraph, a GNN training data loader with caching, graph partition and pipeline

PAGRAPH DESIGN

PaGraph Overview:

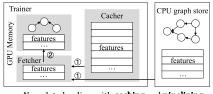
- Graph Store Server for serving graph data
- Trainer working on a GPU for GNN model training
 - Loader for caching and prefetching graph data
- Graph partition for efficient caching on multi-GPU training



PaGraph overview

■ Loader for Caching Graph Data: Cache vertex data with top highest out-degree

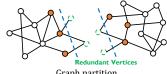
- indicated from neighbor-sampling
- Mini-batch data is prefetched from both CPU shared memory (un-cached data in Graph Store Server) and GPU memory (cached data in Loader)



New data loading with caching and pipelining

Graph Partition:

- Offer full accessing view for Loader
- Avoid cross-partition access during sampling Achieve computation balance and load balance
- Include redundant vertices for each sub-graph
- Adopt stream-based algorithm. Compute scores for each vertex for each sub-graph



Graph partition

Multi-GPU Training:

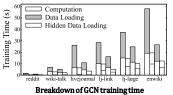
- Leverage data parallelism for GNN model Each trainer works on a GPU and a sub-graph
- Communication among Trainers is only for exchanging
- model gradients at the end of each mini-batch

8

Experiment Setup

- PaGraph is implemented on top DGL and PvTorch
- Evaluate on a multi-GPU server:
 - 2 Intel Xeon E5-2620v4 CPUs, 512GB RAM, 4 GTX 1080Ti GPUs
- CUDA v10.1, DGL v0.4, PyTorch v1.3

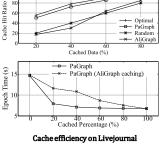
Single GPU Training Performance



- Up to 4.9x of PaGraph over DGL
- Up to 6.2x of PaGraph+PP over DGL+PP
- 78.7% loading time reduction by PaGraph with 54% cached data on enwiki
- Pipelining can further fully hide the loading time into computation

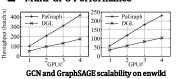
EXPERIMENTAL RESULTS

Cache Performance



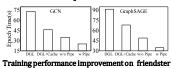
- Outperform AliGraph and Random with a cache hit ratio close to optimal Achieve up to 1.5x performance
- speedup over AliGraph Pipelining can improve the caching efficiency

Multi-GPU Performance



- 2.4x speedup of GCN training on enwiki with 4-GPU over DGL
- Linear scalability

Effects of Optimizations



- Caching achieves up to 1.4x speedup
- Partitioning improves up to 28.1%
- Pipelining further reduces training time up to 29.3%