Mitigating the Effect of Stragglers in Distributed Heterogenous Graph Neural Network Training

Pranjal Naman

April 28, 2023

Introduction and Motivation

Introduction

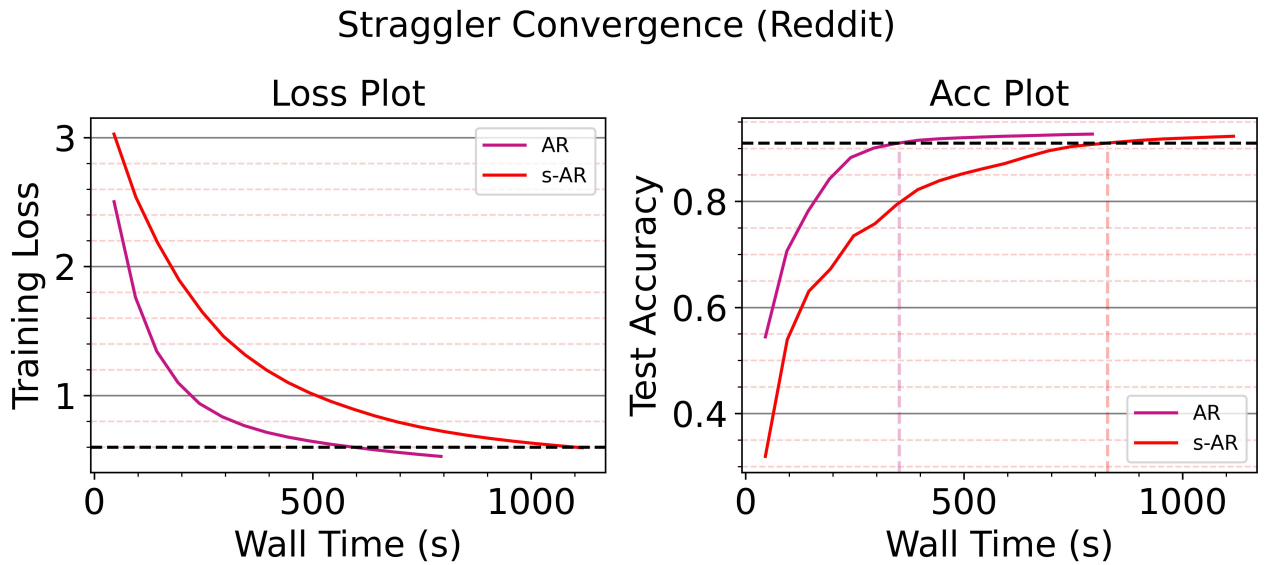
- Storing petabytes of graph data on a single device is not plausible.
- Graph is partitioned into sub-graphs and placed on the worker nodes
- Distributed training widely used for large-scale GNN training using a *data-parallel* approach [4, 1]
- This work focuses on optimizing data parallel GNN training for heterogeneous settings.

Related Work

- Current popular Data Parallel training frameworks make use of the `all-reduce` primitive
- **Centralized** - Parameter Servers, **Decentralized** - `all-reduce` primitive
- **Centralized** - single bottleneck, **Decentralized** - stragglers
- **Asynchronous** - staleness, **Synchronous** - stragglers
- **Grouped Stale Synchronous Parallel (GSSP)** - Introduced in 2022 by Sun et al.[3]
  - Workers with similar performance grouped together
  - Follows an intra-group local and inter-group global synchronization strategy
  - Strikes a balance between staleness & stragglers

Key Motivation

- Commodity clusters have **heterogeneous hardware**.
  - Synchronisation after every batch/epoch is effected by **stragglers**.
  - Asynchronous methods suffer from **staleness problems**.
- Reducing stragglers improves training throughput
- Reducing staleness improves convergence
- Find a **balance between synchronous and asynchronous methods**.
  - Group similar devices together to work in a synchronous manner
  - Groups interact asynchronously with each other.
  - Reduces stragglers, while also mitigating staleness.



Methodology

Grouped ARAR (g-ARAR)

- Group the process according to their training speed.
- Each group carries out `all-reduce` after every batch/epoch, within itself.
- Each group has a chosen leader (lowest rank within group)
- Periodically, the group leaders perform an `all-reduce` amongst themselves.
- Other members get the updated weights in the next iteration synchronization with the group leader.

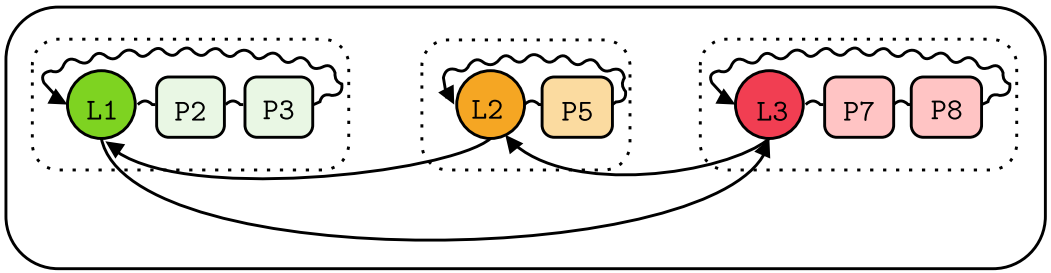
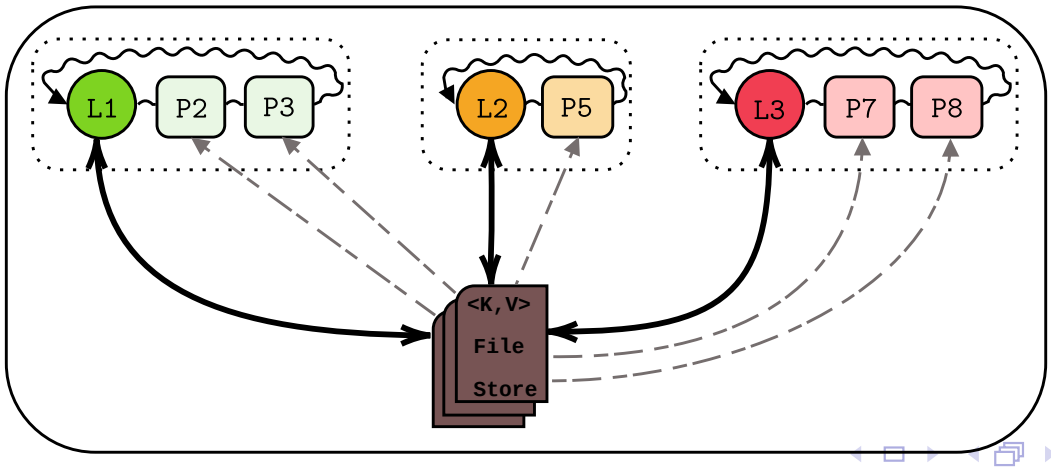


Figure: Schematic for Grouped-ARAR

Grouped ARPS (g-ARPS)

- Decentralized intra-group and centralized inter-group
- Intra-group: all-reduce takes place per iteration
  - no worker has to wait.
  - each worker has a local perception of the model.
- Periodic update of the global PS by the group leaders
  - update the PS parameters.
  - receive the updated parameters from the PS.
  - PS implemented using a distributed $< key, value >$ File Store.
  - workers never have to wait - intra-group waiting time is negligible, inter-group fetching of parameters is independent



Master Worker Partial Reduce (mw-PR)

- Inspired by Miao et al.[2] which implements p-reduce on GPU using shared memory.
- Each worker process communicates its parameters with the master after each iteration.
- Master gets P parameters, averages the parameters and sends them back to the processes
- P has to be chosen carefully
- $P = 1 \rightarrow$ acts as parameter server training
- P too large $\rightarrow$ large number of processes spend time waiting

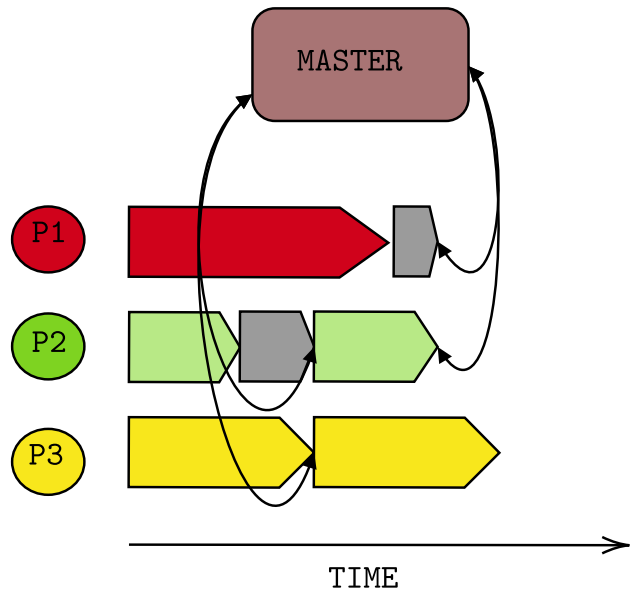


Figure: Schematic for Master Worker Partial Reduce

Implementation

Implementation Overview

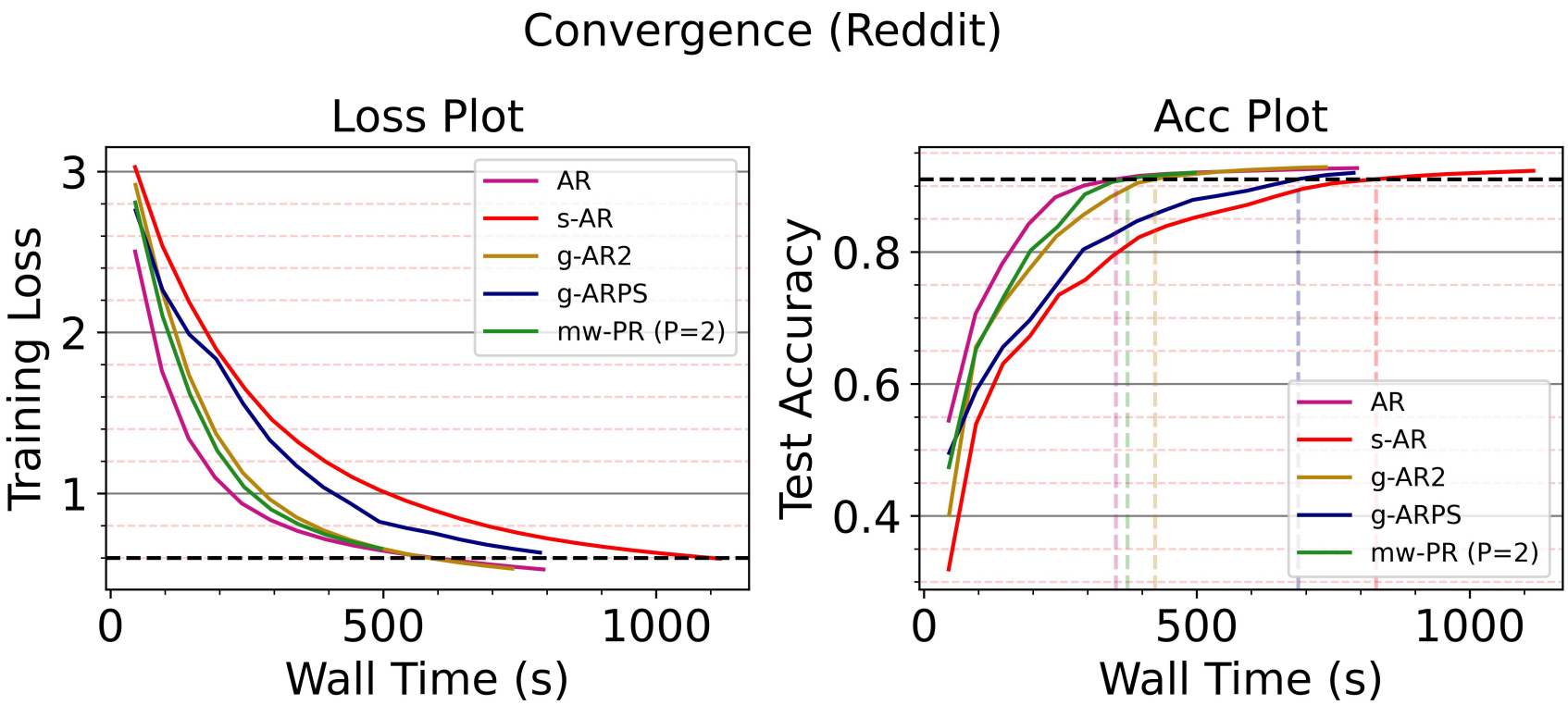
- Used `torch.distributed` for implementation, with the `MPI backend`.
- We implement the following proposed methods:
  - **Grouped-ARAR (g-ARAR)**
  - **Grouped-ARPS (g-ARPS)**
  - **Master Worker Partial Reduce (mw-PR)**
- Classical `all-reduce` with and without stragglers act as the roof-line and the baseline models for evaluation, respectively.
- Each of the methods is implemented for 2 data sets evaluated on a vanilla GCN - Reddit and Cora Full.
- Perform convergence analysis, variation with hyperparameters (like P) and scalability studies for these methods

Evaluation

Setup

- All experiments are run on the IOE cluster (8 compute nodes and 1 head node)
- Heterogeneity simulated by introducing random `sleeps` and launching additional processes
  - `os.nice` to control process priority to simulate heterogeneity.
  - A script runs initially that segregates the processes into three groups - slow, medium & fast.
- Graph data is **hash partitioned** among workers
- **Period of 45 seconds** for all inter-group communications
- Partitioning introduces halo vertices (1 hop vertices)

Convergence



- As expected, `all-reduce` performs best in terms of convergence.
- `mw-PR` seems to outperform `g-ARPS` and `g-ARAR`.
- `g-ARAR` and `g-ARPS` converge very similarly.

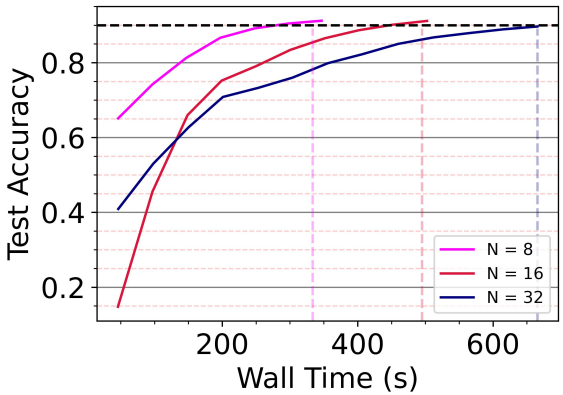
Discussion

- Proposed methods significantly reduce the waiting time of each worker. Thus, increasing work efficiency
- Both grouping variants significantly reduce communication time
- g-ARPS has nearly no wait time at all.
- Compute time remains similar across all methods - expected.
- Small but unpredictable waiting times in mw-PR.

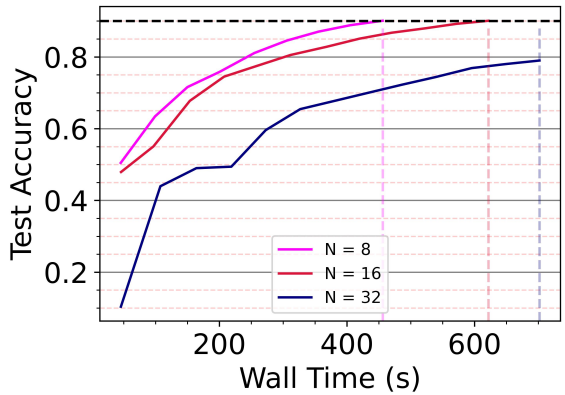
Scalability Study

- Independent subgraphs placed on workers
- `g-ARAR` & `g-ARPS` scale poorly for the GCN model trained with the Reddit dataset
- `mw-PR` also shows poor scalability - further investigation shows not enough compute per epoch $\rightarrow$ communication dominates

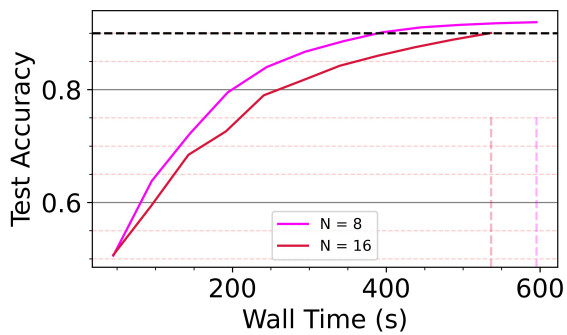
Scalability Plot (g-ARAR on Reddit)



Scalability Plot (g-ARPS on Reddit)

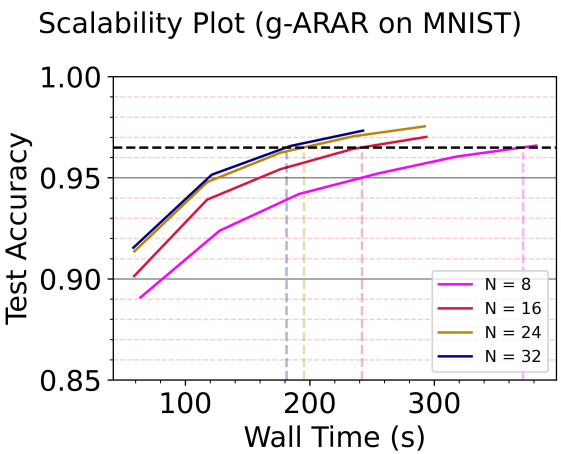


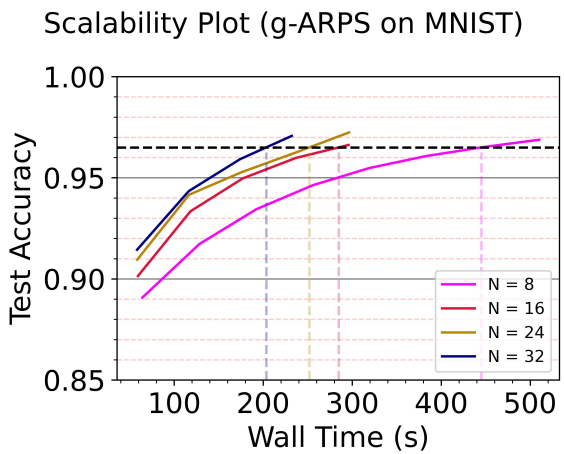
Scalability Plot (mw-PR on Reddit) for P = 3

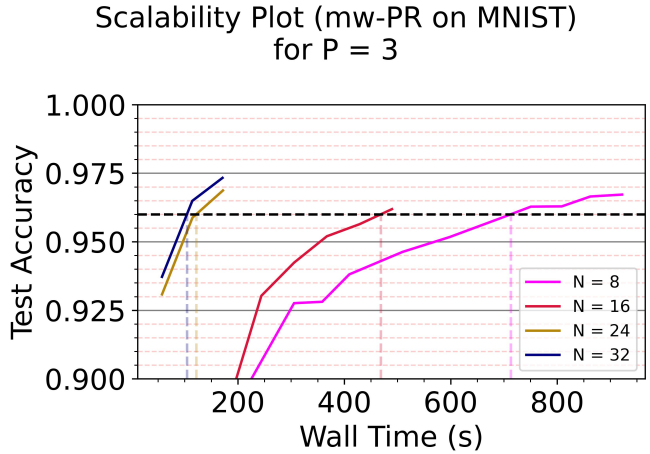


- Study the proposed methods on a DNN to understand this deviation from the expected scaling
- not enough compute per epoch $\rightarrow$ communication dominates

Scalability of a DNN model on MNIST



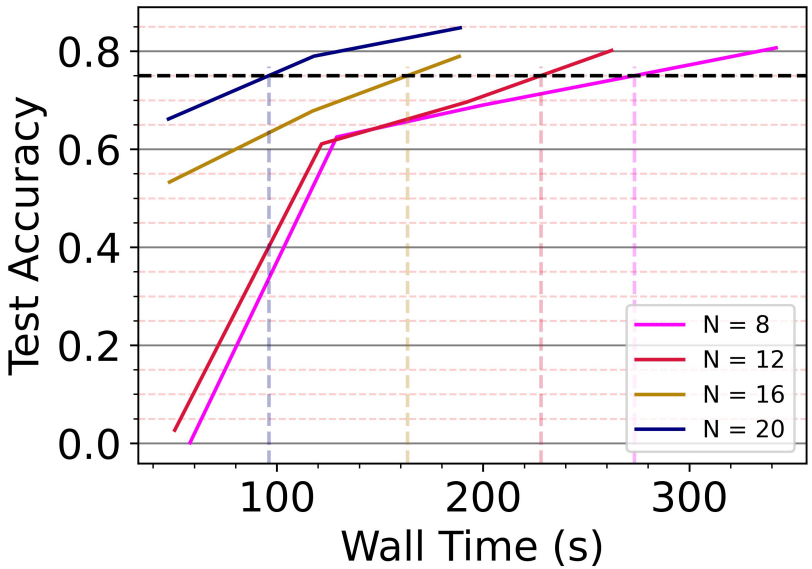




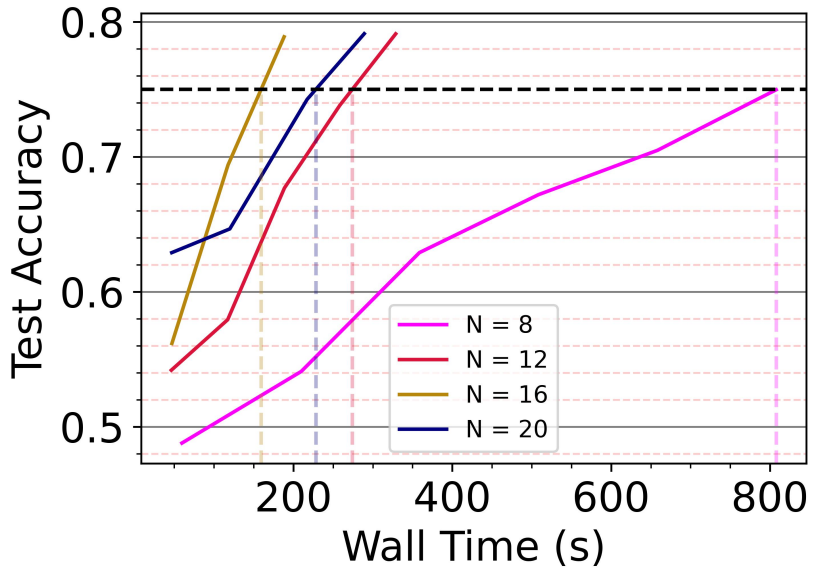
Solution to better understand scalability?

Increase the complexity of the GCN model or increase data

Scalability Plot (g-ARAR on Reddit)



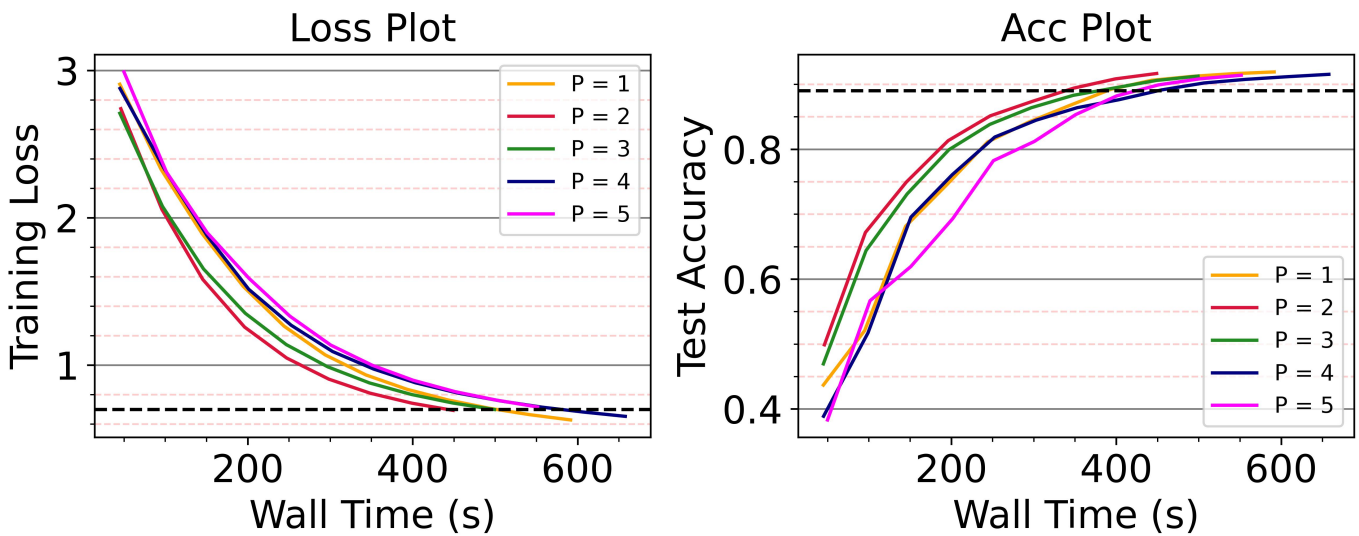
Scalability Plot (g-ARPS on Reddit)



Key Takeaways

- **Novel Work** :
  - `g-ARAR` is inspired by GSSP [3], but takes a synchronous approach.
  - `g-ARPS` takes a locally synchronous, globally asynchronous approach (to try and achieve the best of both worlds).
  - `mw-PR` is a novel implementation, inspired by the simple PS, for distributed settings, that can be applied with or without grouping.
- `g-ARPS` makes no inter-group communication calls.
  - uses a distributed key, value store instead.
  - allows for asynchronous sharing of model weights.

Convergence Plots v P (mw-PR on Reddit)



Key Takeaways

- All proposed methods have improved work efficiency, with g-ARPS being the best.
- g-ARPS has a negligible wait time.
  - Usage of dist. key, value file store removes any inter-group communication.
- g-ARPS outperforms g-ARAR and mw-PR in reducing stragglers.
- Wait time in mw-PR has a significant variance.
  - The p-reduce may take a random amount of time.
  - Wait times are not predictable.
- mw-PR has a better convergence rate than g-ARAR or g-ARPS.

Thank you!

[1] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR 2019 Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[2] Xupeng Miao, Xiaonan Nie, Yingxia Shao, Zhi Yang, Jiawei Jiang, Lingxiao Ma, and Bin Cui. Heterogeneity-aware distributed machine learning training via partial reduce. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 2262–2270. Association for Computing Machinery, 2021.

[3] Haifeng Sun, Zhiyi Gui, Song Guo, Qi Qi, Jingyu Wang, and Jianxin Liao. Gssp: Eliminating stragglers through grouping synchronous for distributed deep learning in heterogeneous cluster. *IEEE Transactions on Cloud Computing*, 10(4):2637–2648, 2022.

[4] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. Distdgl: Distributed graph neural network training for billion-scale graphs. *CoRR*, abs/2010.05337, 2020.