SANCUS: *Staleness*-Aware *Communication*-Avoiding Full-Graph Decentralized Training in Large-Scale Graph Neural Networks

Jingshu Peng
The Hong Kong University of Science and Technology
jpengab@cse.ust.hk

Zhao Chen
The Hong Kong University of Science and Technology
zchenah@cse.ust.hk

Yingxia Shao*
Beijing University of Posts and Telecommunications
shaoyx@bupt.edu.cn

Yanyan Shen*
Shanghai Jiao Tong University
shenyy@sjtu.edu.cn

Lei Chen
The Hong Kong University of Science and Technology
leichen@cse.ust.hk

Jiannong Cao
The Hong Kong Polytechnic University
csjcao@comp.polyu.edu.hk

ABSTRACT

Graph neural networks (GNNs) have emerged due to their success at modeling graph data. Yet, it is challenging for GNNs to efficiently scale to large graphs. Thus, distributed GNNs come into play. To avoid communication caused by expensive data movement between workers, we propose SANCUS, a staleness-aware communication-avoiding decentralized GNN system. By introducing a set of novel bounded embedding staleness metrics and adaptively skipping broadcasts, SANCUS abstracts decentralized GNN processing as sequential matrix multiplication and uses historical embeddings via cache. Theoretically, we show bounded approximation errors of embeddings and gradients with convergence guarantee. Empirically, we evaluate SANCUS with common GNN models via different system setups on large-scale benchmark datasets. Compared to SOTA works, SANCUS can avoid up to 74% communication with at least 1.86× faster throughput on average without accuracy loss.

PVLDB Reference Format:
Jingshu Peng, Zhao Chen, Yingxia Shao, Yanyan Shen, Lei Chen, and Jiannong Cao. SANCUS: *Staleness*-Aware *Communication*-Avoiding Full-Graph Decentralized Training in Large-Scale Graph Neural Networks. PVLDB, 15(9): 1937 - 1950, 2022.
doi:10.14778/3538598.3538614

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/chenzhao/light-dist-gnn.

1 INTRODUCTION

The success of Graph Neural Networks (GNNs) [20] has laid the foundation of recent advancement in the state of the art to model real-life graphs. In essence, GNNs are structure-aware models that construct the network architectures adapted to the original topology of the input graphs. By iteratively aggregating the neighborhood of the targets, GNNs can exploit the structure and feature information at the same time. Despite the promising performance, the major challenge that limits the adoption of GNNs to large-scale graphs lies in the inability to utilize all data in finite time and the scalability of the algorithm itself. To mitigate the ***memory requirement*** with ever-increasing data and model size, distributed GNN processing is the inevitable remedy. Some efforts have been made towards sampling-based distributed GNN training, at the price of information loss [3, 16, 34], sampling overhead [16], and no convergence guarantee [5]. Thus, in this work, we focus on the distributed training of full-GNNs.

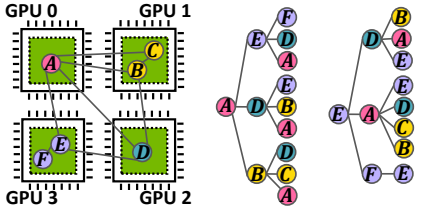


Figure 1: Distributed full-GNN example: nodes in same color are on same GPU. On the left, a 6-node graph is stored on 4 GPUs. On the right are the computational graphs of a 2-layer GNN for Node *A* and *E*. During GNN neighborhood aggregation, intensive cross-device visits (10 times to update Node *A* and 9 for *E*) to fetch neighbor data cause expensive communication overhead.

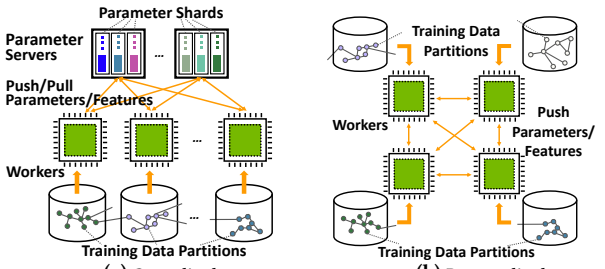
Compared to traditional graph processing or machine learning, new issues have emerged for distributed full-GNNs from the system perspective. Aside from the substantial memory footprint, distributed GNNs are memory-intensive as well as compute-intensive [33, 36] due to coupled irregular neighborhood access and iterative learning procedure. Consequently, the intensive communication, including not only gradients or parameters but also embeddings, makes efficient distributed GNN training more challenging. As exemplified by Figure 1, the training process needs to constantly query the target nodes, their neighbors, and their farther neighbors, to transfer both embeddings and gradients among workers. Thus, by virtue of such data movement in GNN aggregation, cross-device ***data communication*** can be one major archenemy of efficient GNN processing. The incurred communication cost may account for 80% and even more of the total training time [3, 13, 34].

In distributed training, the underlying system architecture of how workers communicate is crucial, especially for GNNs with substantial communication overhead. As illustrated in Figure 2, two approaches exist: centralized and decentralized. Though most distributed GNN systems [13, 16, 27, 30, 43, 45] work in the popular

*Yingxia Shao and Yanyan Shen are the corresponding authors.

Proceedings of the VLDB Endowment, Vol. 15, No. 9 ISSN 2150-8097.
doi:10.14778/3538598.3538614

centralized parameter server (PS) scheme in Figure 2a, they often pay the price of heavy preprocessing and complex workflow, in pursuit of efficiency and scalability. By nature of GNNs, the intensive communication between all the workers and the central PS plus the waiting time for stragglers may lead to high communication overhead [3]. Decentralized architectures, however, can be more robust and easier to deploy, by avoiding the inconvenience in implementing and tuning a PS, centralized bottleneck bandwidth, and single point of failure [22]. Especially for large neural networks, the decentralized scheme is proven to be more superior theoretically [23]. Hence, Tripathy *et al.* [34] offer CAGNET – so far the only SOTA decentralized parallel algorithms [14] adapted to GNNs. In CAGNET, each worker keeps a full copy of parameters to alleviate centralized communication overhead. However, it calls for redundant and unnecessary broadcasts of all embeddings and gradients. Besides, all the workers must wait for stragglers to synchronize, leading to extra communication overhead.



(a) Centralized. (b) Decentralized.

Figure 2: Centralized vs. decentralized GNN processing example. In the centralized scheme, workers periodically send updates to a parameter server. In the decentralized scheme, workers exchange them directly.

In this paper, to fill this gap in efficient GNN processing, we propose SANCUS, a staleness-aware communication-avoiding decentralized GNN training system via adaptively skipping broadcast and caching historical embeddings with bounded staleness. To bypass the irregular data communication between GPUs, we **firstly** revisit the parallel algorithms to distribute GNNs [34] and decrease communication overhead in a fundamentally distinct way. As Figure 3 shows, by regarding the GNN processing purely as a sequence of matrix multiplication operations in a decentralized scheme, each GPU loads the split submatrices without taking the semantic meaning into account. The excessively large adjacency and embedding matrices are sliced into $\mathbf{A}_i$ and $\mathbf{H}_i$ ($i \in [1, 4]$) and distributed to GPU_{i-1} with the full weight matrix W. Then $\mathbf{H}_1$ to $\mathbf{H}_4$ are sequentially one-to-all broadcast to all GPUs in parallel. After 4 broadcasts, the whole embedding matrix $\mathbf{H}$ is updated for a layer. Thus, by moving intact matrix blocks, SANCUS takes advantage of data parallelism to ***avoid communication*** caused by intensive neighbor fetching in Figure 1. **Secondly**, to further ***avoid communication*** under a decentralized scheme, we propose to cache and skip-broadcast the histrocial embeddings We define historical embeddings as the embedding sub-matrices from earlier epochs in each distributed process, i.e., the sub-matrix $\mathbf{H}_i$ individually computed on GPU_{i-1} in Figure 3. We utlize caching and design a novel skip-broadcast operator to support historical embeddings in SANCUS. **Thirdly**, to manage the ***system staleness*** caused by using mixed-version embeddings on each GPU, we propose the generalization of the widely-used *bounded gradient staleness* in centralized schemes [7], to historical embeddings. We introduce a set of novel bounded embedding staleness metrics in decentralized GNNs. Particularly, SANCUS adaptively skip-broadcasts embeddings within bounds and automatically reuses cached historical embeddings to directly avoid communication; otherwise, if the embeddings become too stale, the results are broadcast and updated in cache among GPUs to keep the ***system staleness*** within bounds. Taking Figure 3 as a toy example, the embeddings $\mathbf{H}_3$ is not too stale, then SANCUS can skip broadcast once, now SANCUS only need 7 broadcasts to update all the embeddings in the 2-layer GNN. Again, it should be emphasized that there is no individual embedding fetching in GNN aggregation with SANCUS. Compared to the conventional distributed GNN in Figure 1, only to obtain the embeddings for node ***A*** and ***E***, 10 and 9 request-and-send operations are needed. To update all, 59 request-and-send operations are needed. However, SANCUS only needs as less as 7 broadcasts in total as shown by Figure 3. Also, it should be pointed out that SANCUS has few burden of preprocessing and can be easily applied to any distributed GNNs based on arbitrary matrix blocking and direct matrix operations.

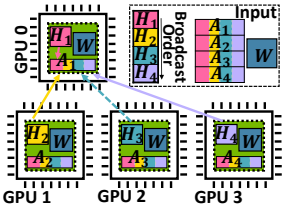


Figure 3: A toy 2-layer GNN example on SANCUS: GPU_{i-1} keeps its shards of $\mathbf{H}_i$ and $\mathbf{A}_i$, with a full W; $\mathbf{H}_i$ are sent to all GPUs in order via one-to-all broadcast (arrows omitted without loss of generality). After 4 sequential broadcasts →,→,→,→ and on-device computation, since the broadcasts are in parallel, all $\mathbf{H}_i$ updates for GPU_{i-1}. Next, SANCUS may tolerate $\mathbf{H}_3$ to skip 1 broadcast as shown by ⇢. In total, only $4 + 3 = 7$ broadcasts are needed.

In summary, our major contributions are: **(1) Problem Exploration.** We share a new perspective to accelerate GNNs by introducing historical embeddings with bounded staleness to decentralized GNNs, treating GNN processing purely as sequential matrix operations. **(2) Novel Metrics.** We introduce a set of novel bounded embedding staleness metrics in decentralized GNNs to effectively manage the system staleness caused by historical embeddings. **(3) New Criterion.** We provide the communication cost bound of SANCUS. We prove the approximation errors of the embeddings and gradients are bounded with convergence guarantee. **(4) Prominent Performance.** We evaluate SANCUS on large-scale benchmark datasets with prevalent GNN models via different system setups, to show its ability to generalize and superiority in efficiency while preserving effectiveness. SANCUS's performance with little or no accuracy loss demonstrates consistency with our theoretical findings.

2 PRELIMINARIES

In this section, we review the related concepts of our target problem, and introduce necessary equations to set the background, then formally define the problem. The key notations are listed in Table 1.

2.1 Graph Neural Networks

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph of order N with a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ and nodes $\mathcal{V} = \{v_1, v_2, \ldots, v_N\}$. Consider the graph adjacency matrix $\mathbf{A}$, where the element A_{ij} in the matrix specifies the relation between the nodes v_i and v_j with $A_{ij} = 1$ if there exists an edge $(v_i, v_j) \in \mathcal{E}$ or otherwise $A_{ij} = 0$. $\mathbf{A}$ is symmetric since $\mathcal{G}$

is undirected. Denote $\hat{\mathbf{A}} = \bar{\mathbf{D}}^{-\frac{1}{2}}\bar{\mathbf{A}}\bar{\mathbf{D}}^{-\frac{1}{2}}$ as the adjacency matrix after symmetric normalization in GCN [20], where $\bar{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ denotes the adjacency matrix with self-connections and $\bar{\mathbf{D}} \in \mathbb{R}^{N\times N} = \mathbf{D} + \mathbf{I}_N$ denotes the diagonal node degree matrix.

Without loss of generality, the ℓ-th layer propagation process of such GNNs [2, 37] can be formulated in matrix form as:

$$\mathbf{H}^{(\ell)} = \sigma\left(\mathbf{H}^{(\ell-1)}, \hat{\mathbf{A}}; \mathbf{W}^{(\ell-1)}\right) \tag{1}$$

where σ denotes the activation function such as ReLU and $\mathbf{W} \in \mathbb{R}^{F\times F}$ denotes the weight matrix. Initially, $\mathbf{H}^{(0)} = \mathbf{X}$ where $\mathbf{X} \in \mathbb{R}^{N\times F}$ is the node embedding matrix whose i-th row represents the length-F embedding vector of node v_i. For convenience, the superscript $^{(\ell)}$ notation is omitted when it is clear from context.

- **Forward Propagation.** Specifically, the neighbors' $(\ell-1)$-th embedding vectors are combined for each node. Based on the iterative scheme of GNNs, the computation process is given:

$$\mathbf{Z}^{(\ell)} = \hat{\mathbf{A}}\mathbf{H}^{(\ell-1)}\mathbf{W}^{(\ell-1)} \tag{2}$$

$$\mathbf{H}^{(\ell)} = \sigma\left(\mathbf{Z}^{(\ell)}\right) \tag{3}$$

- **Backpropagation.** Firstly we derive the recurrence to backpropagate the gradient. Let $\nabla_{\mathbf{Z}^{(\ell)}}\mathcal{L}$ denote the gradient of the loss $\mathcal{L}$ with respect to $\mathbf{Z}^{(\ell)}$. To simplify, we define a factor $\delta^{(\ell)} = \nabla_{\mathbf{Z}^{(\ell)}}\mathcal{L}$. By the chain rule, the relation between $\delta^{(\ell-1)}$ and $\delta^{(\ell)}$ is:

$$\delta^{(\ell-1)} = \frac{\partial\mathcal{L}}{\partial\mathbf{Z}^{(\ell-1)}} = \delta^{(\ell)}\hat{\mathbf{A}}\left(\mathbf{W}^{(\ell-1)}\right)^{\top} \odot \sigma'\left(\mathbf{Z}^{(\ell-1)}\right), \tag{4}$$

where $\sigma'(\cdot)$ is the derivative of the activation function $\sigma(\cdot)$. Then, the gradient $\nabla_{\mathbf{W}^{(\ell)}}\mathcal{L}$ of the loss $\mathcal{L}$ with respect to $\mathbf{W}^{(\ell)}$ is:

$$\nabla_{\mathbf{W}^{(\ell-1)}}\mathcal{L} = \frac{\partial\mathcal{L}}{\partial\mathbf{W}^{(\ell-1)}} = \frac{\partial\mathcal{L}}{\partial\mathbf{Z}^{(\ell)}}\frac{\partial\mathbf{Z}^{(\ell)}}{\partial\mathbf{W}^{(\ell-1)}} = \delta^{(\ell)}\hat{\mathbf{A}}\left(\mathbf{H}^{(\ell-1)}\right)^{\top} \tag{5}$$

In GNNs, one training *epoch* consists of a forward propagation and a backpropagation pass, with subsequent weight update:

$$\mathbf{W}^{(\ell-1)} = \mathbf{W}^{(\ell-1)} - \eta\nabla_{\mathbf{W}^{(\ell-1)}}\mathcal{L} \tag{6}$$

where η represents the learning rate. The GNN trains a number of epochs until the accuracy saturates and the model converges.

Table 1: Summary of the key symbols and notations.

Notation	Description
$\mathbf{A}$	Adjacency matrix of the graph $(N \times N)$
$\hat{\mathbf{A}}$	Adjacency matrix after symmetric normalization $(N \times N)$
$\mathbf{W}^{(\ell)}$	Weight matrix at the ℓ^{th} layer $(F \times F)$
$\mathbf{H}^{(\ell)}$	Embedding matrix at the ℓ^{th} layer $(N \times F)$
$\mathbf{Z}^{(\ell)}$	Input matrix to activation function at the ℓ^{th} layer $(N \times F)$
$\mathbf{T}^{(\ell)}$	Intermediate result matrix of the multiplication $\hat{\mathbf{A}}\mathbf{H}^{(\ell)}$
$\nabla_{\mathbf{Z}^{(\ell)}}\mathcal{L} = \delta^{(\ell)}$	Gradient matrix of the loss $\mathcal{L}$ with respect to $\mathbf{Z}^{(\ell)}$ at the ℓ^{th} layer
$\nabla_{\mathbf{W}^{(\ell)}}\mathcal{L}$	Gradient matrix of the loss $\mathcal{L}$ with respect to $\mathbf{W}^{(\ell)}$ at the ℓ^{th} layer
$\mathcal{L}$	Loss of the GNN
η	Learning rate
ϵ	Staleness bound
$P(i)$	i-th process in distributed GNN

2.2 Problem Definition

In this work, we focus on the problem of node-level graph representation learning with distributed GNNs.

Definition 1 (Node-level Graph Representation Learning).

Input: *A L-layer GNN model, with the undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, adjacency matrix $\mathbf{A}$, initial feature matrix $\mathbf{X}$, and labels $\mathbf{Y}$.*

Output: *The representation matrix $\mathbf{H}^{(L)}$ composed of learned low-dimensional vector representation for each node.*

Formally, given a loss function $\mathcal{L}$ and the target $\mathbf{Y}$, the objective function to solve is defined below:

$$\min_{\mathbf{W}} \quad \mathcal{L}\left(\mathbf{H}^{(L)}, \mathbf{Y}\right)$$
$$s.t. \quad \mathbf{H}^{(\ell)} = \sigma\left(\mathbf{H}^{(\ell-1)}, \hat{\mathbf{A}}; \mathbf{W}^{(\ell-1)}\right) \quad \forall\ell \in [1, L].$$

3 THE SANCUS FRAMEWORK

First, we overview Sancus step by step in Section 3.1. Algorithm 1 introduces the complete staleness-aware communication-avoiding decentralized full-GNN training algorithm. To further elaborate on avoiding communication, we propose historical embeddings and skip-broadcast accordingly in Section 3.3 and Section 3.4. To manage system staleness caused by historical embeddings, we propose a set of novel metrics on bounded embedding staleness in Section 3.5.

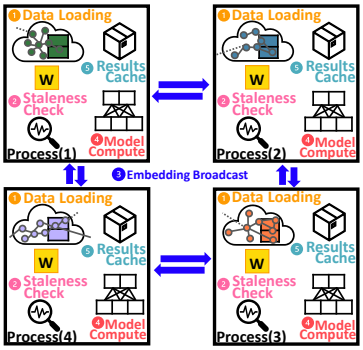


Figure 4: The overall architecture. The full graph and its node features are distributed to each process on the workers. Sancus has five major steps: (1) Data Loading: load the split graph, embedding martix blocks, and full model into each GPU; (2) Staleness Check: check if the embeddings become too stale for the root GPU; (3) Embedding Broadcast: if the embeddings is too stale, broadcast the up-to-date results among GPUs, otherwise other GPUs reuse the cached historical results from current root GPU; (4) Model Computation: compute the model either based on the latest or cached stale results; (5) Results Cache: update cache accordingly.

3.1 Overview

In this work, we propose Sancus, an adaptive staleness-aware communication-avoiding decentralized GNN system. Fundamentally, Sancus is simple yet effective which caches and reuses the stale historical embeddings and skips broadcast accordingly during the decentralized GNN training, based on a general communication-avoiding matrix blocking algorithm for parallel computing.

We provide the overview of Sancus in Figure 4. Primarily, there are five steps: (1) data loading, (2) staleness bound checking, (3) embedding broadcasting, (4) GNN model computing, and (5) results caching. Here, we briefly clarify these steps: (1) to begin with, the whole sparse adjacency matrix of the full graph and the dense embedding matrix are split into matrix blocks, then loaded to individual worker. Each worker keeps its own replica of the full model; (2) on each GPU, before broadcasting the last computing results, we check whether the staleness of historical embeddings are within proposed bounds. If the staleness is within bounds, the embedding broadcast is skipped and the cached historical embeddings are reused for this iteration's model computing; (3) otherwise, if the staleness exceeds the limit, the latest results are broadcast to all workers and updated in cache; (4) thus either latest embeddings or cached historical embeddings are loaded to the GNN model to compute; (5) finally, updated embeddings are dispatched to next iteration's staleness check before the broadcast.

3.2 Staleness-Aware Communication-Avoiding Decentralized Training

First, we present the comprehensive staleness-aware communication avoiding decentralized full-graph GNN Training in Algorithm 1 and elaborate on its details. There are three keys: **(1)** Worker state

flag F(i) is equipped to indicate the worker state. The state is recorded as either ACTIVE or STALE to support the Skip-Broadcast operation in Section 3.4; **(2)** cache is utilized to store the historical embeddings from other workers that can be repeatedly utilized for future iterations to avoid communication; **(3)** bounded embedding staleness is tolerated to manage system staleness, where each worker may use embeddings from different iterations.

Algorithm 1 The decentralized stale parallel GNN training algorithm based on arbitrary general block row decomposition preprocessing strategy with a foward pass procedure to compute $\mathbf{Z}$ in Equation (2) and $\mathbf{H}$ in Equation (3), a backpropagation procedure to compute the gradients δ in Equation (4) and $\nabla_{\mathbf{W}}\mathcal{L}$ in Equation (5), and the final weight update in Equation (6). The matrices $\hat{\mathbf{A}}$ and $\mathbf{H}$ are distributed on a $p \times p$ process grid, where each process $P(i)$ receives N/p consecutive block rows.

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; Sparse adjacency matrix $\hat{\mathbf{A}}$; Dense feature matrix $\mathbf{H}^{(0)}$; Dense weight matrix $\mathbf{W}$; **Output**: Node embedding matrix $\mathbf{H}^{(L)}$;

1: **Preprocessing**: Block row partition;
2: **for all** process $P(i)$ **in parallel do**
3: **procedure** FORWARD PASS
4: **for** $\ell = 1, \cdots, L$ **do**
5: **for** $j = 1$ **to** p **do**
6: **if** F(j) == ACTIVE **then** ▷ Worker state flag
7: BROADCAST($\mathbf{H}_j^{(\ell-1)}$)
8: CACHE($\mathbf{H}_j^{(\ell-1)}$) ▷ Cache latest $\mathbf{H}_j^{(\ell-1)}$ from $P(j)$
9: ▷ Compute intermediates with latest $\mathbf{H}_j^{(\ell-1)}$
10: $\mathbf{T}_i^{(\ell-1)} \leftarrow \mathbf{T}_i^{(\ell-1)} + \hat{\mathbf{A}}_{ij}\mathbf{H}_j^{(\ell-1)}$
11: **else** \\ Skip-Broadcast
12: ▷ Compute intermediates with historical $\tilde{\mathbf{H}}_j^{(\ell-1)}$
13: $\mathbf{T}_i^{(\ell-1)} \leftarrow \mathbf{T}_i^{(\ell-1)} + \hat{\mathbf{A}}_{ij}\tilde{\mathbf{H}}_j^{(\ell-1)}$
14: $\mathbf{Z}_i^{(\ell)} \leftarrow \mathbf{T}_i^{(\ell-1)}\mathbf{W}^{(\ell-1)}$ ▷ Input to activation
15: $\mathbf{H}_i^{(\ell)} \leftarrow \sigma\left(\mathbf{Z}_i^{(\ell)}\right)$ ▷ Update the embeddings
16: F(i) $\leftarrow$ STALE($\mathbf{H}_i^{(\ell)}$) ▷ Update state of worker i
17: **procedure** BACKWARD PASS
18: **for** $\ell = L-1, \cdots, 0$ **do**
19: $\delta_i^{(\ell)} \leftarrow$ GRADIENT_CLIP($\delta_i^{(\ell)}$)
20: BROADCAST($\delta_i^{(\ell)}$) and Update weights $\mathbf{W}^{\ell}$ by gradients

For the preprocessing, SANCUS supports any partitioning algorithms that split the graph and feature matrices into matrix blocks, such as the classical METIS [18] adopted in most existing distributed systems including DistDGL [43] and AliGraph [45], or efficient random partitioning. As shown in Figure 5, SANCUS treats GNN processing purely as sequential matrix multiplication operations to avoid intensive neighbor fetching during GNN aggregation. To start with, the sparse adjacency matrix $\hat{\mathbf{A}}$ and the dense embedding matrix $\mathbf{H}$ are distributed to each processes $P(i)$ where $i \in [1, p]$ on workers. To further illustrate, recall that N denotes the node number and F denotes the feature embedding length, then the $(N \times N)$ matrix $\hat{\mathbf{A}}$ is computed with p row partitions and p column partitions while the $(N \times F)$ matrix $\mathbf{H}$ is computed with p row partitions as shown in Figure 5. The $(F \times F)$ dense weight matrix $\mathbf{W}$, however, is fully replicated throughout every process $P(i)$. Additionally, we define the intermediate results $\mathbf{T}_i^{(\ell)}$ of the matrix multiplication $\hat{\mathbf{A}}\mathbf{H}^{(\ell)}$ as $\mathbf{T}_i^{(\ell)} = \sum_{j=1}^{p} \hat{\mathbf{A}}_{ij}\mathbf{H}_j^{(\ell)}$ for each process $P(i)$.

For each distributed process $P(i)$ in parallel, SANCUS proceeds the forward pass and backpropagation with the help of collective operations such as ring-based pipelined Broadcast and AllReduce. At the beginning, the staleness of the intermediate embedding results $\mathbf{H}_j^{(\ell-1)}$ of process j is checked in Line 6 before broadcasting to other workers. If the process state is ACTIVE, then $\mathbf{H}_j^{(\ell-1)}$ is sent to all workers in Line 7 from the root rank and copied to all ranks via a One-to-All Broadcast sequentially and cached accordingly in Line 8. Otherwise if the process state is STALE, SANCUS performs Skip-Broadcast to swap out the process j from the communication topology but leave it in the broadcast graph so that the worker j can still receive updates. The process j stops broadcasting out $\mathbf{H}_j^{(\ell-1)}$ for this round, so all other workers repeatedly use their cached version of stale embeddings $\tilde{\mathbf{H}}_j^{(\ell-1)}$. Thus, either the up-to-date results from the last epoch are used in Line 10 or the cached stale results from earlier epochs are automatically repeated for local computation in Line 13. Next, the intermediate results $\mathbf{T}_i^{(\ell-1)}$ are used to compute the embeddings $\mathbf{H}_i^{(\ell)}$ for each process $P(i)$. After computing the latest embeddings $\mathbf{H}_i^{(\ell)}$ for process i, SANCUS checks whether $\mathbf{H}_i^{(\ell)}$ is within the staleness bound so that the worker state flag F(i) remains STALE to keep Skip-Broadcast $\mathbf{H}_i^{(\ell)}$ or becomes ACTIVE to send out $\mathbf{H}_i^{(\ell)}$ in Line 16. Note that for staleness checking, SANCUS either inspects staleness defined in Definition 2, Definition 3, or Definition 4 in Line 16, respectively.

In the backward pass, gradients δ_j are broadcast similarly. To reduce communication of gradients sending, gradient clipping is performed locally in Line 19, also as a regularizer of historical embeddings. Conventionally, gradients are clipped whenever the L2-norm $||\cdot||_2$ exceeds a threshold th. With decentralized P workers, denote the local threshold as th_i. Assume *i.i.d.* gradients on each worker with variance σ^2, then the sum of gradients on all workers has variance $P\sigma^2$. Hence, $\mathbb{E}||\delta_i||_2 \approx \sigma$ and $\mathbb{E}||\delta||_2 \approx P^{\frac{1}{2}}\sigma$. It follows that the local gradient threshold is scaled by $P^{\frac{1}{2}}$, that is, $th_i = P^{\frac{1}{2}}th$. Finally to update the model, an AllReduce operation is performed to combine gradients from all workers and send them to all ranks.

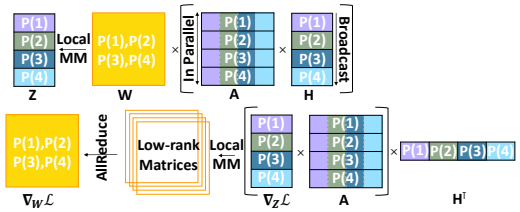


Figure 5: Communication-Avoiding Data-Parallel GNN training, from a sequential matrix multiplication processing perspective.

Since the GNN processing is treated purely as sequential matrix multiplication operations as shown in Figure 5, matrix blocks are directly moved among decentralized workers. Thus, SANCUS avoids the irregular and complex request-send communication to fetch neighbors in vertex-centric distributed GNNs. To further avoid communication, we define historical embeddings in Section 3.3 so that SANCUS can cache and reuse historical embeddings.

3.3 Historical Embeddings

With P decentralized workers, the embedding matrix $\mathbf{H}$ split by rows is denoted as $\mathbf{H}_i$ where $i \in [1, p]$ and is distributed to each $P(i)$ process. To compute the embedding $\mathbf{H}^{(\ell)}$ in a general GNN in Equation (1), we need to combine the matrix block $\mathbf{H}_i$ on each GPU. Inspired by historical embeddings $\tilde{\mathbf{h}}^{(\ell)}$ [5, 12], we generalize the idea of historical embeddings as stale intermediate embedding results computed by other workers in distributed GNNs. Thus, the embedding matrix $\mathbf{H}^{(\ell)}$ in Equation (2) consists of two parts – the latest embedding submatrices from active workers which just broadcast the results and the historical embedding submatrices from stale workers whose embedding variation is small enough to

be neglected. The historical embeddings are stored in the cache on each GPU, only preserving the fresh ones.

Let [] denotes the vertical concatenation of matrix blocks. The processor state ACTIVE and STALE denote whether the processor $P(i)$ is active to broadcast the latest result of embedding submatrix $\mathbf{H}_i^{(\ell)}$ or stale so the history $\tilde{\mathbf{H}}_i^{(\ell)}$ is used:

$$\begin{aligned}\mathbf{H}^{(\ell)} &= \sigma\left(\left[\mathbf{H}_i^{(\ell-1)}\right]_{i=1}^{P}, \hat{\mathbf{A}}; \mathbf{W}^{(\ell-1)}\right) \\ &\approx \sigma\left(\left[\mathbf{H}_{i:P(i)\Leftarrow\texttt{ACTIVE}}^{(\ell-1)} \mid \tilde{\mathbf{H}}_{i:P(i)\Leftarrow\texttt{STALE}}^{(\ell-1)}\right]_{i=1}^{P}, \hat{\mathbf{A}}; \mathbf{W}^{(\ell-1)}\right).\end{aligned} \tag{7}$$

3.4 Skip-Broadcast

With the decentralized scheme, the question is how we can adjust the communication operation such as one-to-all broadcast to support historical embeddings with bounded staleness. Since most implementations of such decentralized scheme [34] is based on bulk synchronism, it is challenging to directly enforce historical embeddings. Thus, we propose a communication primitive that is efficient to implement and requires no centralized parameter servers. Particularly, a Skip-Broadcast scheme is designed, allowing seamless reshaping of the communication topology during training. To realize Skip-Broadcast, Sancus keeps the state flag `Flag(i)` on each worker i to indicate the corresponding worker status for the embeddings $\mathbf{H_i}$ computed on that worker, where $i \in [1, p]$. Specifically, `Flag(i)==ACTIVE` means worker i needs to broadcast its latest version of embeddings $\mathbf{H_i}$. During the broadcast, the latest embeddings $\mathbf{H_i}$ should be sent to all other workers and cached there respectively. If `Flag(i)` turns to STALE, Sancus can Skip-Broadcast $\mathbf{H}_i$ and let other workers utilize their cached stale embeddings.

Take Figure 6 as an example, GPU 2 is notified with `Flag(2)` with STALE state so that other workers rely on their cached stale version of the historical embeddings $\mathbf{H_2}$. Then the ring-based communication topology is reshaped seamlessly to skip GPU 2 and connect its neighbors directly. To receive updated embeddings and gradients from other workers, it should be pointed out that GPU 2 is still preserved in the graph. Therefore, the Skip-Broadcast is performed in replacement of the original broadcast operation whenever the portion of embeddings computed by the corresponding worker is within the bounded staleness. By bypassing the STALE worker to broadcast its stale embeddings, Sancus further reduces the communication overhead. The stale flag `Flag(i)` is checked in every iteration and updated if needed to help reshape the ring-based communication topology.

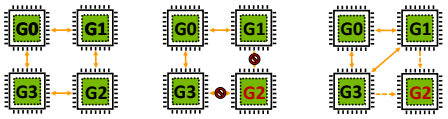


Figure 6: Skip-Broadcast Example, with STALE GPU worker marked red.

3.5 Bounded Embedding Staleness

With skip-broadcast to support embedding staleness, workers can result with embeddings of different iterations from others. To manage system staleness with such mixed-version issue, Sancus supports bounded embedding staleness. Though bounded gradient staleness is deeply investigated [7, 17, 38] in traditional distributed ML for stochastic gradient descent (SGD), its main purpose is to help SGD converge, mitigating negative effects from stale gradients. However, we actively utilize stale embeddings to avoid communication. By introducing a set of novel bounded embedding staleness metrics ϵ, we can control the errors caused by stale embeddings.

3.5.1 *Bounded Staleness Definition Variants.*

We firstly provide the definitions of three staleness of historical embeddings, including one measured by the variation gap of embeddings. The first epoch-fixed embedding staleness of one local update on each processor $P(i)$ is formalized as follows:

Definition 2 (Epoch-Fixed Embedding Staleness). *For all processors in the decentralized GNNs, let $\tilde{e}$ and e denote the epoch number of the intermediate stale embeddings and the current epoch respectively, the maximum number ϵ_E of stale epochs that can be tolerated is defined as the fixed-epoch embedding staleness: $|\tilde{e} - e| \leq \epsilon_E$, where the intermediate embeddings during model computation are only broadcast after every ϵ_E epochs.*

However, all workers can still be regarded as working in a fully synchronous fashion. Thus, we propose two more flexible metrics for the decentralized scheme, so workers may rely on stale embeddings of adaptive iterations from others. Thus, the system can better manage its staleness where workers work at different speed.

Definition 3 (Epoch-Adaptive Embedding Staleness). *For each processor $P(i)$ in the decentralized GNNs, let the maximum epoch gap ϵ_A of embeddings between $P(i)$ and all its in-coming neighbor processors be the epoch-adaptive embedding staleness. $P(i)$ must broadcast its latest results to others after receiving stale embeddings from all its in-comoing neighbors at most ϵ_A epochs ago.*

Definition 4 (Epoch-Adaptive Variation-Gap Embedding Staleness). *For each processor $P(i)$ in the decentralized GNNs, let the maximum variation gap ϵ_H in the values of the stale embeddings that can be tolerated be defined as the epoch-adaptive variation-gap embedding staleness: $\|\tilde{\mathbf{H}}_i^{(\ell)} - \mathbf{H}_i^{(\ell)}\| \leq \epsilon_H$, where the intermediate embeddings are adaptively broadcast whenever the embedding variation gap exceeds ϵ_H regardless of the number of epochs skipped.*

Since all above metrics are defined locally on each worker, we need no centralized or global monitor to break the decentralized scheme. Particularly, one can easily adapt the general definitions above to any specific distributed GNN systems as the metrics to study how the stale results affect the distributed training.

3.5.2 *Bounded Staleness Check Procedure.*

Now, we introduce the procedure to check whether the cached historical embeddings exceed the staleness bound based on definitions in Section 3.5.1, to manage system staleness caused by the mixed-version problem. One crucial distinction from traditional distributed ML is that the bounded staleness is enforced on the intermediate embeddings $\mathbf{H}_i$ instead of gradients, with a new perspective to avoid unnecessary communication in distributed GNNs by taking the initiative to utilize stale embeddings. However, we need to control the errors caused by using stale embeddings. To allow bounded embedding staleness in a decentralized setting, it is natural to design a light-weighted local state tracker on each worker for efficient bounded embedding staleness check in Sancus.

From the database community, we adapt the idea of version control [19] but in a decentralized approach to monitor the staleness of the system. We use `Ver` to denote the current training epoch number. Then on each worker i, upon the arrival of each latest $\mathbf{H}_j$ from the worker j, we stamp $\mathbf{H}_j$ with version $\texttt{Ver}_i(j)$, i.e., the epoch

number where $\mathbf{H}_j$ is computed, correspondingly. We keep track of the version number for all the $\mathbf{H}_j$ where $j \in [1, \cdots, p]$.

Algorithm 2 Embedding Staleness Check based on Definition 2.

Input: Current epoch number Ver; Cached embedding version Ver(j)
1: **procedure** STALE( )
2: **if** Ver − Ver(j) > ϵ_E **then** ▷ Bounded staleness exceeded
3: F(i) ← ACTIVE ▷ Set flag ACTIVE to indicate too-stale state
4: **else** F(i) ← STALE ▷ Staleness within tolerance

Firstly, we introduce the procedure to check the epoch-fixed embedding staleness in Definition 2. The first modification to the decentralized stale parallel Algorithm1 is in Line 8: from CACHE($\mathbf{H}_j^{(\ell-1)}$) to CACHE($\mathbf{H}_j^{(\ell-1)}$, Ver(j)). The version (epoch number) of latest broadcast $\mathbf{H}_j^{(\ell-1)}$ is stamped with Ver(j). After a forward and backward pass are finished, a new epoch is proceeded. The algorithm to check the epoch-fixed embedding staleness is shown in Algorithm 2. Notice that the subscript i for Ver$_i(j)$ is omitted since this basic staleness bound is unified for all workers. An $\epsilon_E = 1$ example in Figure 7(a) shows the embeddings skip-broadcast every other epoch.

Algorithm 3 Embedding Staleness Check based on Definition 3.

Input: Current epoch number Ver; Cached embedding version Ver$_i(j)$
1: **procedure** STALE( )
2: **for all** process $P(i)$ **in parallel do**
3: **for** $j = 1$ **to** p **do**
4: ▷ Worker i exceeds staleness bound on cached $\tilde{\mathbf{H}}_j^{\ell-1}$
5: **if** Ver − Ver$_i(j)$ > ϵ_A **then**
6: F(j) ← ACTIVE ▷ Indicate too-stale state for $\tilde{\mathbf{H}}_j^{\ell-1}$
7: **EndProcedure** for $P(j)$

Secondly, for the epoch-adaptive embedding staleness in Definition 3, the modification to Algorithm1 is also in Line 8: from CACHE($\mathbf{H}_j^{(\ell-1)}$) to CACHE($\mathbf{H}_j^{(\ell-1)}$, Ver$_i(j)$). The latest broadcast $\mathbf{H}_j^{(\ell-1)}$ is stamped with Ver$_i(j)$ for each $\mathbf{H}_j^{(\ell-1)}$ on processor i. Algorithm 3 depicts the staleness checking procedure. If the embeddings $\tilde{\mathbf{H}}_j^{\ell-1}$ of any other worker j cached on worker i is from ϵ_A epochs ago (Line 5), the worker i is running too fast for the worker j and $\tilde{\mathbf{H}}_j$ is too stale, so F(j) is turned to ACTIVE to send out $\mathbf{H}_j$ soon. This ensures that the worker only broadcasts latest results proceeded in the new epoch when it receives updates from other workers at most ϵ_A epochs ago. An example with $\epsilon_A = 1$ is shown in Figure 7(b). In Epoch 9, the results $\mathbf{H}_{2\_7}$ of worker 2 become too stale since $9 - 7 = 2 > \epsilon_A = 1$, worker 2 becomes ACTIVE and broadcasts updated results to all others as shown in Epoch 10. Then worker 1 becomes ACTIVE because others are using $\mathbf{H}_{1\_8}$ which is too stale. Similarly, work 3 and 4 are designated as ACTIVE in Epoch 11.

Algorithm 4 Embedding Staleness Check based on Definition 4.

Input: Current $\mathbf{H}_i^\ell$ computed in $P(i)$; Cached embeddings $\tilde{\mathbf{H}}_i^\ell$
1: **procedure** STALE( )
2: **if** $||\mathbf{H}_i^\ell - \tilde{\mathbf{H}}_i^\ell|| > \epsilon_H$ **then** ▷ Bounded staleness exceeded
3: F(i) ← ACTIVE ▷ Set flag ACTIVE to indicate too-stale state
4: **else** F(i) ← STALE ▷ Staleness within tolerance

Lastly, for the epoch-adaptive variation-gap embedding staleness in Definition 4, the staleness check is purely based on the embedding variation locally on each worker. The checking procedure is elaborated in Algorithm 4. If the embedding variation of $\mathbf{H}_i^\ell$ is within the bound, latest results need no broadcasting, and other workers can use the stale historical embedding $\tilde{\mathbf{H}}_i^\ell$. Otherwise, the embedding variation becomes too large, then the latest $\mathbf{H}_i^\ell$ needs broadcasting and caching. It is noticed that no version tracker is required here. As Figure 7(c) shows, since the staleness is only based on the embedding variation gap, workers may become STALE after an adaptive number of epochs.

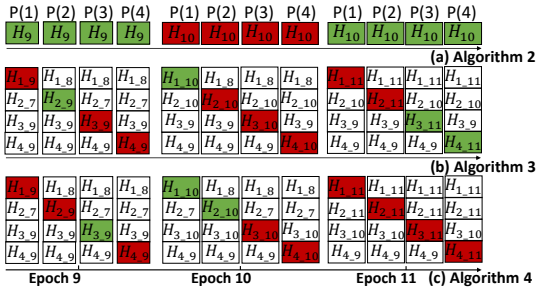


Figure 7: Example training based on each bounded embedding staleness: worker i is denoted by $P(i)$ where $i \in [1, 4]$ and H_{i_e} denotes the results obtained from process i in epoch e; ACTIVE worker is colored green while STALE worker is colored red. For Algorithm 3 and 4, the staleness bound $\epsilon_E = \epsilon_A = 1$. For Algorithm 5, a toy example is given.

4 THEORETICAL RESULTS

In this section, we present the theoretical results. Firstly, we bound the communication cost based on the $\alpha - \beta$ communication cost analyzing model in Section 4.1. Then, we bound the approximation errors of the embeddings and gradients and guarantee the convergence with SANCUS in Section 4.2.

4.1 Communication Cost Analysis

The reduced communication cost relates to the staleness tolerance defined in Section 3.5. Naturally, the communication cost depends on the scale of the tolerance: the larger the tolerance, the more the communication overhead can be saved. To derive the bound for the communication cost with staleness, we first deduce the communication cost without staleness based on the analysis by Tripathy *et al.*[34] and Chan *et al.*[4], using the conventional communication analyzing model $\alpha - \beta$ [4, 34]. Specifically, in this analyzing model, α is the latency of constant time units despite the message size and β is the bandwidth which denotes the corresponding time units taken per word for making to the target worker. Note that the lower bound for the communication cost of one single F-word broadcast message to p processes is $O(\alpha + F\beta)$ [34], where F is the embedding vector length. Also, the lower bound for the communication cost of all-reduce COST(ar) in Equation (5) is $O\left(\log p\ \alpha + 2\frac{p-1}{p}F\beta\right)$ [4].

Thus, if without tolerating staleness, the lower bound on the communication cost for each process in Equation (2) in the forward pass is $(p-1)\left(\alpha + \frac{N}{p}F\beta\right)$ to broadcast the intermediate results $\mathbf{H}_j^{(\ell)}$ to other $(p-1)$ processes, while in the backpropagation it is $(p-1)\left(\alpha + \frac{N}{p}F\beta\right)$ for Equation (4) and $\left(\log p\ \alpha + 2\frac{p-1}{p}F^2\beta\right)$ for Equation (5) which requires AllReduce on $(F \times F)$-matrices.

Hence, the total communication COST($comm$) has a lower bound:

$$O\left(L\left((2(p-1) + \log p)\,\alpha + 2\frac{p-1}{p}F(N+F)\,\beta\right)\right). \tag{8}$$

Now assume that e_T is the total epoch number and e_R denotes the number of epochs performing skip-broadcast and re-utilizing historical embeddings, we can derive that the lower bound of the communication cost COST($comm^\star$) for the proposed stale parallel algorithms. As shown in the following, the cost is composed of two parts: **1)** for the normal epochs using latest results, the communication cost is the same as the COST($comm$) in Equation (8); **2)** for the remaining epochs using historical embeddings, the communication cost of broadcast is bypassed, leaving only the cost of AllReduce

in Equation (5) on low-rank matrices COST(ar). Thus, the bound for the communication cost is deduced as:

$$\begin{aligned}\text{COST}(comm^{\star}) &= (e_T - e_R)\,\text{COST}(comm) + e_R L\ \text{COST}(ar)\\ &= L\,\{[2(p-1)(e_T - e_R) + e_T \log p]\,\alpha\\ &\qquad + 2\,(p-1)/p\ F\,[N(e_T - e_R) + e_T F]\,\beta\}\,.\end{aligned}$$

Therefore, we obtain a tighter bound though the relative communication saving may still be obscure. It is left to be established by the empirical studies in Section 5.3.1 that our framework can save up to 74% communication cost.

4.2 Convergence Analysis

4.2.1 Proof Roadmap. In this section, on the basis of bounded embedding staleness, we provide theoretical guarantees of SANCUS: the approximation error bounds on the intermediate embedding results and gradients, as well as its ensured convergence. We show that the benefits of caching and repeatedly utilizing historical information to reduce communication come at approximation errors that can be bounded. Specifically, we deduce the convergence guarantee with the following procedures in Section 4.2.2:

- **Proposition 1** lays out necessary and basic inequality operations required in this theoretical analysis;
- **Lemma 1** states that with bounded staleness on the embeddings, the approximations of the intermediate matrix results are close to the exact ones in the training process;
- **Lemma 2** further indicates the approximations of gradients in the training process are close to the exact ones;
- **Theorem 1** concludes that the changes of the weights during training are slow enough for the gradients to be asymptotically unbiased, thus guarantees the convergence.

We only present the theoretical analysis on the stricter Definition 4 due to space limit. The corresponding theoretical results of the basic fixed-epoch staleness can be derived by the stricter one via setting $\epsilon_H = \max \Delta_{\|\mathbf{H}\|} \times \epsilon_E$ where $\max \Delta_{\|\mathbf{H}\|}$ denotes the maximum value variation of $\mathbf{H}$ after any epoch.

4.2.2 Details. We derive Theorem 1 based on the proof [5, Theorem 2]. Nonetheless, our proof is distinct from the proof [5] in the following aspects, based on the assumption that the activation $\sigma(\cdot)$ and the gradients $\nabla\mathcal{L}$ are ρ-Lipschitz continuous in view of their possibly high non-linearity: **1)** we perform full-batch GNN training without sampling; **2)** we train in a decentralized parallel fashion; **3)** we adaptively reuse the historical embeddings, where the number of stale epochs is automatically determined rather than just use stale activations from one epoch away in most works.

First of all, we extract the proposition [5, Proposition B] below as it is needed in our proof:

Proposition 1. *Denote* $\|\mathbf{A}\|_\infty = \max_{i,j} |\mathbf{A}(i,j)|$ *and* $col(\mathbf{A})$ *as the column number of matrix* $\mathbf{A}$*, we have* $\|\mathbf{A}+\mathbf{B}\|_\infty \le \|\mathbf{A}\|_\infty + \|\mathbf{B}\|_\infty$, $\|\mathbf{A}\odot\mathbf{B}\|_\infty \le \|\mathbf{A}\|_\infty \|\mathbf{B}\|_\infty$, *and* $\|\mathbf{AB}\|_\infty \le col(\mathbf{A})\|\mathbf{A}\|_\infty \|\mathbf{B}\|_\infty$.

The proof [5, Appendix C] is omitted. We further denote C as the maximum number of columns that exists in our proof:

$$\begin{aligned}C := \max\{&col(\hat{\mathbf{A}}), col(\mathbf{H}^{(0)}), \cdots,\\ &col(\mathbf{H}^{(L)}), col(\mathbf{W}^{(0)}), \cdots, col(\mathbf{W}^{(L)})\}.\end{aligned}$$

Next, we derive that with bounded embedding staleness, the approximation error of the intermediate results is bounded in decentralized training. Thus, we have theoretical grounds for skip-broadcasting and embedding reuse without sacrificing much.

Lemma 1. *For any* N/p *input activations* $\tilde{\mathbf{H}}_i^{(\ell)}$ *in each process* $P(i)$ *where* $i \in \{1,\ldots,p\}$*, given that* ***1)*** *the activation function* $\sigma(\cdot)$ *is* ρ*-Lipschitz continuous;* ***2)*** *the matrices* $\hat{\mathbf{A}}_i, \tilde{\mathbf{H}}_i, \mathbf{H}_i$*, and* $\mathbf{W}_i$ *are bounded by some constant* B*;* ***3)*** *the historical embeddings* $\tilde{\mathbf{H}}_i^{(\ell)}$ *are close to the exact embeddings* $\mathbf{H}_i^{(\ell)}$ *with the staleness bound* ϵ_H *where* $\|\tilde{\mathbf{H}}_i^{(\ell)} - \mathbf{H}_i^{(\ell)}\|_\infty \le \epsilon_H$*: then the approximation error of the intermediate outputs* $\tilde{\mathbf{T}}_i$ *and* $\tilde{\mathbf{Z}}_i$ *are also bounded by some constants:* ***(1)*** $\|\tilde{\mathbf{T}}_i^{(\ell)} - \mathbf{T}_i^{(\ell)}\|_\infty \le pCB\epsilon_H$*;* ***(2)*** $\|\tilde{\mathbf{Z}}_i^{(\ell)} - \mathbf{Z}_i^{(\ell)}\|_\infty \le C^2B^2\epsilon_H$*, where* $\ell \in [1, L]$ *denotes the* ℓ*-th layer.*

PROOF. In our decentralized GNN training, $\tilde{\mathbf{H}}_i$ is either composed of the up-to-date embeddings $\mathbf{H}_i$ or the historical embeddings $\tilde{\mathbf{H}}_i = \mathbf{H}_s$ from stale epochs $s < i$ where $\|\mathbf{H}_i - \mathbf{H}_s\| \le \epsilon_H$.

By Proposition 1, we have

$$\begin{aligned}\|\tilde{\mathbf{T}}_i^{(\ell)} - \mathbf{T}_i^{(\ell)}\|_\infty &= \|\sum_{j=1}^{p} \hat{\mathbf{A}}_{ij}\tilde{\mathbf{H}}_j^{(\ell)} - \sum_{j=1}^{p} \hat{\mathbf{A}}_{ij}\mathbf{H}_j^{(\ell)}\|_\infty\\ &\le p \max_{j=1}^{p} \|\hat{\mathbf{A}}_{ij}\left(\tilde{\mathbf{H}}_j^{(\ell)} - \mathbf{H}_j^{(\ell)}\right)\|_\infty \le pCB\epsilon_H,\end{aligned}$$

and also

$$\begin{aligned}\|\tilde{\mathbf{Z}}_i^{(\ell)} - \mathbf{Z}_i^{(\ell)}\|_\infty &= \|\hat{\mathbf{A}}_i\tilde{\mathbf{H}}_i^{(\ell-1)}\mathbf{W}_i^{(\ell-1)} - \hat{\mathbf{A}}_i\mathbf{H}_i^{(\ell-1)}\mathbf{W}_i^{(\ell-1)}\|_\infty\\ &\le C^2\|\hat{\mathbf{A}}_i\|_\infty\|\tilde{\mathbf{H}}_i^{(\ell-1)} - \mathbf{H}_i^{(\ell-1)}\|_\infty\|\mathbf{W}_i^{(\ell-1)}\|_\infty \le C^2B^2\epsilon_H.\end{aligned}$$

□

Then, we show that the approximate gradients $\tilde{\delta}^{(\ell)} = \nabla_{\tilde{\mathbf{Z}}^{(\ell)}}\tilde{\mathcal{L}}$ are close to the exact gradients $\delta_i^{(\ell)} = \nabla_{\mathbf{Z}_i^{(\ell)}}\mathcal{L}$.

Lemma 2. *For each process* $P(i)$ *where* $i \in [1,p]$*, given that* ***1)*** *the activation function* $\sigma(\cdot)$ *and the gradient* $\nabla\mathcal{L}$ *are* ρ*-Lipschitz continuous;* ***2)*** *the matrices* $\hat{\mathbf{A}}_i, \mathbf{W}_i, \delta^{(\ell)}$ *and* $\sigma'(\mathbf{Z}_i)$ *are bounded by some constant* B*: then the approximation error of the gradients* $\nabla_{\tilde{\mathbf{Z}}_i^{(\ell)}}\tilde{\mathcal{L}}$ *and* $\nabla_{\mathbf{W}_i^{(\ell)}}\tilde{\mathcal{L}}$ *are also bounded by some constants:* ***(1)*** $\|\nabla_{\tilde{\mathbf{Z}}_i^{(\ell)}}\tilde{\mathcal{L}} - \nabla_{\mathbf{Z}_i^{(\ell)}}\mathcal{L}\|_\infty \le K$*;* ***(2)*** $\|\nabla_{\mathbf{W}_i^{(\ell)}}\tilde{\mathcal{L}} - \nabla_{\mathbf{W}_i^{(\ell)}}\mathcal{L}\|_\infty \le K$*; where* $\ell \in \{1,\cdots,L\}$ *denotes the* ℓ*-th GNN layer and* K *depends on* ρ, C, B*, and* ϵ_H*.*

PROOF. By the ρ-Lipschitz continuity of $\sigma(\cdot)$ and $\nabla_{\mathbf{Z}^{(\ell)}}\mathcal{L}$, Equation (4), Proposition 1 and Lemma 1, we prove the approximation error bound of $\nabla_{\tilde{\mathbf{Z}}^{(\ell)}}\tilde{\mathcal{L}}$ by induction.

Base case. For the final layer L, by the ρ-Lipschitz continuity of $\nabla_{\mathbf{Z}^{(\ell)}}\mathcal{L}$ and Lemma 1, we have

$$\|\nabla_{\tilde{\mathbf{Z}}_i^{(L)}}\tilde{\mathcal{L}} - \nabla_{\tilde{\mathbf{Z}}_i^{(L)}}\mathcal{L}\|_\infty \le \rho\|\tilde{\mathbf{Z}}_i^{(L)} - \mathbf{Z}_i^{(L)}\|_\infty \le \rho C^2B^2\epsilon_H.$$

So the statement holds for $\ell = L$, where $K_{(L)} = \rho C^2B^2\epsilon_H$.

Induction hypothesis. Assume that $\|\nabla_{\tilde{\mathbf{Z}}_i^{(\ell')}}\tilde{\mathcal{L}} - \nabla_{\tilde{\mathbf{Z}}_i^{(\ell')}}\mathcal{L}\|_\infty \le K_{(\ell')}$ holds for all $\ell' > \ell$. Then for layer ℓ, by Equation (4), Proposition 1, and Lemma 1, we have

$$\begin{aligned}\|\nabla_{\tilde{\mathbf{Z}}_i^{(\ell)}}\tilde{\mathcal{L}} - \nabla_{\tilde{\mathbf{Z}}_i^{(\ell)}}\mathcal{L}\|_\infty &= \|\tilde{\delta}_i^{(\ell+1)}\hat{\mathbf{A}}_i\left(\mathbf{W}_i^{(\ell)}\right)^{\intercal} \odot \sigma'\left(\tilde{\mathbf{Z}}_i^{(\ell)}\right)\\ &\qquad - \delta_i^{(\ell+1)}\hat{\mathbf{A}}_i\left(\mathbf{W}_i^{(\ell)}\right)^{\intercal} \odot \sigma'\left(\mathbf{Z}_i^{(\ell)}\right)\|_\infty\\ &\le C^2\{\|\tilde{\delta}_i^{(\ell+1)}\|_\infty\|\hat{\mathbf{A}}_i\|_\infty\|\left(\mathbf{W}_i^{(\ell)}\right)^{\intercal}\|_\infty\|\sigma'\left(\tilde{\mathbf{Z}}_i^{(\ell)}\right) - \sigma'\left(\mathbf{Z}_i^{(\ell)}\right)\|_\infty\\ &\quad + \|\tilde{\delta}_i^{(\ell+1)} - \delta_i^{(\ell+1)}\|_\infty\|\hat{\mathbf{A}}_i\|_\infty\|\left(\mathbf{W}_i^{(\ell)}\right)^{\intercal}\|_\infty\|\sigma'\left(\mathbf{Z}_i^{(\ell)}\right)\|_\infty\}\\ &\le C^2\left(B^3\rho(C^2B^2\epsilon_H) + K_{(\ell+1)}B^3\right) = C^2B^3\left(K_{(L)} + K_{(\ell+1)}\right).\end{aligned}$$

Setting $K_{(\ell)} = C^2B^3\left(K_{(L)} + K_{(\ell+1)}\right)$, we find the inequality also holds for layer ℓ, which completes the inductive step.

Conclusion. The above concludes the proof by induction.

The bound of the approximation error of $\nabla_{\mathbf{W}_i^{(\ell)}}\tilde{\mathcal{L}}$ can be obtained in a similar fashion, which is omitted due to space limitations. □

Finally, we conclude the convergence guarantee of proposed decentralized GNN training with bounded embedding staleness.

Theorem 1. *With the GNN model of L layers, given the local minimizer* $\mathbf{W}^\star$*, the initial weights* $\mathbf{W}_{(1)}$*, and the staleness bound* ϵ_H*, suppose that* ***1)*** *the activation function* $\sigma(\cdot)$ *and the gradient* $\nabla\mathcal{L}$ *are* ρ*-Lipschitz continuous;* ***2)*** *for the matrices* $\hat{\mathbf{A}}$, $\mathbf{H}$, *and* $\mathbf{W}$*, all the gradients of the loss function with respect to the weight matrix* $\|\nabla_{\mathbf{W}}\tilde{\mathcal{L}}\|_\infty$, $\|\nabla_{\mathbf{W}}\mathcal{L}\|_\infty$ *and* $\|\nabla\mathcal{L}(\mathbf{W})\|_\infty$ *are bounded by some constant* $G > 0$*;* ***3)*** *the loss* $\mathcal{L}(\mathbf{W})$ *is* ρ*-smooth: then there exists a constant* $K > 0$ *such that* $\forall N > L\epsilon_H$*, if the distributed GNN is trained in parallel correspondingly with bounded staleness for* $R \leq N$ *iterations, where* $R \in [1, \cdots, N]$ *is chosen uniformly and the learning rate* $\eta = \min\{\frac{1}{\rho}, \frac{1}{\sqrt{N}}\}$*, we have*

$$\mathbb{E}_R\|\nabla\mathcal{L}(\mathbf{W}_{(R)})\|_F^2 \leq 2\frac{\mathcal{L}(\mathbf{W}_{(1)}) - \mathcal{L}(\mathbf{W}^\star) + \frac{\rho K}{2}}{\sqrt{N}}, \tag{9}$$

where the learning rate $\eta = \min\{\frac{1}{\rho}, \frac{1}{\sqrt{N}}\}$.

Proof. For convenience, let $\Delta_{(i)} = \nabla_{\mathbf{W}_{(i)}}\tilde{\mathcal{L}} - \nabla\mathcal{L}(\mathbf{W}_{(i)})$, by Lemma 2 and the ρ-smoothness of the $\mathcal{L}(\mathbf{W})$, we derive $\mathcal{L}(\mathbf{W}_{(i+1)})$

$$\begin{aligned}
&\leq \mathcal{L}(\mathbf{W}_{(i)}) + \langle\nabla\mathcal{L}(\mathbf{W}_{(i)}), \mathbf{W}_{(i+1)} - \mathbf{W}_{(i)}\rangle + \frac{\rho}{2}\eta^2\|\nabla_{\mathbf{W}_{(i)}}\tilde{\mathcal{L}}\|_F^2 \\
&= \mathcal{L}(\mathbf{W}_{(i)}) - \eta\langle\nabla\mathcal{L}(\mathbf{W}_{(i)}), \nabla_{\mathbf{W}_{(i)}}\tilde{\mathcal{L}}\rangle + \frac{\rho}{2}\eta^2\|\nabla_{\mathbf{W}_{(i)}}\tilde{\mathcal{L}}\|_F^2 \\
&= \mathcal{L}(\mathbf{W}_{(i)}) - \eta\langle\nabla\mathcal{L}(\mathbf{W}_{(i)}), \Delta_{(i)}\rangle - \eta\|\nabla\mathcal{L}(\mathbf{W}_{(i)})\|_F^2 \\
&+ \frac{\rho}{2}\eta^2\left(\|\Delta_{(i)}\|_F^2 + \|\nabla\mathcal{L}(\mathbf{W}_{(i)})\|_F^2 + 2\langle\Delta_{(i)}, \nabla\mathcal{L}(\mathbf{W}_{(i)})\rangle\right) \\
&\leq \mathcal{L}(\mathbf{W}_{(i)}) - (\eta - \frac{\rho}{2}\eta^2)\|\nabla\mathcal{L}(\mathbf{W}_{(i)})\|_F^2 + \frac{\rho}{2}\eta^2\|\Delta_{(i)}\|_F^2.
\end{aligned}$$

By Lemma 2, $\|\Delta_{(i)}\|_F^2 \leq \|\nabla_{\mathbf{W}_{(i)}}\tilde{\mathcal{L}}\|_\infty + \|\nabla\mathcal{L}(\mathbf{W}_{(i)})\|_\infty \leq 2G^2 \leq K$. Thus,

$$\mathcal{L}(\mathbf{W}_{(i+1)}) \leq \mathcal{L}(\mathbf{W}_{(i)}) - (\eta - \frac{\rho}{2}\eta^2)\|\nabla\mathcal{L}(\mathbf{W}_{(i)})\|_F^2 + \frac{\rho}{2}\eta^2 K. \tag{10}$$

For all i, we sum up Equation (10) and rearrange the terms:

$$(\eta - \frac{\rho}{2}\eta^2)\sum_{i=1}^{N}\|\nabla\mathcal{L}(\mathbf{W}_{(i)})\|_F^2 \leq \mathcal{L}(\mathbf{W}_{(1)}) - \mathcal{L}(\mathbf{W}^\star) + \frac{\rho}{2}\eta^2 KN. \tag{11}$$

Recall $\eta = \min\{\frac{1}{\rho}, \frac{1}{\sqrt{N}}\}$, we divide both sides of Equation (11) by $N(\eta - \frac{\rho}{2}\eta^2)$, then we obtain $\mathbb{E}_R\|\nabla\mathcal{L}(\mathbf{W}_{(R)})\|_F^2$:

$$\begin{aligned}
&= \frac{1}{N}\sum_{i=1}^{N}\|\nabla\mathcal{L}(\mathbf{W}_{(i)})\|_F^2 \leq 2\frac{\mathcal{L}(\mathbf{W}_{(1)}) - \mathcal{L}(\mathbf{W}^\star) + \frac{\rho}{2}\eta^2 KN}{N\eta(2-\rho\eta)} \\
&\leq 2\frac{\mathcal{L}(\mathbf{W}_{(1)}) - \mathcal{L}(\mathbf{W}^\star)}{N\eta} + \rho\eta K \leq 2\frac{\mathcal{L}(\mathbf{W}_{(1)}) - \mathcal{L}(\mathbf{W}^\star) + \frac{\rho K}{2}}{\sqrt{N}}.
\end{aligned}$$

In particular, $\mathbb{E}_R\|\nabla\mathcal{L}(\mathbf{W}_{(R)})\|_F^2 \to 0$ when $N \to \infty$. The above concludes that the convergence is guaranteed. □

5 EXPERIMENTS

We empirically evaluate the proposed framework Sancus, targeting at answering the following major questions:

- Can it effectively avoid communication under different setups? In comparison to SOTA works? (Section 5.3.1 and 5.3.2)
- Can it avoid communication while maintaining the accuracy with staleness-awareness? (Section 5.3.3)
- What is the distribution of the skipped epochs in Sancus? Is it adaptive to manage staleness? (Section 5.3.4)
- How is the memory footprint with the proposed historical embedding caching? (Section 5.3.5)

5.1 Datasets

We evaluate Sancus on five commonly-used [2] large-scale benchmark datasets [15, 39], listed in Table 2. The task on Flickr and Reddit is single-class node classification, while on Amazon, ogbn-products, and ogbn-papers100M is multi-class classifications. Specifically, Flickr models the relations between images uploaded with common properties. Reddit dataset consists of posts and user comments to predict the topical communities that the posts belong to. On Amazon and ogbn-product datasets with node representing product and edge representing products purchased by one customer, we need to categorize the product nodes with multiple labels. Ogbn-papers100M is a citation graph to predict subject areas of papers. All datasets follow the "fixed-partition" splits [15, 39].

Table 2: Summary of the graph data statistics used in our experiments to evaluate the proposed framework ("m": multi-class classfication)

Dataset	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert F\rvert$	# Class	Byte size
Flickr	89,250	899,756	500	7	529 MB
Reddit	232,965	11,606,919	602	41	3.53 GB
Amazon	1,598,960	132,169,734	200	107 (m)	2.34 GB
ogbn-products	2,449,029	61,859,140	100	47	1.38 GB
ogbn-papers100M	111,059,956	1,615,685,872	200	172	56.2 GB

5.2 Implementation and Setups

We implement Sancus on top of a general PyTorch [31] implementation of the classical parallel algorithms [14] adapted to distributed GNN training [34]. Fundamentally, Sancus is different, since it avoids communication adaptively via the skip-broadcast of cached historical embeddings based on proposed bounded embedding staleness. We implement bounded staleness ϵ_E in Definition 2 with Algorithm 1 and 2 as a baseline **SCS-E** that automatically skip-broadcasts and reuses the cached stale results until every ϵ_E epochs. Then, we implement ϵ_A in Definition 3 with Algorithm 1 and 3 as **SCS-A**, and ϵ_H in Definition 4 with Algorithm 1 and 4 as **SCS-H**, that skip-broadcast cached historical embeddings for an adaptive number of epochs. Additionally, we implement **SkipG** [29] on bounded gradient staleness from traditional distributed training to prove bounded embedding staleness is more superior.

Particularly, Sancus supports the following in decentralized training: cache for historical embeddings; version controller and bounded staleness checker; Skip-Broadcast operator. At each ℓ-th layer $\ell \in [1, L]$, the staleness checker examines the gap between the current epoch number and the version of cached results. Or it compares the latest embeddings $\mathbf{H}_i^{\ell-1}$ to the historical embeddings to check if the staleness bound $\epsilon_E/\epsilon_A/\epsilon_H$ is exceeded. The cache of historical embeddings is updated correspondingly if the results turn too stale, where we only store the newly received embeddings. Following the convention [5, 8, 12], warm-up epochs without skipping any broadcasting are set up to protect the performance, since the variation in results may be large in the early stage of learning. Notably, Sancus is orthogonal to any system that supports arbitrary partitioning of matrices. Sancus is easy to deploy and extend due to the wide adoption of PyTorch in communities.

Our experiments are performed on four different GPU configurations: ① eight RTX 2080 Ti connected by PCIe 3.0 × 16; ② two servers connected by 10Gbps Ethernet - each has four RTX 2080 Ti via PCIe 3.0; ③ four A100 40GB via NVLink; ④ four V100 32GB via NVLink. We evaluate Sancus with ① on Flickr, Reddit, Amazon, and ogbn-products, and ①②③ on ogbn-products. For the largest ogbn-papers100M, ③ is used, while ④, as a commonly used configuration, is used for the overall comparison with other SOTA systems [3, 16, 24, 33, 34]. We adopt the GCN model [20] of 3 or 4 layers with variations on the hidden feature size (i.e., 16 for Flickr and Reddit, 256 for Amazon, 128 for ogbn-products, and {16, 32, 64} for

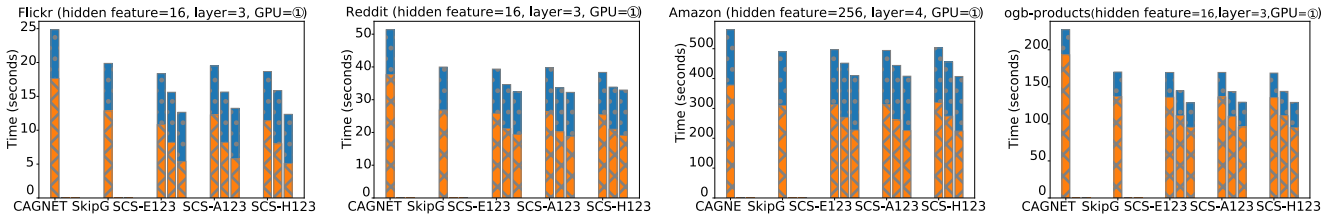
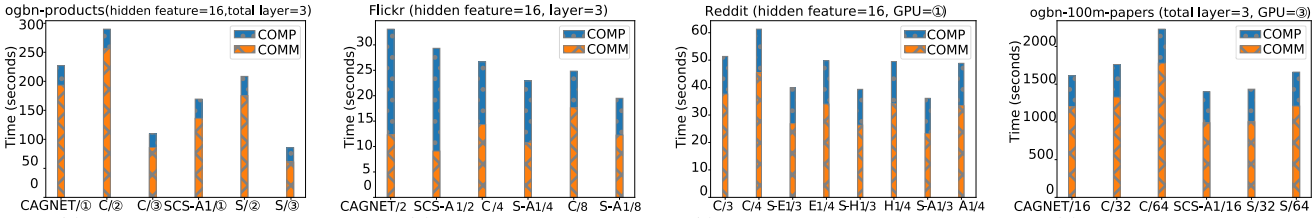


Figure 8: **Communication-avoiding performance using all 8 GPUs on Flickr/Reddit/Amazon/ogbn-products datasets. In each subplot, x-axis denotes the methods compared: CAGNET [34], SkipG [29], SCS-E1/E2/E3 with $\epsilon_E = \{1, 2, 3\}$, SCS-A1/A2/A3 with $\epsilon_A = \{1, 2, 3\}$, SCS-H1/H2/H3 with $\epsilon_H = \{0.01, 0.02, 0.03\}$; y-xis denotes the time proportion during training, where blue bar denotes model computation cost and orange bar denotes communication cost.**



(a) GPU Configurations. **(b) Number of GPUs.** **(c) Number of Total GNN Layers.** **(d) Size of Hidden Feature.**

Figure 9: **The performance with system architecture variants (GPU configurations/number of GPUs/GNN layer number/GNN hidden feature size) compared to CAGNET on ogbn-products/Flickr/Reddit/ogbn-papers100M datasets, respectively. In each subplot, y-axis denotes the time proportion during training, where blue bar denotes model computation cost and orange bar denotes communication cost. For the x-axis, we denote the method/GPU configurations in Fig. 9a, method/number of GPUs in Fig. 9b, method/layer number in Fig. 9c, and method/hidden feature size in Fig. 9d.**

ogb-papers100M). Moreover, we use 2/4/8 GPUs in ① for Flickr as well as ①②③ for ogbn-products. We also implement GAT [35] to verify the generality of Sancus on different GNN models.

The total training epoch number e_T is 300/300/400/500/200 for Flickr, Reddit, Amazon, ogbn-products, and ogbn-papers100M. Following the convention to set warm-up epochs in staleness-tolerant training [5, 8, 12], we set 50 warm-up epochs for all datasets. For the staleness bound, we choose ϵ_E in $[1, 7]$, ϵ_A in $[1, 5]$ and ϵ_H in $[0.01, 0.05]$ to control the variation amplitude. Without loss of generality, we show the main outcomes with $\epsilon_E = \{1, 5\}$, $\epsilon_A = \{1, 5\}$ and $\epsilon_H = \{0.01, 0.05\}$ in Section 5.3.1 to 5.3.5, accompanied by an analysis of the influence on staleness-aware training in Section 5.3.7.

5.3 Results

We answer the questions at the beginning of Section 5, specifically:

5.3.1 ***Sancus is Effective.*** First, we demonstrate the effectiveness on communication reduction of Sancus on different benchmark datasets. Note that the datasets are with increasing order of magnitude to show the scalability. In Figure 8, we show results with accuracy loss within 0.01. Compared to the SOTA CAGNET and bounded-gradient method SkipG, all our system variants further avoid communication by at least 35% to 74% with bounded embedding staleness. Though SkipG via traditional gradient stalenss also outperforms CAGNET since it skip-broadcasts gradients during the training, it avoids communication sightlessly at the price of accuracy. Compared to Sancus variants, SkipG suffers from serious accuracy deterioration up to 5% as shown in Figure 10, while Sancus can preserve the GNN performance on all datasets. Besides, as Figure 8 shows, Sancus variants can still outperform SkipG for 29% to 63% more communication reduction on different datasets.

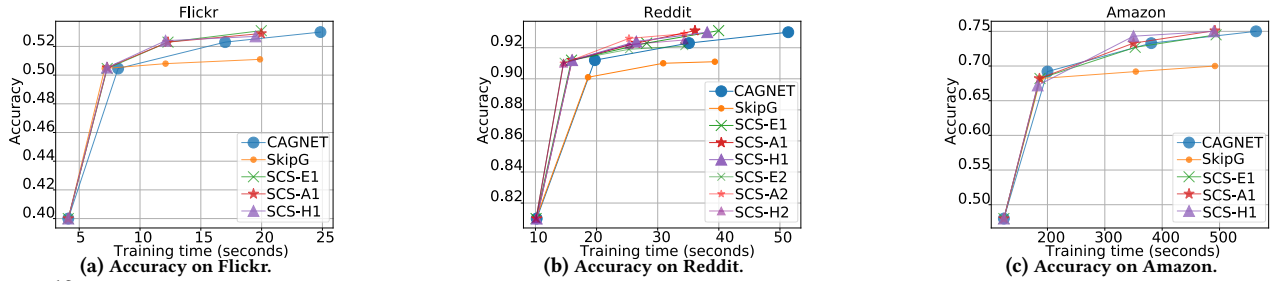
To highlight the best results, we avoid about 74% of communication on Flickr with SCS-H3 with $\epsilon_H = 0.03$, 48% on Reddit with SCS-A3, and 50% on ogbn-products with SCS-H3. It should be noticed that though SCS-E shows competitive communication avoiding performance, its accuracy degrades drastically with the increasing ϵ_E as Figure 10b, 11a, and 12b shows, compared to SCS-A/H. Also, the 35% communication avoding on Amazon and ogbn-papers100M seems not impressive since Amazon is trained on a deeper 4-layer GNN with the largest hidden feature size 256 to guarantee the accuracy and ogbn-papers100M is the largest dataset, both with intrinsically larger communication volume. Considering the better performance preserving, the more adaptive SCS-A/H are in fact more robust. Above analysis verifies the effectiveness of bounded staleness and the adaptive communication reduction.

Without loss of generality, we demonstrate the generality of Sancus with its worst-behaved baselines SCS-E1/A1/H1 of the least communication avoiding, since our goal is to study the influence of different system configurations. We evaluate the varaints of system architectures in terms of GPU configurations, number of GPUs, number of total GNN layers, and hidden feature size on ogbn-products, Flickr, Reddit, and ogbn-papers100M in Figure 9. To further show the training time improvement, we present the epoch time breakdown with computation and communication in Table 3.

Table 3: **Summary of detailed time breakdown (second) measured in seconds in one epoch with SCS-A/E/H compared to CAGNET, on Reddit dataset.**

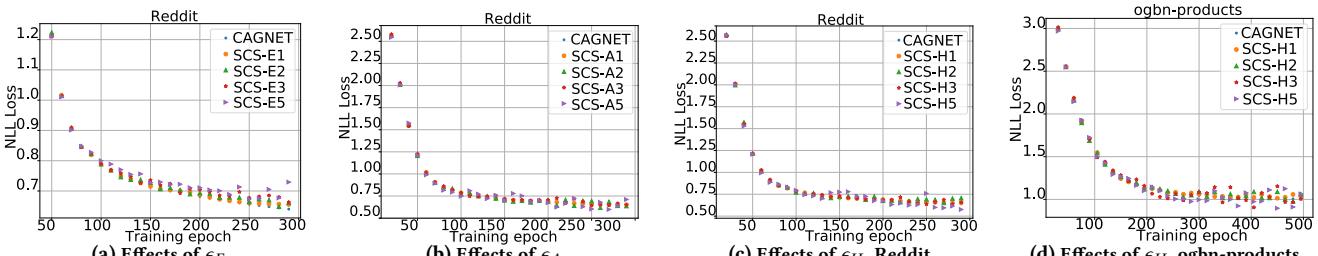
Config	Operation	CAGNET	SCS-A1	SCS-E	SCS-H
① **8*1**	**compute**	0.365	0.359	0.359	0.343
p2p: no	**communicate**	1	0.687	0.697	0.675
② **4*2**	**compute**	0.093	0.093	0.092	0.090
p2p: no	**communicate**	1	0.717	0.714	0.698
③ **4*1**	**compute**	0.437	0.431	0.431	0.425
p2p: yes	**communicate**	1	0.703	0.708	0.692

Specifically, in Figure 9a, the communication time under different GPU configurations varies greatly. However, Sancus can achieve consistent communication avoiding in all the configurations ①②③ – with both single-machine multiple-GPUs and multi-server environment regardless of the specific GPU tpyes, though GPU configurations have a great impact on the communication itself. Further, we examine the detailed time breakdown under ①②③ in Table 3.



(a) Accuracy on Flickr. (b) Accuracy on Reddit. (c) Accuracy on Amazon.

Figure 10: The accuracy results using all eight GPUs on Flickr/Reddict/Amazon datasets, respectively. In each subplot, x-axis denotes the methods compared: CAGNET [34], SkipG [29], SCS-E1/E2 denotes $\epsilon_E = \{1, 2\}$, SCS-A1/A2 $\epsilon_A = \{1, 2\}$, SCS-H1/H2 $\epsilon_H = \{0.01, 0.02\}$; y-xis denotes the time proportion.



(a) Effects of ϵ_E (b) Effects of ϵ_A (c) Effects of ϵ_H, Reddit (d) Effects of ϵ_H, ogbn-products

Figure 11: The scatter plots of NLL loss with epochs performing broadcast on Reddit and ogbn-products: SCS-E1/E2/E3/E5 denotes $\epsilon_E = \{1, 2, 3, 5\}$; SCS-A1/A2/A3/A5 denotes $\epsilon_A = \{1, 2, 3, 5\}$; SCS-H1/H2/H3/H5 denotes $\epsilon_H = \{0.01, 0.02, 0.03, 0.05\}$

SCS-A1 can achieve the least overhead consistently in all the settings, including the multi-server one. As Figure 9b shows, with increasing GPU number, the total cost compared to CAGNET are reduced continually. Though the communication cost increase with the increasing GPU number, with SCS-H, we can reduce the communication cost using all 8 GPUs to get close to the communication cost of CAGNET using 2 GPUs, together with 67% reduction on the computation cost. Importantly, the communication proportion we avoid increases with the number of GPU used, which is hard for the centralized PS architectures [23] to achieve. For the influence of total GNN layers in Fig. 9c, we show that communication is avoided for different GNN depths. Further, when increasing hidden feature size, Fig. 9d shows that compared to CAGNET, the communication increasing of our epoch-adaptive strategy is much less and slower. Above concludes the robustness and generality of our framework on different system variants.

5.3.2 ***SANCUS Outperforms SOTA Systems.*** In Table 4, we give an overall throughput (epochs/second) comparison of SANCUS (SCS-A1) to five related SOTA distributed systems over their commonly-used Reddit dataset, including: CAGNET [34], RoC [16], Dorylus [33], sampling-based PaGraph [24], and DGCL [3]. RoC proposes a sophisticated memory management method while PaGraph exploits static caching of nodes with higher degree in the GPU memory, both leveraging a nontrivial partitioning algorithm to balance the workload and reduce the cross-device data visits. Dorylus adopts bounded staleness asynchrony for low-cost distributed training with CPU servers, while DGCL reduces communication by finding optimal communication routes in the specific system topology for every node in the entire graph. For all the works, we report their best results that can be compared.

The worst-behaved baseline SCS-A1 with the least communication avoiding, in fact, still outperforms all the related SOTA distributed GNN systems. We can process the fastest 10.3 epochs per second with SCS-A1 with an average 1.86× throughput. As compared to Dorylus which aims at low-cost full-batch distributed GNN training, we are 68.7× faster and 80% cheaper.

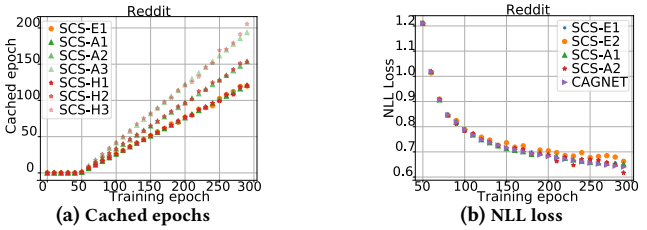
5.3.3 ***SANCUS Converges Fast and Reserves the GNN Performance.*** With repeated usage of historical embeddings to skip-broadcast adaptively, SANCUS avoids communication while preserving the GNN accuracy. In Figure 10 and 12b, all proposed variants converge to a very close ($\leqslant 0.005$), even the same, sometimes the better accuracy results with communication avoiding, compared to CAGNET. Besides, the convergence time to reach satisfying accuracy is much faster. We also show that the traditional bounded gradient method SkipG suffers far more accuracy loss ($\leqslant 0.02$) compared to our skip-broadcast historical embeddings. In general, we draw the conclusion on the extremely close proximity of the model performance between the proposed training framework and the original GNN training, where the empirical results are in consistent with the theoretical results as presented in Section 4.2.

Table 4: Throughput (epochs/second) of GCN on Reddit dataset over different distributed GNN systems. Note: 1) For RoC result, P100 (4.7 TFLOPs) is 67% as fast as V100 (7 TFLOPS); 2) Dorylus costs 0.085/h*860s = $0.2 for 130 epochs, while SCS-A costs 3.06/h*4*13s = $0.04 ; 3) PaGraph is sampling-based.

System	Config	Throughput	Reference
SCS-A	V100*4	1000/97.4 = 10.3	–
CAGNET	V100*4	1/0.11 = 9	Fig 1 [34]
RoC	P100*4	5	Fig 5 [16]
Dorylus	Lambda on CPU*2	130/860 = 0.15	Sec 7.2 Table 4 [33]
PaGraph	1080ti*4	1 ∗ 4.9 = 5	Sec 5.2 5.5; Fig 9 [24]
DGCL	V100*4	∼ (1000/150) = 7	Fig 8(a) [3]

5.3.4 ***SANCUS is Staleness-Aware.*** We show the adaptivity in the number of epochs that SANCUS skips when using cached historical embeddings to manage system staleness. As illustrated in Figure 12a, we plot the corresponding epochs that actually perform broadcasting in the training of the proposed framework. As the control group,

epoch-fixed SCS-E1 in Figure 12a broadcast the latest results in every 2 iterations. Thus its cached epochs – the orange dots, increase regularly. As for epoch-adaptive schemes SCS-A and SCS-H, we find the broadcasts exhibiting irregular patterns, by observing that the interspaces between dots are at varying distances, especially for SCS-H. The explanation is that the staleness in epoch number is adaptively controlled by the staleness tolerance ϵ_H, also shown by the earlier example in Figure 7. In accordance with Figure 8, SCS-E1/A1/H1 cache the least epochs thus avoid the least communication as Figure 8 shows. Compared to the epoch-fixed SCS-E, epoch-adaptive methods provide higher accuracy results (Figure 10). It shows that managing system staleness can lead to better preservation of effectiveness. Notably, though SCS-E3 caches similar epochs with similar communication avoiding in Figure 8, it suffers from the most accuracy loss in Figure 10. Taking all above into account, we conclude that the adaptive strategy with staleness-awareness is more robust and advantageous in communication avoiding with little and even no loss of accuracy, sometimes even better accuracy.



(a) Cached epochs **(b) NLL loss**

Figure 12: The scatter plots with x-axis denoting epochs performing broadcast on Reddit, to show adaptive staleness-awareness. SCS-E1/E3 denotes $\epsilon_E = \{1, 2\}$; SCS-A1/A2/a3 denotes $\epsilon_A = \{1, 2, 3\}$; SCS-H1/H2/H3 denotes $\epsilon_H = \{0.01, 0.02, 0.03\}$.

5.3.5 ***SANCUS is Memory-Efficient.*** Due to the GPU memory constraints, the scalability of such distributed GNN training is instructed by the memory cost on GPUs. During SANCUS training, there are three parts of GPU memory footprints: local data (i.e., embeddings, local adjacency matrix, and full weight matrix), memory for matrix opertation, and cache of historical embeddings. Compared to CAGNET, the only extra memory SANCUS consumes is the cache of historical embeddings. Thus, we show the cache memory footprints in Table 5. Generally, the cache memory cost is a particularly small proportion for modern GPUs on most datasets.

5.3.6 ***SANCUS is Easy to Generalize.*** Besides GCN, We additionally generalize to another distinct full-GNN architecture GAT in Figure 13a and 13b. We simply verify that SANCUS can help avoid communication on GAT while preserving accuracy. Note that GAT does not achieve its best performance because machine learning parameter-tuning is not a focus in our work. Also, we show the performance of 1.5D CATNET too since 1D and 1.5D are reported the best [34]. Though SANCUS can still avoid communication with little harm to accuracy, the improvements on 1.5D is less due to its intrinsically less broadcasts in the algorithm. Thus, SANCUS indeed can be easily generalize to other models, where we stress that if with parameter-tuning, the results can improve.

5.3.7 ***Analysis of the Effects on Bounded Staleness.*** We study how the bounded staleness metrics affect the training in Figure 11, with the effects of ϵ_E in Figure 11a, ϵ_A in Figure 11b and ϵ_H in Figure 11c and 11d, respectively. Particularly, with the increasing value of the staleness bound, the convergence becomes more volatile with larger fluctuation. Though the convergence speed may decrease, the total training time is still less since more broadcasts are skipped as shown in Fig. 10. With a larger value, the avoided communication becomes more as Fig. 8 shows, with negligible accuracy loss in Fig. 10. Thus, combining the results in Figure 8 and 10, one can adapt ϵ_E, ϵ_A and ϵ_H accordingly to specific applications, where some focuses more on the accuracy (i.e., with smaller bound) while others are more interested in the efficiency (i.e., with larger bound), respectively.

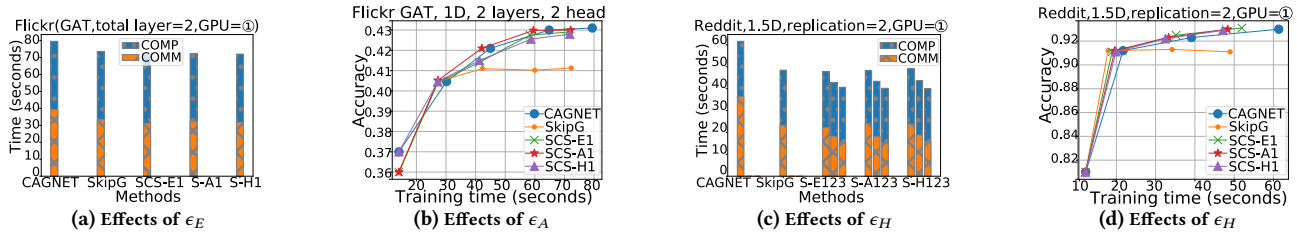
Choice of Staleness Bound. The tradeoff between the runtime and error is difficult to understand due to the entanglement of model convergence and system performance, especially under the distributed settings. Larger bounded staleness may help improve the throughput, but it can pose a negative impact on the accuracy [7, 9, 17, 21, 26]. Moreover, differences in applications, datasets, implementations, and configurations may affect the practical staleness in the GNN training as well. Take Figure 9 as an example, we can have a glimpse of the changes that may happen. Despite different datasets response differently to varying staleness bounds as Figure 11c and 11d show, a satisfying model performance can be achieved for all our experiments with a negligible deceleration in convergence. Though it still remains an open research problem [9, 21], it can be concluded that in our GNN training, the more superior adaptive strategies with $\epsilon_A = \{2, 3\}$ and $\epsilon_H = \{0.02, 0.03\}$ can strike an acceptable balance between the runtime-error tradeoff.

Table 5: Cache Memory Footprints with ①. Ogbn-paper100M is omitted from the table since it only runs on 4 GPUs with 31GB cache size.

Dataset	2 GPUs	4 GPUs	8 GPUs
Flickr	170.2 MB	255.3 MB	297.9 MB
Reddit	0.5 GB	0.8 GB	0.9 GB
Amazon	1.2 GB	1.8 GB	2.1 GB
ogbn-products	0.9 GB	1.4 GB	1.6 GB

6 RELATED WORK

Distributed Graph Neural Networks. The distributed GNNs are still in its infancy [2], with a few prior works on GPU-based systems. Compared to distributed systems for large graph analysis [6, 10, 11, 28, 32], the communication overhead in distributed GNNs are even more challenging, since the intensive data movement among workers to fetch neighbor embeddings is expensive. Currently, most existing systems utilized a centralized architecture. For example, NeuGraph [27] proposes the GNN training framework on a multi-GPU single machine with METIS [18] partitioning and specialized optimization in scheduling and pipelining. However, it is not released for public access. RoC [16] dynamically partitions the graph with an online regression method and proposes a sophisticated memory management method among workers, at the cost of complex workflow. PaGraph [24] exploits static caching of nodes with higher degree in the GPU memory, leveraging a nontrivial partitioning algorithm to balance the workload and reduce the data movement in cross-device visits. G^3 [25] leverages parallel graph optimizations to improve graph operations in GPU systems, and Zhou *et al.* [44] utilize channel pruning to accelerate GNN inference, while Grain [42] focuses on GNN data selection via social influence maximization and RDD [41] uses unlabeled data. Yet,



(a) Effects of ϵ_E (b) Effects of ϵ_A (c) Effects of ϵ_H (d) Effects of ϵ_H

Figure 13: Generalization to GAT [35] and 1.5D [34].

their evaluation does not focus on distributed training. AliGraph [45] utilizes static cache as well but only supports CPU servers, while AGL [40] uses MapReduce and optimizes both training and inference. To reduce and balance the communication, DistDGL [43] leverages partitioning with load balancing, while Min *et al.* [30] present a GPU-oriented communication reduction via zero-copy access. These specialized systems often come with heavy preprocessing and complex workflow. Moreover, such newly proposed frameworks often pose challenges in their deployment and extension. All of aforementioned distributed GNN systems adopt a centralized design, which may lead to centralized communicaion, high communication overhead, and a single point of failure. Also, it should be noted that only NeuGraph and Roc support full-GNN processing, while all others need sampling. More recently, DGCL [3], a communication library for distributed full-GNN training, tries to reduce the communication by finding optimal communication routes in specific system topology for every node in the entire graph. Regardless of the substantial overhead caused by planning communication for each node before every execution, it still follows the conventional message passing paradigm for vertex-centric computation. Tripathy *et al.* [34] incorporate matrix blocking techniques into a set of parallel algorithms [14], to suit the sparse matrix and dense matrix operations in distributed GNNs. Though being a general implementation of distributed GNNs with high extensibility, their proposed CAGNET still struggles in scalability, owing to the communication bottleneck. Thus, we adapt the powerful parallel algorithm [14, 34] and abstract the GNN processing as sequential matrix multiplication so that its intermediate historical embeddings are cached and re-utilized to reduce the communication overhead further in a system environment with staleness for the first time.

Historical Embeddings and Bounded Staleness. VR-GCN [5] firstly maintains historical embeddings in sampling-based GNNs to reduce the variance and further control the number of nodes to sample, with a thorough theoretical analysis on variance and convergence. Then, MVS-GNN [8] combines this method into one-shot sampling without iterative exploration of nodes in each layer. More recently, GNNAutoScale [12] proposes to use historical embeddings to approximate the missing out-of-mini-batch information for each layer, with theoretical analysis on the model expressiveness. GNNAutoScale aims to fuse their model into distributed training in the future. However, all above models are sampling-based method of stochastic training in single-memory systems.

Meanwhile, bounded staleness is widely studied in traditional distributed machine learning to support bounded asynchrony, especially for the centralized PS architecture [7, 17, 38]. Only recently, the decentralized scheme begins to draw more attention after its proved superiority to the centralized architecture [23, 26]. Yet, traditional distributed ML systems do not consider the cross-device embedding fetching, which limits their adoption on GNNs. Though one recent work, Dorylus [33], adopts bounded asynchrony with tolerance of stale embeddings in full-GNN training. It targets only at low-cost distributed CPU servers using serverless threads by AWS Lambdas [1]. Instead, we explore historical embeddings with a set of novel bounded embedding staleness metrics in decentralized full-GNN training accelerated by GPUs.

7 CONCLUSION

In this paper, we present Sancus, the first staleness-aware communication avoiding scheme for decentralized GNN systems that can adaptively avoid communication by caching historical embeddings and manage system staleness by bounded embedding staleness, while preserving the GNN model performance. We propose a set of novel bounded embedding staleness metrics: the epoch-fixed staleness, epoch-adaptive staleness, and epoch-adaptive variation-gap staleness. Then, we integrate the historical embedding cache and bounded embedding staleness check into decentralized GNNs to adaptively skip broadcast among GPUs. Specifically, we reuse the cached historical embeddings within the staleness bound. Whenever the staleness exceeds the bound, we broadcast the latest results, and update the cache. Theoretical analysis on the bound of communication costs and approximation errors are presented, while extensive experiments over large-scale benchmark graph datasets are conducted to demonstrate the efficiency and effectiveness of Sancus, as well as the necessity of the adaptive strategy to manage system staleness. We also show that the cache to store histrocial embeddings is memory-efficient by summarizing the memory footprints. One interesting future direction is to explore the optimization on partitioning algorithms to further improve the efficiency.

ACKNOWLEDGMENTS

Yingxia Shao's work is supported by the National Natural Science Foundation of China (Nos. U1936104, 62192784) and CCF-Baidu Open Fund. Yanyan Shen is partially supported by Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102). Lei Chen's work is partially supported by National Key Research and Development Program of China Grant No. 2018AAA0101100, the Hong Kong RGC GRF Project 16202218, CRF Project C6030-18G, C1031-18G, C5026-18G, CRF C2004-21GF, AOE Project AoE/E-603/18, RIF Project R6020-19, Theme-based project TRS T41-603/20R, China NSFC No. 61729201, Guangdong Basic and Applied Basic Research Foundation 2019B151530001, Hong Kong ITC ITF grants ITS/044/18FX and ITS/470/18FX, Microsoft Research Asia Collaborative Research Grant, HKUST-NAVER/LINE AI Lab, Didi-HKUST joint research lab, HKUST-Webank joint research lab grants and HKUST Global Strategic Partnership Fund (2021 SJTU-HKUST).

REFERENCES

[1] 2022. Lambda. Retrieved May 15, 2022 from https://aws.amazon.com/lambda/

[2] Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, and Eduard Alarcón. 2022. Computing Graph Neural Networks: A Survey from Algorithms to Accelerators. *ACM Comput. Surv.* 54, 9 (2022), 191:1–191:38. https://doi.org/10.1145/3477141

[3] Zhenkun Cai, Xiao Yan, Yidi Wu, Kaihao Ma, James Cheng, and Fan Yu. 2021. DGCL: an efficient communication library for distributed GNN training. In *EuroSys '21: Sixteenth European Conference on Computer Systems, Online Event, United Kingdom, April 26-28, 2021*. ACM, 130–144.

[4] Ernie Chan, Marcel Heimlich, Avi Purkayastha, and Robert A. van de Geijn. 2007. Collective communication: theory, practice, and experience. *Concurr. Comput. Pract. Exp.* 19, 13 (2007), 1749–1783.

[5] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic Training of Graph Convolutional Networks with Variance Reduction. In *ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research)*, Vol. 80. PMLR, 941–949. http://proceedings.mlr.press/v80/chen18p.html

[6] Rong Chen, Jiaxin Shi, Yanzhe Chen, Binyu Zang, Haibing Guan, and Haibo Chen. 2018. PowerLyra: Differentiated Graph Computation and Partitioning on Skewed Graphs. *ACM Trans. Parallel Comput.* 5, 3 (2018).

[7] James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Gregory R. Ganger, Garth Gibson, Kimberly Keeton, and Eric P. Xing. [n.d.]. Solving the Straggler Problem with Bounded Staleness. In *HotOS XIV, Santa Ana Pueblo, New Mexico, USA, May 13-15, 2013*. USENIX Association.

[8] Weilin Cong, Rana Forsati, Mahmut T. Kandemir, and Mehrdad Mahdavi. 2020. Minimal Variance Sampling with Provable Guarantees for Fast Training of Graph Neural Networks. In *KDD '20: The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Virtual Event, CA, USA, August 23-27, 2020*. ACM, 1393–1403. https://doi.org/10.1145/3394486.3403192

[9] Wei Dai, Yi Zhou, Nanqing Dong, Hao Zhang, and Eric P. Xing. 2019. Toward Understanding the Impact of Staleness in Distributed Machine Learning. In *ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=BylQV305YQ

[10] Wenfei Fan, Jingbo Xu, Yinghui Wu, Wenyuan Yu, and Jiaxin Jiang. 2017. GRAPE: Parallelizing Sequential Graph Computations. *Proc. VLDB Endow.* 10, 12 (2017), 1889–1892. http://www.vldb.org/pvldb/vol10/p1889-fan.pdf

[11] Wenfei Fan, Jingbo Xu, Yinghui Wu, Wenyuan Yu, Jiaxin Jiang, Zeyu Zheng, Bohan Zhang, Yang Cao, and Chao Tian. 2017. Parallelizing Sequential Graph Computations. In *SIGMOD 2017, Chicago, IL, USA, May 14-19*. ACM.

[12] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Jure Leskovec. 2021. GNNAutoScale: Scalable and Expressive Graph Neural Networks via Historical Embeddings. In *ICML 2021, 18-24 July 2021, Virtual Event (Proceedings of Machine Learning Research)*, Vol. 139. PMLR, 3294–3304. http://proceedings.mlr.press/v139/fey21a.html

[13] Swapnil Gandhi and Anand Padmanabha Iyer. 2021. P3: Distributed Deep Graph Learning at Scale. In *15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2021, July 14-16, 2021*. USENIX Association, 551–568. https://www.usenix.org/conference/osdi21/presentation/gandhi

[14] Amir Gholami, Ariful Azad, Peter H. Jin, Kurt Keutzer, and Aydin Buluç. 2018. Integrated Model, Batch, and Domain Parallelism in Training Neural Networks. In *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*. ACM, 77–86.

[15] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. [n.d.]. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. https://proceedings.neurips.cc/paper/2020/hash/fb60d411a5c5b72b2e7d3527cfc84fd0-Abstract.html

[16] Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. 2020. Improving the Accuracy, Scalability, and Performance of Graph Neural Networks with Roc. In *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*. mlsys.org. https://proceedings.mlsys.org/book/300.pdf

[17] Jiawei Jiang, Bin Cui, Ce Zhang, and Lele Yu. 2017. Heterogeneity-aware Distributed Parameter Servers. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD Conference 2017, Chicago, IL, USA, May 14-19, 2017*. ACM, 463–478. https://doi.org/10.1145/3035918.3035933

[18] George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* 20, 1 (1998).

[19] Won Kim and Frederick H. Lochovsky (Eds.). 1989. *Object-Oriented Concepts, Databases, and Applications*. ACM Press and Addison-Wesley.

[20] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=SJU4ayYgl

[21] Joo Hwan Lee and Hyesoon Kim. 2019. Empirical Investigation of Stale Value Tolerance on Parallel RNN Training. In *ISPASS 2019, Madison, WI, USA, March 24-26, 2019*. IEEE, 153–164. https://doi.org/10.1109/ISPASS.2019.00029

[22] Hao Li, Asim Kadav, Erik Kruus, and Cristian Ungureanu. [n.d.]. MALT: distributed data-parallelism for existing ML applications. In *EuroSys 2015, Bordeaux, France, April 21-24, 2015*. ACM.

[23] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. 2017. Can Decentralized Algorithms Outperform Centralized Algorithms? A Case Study for Decentralized Parallel Stochastic Gradient Descent. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 5330–5340. https://proceedings.neurips.cc/paper/2017/hash/f75526659f31040afeb61cb7133e4e6d-Abstract.html

[24] Zhiqi Lin, Cheng Li, Youshan Miao, Yunxin Liu, and Yinlong Xu. 2020. PaGraph: Scaling GNN training on large graphs via computation-aware caching. In *SoCC '20: ACM Symposium on Cloud Computing, Virtual Event, USA, October 19-21, 2020*. ACM, 401–415. https://doi.org/10.1145/3419111.3421281

[25] Husong Liu, Shengliang Lu, Xinyu Chen, and Bingsheng He. 2020. G3: When Graph Neural Networks Meet Parallel Graph Processing Systems on GPUs. *Proc. VLDB Endow.* 13, 12 (2020). http://www.vldb.org/pvldb/vol13/p2813-liu.pdf

[26] Qinyi Luo, Jiaao He, Youwei Zhuo, and Xuehai Qian. 2020. Prague: High-Performance Heterogeneity-Aware Asynchronous Decentralized Training. In *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*. ACM, 401–416.

[27] Lingxiao Ma, Zhi Yang, Youshan Miao, Jilong Xue, Ming Wu, Lidong Zhou, and Yafei Dai. 2019. NeuGraph: Parallel Deep Neural Network Computation on Large Graphs. In *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*. USENIX Association.

[28] Grzegorz Malewicz, Matthew H. Austern, Aart J. C. Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. 2010. Pregel: a system for large-scale graph processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010*. ACM, 135–146. https://doi.org/10.1145/1807167.1807184

[29] Xupeng Miao, Xiaonan Nie, Yingxia Shao, Zhi Yang, Jiawei Jiang, Lingxiao Ma, and Bin Cui. 2021. Heterogeneity-Aware Distributed Machine Learning Training via Partial Reduce. In *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. ACM, 2262–2270. https://doi.org/10.1145/3448016.3452773

[30] Seungwon Min, Kun Wu, Sitao Huang, Mert Hidayetoglu, Jinjun Xiong, Eiman Ebrahimi, Deming Chen, and Wen-mei W. Hwu. 2021. Large Graph Convolutional Network Training with GPU-Oriented Data Communication Architecture. *Proc. VLDB Endow.* 14, 11 (2021), 2087–2100. https://doi.org/10.14778/3476249.3476264

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 8024–8035.

[32] Mo Sha, Yuchen Li, Bingsheng He, and Kian-Lee Tan. 2017. Accelerating Dynamic Graph Analytics on GPUs. *Proc. VLDB Endow.* 11, 1 (2017). http://www.vldb.org/pvldb/vol11/p107-sha.pdf

[33] John Thorpe, Yifan Qiao, Jonathan Eyolfson, Shen Teng, Guanzhou Hu, Zhihao Jia, Jinliang Wei, Keval Vora, Ravi Netravali, Miryung Kim, and Guoqing Harry Xu. [n.d.]. Dorylus: Affordable, Scalable, and Accurate GNN Training with Distributed CPU Servers and Serverless Threads. In *OSDI 2021, July 14-16, 2021*. USENIX Association, 495–514.

[34] Alok Tripathy, Katherine A. Yelick, and Aydin Buluç. 2020. Reducing communication in graph neural network training. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event, November 9-19, 2020*. IEEE/ACM.

[35] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. [n.d.]. Graph Attention Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=rJXMpikCZ

[36] Zhao Wang, Yijin Guan, Guangyu Sun, Dimin Niu, Yuhao Wang, Hongzhong Zheng, and Yinhe Han. 2020. Gnn-pim: A processing-in-memory architecture for graph neural networks. In *Conference on Advanced Computer Architecture*. Springer, 73–86.

[37] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. 2021. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Networks Learn. Syst.* 32, 1 (2021), 4–24.

[38] Lintao Xian, Bingzhe Li, Jing Liu, Zhongwen Guo, and David H. C. Du. 2021. H-PS: A Heterogeneous-Aware Parameter Server With Distributed Neural Network Training. *IEEE Access* 9 (2021), 44049–44058.

[39] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=BJe8pkHFwS

[40] 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 (2020), 3125–3137. http://www.vldb.org/pvldb/vol13/p3125-zhang.pdf

[41] Wentao Zhang, Xupeng Miao, Yingxia Shao, Jiawei Jiang, Lei Chen, Olivier Ruas, and Bin Cui. 2020. Reliable Data Distillation on Graph Convolutional Network. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 1399–1414. https://doi.org/10.1145/3318464.3389706

[42] Wentao Zhang, Zhi Yang, Yexin Wang, Yu Shen, Yang Li, Liang Wang, and Bin Cui. 2021. Grain: Improving Data Efficiency of Graph Neural Networks via Diversified Influence Maximization. *Proc. VLDB Endow.* 14, 11 (2021), 2473–2482. http://www.vldb.org/pvldb/vol14/p2473-zhang.pdf

[43] Da Zheng, Chao Ma, Minjie Wang, Jinjing Zhou, Qidong Su, Xiang Song, Quan Gan, Zheng Zhang, and George Karypis. 2020. DistDGL: Distributed Graph Neural Network Training for Billion-Scale Graphs. (2020), 36–44. https://doi.org/10.1109/IA351965.2020.00011

[44] Hongkuan Zhou, Ajitesh Srivastava, Hanqing Zeng, Rajgopal Kannan, and Viktor K. Prasanna. 2021. Accelerating Large Scale Real-Time GNN Inference using Channel Pruning. *Proc. VLDB Endow.* 14, 9 (2021), 1597–1605. http://www.vldb.org/pvldb/vol14/p1597-zhou.pdf

[45] 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 (2019), 2094–2105. http://www.vldb.org/pvldb/vol12/p2094-zhu.pdf