Performance Evaluation of Distributed Graph Neural Network Inference

Yitao Lei
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
y.lei2@student.vu.nl

Mingjiang Ma
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
m.ma@student.vu.nl

Ivy Rui Wang
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
r.wang3@student.vu.nl

Zhiyan Yao
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
z.yao2@student.vu.nl

Zhongyu Shi
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
z.shi3@student.vu.nl

Delong Yuan
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
d.yuan@student.vu.nl

Prof. dr. ir. Alexandru Iosup
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
A.Iosup@atlarge-research.com

Sacheendra Talluri
Vrije Universiteit Amsterdam
Amsterdam, Netherlands
S.Talluri@atlarge-research.com

***Abstract*—The advancement in graph-based data analytics has elevated the Graph Neural Network (GNN) to a prominent position for extracting insights from complex, interconnected data structures. Addressing scalability challenges inherent in large-scale graph datasets requires leveraging distributed computing frameworks. This paper delves into a comprehensive performance evaluation of GNN inference within a distributed computing context, crucial for understanding the impact of such environments on the efficiency, accuracy, and reliability of a GNN model. Based on architecture and node-node operation, our approach develops communication system with reasonable overhead. Additionally, we place a strong emphasis on key metrics such as load balancing, redundancy, and fault tolerance.**

***Index Terms*—GNN, Distributed system, Performance Evaluation, Load Balancing, Fault Tolerance**

I. Introduction

Graph data is ubiquitous in many domains, such as social networks, bioinformatics, knowledge graphs, and recommendation systems. Graph data often exhibits complex and irregular structures that are challenging to analyze and understand. Graph Neural Networks (GNNs) are a class of deep learning models that can effectively capture the structural and semantic information of graphs, and have shown promising results in various graph-based tasks, such as node classification, link prediction, and graph generation.

However, applying GNNs to large-scale graphs poses significant challenges in terms of computational and communication costs. To address these challenges, distributed systems are often employed to parallelize the GNN inference process across multiple machines or devices. However, distributed systems also introduce new issues that need to be carefully considered, such as how to partition the graph, how to synchronize the model parameters, how to balance the workload, and how to handle failures.

In this paper, we present a comprehensive performance evaluation of GNN inference in a distributed system setting. We focus on three representative GNN models: Graph Attention Network (GAT), Graph Convolutional Network (GCN), and Message Passing Neural Network (MPNN). We implement these models using a popular distributed deep learning framework, Torch Distributed, and evaluate them on several real-world graph datasets. We measure the performance of these models in terms of accuracy, efficiency, availability and scalability. We also investigate the impact of different factors on the performance of GNN inference, such as graph partitioning strategies, Data discrepancy, and different network conditions. The main contributions of this paper are:

- We implement a peer-to-peer architecture distributed GNN inference pipeline and apply it on GCN, MPNN and GAT.

- We provide a comprehensive performance evaluation of GNN inference in a distributed system setting, covering three representative GNN models, several real-world graph datasets, and various performance metrics.

- We analyze the effect of different factors on the performance of GNN inference in a distributed system setting, such as graph partitioning strategies, load balancing schemes, redundancy levels, and fault tolerance mechanisms.

- We identify the challenges and opportunities for future research on GNN inference in distributed systems.

The open source code of the project can be found on https://github.com/yleinl/DSassignment

II. Background

Graph Neural Networks (GNNs) have revolutionized the way we handle graph-structured data in various fields such as social networking, bioinformatics, and complex network analysis. Unlike traditional neural networks, GNNs are specifically designed to capture the complex relationships and patterns within graph data, considering both the features of individual nodes and the overall structure of the graph [1].

The deployment of GNNs in real-world applications often involves dealing with large-scale graph datasets. Due to device storage limitations, it is difficult to process a graph with billions of nodes in a single machine. Thus distributed system is required for distributed inference of large graph datasets. This integration brings forth several key challenges and considerations:

1) Scalability and Data Management: As graph datasets grow in size, they often exceed the capacity of a single machine. Distributed systems offer a solution by partitioning the graph across multiple machines. However, this introduces complexity in data management, requiring efficient strategies for data distribution, partitioning, and retrieval.
2) Network Latency in Data Access: In distributed GNN applications, accessing node information scattered across different machines can introduce significant latency. This is particularly challenging for GNNs, where the neighborhood information of each node is crucial for effective processing. Minimizing latency and optimizing data access patterns are essential for maintaining GNN performance.
3) Resource Allocation and Load Balancing: Effective resource utilization is critical in a distributed setting. GNNs require balanced allocation of computational and memory resources across the network to ensure efficient processing. Load balancing becomes a crucial aspect, especially when dealing with dynamic and unevenly distributed graph data.
4) Consistency and Synchronization: Ensuring data consistency in the face of concurrent operations in a distributed environment is a significant challenge. GNNs require up-to-date and synchronized information from various parts of the graph, necessitating robust synchronization mechanisms to maintain data integrity and accuracy.
5) Fault Tolerance and System Reliability: Distributed systems are prone to failures, including node outages and network issues. Developing fault-tolerant mechanisms is essential to ensure that GNN operations are not severely disrupted and can recover or continue processing in the face of such failures.
6) Performance Metrics and Optimization: Evaluating the performance of GNNs in distributed settings involves considering various metrics such as computational efficiency, response time, and accuracy of graph analysis. Optimizing these metrics in the face of network variability and distributed system complexities is key to leveraging the full potential of GNNs.

We can see that scientists are addressing these issues. In order to exploit the collaborative and cooperative nature of intelligent agents for anomaly detection, a multi-agent system, with each agent implementing a Graph Neural Network, is designed [2]. For distributed graph neural network training, we can also refer to the survey that handles major challenges in distributed GNN training such as massive feature communication, the loss of model accuracy, and workload imbalance [3]. A new system for training GNNs in a mini-batch fashion on a cluster of machines has been developed. For instance, the DistDGL, based on the Deep Graph Library (DGL), distributes the graph and its associated data (initial features and embeddings) across the machines and uses this distribution to derive a computational decomposition by following an owner-compute rule [4].

Based on the surveys before, we propose a new application for evaluating distributed graph neural network inference.

III. System Design

This section goes over the design for the distributed inference system pipeline. Firstly, we discuss the architecture of the distributed inference and give a high-level view. Then we go into details about the tasks and communications on each node. Lastly, we will discuss the key features of our distributed inference system pipeline. In the end, we will analyze some concepts regarding to our system performance bottleneck and predict their impacts on system performance. We achieved the design through the long-term iterative AtLarge [5] design process.

A. Requirements

Our system design is driven by a set of functional and non-functional requirements that guide the development of the distributed inference system pipeline.

1) Functional Requirements:

1) **FR-01:** The system must allow for the partitioning of large graph datasets.
2) **FR-02:** It should support real-time inference on distributed clusters.
3) **FR-03:** The system should perform distributed inference across different sample sizes.
4) **FR-04:** The system should perform distributed inference on different variants of the GNN model.
5) **FR-05:** The system should allow for dynamic scaling of computing nodes as needed.

2) Non-Functional Requirements:

1) **NFR-01:** The system must achieve low-latency communication between distributed nodes to minimize overhead.
2) **NFR-02:** The system should realize at least weak scalability, allowing the system to handle growing sizes of datasets with more computing resources.
3) **NFR-03:** The system should work even under bad network conditions. The latency should be limited and the accuracy cannot lose too much.

B. Architecture

Our system consists of a set of machines or system nodes. Each node possesses a portion of the graph data and feeds it into a graph neural network for inference. During the design process, we consider two architectures, centralized server architecture and decentralized server architecture. centralized server architecture in which all node features are sent to the master node, and the master node is responsible for synchronization among nodes. in decentralized server architecture, each node is responsible for synchronization of a part of the features by itself and sends them to the nodes that need them. Compared to decentralized architecture, centralized

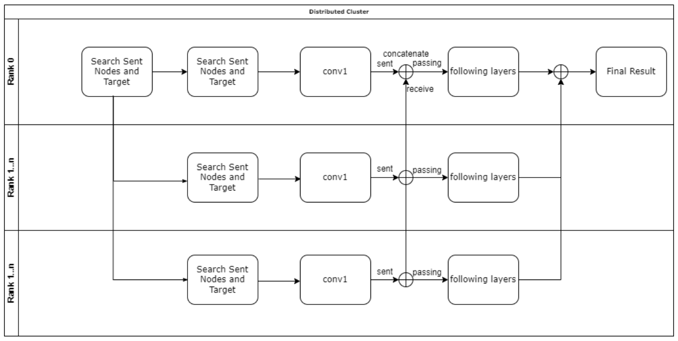

Fig. 1. Node-Node Operation

architecture makes it easier to manage node features, but requires more communication overhead. We found experimentally that inference is very fast even with graph data that requires considerable memory/explicit memory. The most significant reason limiting graph neural network inference for large-scale graphs is memory constraints. Therefore repeatedly processing all large amounts of graph data by the master node would greatly increase the burden on the master node, and likewise lead to a large memory overhead. Therefore, we chose a decentralized architecture where each partial node is responsible for a portion of the graph data inference and consistency maintenance.

In addition to the data partitioning operations performed on rank 0, the operations on each computational node consist of synchronization and inference. Each computational node needs to know which part of the node's features it needs to maintain and which computational node it needs to synchronize this part of the node's features. Therefore, each node needs to first compute the nodes and the corresponding division partitions that need to be synchronized and send the new features to the distributed nodes that need the features of these points after each stage of inference (e.g., convolution) is completed.

C. Node-Node Operation

Before the specific inference and synchronization tasks start, the node with rank0 (the first compute node in the distributed cluster) takes care of the graph partitioning to complete the task distribution. This task is extra compared to the other nodes, so there will be a short idle period for the other nodes at the beginning. After receiving the data, the communication between the nodes is realized through `torch.distributed`, a module provided by Pytorch to support distributed training.

Although `torch.distributed` does not have direct support for distributed inference in deep learning models, we can still use a series of communication primitives provided by it to realize distributed communication. We use non-blocking communication to achieve message sending and receiving to accomplish synchronization of features. The figure shows the nodes' operation and communication with each other.

D. Communication

After our experiments on a single machine, we found that single inference is very fast even for Reddit datasets that require hundreds of gigabytes of memory. Thus, the communication overhead is very important in the overall computing time. To reduce the communication between distributed nodes, we need to reduce the coupling between subgraphs after partitioning. We employ the louvain algorithm [6] for community detection across the graph structure to maximize the modularity of the graph structure. Modularity is a measure of the quality of a network's community structure, which measures the difference between the actual number of edges in the network and the expected number of edges. By maximizing the modularity degree, the algorithm tries to find a way of grouping nodes in which nodes are more closely connected to each other and communities (i.e., our completed divisions) are more sparsely connected to each other so that communication can be minimized.

E. Load Balancing

Load balancing is also an important requirement for a distributed system. Our load balancing is mainly achieved by an average partition method for nodes. In graph neural network inference, the convolution of a graph is a convolution operation on each node, so the inference complexity of graph neural networks is directly linked to the number of nodes. Through the average division of nodes, we can achieve a certain degree of load balancing.

F. Redundancy and Fault Tolerance

In traditional graph partitioning, the entire graph is divided into parts and each partition owns a portion of nodes. The following figure shows a traditional way of graph partitioning. When synchronizing features, you need to send nodes that have connection relationships with other graphs, and splice up the connection relationships and node features by concatenation before performing operations. However, this approach is very risky, once the message is sent incompletely due to network or other reasons, some node features are not received, which will lead to the connection relationship and the number of nodes do not correspond to each other, and the neural network inference reports an error, leading to the failure of the whole distributed system.

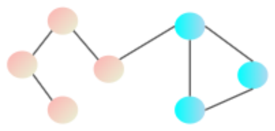

Fig. 2. Original Graph

In order to prevent the system failure due to incomplete number of nodes, we use redundancy to realize the division. As shown in the figure, each redundant node will have a copy in all the graph partitioning partitions that have connections with it, so the synchronization process simply replaces the copies of

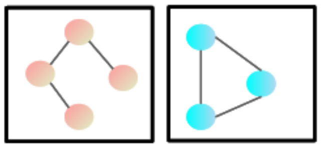

Fig. 3. Normal Partition

the features. Even if it is because of incomplete acceptance of the network condition nodes, the original copy of the features can support the model inference to proceed normally, only a part of the connection relationship of the boundary nodes is lost, which may have a certain impact on the accuracy of the inference results.

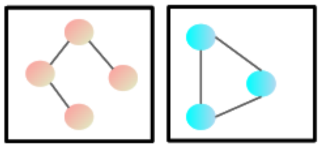

Fig. 4. Redundant Partition

G. Design Problems

To reduce the memory workload on a single machine, we apply the peer-to-peer architecture. However, determining the nodes that need to be synchronized and the synchronization target is a rather heavy task, with a complexity of up to O(nEV), where n is the number of partitions, E is the number of edges, and V is the number of points in each partition. This will take much more time than the GNN inference itself. However, the client-server architecture also has flaws. The server node that synchronizes the results needs to process the entire graph, which violates our original intention of avoiding processing large graphs. Therefore, our future improvement direction can focus on reducing the complexity of preprocessing operations and using a less complex way to determine the synchronization target.

IV. Experiment Design

A. Goals and System-Under-Test

In our experiment, we investigate three benchmark Graph Neural Network (GNN) models: Graph Convolutional Network (GCN) [7], Message Passing Neural Network (MPNN) [8], and Graph Attention Network (GAT) [9]. Each model has distinct characteristics:

- **MPNN:** A foundational GNN that aggregates features of connected nodes using a mean approach.
- **GCN:** A variant of MPNN, which assigns weights to connected nodes based on their degree for weighted average calculation. This process is dependent on the graph structure.
- **GAT:** Incorporates a graph attention mechanism, leading to a more complex structure than GCN and MPNN.

We utilize three datasets in our experiments: Cora, PubMed, and Reddit, which are provided by PyGeometric [10]. Due to resource constraints, we sample the Reddit dataset. The table below summarizes the number of nodes and edges for these datasets:

Dataset	Number of Nodes	Number of Edges
Cora	2708	5429
PubMed	19717	44338
Reddit (Sampled)	2329	11148
Reddit (Unsampled)	232965	114615892

TABLE I
DATASET STATISTICS FOR OUR GNN EXPERIMENTS.

We aim to evaluate the impact of graph size and sparsity on the performance of distributed GNNs, and also how the system would performance in different network environments such as increased latency, packet loss and limited bandwidth. Our experiments are based on the DAS-6 cluster.

B. Experiment 1: Availability and Scalability

This experiment partitioned large graph datasets and performed distributed inference on clusters. The entire workload of inference is distributed among a set of machine nodes that coordinate the parallel inference of the workload. This experiment validated the **FR-01, FR-02, FR-05, and NFR-02**. The number of machine nodes ranges from 3 to 14. In terms of evaluation, we not only focus on the accuracy of inference but also on performance metrics such as inference time and total time.

C. Experiment 2: Generalization

Considering the diverse variants of GNNs, we conducted distributed inference on three major GNN variants (GCN, GAT, MPNN) and evaluated the accuracy and performance. To gain insights into the performance of each GNN variant in distributed inference, we examined their applicability on different graph datasets individually. This experiment validates the **FR-04**. In addition to accuracy, we consider inference and total time in our evaluations.

D. Experiment 3: Different Datasets

Considering the characteristics of different graph structures, we conducted distributed inference and evaluated the effectiveness and performance of three major graph benchmark datasets (Cora, PubMed, Reddit). This experiment can validate the **FR-03**. By conducting experiments on three datasets—Cora, PubMed, and Reddit—with distinct characteristics, we comprehensively assessed the model's generalization capability and inference performance. We take into account both accuracy as well as inference and total time when we evaluate.

E. Experiment 4: Different Network Conditions

In this experiment, we explored the performance of distributed GNNs under certain network conditions: network delay (every 10ms from 0ms to 50ms), loss (every 4% from 0% to 20%), and bandwidth (no limit, 1Mbps, 500Kbps, 200Kbps, 100Kbps, 50Kbps). The performance is based on the inference time and the total task time for communication evaluation. This experiment focuses on the two NFRs of the design: **NFR-01** (low-latency communication) and **NFR-03** (perform well under bad network conditions).

All of the distributed GNN inference tasks are done on 4 compute nodes. To simulate different network conditions, we choose *tc* (traffic control tool) and *netem* (network simulation tool) to control and simulate different network traffic. With the help of them, we can change the latency, packet loss rates, and bandwidth on a certain network interface, also specify to restrict a specific ip.

Since we're using 4 nodes on DAS-6, the way to limit communication between these nodes is use tc and netem to change the traffic limiting parameters for those nodes' ips and apply them to the network devices accessing the corresponding ip.

In more detail, the setup of different delay and loss is alike: use a priority queue and add a restriction to one of its sub-queues. Finally, use a filter to direct all matching traffic to that sub-queue. For bandwidth setup, we use a hierarchical token bucket queueing rule. We set a high-speed main category with no bandwidth limitations, and create a subcategory with certain bandwidth limitations under the main category. The filter is the same with delay/loss setup.

The testing results are shown in the Fig 9, 10, 11, and 12.

V. RESULTS

A. Results 0: Evaluation Baseline

To enable comparisons in a distributed environment, we first conducted inference processes in a single-machine environment. We recorded the total runtime, inference time, and accuracy as benchmark values, which were later used as baselines in a distributed environment. The specific results are presented in the table II.

B. Results 1: Availability and Scalability

Distributed inference of GNN models was conducted across 3 to 14 machine nodes. Overall, the accuracy of distributed inference showed differences from the baseline within an acceptable range(Figure 8), and in some cases, even surpassed the baseline (notably, with an increase in the number of nodes, GCN demonstrated a gradual improvement in accuracy on the Cora dataset and consistently outperformed the baseline (rightmost figure in 8)). Detailed analysis of the results is provided in the discussion section. In most cases, there was a slight decrease in accuracy with an increase in the number of nodes, but the decline was not excessively large.

However, it is worth noting that as the number of nodes increased, both total runtime and inference time exhibited an upward trend (Figure 5, 6, 7). This could be attributed to additional overhead from communication and coordination between nodes, leading to an overall increase in the time required to complete the inference task.

C. Results 2: Generalization

We attempted distributed inference on variants of three major GNN models. These variants exhibit differences in computation processes at each layer, ultimately leading to distinct patterns in the results. Overall, the accuracy of GCN is most affected by distribution, while the accuracy of GAT and MPNN is least affected(Figure 8). This may be attributed to the fact that in the GCN model, normalization with respect to node degree is applied during each layer's aggregation process, requiring information from the overall graph structure. Due to the partitioning of the graph on worker nodes, structural information becomes incomplete, leading to fluctuations in accuracy. In contrast, in the other two models, worker nodes aggregate features by averaging over connected nodes, requiring only feature information, which has already been synchronized.

From an efficiency standpoint, distributed inference does not seem to bring stable additional gains(Figure 5, 6, 7). In some cases, both total runtime and inference time are reduced, while in other cases, there is no efficiency improvement, and even instances of efficiency decline.

D. Results 3: Different Sample Sizes

We conducted distributed inference on different complex graph datasets. The Cora dataset represents a sparse small graph, PubMed represents a sparse large graph, and the original Reddit graph is extremely large and dense, making it unable to run smoothly on DAS-6 (possibly due to machine limitations causing the execution to hang). Therefore, we resampled it at a ratio of 0.01, representing a dense small graph.

Observing the accuracy results of the GCN model (Figure 8), we found that the distributed inference accuracy on the Cora dataset is higher than the baseline. This suggests that in sparse graphs, the impact of worker nodes lacking some connections is minimal. In some cases, if some interfering data is lost, accuracy may even increase. However, for dense graphs like Reddit, losing connection relationships may lead to a drastic decrease in accuracy. In contrast, for the MPNN and GAT models, the accuracy of distributed inference on these three datasets remains stable. This is likely because in these two models, worker nodes have less demand for global information, so distributed inference does not result in significant information loss.

E. Result 4: Different Network Conditions

From Fig 9, 10, 11, we can see as the network delay or network loss grows, the total time and inference time for three models on three databases are both increasing linearly. When the bandwidth is decreasing by about half each time, the distribution of time and bandwidth shows an exponential shape.

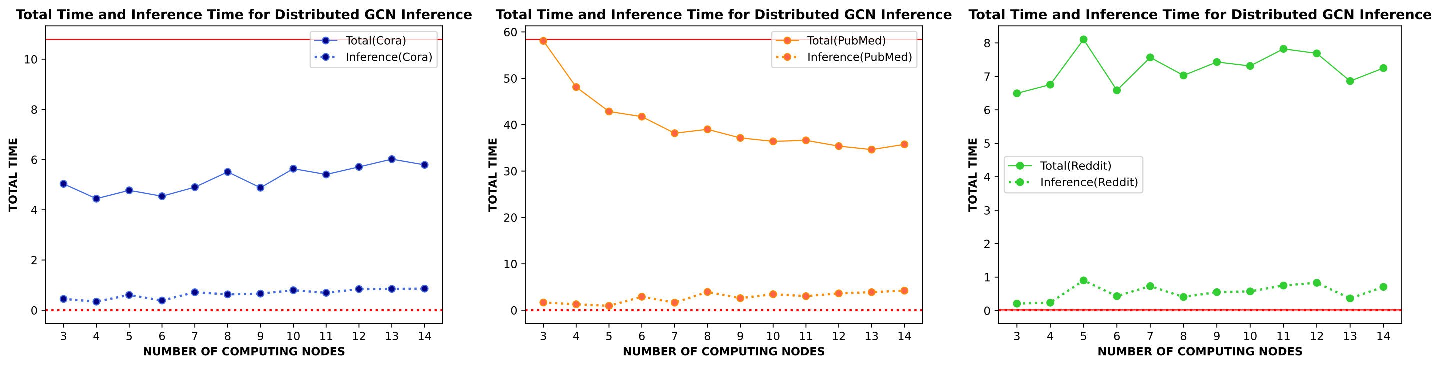

Fig. 5. **Illustration of total time and inference time for running distributed inference pipeline of GCN over three different sizes of complex graph data(Cora, PubMed, and Reddit datasets) with different numbers of distributed computing nodes from 3 to 14.** The leftmost figure shows total and inference time changes on the Cora dataset. The figure in the middle shows total and inference time changes on the PubMed dataset. The rightmost figure shows total and inference time changes on the Reddit dataset. The red solid line represents the baseline for total time and the red dotted line represents the baseline for inference time. The baseline records total time and inference time in a single-machine (non-distributed) environment.

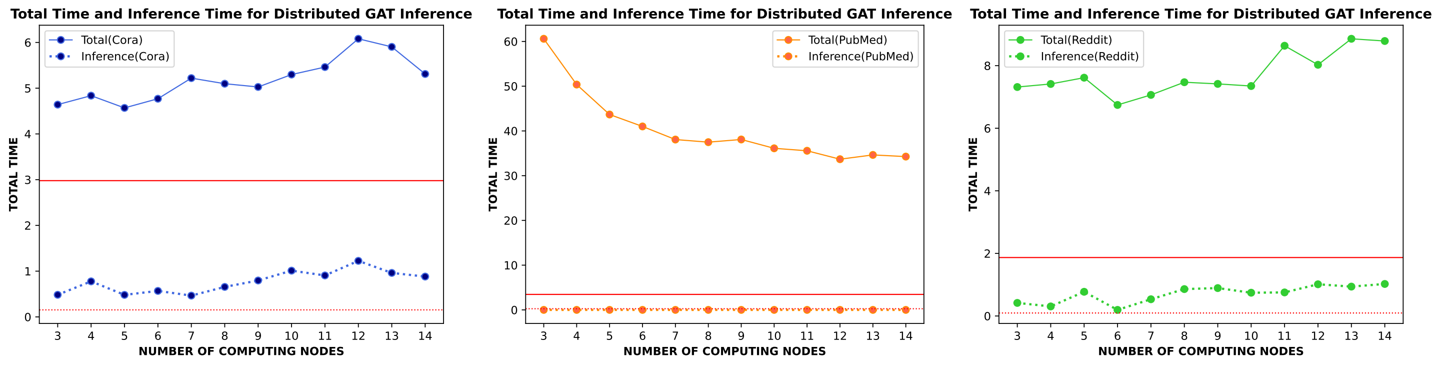

Fig. 6. **Illustration of total time and inference time for running distributed inference pipeline of GAT over three different sizes of complex graph data(Cora, PubMed, and Reddit datasets) with different numbers of distributed computing nodes from 3 to 14.** The leftmost figure shows total and inference time changes on the Cora dataset. The figure in the middle shows total and inference time changes on the PubMed dataset. The rightmost figure shows total and inference time changes on the Reddit dataset. Red solid line represents the baseline for total time and red dotted line represents the baseline for inference time. The baseline records total time and inference time in a single-machine (non-distributed) environment.

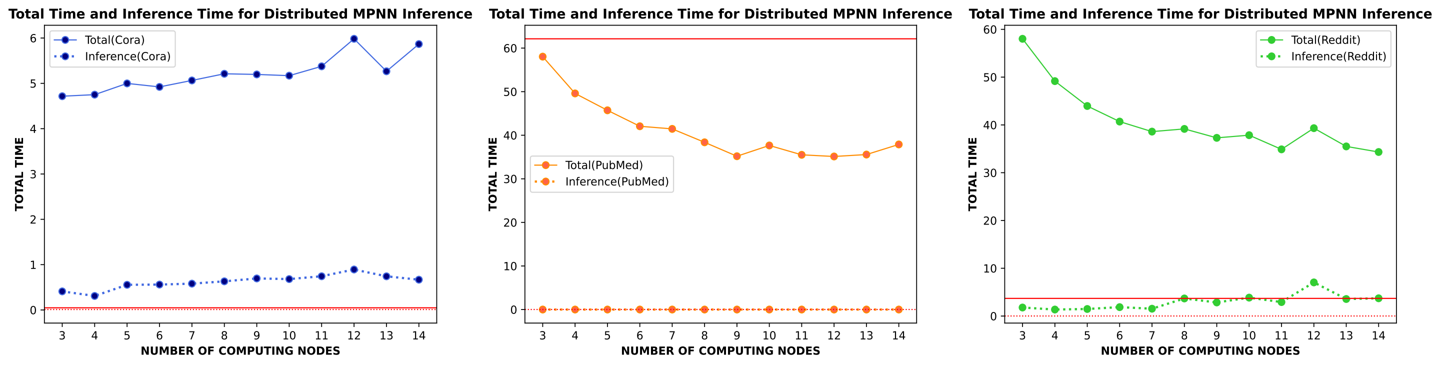

Fig. 7. **Illustration of total time and inference time for running distributed inference pipeline of MPNN over three different sizes of complex graph data(Cora, PubMed, and Reddit datasets) with different numbers of distributed computing nodes from 3 to 14.** The leftmost figure shows total and inference time changes on the Cora dataset. The figure in the middle shows total and inference time changes on the PubMed dataset. The rightmost figure shows total and inference time changes on the Reddit dataset. Red solid line represents the baseline for total time and red dotted line represents the baseline for inference time. The baseline records total time and inference time in a single-machine (non-distributed) environment.

	Total Time(seconds)			Inference Time(seconds)			Accuracy		
	Cora	PubMed	Reddit	Cora	PubMed	Reddit	Cora	PubMed	Reddit
GCN	10.787	58.389	0.014	0.003	0.012	0.020	0.799	0.788	0.746
GAT	2.978	3.461	1.869	0.153	0.259	0.096	0.826	0.776	0.727
MPNN	0.047	62.133	3.695	0.009	0.009	0.0141	0.802	0.782	0.704

TABLE II

EVALUATION BASELINE. THE TABLE SHOWS TOTAL TIME, INFERENCE TIME, AND ACCURACY IN A SINGLE-MACHINE (NON-DISTRIBUTED) ENVIRONMENT.

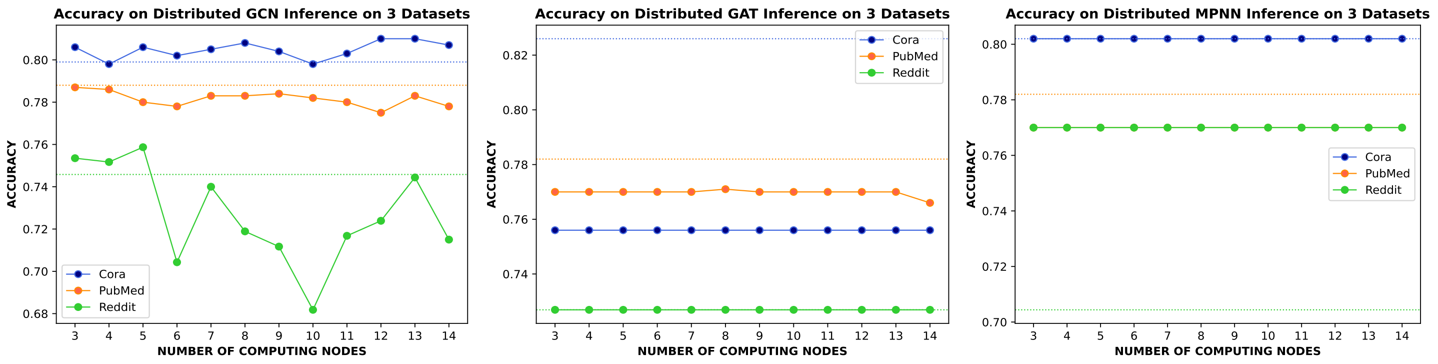

Fig. 8. **Accuracy on three GNN models (GCN, GAT and MPNN) on 3 datasets (Cora, PubMed and Reddit) with different numbers of distributed computing nodes from 3 to 14.** The leftmost figure shows accuracy on GCN. The figure in the middle shows accuracy on GAT. The rightmost figure shows accuracy on MPNN. Dotted lines denote the baselines in a single-machine (non-distributed) environment.

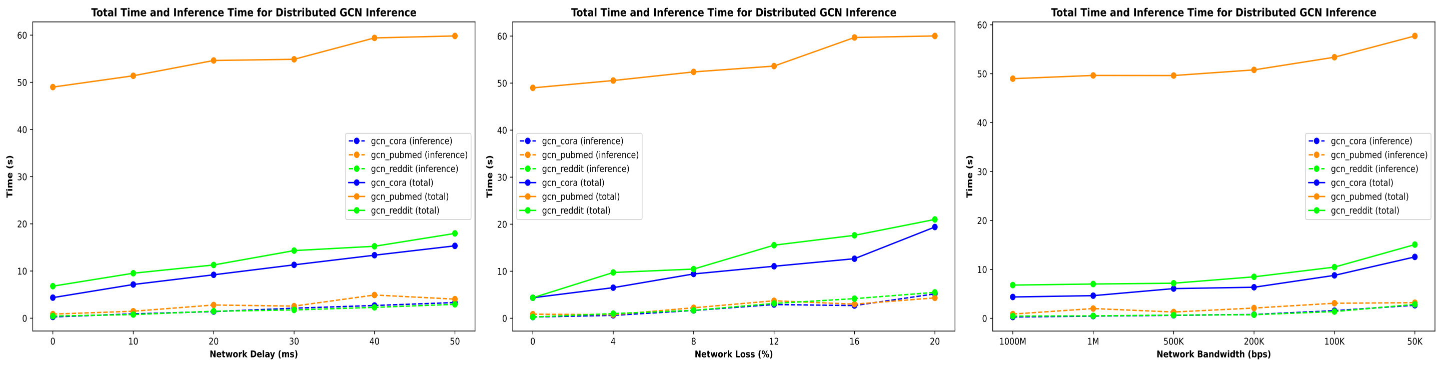

Fig. 9. **Illustration of total time and inference time for running distributed inference pipeline of GCN over three different sizes of complex graph data(Cora, PubMed, and Reddit datasets) under different network circumstances.** The leftmost figure shows accuracy when changing the network delay from 0 to 50 ms. The figure in the middle shows accuracy when changing the network loss from 0 to 20%. The rightmost figure shows accuracy when network bandwidth differs from no limit to 50Kbps.

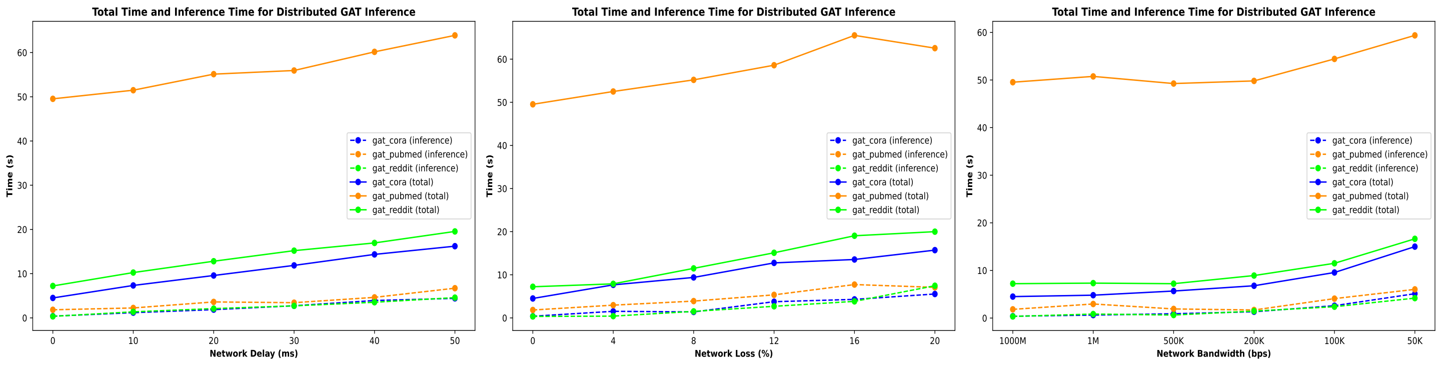

Fig. 10. **Illustration of total time and inference time for running distributed inference pipeline of GAT over three different sizes of complex graph data(Cora, PubMed, and Reddit datasets) under different network circumstances.** The leftmost figure shows accuracy when changing the network delay from 0 to 50 ms. The figure in the middle shows accuracy when changing the network loss from 0 to 20%. The rightmost figure shows accuracy when network bandwidth differs from no limit to 50Kbps.

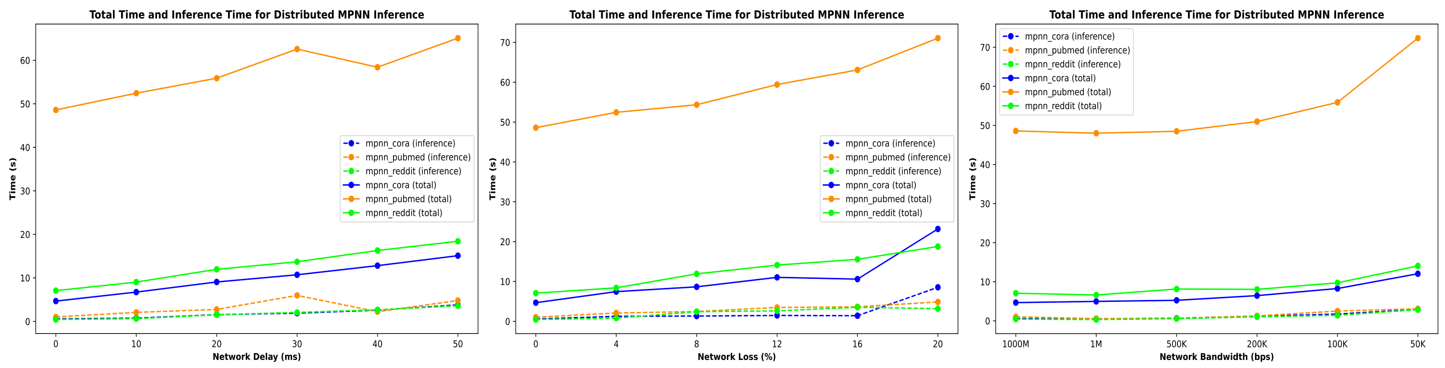

Fig. 11. **Illustration of total time and inference time for running distributed inference pipeline of MPNN over three different sizes of complex graph data(Cora, PubMed, and Reddit datasets) under different network circumstances.** The leftmost figure shows accuracy when changing the network delay from 0 to 50 ms. The figure in the middle shows accuracy when changing the network loss from 0 to 20%. The rightmost figure shows accuracy when network bandwidth differs from no limit to 50Kbps.

Though accuracy is not highly considered in this experiment, in Fig 12, we can see the accuracy of all the works is relatively stable. For MPNN on the Reddit database, it fluctuates occasionally, but the overall accuracy is maintained.

From these observations, the system's performance is relatively stable, due to the robust model. So the NFR-03 (The system should work even under bad network conditions; The latency should be limited and the accuracy cannot lose too much) is satisfied by the system.

VI. Discussion

Our work revolves around the comprehensive performance evaluation of GNN inference within a distributed computing context. Our distributed GNN inference system shows great scalability, generalization, fault tolerance, and efficiency with acceptable performance drops based on the results shown in section V.

In terms of scalability, our system can be perfectly extended to multiple nodes and achieves weak scalability, which means the system can deal with more workloads, i.e., bigger graphs when given more resources. This property makes GNN inference for billion-scale Graphs possible.

In terms of generalization, several mainstream GNN models including GCN, GAT and MPNN are tested on datasets of different scales. These experiments highlight the compatibility of our system and prove the potential of our system to be used more generally.

In terms of fault tolerance, experiments under bad network conditions show our redundancy method works well. However, the system still needs more diversified fault tolerance mechanisms to keep the whole system steady. This is our future direction of improvement.

In terms of efficiency, almost all the models on all the datasets we tested show different degrees of decay in accuracy and latency. This is reasonable because we lack the features of neighbouring nodes and we need to communicate the needed features under a distributed computing context which may lose messages and thus accuracy drops. In addition, such communication overhead will induce worse latency. This is also our future direction of improvement.

In terms of tradeoffs inherent in the design of our system, we have identified several key challenges and considerations. These include scalability VS. performance, generalization VS. complexity and etc. In general, we can sacrifice performance and system simplicity in order to get the ability of tackling more workloads, for example, the ability to do inference of billion-scale graphs where they cannot even be stored on single machine.

Regarding the decision to use a distributed GNN inference system, it is important to weigh the tradeoffs. While a distributed GNN inference system offers the potential for scalability and ability to solve billion-scale graph inferencing, performance will drop within a small margin. It also introduces complexity in data management, communication overhead, and the need for robust synchronization mechanisms. Additionally, ensuring fault tolerance and system reliability becomes crucial in a distributed environment. Therefore, the decision to use a distributed system should be based on a thorough evaluation of the specific requirements, workload characteristics, and tradeoffs involved.

Extrapolating from the results reported in Section V for system workloads that are orders of magnitude higher than what has been tried in real-world experiments, it becomes essential to consider the potential challenges and limitations of scaling the system. As workloads increase significantly, issues such as data distribution, communication overhead, load balancing, and fault tolerance may become more pronounced. Therefore, careful consideration and potentially further experimentation would be necessary to assess the feasibility and effectiveness of using a distributed system for such high workloads.

VII. Conclustion

Our distributed GNN inference system brings a feasible solution of GNN inference for billion-scale graphs with good scalability, generalization, and fault tolerance. However, the accuracy and speed drop are urgent problems, and how to

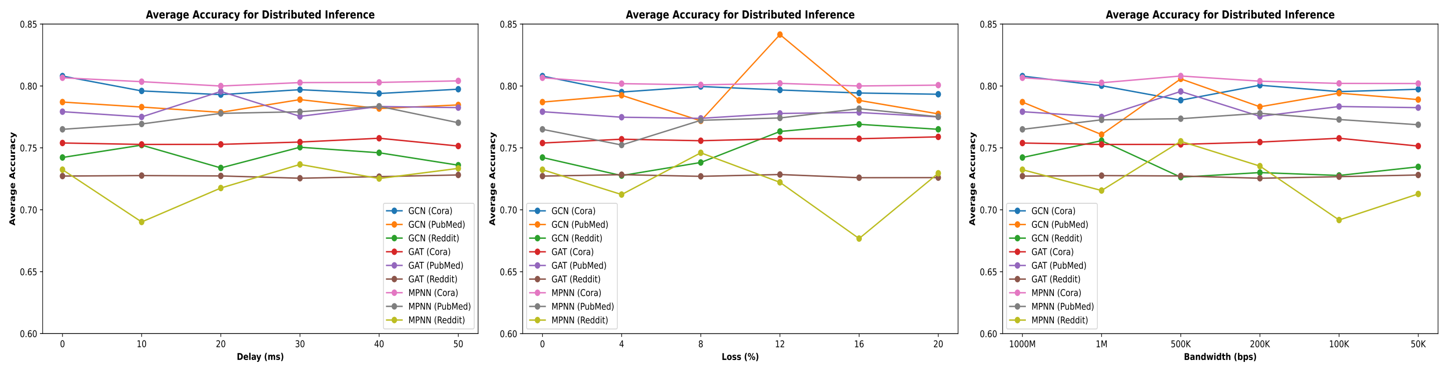

Fig. 12. **Accuracy on three GNN models (GCN, GAT and MPNN) on 3 datasets (Cora, PubMed and Reddit) under different network circumstances.** The leftmost figure shows accuracy when changing the network delay from 0 to 50 ms. The figure in the middle shows accuracy when changing the network loss from 0 to 20%. The rightmost figure shows accuracy when network bandwidth differs from no limit to 50Kbps.

improve them under a distributed computing context is our direction of future work.

VIII. Appendix

A. Time Sheet

Activity	Time (Hours)
Total Time	160
Think Time	30
GNN Learning Time 20	
Dev Time	30
XP Time	20
Analysis Time	25
Write Time	15
Wasted Time	20

TABLE III
Time Allocation for the Assignment

B. Open Source

Our code is open source on GitHub, you can get it by https://github.com/yleinl/DSassignment. It is implemented based on the GCN and GAT implementation from Data Science Group, IIT Roorkee. [11].

References

[1] J. Vatter, R. Mayer, and H.-A. Jacobsen, "The evolution of distributed systems for graph neural networks and their origin in graph processing and deep learning: A survey," *ACM Computing Surveys*, 2023.

[2] A. Protogerou, S. Papadopoulos, A. Drosou, D. Tzovaras, and I. Refanidis, "A graph neural network method for distributed anomaly detection in iot," *Evolving Systems*, vol. 12, pp. 19–36, 2021.

[3] Y. Shao, H. Li, X. Gu, H. Yin, Y. Li, X. Miao, W. Zhang, B. Cui, and L. Chen, "Distributed graph neural network training: A survey," *arXiv preprint arXiv:2211.00216*, 2022.

[4] D. Zheng, C. Ma, M. Wang, J. Zhou, Q. Su, X. Song, Q. Gan, Z. Zhang, and G. Karypis, "Distdgl: Distributed graph neural network training for billion-scale graphs," in *2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, 2020, pp. 36–44.

[5] A. Iosup, L. Versluis, A. Trivedi, E. Van Eyk, L. Toader, V. Van Beek, G. Frascaria, A. Musaafir, and S. Talluri, "The atlarge vision on the design of distributed systems and ecosystems," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 1765–1776.

[6] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[7] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[8] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1263–1272.

[9] P. Velivckovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[10] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *arXiv preprint arXiv:1903.02428*, 2019.

[11] A. Dagar, A. Pant, S. Gupta, and S. Chandel, "graph_nets," https://github.com/dsgiitr/graph_nets, 2020.