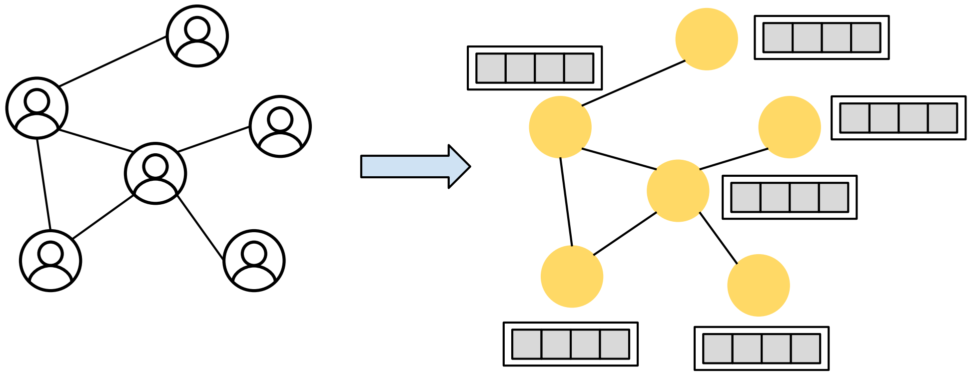
# Rethinking Graph Data Placement for Graph Neural Network Training on Multiple GPUs

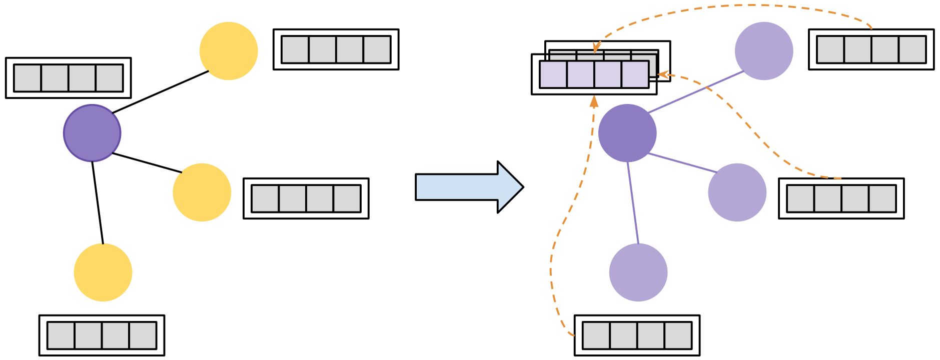
**Shihui Song and Peng Jiang**

University of Iowa

# Graph Neural Network (GNN)

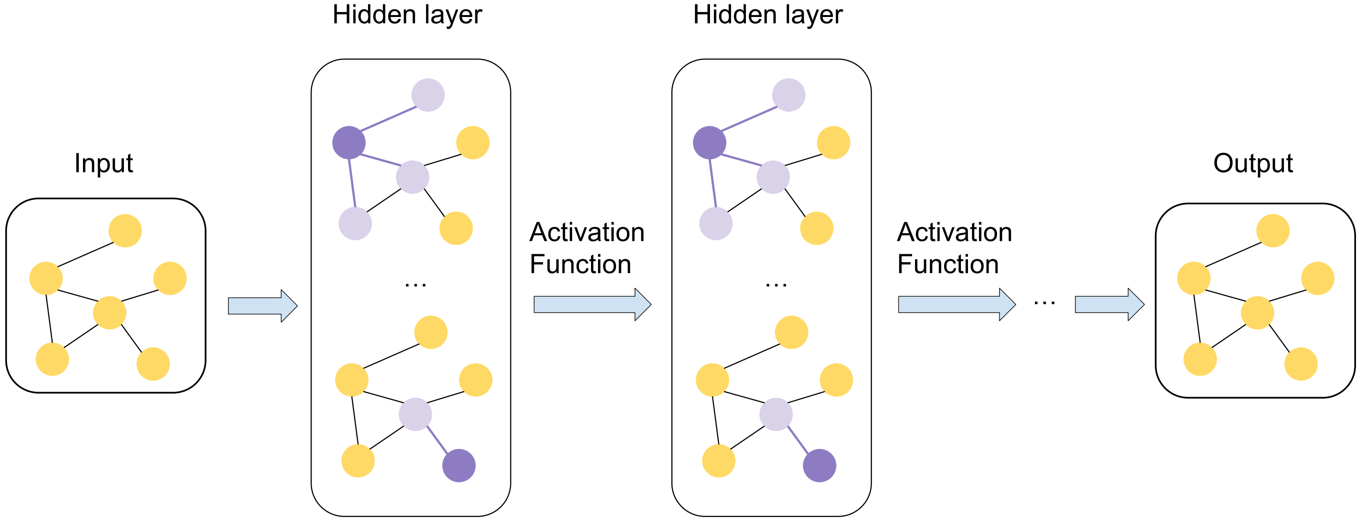


Graph Construction

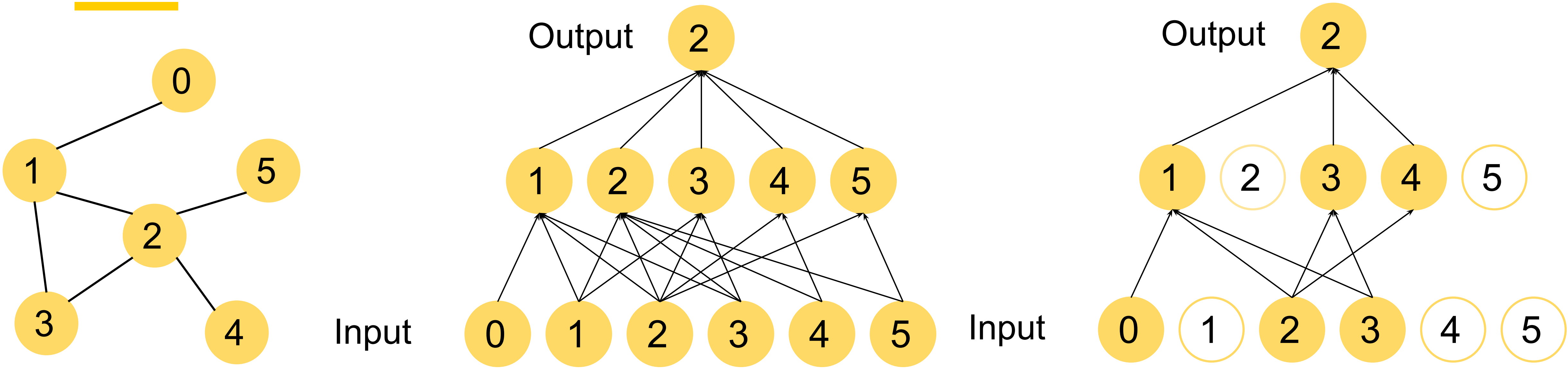


Feature Aggregation

Graph Neural Network Model

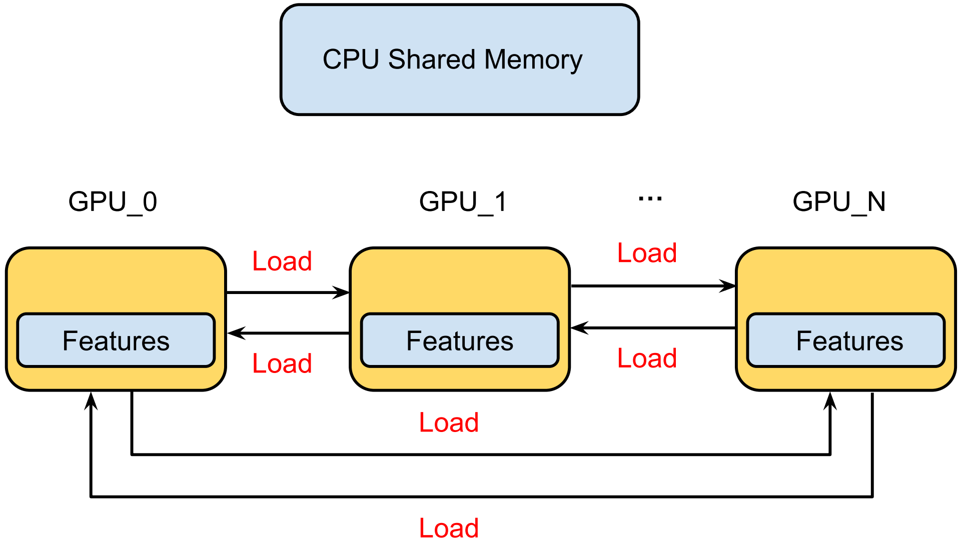


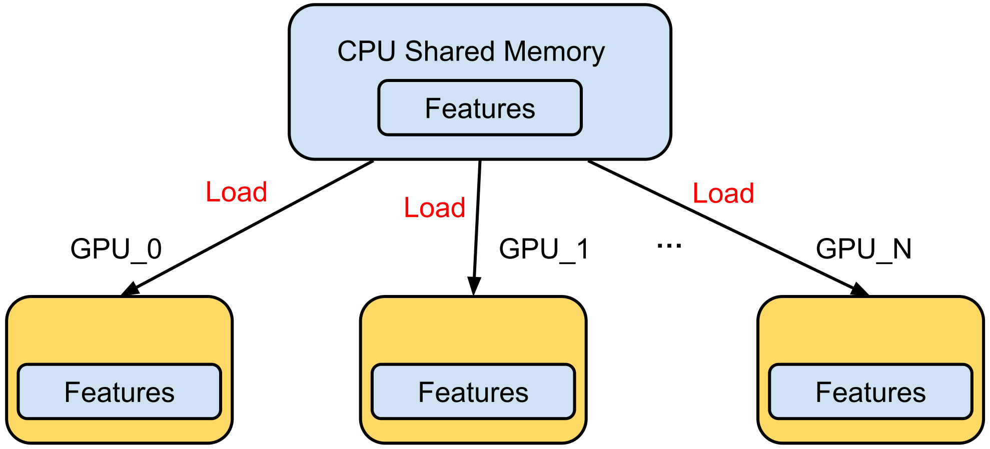
# Sampling-based GNN training




- To reduce the computation, sampling-based GNN training samples a subset of neighbors and estimates the aggregation results based on the sampled nodes.

# Analysis of DGL and PaGraph (PG)




**DGL**

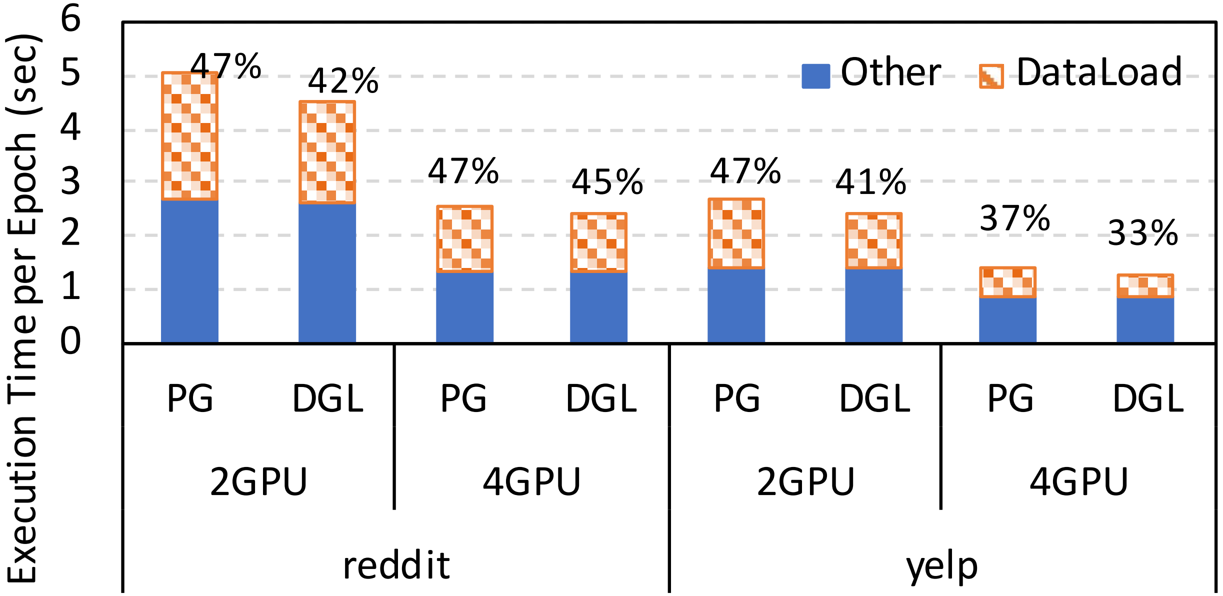
- Adopts METIS graph partitioning
- Assumes that the graph can be entirely stored on multiple GPUs

**PaGraph**

- Stores the graph on CPU and buffers the most frequently accessed nodes of each partition on GPU

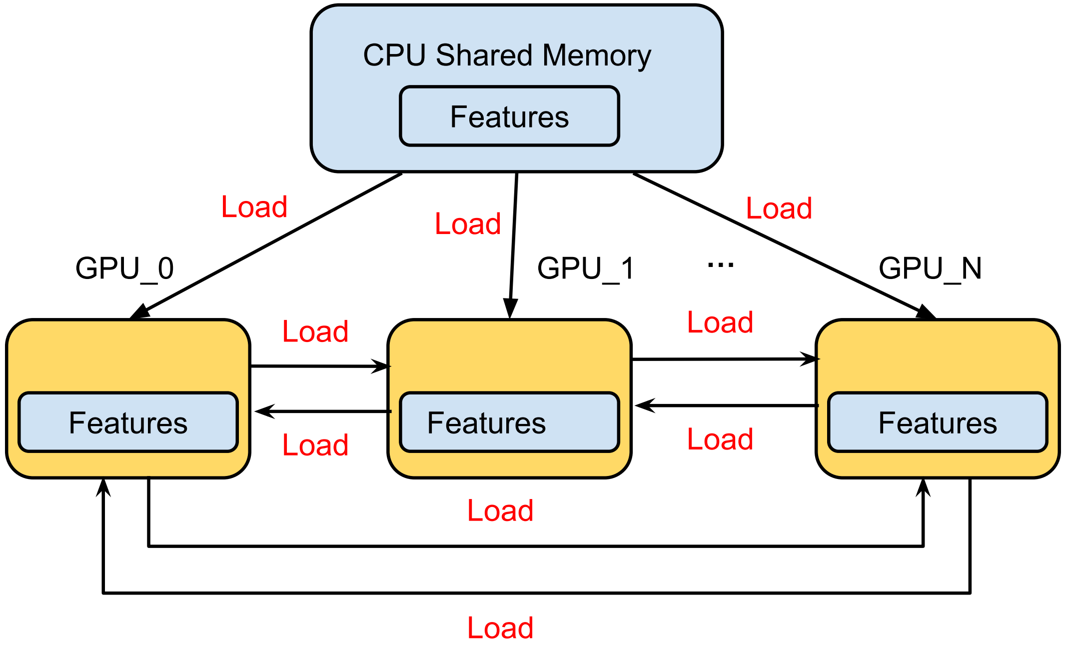
# Motivation

- Assume that the GPU memory is small and we can only store 20% of nodes that are most frequently accessed on each GPU




Loading features is a bottleneck of training

# Performance Model




Goal → Find the optimal configuration of $B_i$

**Cost Function of GPU_i:**

$$C_i(B) = \begin{cases} C_{cpu}\left|R_i \setminus B_{gpu}\right| + C_{gpu}\left|R_i \cap B_{gpu}\right|, & \text{if } C_{gpu} < C_{cpu}; \\ C_{cpu}\left|R_i\right| & \text{if } C_{gpu} \geq C_{cpu}. \end{cases}$$

$R_i$ : The nodes GPU_i needs to read from CPU and other GPUs

$B_{gpu}$: The nodes saved on GPUs

$C_{cpu}$ : The cost of reading a node on CPU

$C_{gpu}$ : The cost of reading a node on a different GPU

**Optimization Problem:**

$$\min \quad \max_{i \in [1,n]} (\mathrm{E}_{S_i \sim D}[C_i(B)]),$$

$$\text{subject to} \quad |B_i| \leq BSIZE, \quad i = 1, \ldots, n$$

# Case1 $C_{gpu} \geq C_{cpu}$

**Optimization Problem:**

$$
\begin{aligned}
\mathrm{E}_{S_i}[C_i] &= C_{cpu}\mathrm{E}_{S_i}[|R_i|] \\
&= C_{cpu}\mathrm{E}_{S_i}[|S_i \setminus B_i|] \\
&= C_{cpu}\mathrm{E}_{S_i}[|S_i|] - C_{cpu}\mathrm{E}_{S_i}[|S_i \cap B_i|]
\end{aligned}
$$

$S_i$ : The nodes GPU_i needs to read

$B_i$ : The nodes saved on GPU_i local memory

When $\mathrm{E}_{S_i}[|S_i \cap B_i|]$ is maximized for every GPU_i, this formular can achieve the minimum value.

Rule1: We store nodes with the highest sampling probability on it

# Case2 $C_{gpu} < C_{cpu}$

**Optimization Problem:**

$$
\begin{aligned}
&\mathrm{E}_{S_i}[C_i] \\
&= C_{cpu}\mathrm{E}_{S_i}\left[\left|R_i \setminus B_{gpu}\right|\right] + C_{gpu}\mathrm{E}_{S_i}\left[\left|R_i \cap B_{gpu}\right|\right] \\
&= C_{cpu}\mathrm{E}_{S_i}\left[\left|R_i\right|\right] - \left(C_{cpu} - C_{gpu}\right)\mathrm{E}_{S_i}\left[\left|R_i \cap B_{gpu}\right|\right] \\
&= C_{cpu}\mathrm{E}_{S_i}\left[\left|S_i\right|\right] - \underline{\left(C_{cpu} - C_{gpu}\right)\mathrm{E}_{S_i}\left[\left|S_i \cap B_{gpu}\right|\right]} - \underline{C_{gpu}\mathrm{E}_{S_i}\left[\left|S_i \cap B_i\right|\right]}
\end{aligned}
$$

Tradeoff

Should store as many nodes as possible on all GPUs

Each GPU stores the same set of nodes with the highest sampling probability

# Case2 Algorithm

**Algorithm 1:** Distributing node features onto multiple GPUs with fast interconnects

**Input:** $\alpha$; #nodes: $N$; #GPUs: $n$; Sampling probability: $p$; Buffer size: $BSIZE$

**Output:** $B = \{B_1, \ldots, B_n\}$

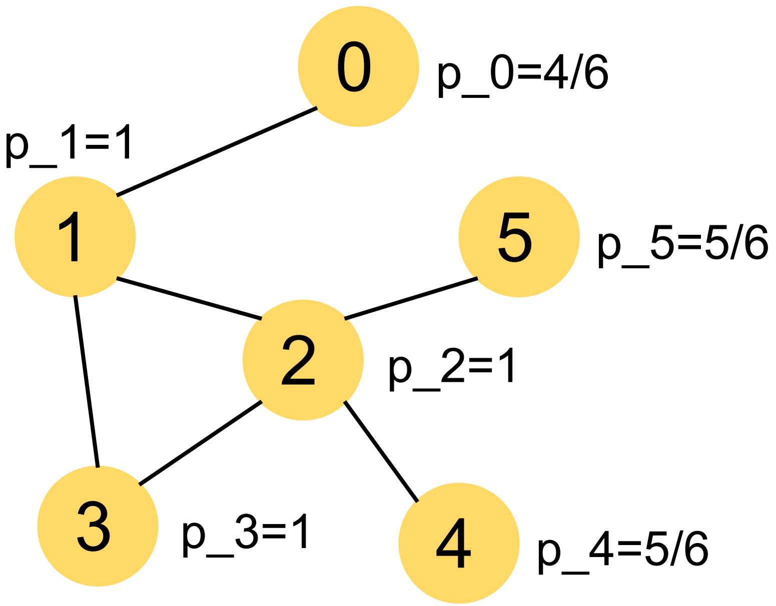
```
/* Sort nodes by probability p in descending order */
V = sort_nodes(N, p);
/* Initialize buffer on each GPU with nodes of highest sampling probabilities */
for i = 1 to n do
    B_i = [V[0], V[1], ... V[BSIZE − 1]];
p_sum = [0.0, ..., 0.0];
for i = 0 to (min(N, n · BSIZE) − BSIZE − 1) do
    if i mod n == 0 then
        /* Sort GPUs by p_sum in ascending order */
        ordered_devices = sort_device(n, p_sum);
    /* Do not change last device in each round */
    if i mod n == n − 1 then continue;
    /* Get device in the sorted order */
    device = ordered_devices[i mod n];
    /* Select the next node in V */
    new_node = V[i + BSIZE];
    /* Select a duplicate node on device */
    old_node_idx = BSIZE − 1 − ⌊i/n⌋;
    old_node = V[old_node_idx];
    /* Check if the replacement is beneficial */
    if p_(new_node) > α · p_(old_node) then
        /* Replace old_node on device with new_node */
        B_device[old_node_idx] = new_node;
        /* Update p_sum */
        p_sum[device] += p_(new_node)
    else break;
return {B_1, ..., B_n}
```

Increase of the second term $C_{gpu}\left(p(old\_node) - p(new\_node)\right)$

Decrease of the last term $\left(C_{cpu} - C_{gpu}\right)p(new\_node)$

Beneficial condition $p(new\_node) > \dfrac{C_{gpu}}{C_{cpu}}\,p(old\_node)$

# Case2 Example




The ordered nodes: [1, 2, 3, 4, 5, 0]

GPU_0: 1 3 p_sum_0 = 0

GPU_1: 4 2 p_sum_1 = 5/6

4 Beneficial? p_4 > 0.3*p_2 ($\alpha$=0.3) ✓

Which GPU? p_sum_0 > p_sum_1 0

Keep at least one copy of 1 2

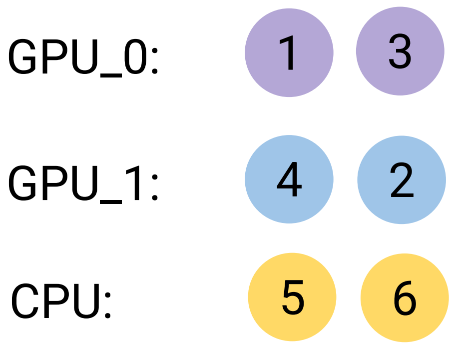
# Limitation of data placement

If the access frequency is less skewed and the GPU memory is small, the data loading might be expensive even with algorithm1

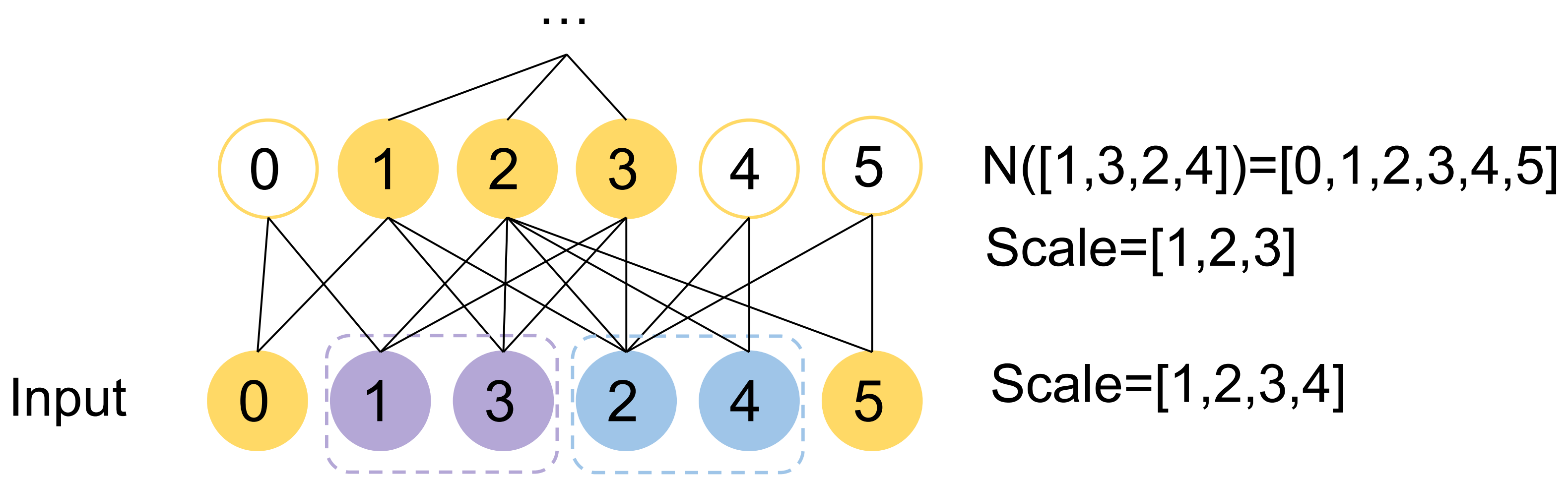
- We still need to load feature embeddings from CPU in most of the cases

A locality-aware neighbor sampling technique to further reduce the data movement overhead

# Locality-Aware Neighbor Sampling




The ordered nodes: [1, 2, 3, 4, 5, 0]




Multiply the sampling probabilities of the neighbor set of B with an adjustable factor

1 2 4 … Maximum

# Experimental Setup

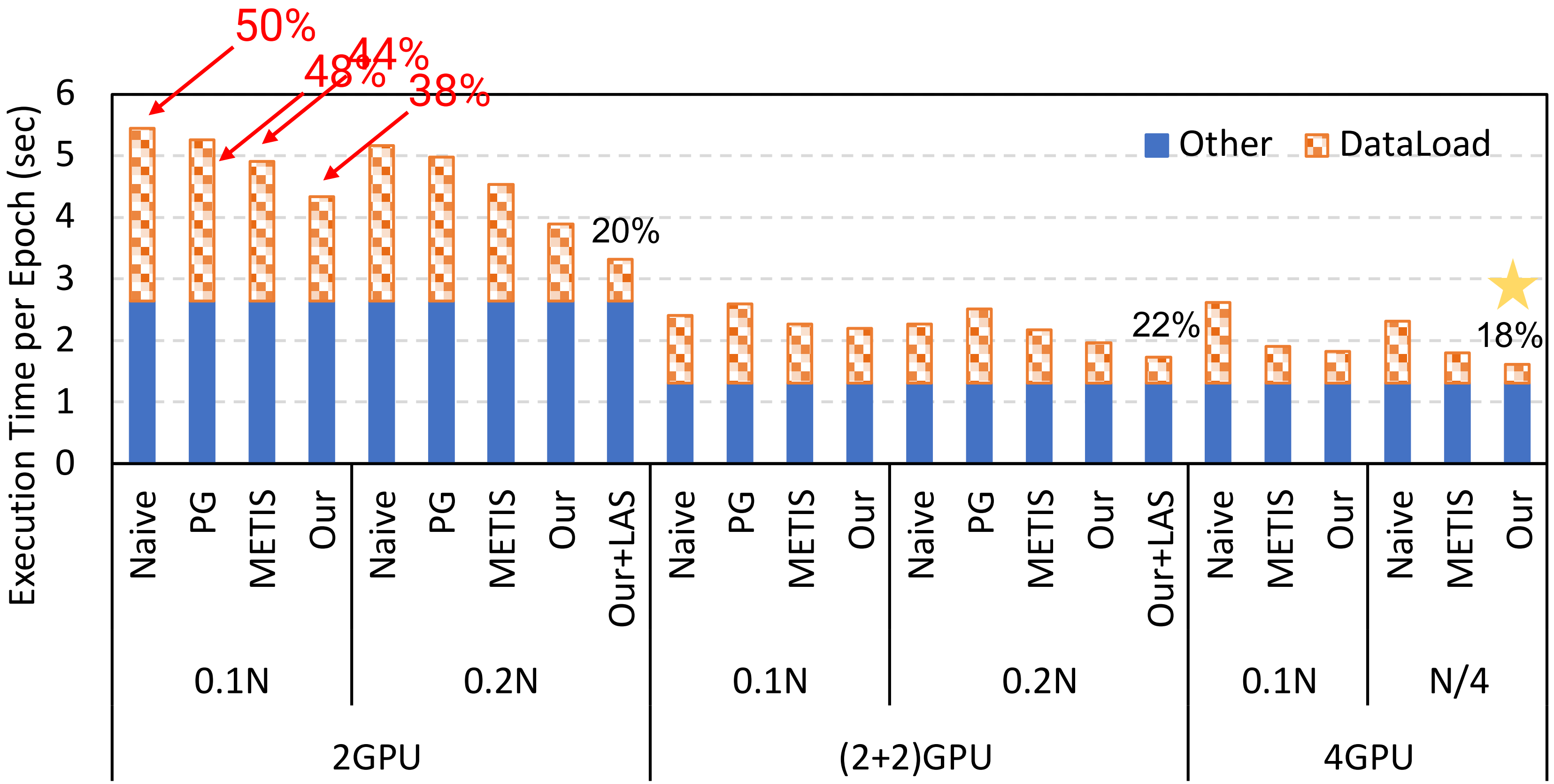
- Platform

  A single machine with two Intel Xeon Gold 6248 CPUs and eight Nvidia Tesla V100 GPUs
  - GPUs connected with NVLink Bridge: (2+2)GPU, (2*4)GPU
  - GPUs connected with NVSwitch: 2GPU, 4GPU, 8GPU
- Dataset

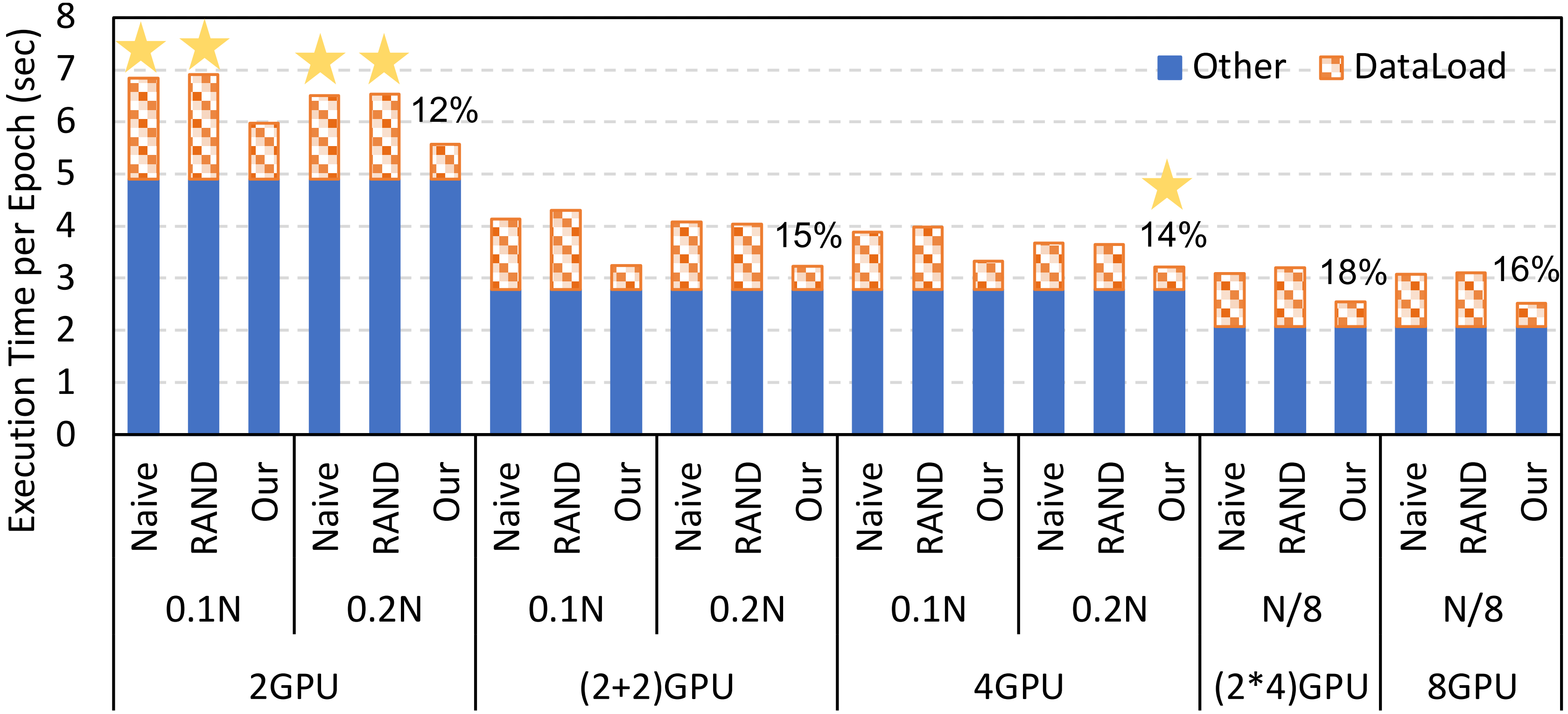
| | Reddit | Yelp | Products | Papers100M | MAG240M |
|---|---|---|---|---|---|
| #nodes | 233K | 717K | 2.4M | 111M | 122M |
| #edges | 11.6M | 7.0M | 62M | 1.6B | 1.3B |
| feat_size | 535MB | 820MB | 934MB | 53GB | 175GB |

- Baseline
  - Naive partitioning
  - Random partitioning
  - METIS partitioning
  - PaGraph partitioning (Lin et al., SoCC'20)

# Evaluation: Speedup on Reddit

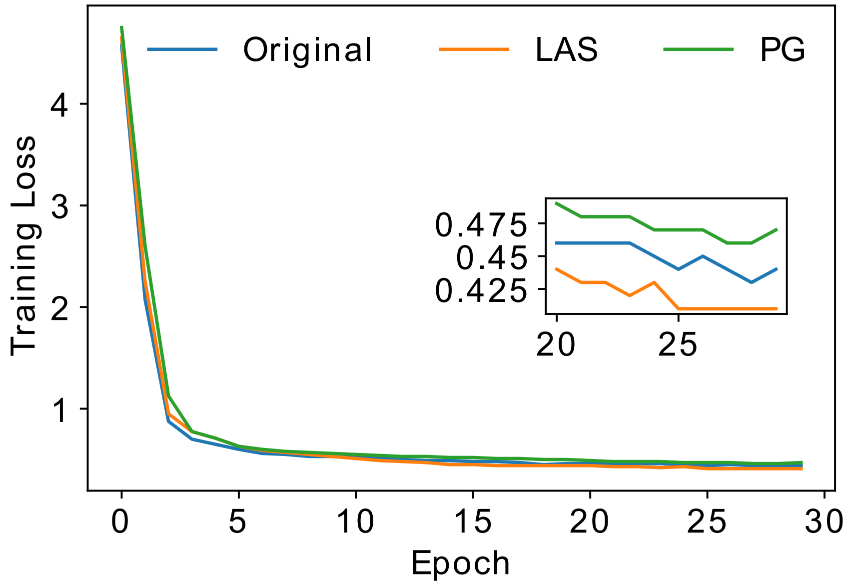


# Evaluation: Speedup on Papers100M

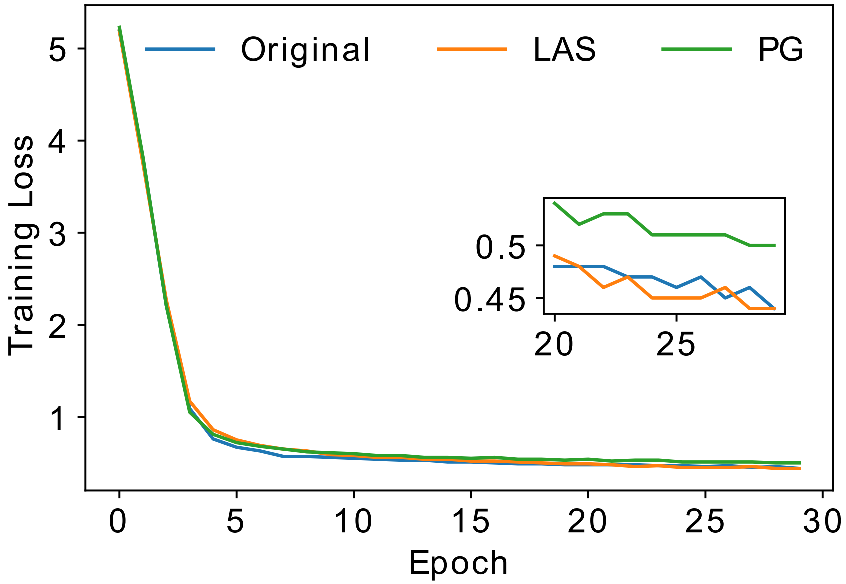



Our strategy reduces the data loading time by 2.4x to 4.0x compared to random

# Evaluation: Accuracy




Training loss on Reddit (2GPU)




Training loss on Reddit (2+2GPU)

LAS has similar convergency speed

LAS has smaller loss at the end of training

# Evaluation: Preprocess overhead

Our algorithm is much faster than the previous graph partitioning algorithms

The execution time for dividing the graphs into four parts

| | Reddit | Yelp | Products |
|---|---|---|---|
| PaGraph | 382 | 1976 | 4753 |
| METIS | 17 | 15 | 83 |
| Our | 0.49 | 0.76 | 3.6 |

- PaGraph has $O(N^2)$ time complexity
- We have $O(N)$ time and space complexity

# Summary

- Aim to reduce the data loading overhead for largescale GNN training on multiple GPUs
- Propose a performance model of the data movement among CPU and GPUs and provide an efficient algorithm to find an optimal data placement strategy
- Propose a locality-aware neighbor sampling technique to further reduce the data loading overhead
- Reduce data movement overhead by 1.2x to 3.3x times with data placement strategy, and achieve up to 4.4x times speedup with locality-aware sampling

For information, doubts and clarifications, contact: shihui-song@uiowa.edu