HET-GMP: A Graph-based System Approach to Scaling Large Embedding Model Training

Xupeng Miao, Yining Shi, Hailin Zhang, Xin Zhang, Xiaonan Nie, Zhi Yang, Bin Cui*
School of Computer Science & Key Lab of High Confidence Software Technologies (MOE), Peking University
*Institute of Computational Social Science, Peking University (Qingdao).
{xupeng.miao, shiyining, z.hl, seanzhang, xiaonan.nie, yangzhi, bin.cui}@pku.edu.cn

ABSTRACT

Embedding models have been recognized as an effective learning paradigm for high-dimensional data. However, a major embedding model training obstacle is that updating and retrieving the shared large-scale embedding parameters usually dominates the distributed training cycle, leading to significant scalability issues. This paper presents HET-GMP, a distributed system on training embedding models. Uniquely, HET-GMP takes advantage of a graph-based approach to efficiently increase scalability. The key insight guiding our design is the "graph way of thinking". HET-GMP creates a bigraph abstraction to represent the access relationships between data samples and embedding vectors. This enables HET-GMP to embrace graph locality and skewness as new performance opportunities and to exploit graph-based replication/partitioning and bounded-asynchronous synchronization to reduce communication overhead. We evaluate the system on the embedding models for click-through rate (CTR) prediction, which presents the most significant challenge and communication bottleneck due to heavy access concurrency to a huge embedding table. The result shows that HET-GMP supports embedding model training with 10^{11} parameters, achieving a reduction in communication up to 87.5% and an up-to 27.5× speedup over the state-of-the-art baseline systems.

CCS CONCEPTS

• **Information systems** → **Data management systems**; **Computational advertising**; • **Computing methodologies** → **Distributed computing methodologies**.

KEYWORDS

Deep embedding model, Model parallelism, Graph partition

ACM Reference Format:
Xupeng Miao, Yining Shi, Hailin Zhang, Xin Zhang, Xiaonan Nie, Zhi Yang, Bin Cui. 2022. HET-GMP: A Graph-based System Approach to Scaling Large Embedding Model Training. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3514221.3517902

SIGMOD '22, June 12–17, 2022, Philadelphia, PA, USA.

ACM ISBN 978-1-4503-9249-5/22/06…$15.00
https://doi.org/10.1145/3514221.3517902

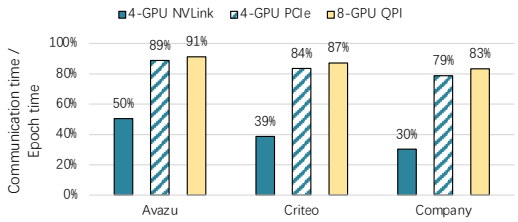


Figure 1: Communication overheads in WDL model training on HugeCTR [7]. Avazu [3], Criteo [2] and Company are different recommendation datasets.

1 INTRODUCTION

Embeddings are often used to handle representation learning problems on high-dimensional data, such as words in a corpus of text, or users and items in recommendation systems. Deep embedding techniques use continuous vectors to represent discrete variables and have large amounts of practical applications, such as click-through-rate (CTR) prediction system [46], graph processing [42] and information extraction [47, 48]. For example, the "wide and deep learning (WDL)" model [13] creates latent vectors from cross-product transformations on categorical features and provides semantic modalities and meaningful representations of these categories in the transformed space. However, with the growing size of deep embedding models and the increasing volume of input data, building a huge embedding model training system is more challenging in regard to both effectiveness and efficiency. For example, the production platform in Facebook has proposed a deep learning recommendation model (DLRM) [36] with trillions of parameters and terabytes in size, which poses a serious scalability challenge.

Modern distributed machine learning (ML) systems (e.g., TensorFlow [8], PyTorch Distributed [26], HET [34], and HugeCTR [7]) typically use parameter server (PS) or AllReduce based approaches to scale-out models. However, these systems face a scalability issue for large embedding models. The greatest inefficiency comes from the sparse reads and updates of the shared embedding parameters through a limited bandwidth link. For example, HugeCTR is a speed-of-light click-through-rate (CTR) model framework that can outperform popular systems such as TensorFlow. Figure 1 shows the distributed training efficiency of HugeCTR on WDL under different inter-GPU connections. We see that up to 90% of training time is spent on fetching and updating embedding parameters, which dominates the training cycle. With the increasing gap between emerging powerful accelerators and the slow growth of network bandwidth, the communication bottleneck would become even more severe.

In this paper, we present HET-GMP, a novel graph-based, data-driven partitioning system for large-scale huge embedding model training. HET-GMP proposes a novel bigraph representation to

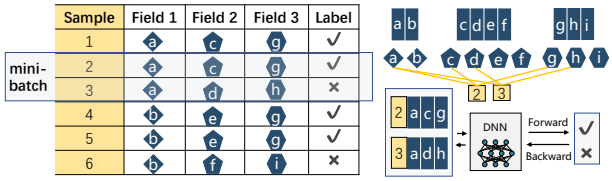


Figure 2: Graph representation of deep CTR model. Sample 2 and 3 formulate a bipartite graph with their embeddings including a, c, g, d, and h. During training, the embedding vectors for each data sample are organized in a row and then feed into the rest of DNN.

manage the input data and embedding parameters. We take a deep CTR model as an example and illustrate the mini-batch based training process with the bigraph representation in Figure 2. Each vertex in the graph represents a data sample or an embedding vector. Each edge connects a sample vertex with an embedding vertex used by the data sample. With the graph representation, we observe two critical characteristics: access *locality* and *skewness*. In particular, a specific embedding is mostly related to only a small subset of data, and the access distribution of embeddings is often highly skewed, which implies two performance opportunities: (1) with a locality-conscious data layout optimization, most embeddings can be stored only on few workers and be updated locally to reduce communication, and (2) replicating hot embeddings at each worker can effectively save network bandwidth (i.e., caching). To leverage such opportunities, HET-GMP introduces two new mechanisms:

Hybrid Graph Partition. To effectively alleviate the communication bottleneck, we provide a hybrid graph partitioning algorithm to find partitions with good locality and workload balance, which combines edge-cut (for evenly distributing vertices) and vertex-cut (for evenly distributing edges of high-degree vertices). Specifically, we first perform 1D edge-cut partitioning to evenly assign data and embedding vertices of a graph to workers to minimize the number of edges spanning workers. We define a new score formula to guide the assignment process and balance the resource requirements among all workers (e.g., the number of embeddings, the communication costs, the computation workloads). To further improve locality, we perform an additional 2D vertex-cut partitioning to evenly assign edges of high-degree embedding nodes across workers by cutting and replicating nodes.

Graph-based Consistency Model. Vertex-cut may cut and replicate embedding nodes, raising the problem of consistency in the presence of writes. To unleash the full power of vertex-cut partitioning, HET-GMP performs a unique form of bounded asynchrony to relax the consistency guarantees. Although asynchronous processing has been widely used in the past, HET-GMP faces a unique technical difficulty that no other systems have dealt with: the update dependencies across embeddings revealed by the underlying graph structures, which has not been investigated in conventional consistency models such as Stale Synchronous Parallel (SSP) [21] without graph views. HET-GMP proposes a novel graph-based bounded asynchrony by introducing two synchronization points where staleness can be tolerated: *intra-embedding synchronization* across the replicas of a specific embedding vertex when reading it, and *inter-embedding synchronization* across those used by a specific data sample vertex when performing computation. We have formally proved the convergence of our asynchronous model.

We summarize our contributions as follows: First, we propose a novel **graph-based system approach** designed to improve the scalability of training huge embedding models. Second, we provide an efficient **hybrid graph partitioning** algorithm that combines edge-cut and vertex-cut for balanced and locality-conscious distributed training (model parallelism). Third, we introduce **graph-based bounded asynchrony** with intra-and-inter embedding synchronizations and prove convergence. Finally, we build the **HET-GMP system** that implements the proposed graph-based system approach. The result shows that HET-GMP can reduce by up to 87.5% the communication related to embeddings and support embedding model training with 10^{11} parameters. HET-GMP achieves $1.64-2.66\times$ end-to-end convergence time speedup and up to $27.5\times$ throughput improvement over the state-of-the-art baseline systems.

2 OBJECTIVE

Let $\xi \in \mathbb{R}^{n\times m}$ be the training dataset, where n is the number of data samples, and m is the number of features. The embedding model will represent each feature with a vector of parameters $\mathbf{x}$. Let d be a small number of latent dimensions, we consider the problem of finding embedding $\mathbf{X} \in \mathbb{R}^{m\times d}$ which minimizes:

$$\min_{\mathbf{X}} \left[F(\mathbf{X}) := \frac{1}{|\xi|} \sum_{i} f(\mathbf{X}; \xi_i) \right] \tag{1}$$

where $f(\cdot)$ is the loss function defined by the model. For example, in deep CTR embedding models, each data sample is represented by a feature vector that consists of several categorical fields. Embeddings are used to learn the low-dimensional representation of the categorical features by minimizing the prediction loss function.

A large number of embedding workloads can be easily modeled using the above problems. They create a low-dimensional vector of "parameters" for each high-dimensional one-hot vector (indicating a feature, word, or node), such as CTR models, ML models like topic modeling, and graph embedding models like knowledge graph embedding. In the following, we focus on CTR embedding models to drive our design, because they are common in modern web companies and present the most significant challenge and communication bottleneck due to heavy access concurrency to the huge embedding table. For example, in knowledge graph embeddings, a data sample only needs to access two embeddings for an edge, whereas a data sample may access tens to hundreds of embeddings in CTR models. Other embedding models where (1) embedding is non-parametric (such as GNNs [24, 32, 35, 49] and BERT [15]), or (2) computation is the bottleneck (e.g., with very deep network architectures for training), are not included in our target workloads.

3 RELATED WORK

In this section, we first briefly introduce the current practice for large embedding model training. Then we explain why the traditional optimization methods (such as parameter allocation, graph partitioning, and relaxed consistency) cannot be directly applied to embedding models. Finally, we introduce the differences between HET-GMP and graph learning systems.

Distributed Training. Distributed training systems have been extensively studied recently to scale up ML for big data and large models. The PS [25] architecture stores a global model and each worker communicates with PS to fetch or update the model. The flexible

communication pattern enables PS to support efficient sparse communications, i.e., only a small subset of the model parameters are accessed by each worker in one iteration. Another architecture is based on collective communication primitives. AllReduce is one of the most representative methods and uses efficient communication techniques for model synchronization, such as Ring-Reduce where the workers are connected with a ring [17, 33]. It has shown superior performance in many DL training workloads but degenerates to inefficient AllGather primitives for sparse communication.

Training large embedding models usually suffers from a communication bottleneck. Parallax [23] considers the difference in the sparsity of model parameters and proposes a hybrid communication architecture that leverages PS and AllReduce to transfer sparse and dense parameters, respectively. Kraken [45] follows the hybrid architecture and optimizes the embedding memory usage. These approaches incur frequent CPU-GPU communication overheads.

To alleviate such costs, HET [34] proposes an embedding-cache-enabled architecture with fine-grained consistency to improve the training scalability. HugeCTR [7] distributes the whole embedding table into the memory of multi-GPUs to accelerate the training of CTR models. Several industrial companies follow the GPU-based model-parallelism design of HugeCTR and have developed their large-scale embedding training systems, such as ZionEX [36] by Facebook and HierPS [50] by Baidu. Although these systems improve the scalability of training embedding models through memory hierarchy (e.g., CPU main memory or SSD), they are still suffering from the CPU-GPUs or GPU-GPU communication bottleneck.

Parameter Allocation. To reduce the network communication when accessing parameters in traditional PS, dynamic parameter allocation [38, 39] has been proposed to utilize the parameter access locality and re-allocate the model parameters during training. However, these studies are limited to some special ML tasks naturally supporting parameter-block-based training such as matrix factorization (MF), tensor factorization, and latent dirichlet allocation. Unfortunately, data samples in DL models might require updating a very large number of different embeddings, making it hard to enforce local access at the coarse granularity of blocks. By contrast, HET-GMP fundamentally improves embedding model training efficiency by exploiting the access locality and skewness characteristics at the fine granularity of embedding, via partition and synchronization using a new graph-based approach.

Graph Partitioning. Many graph computation systems rely on graph partitioning to minimize communication and ensure balanced computation [16, 37]. To exploit locality during computation, both Pregel [1] and GraphLab [31] use edge-cut to accumulate all resources (i.e., messages or replicas) of a vertex in a single machine. To further address the load imbalance issue given skewed distribution in natural graphs, PowerGraph [18] and GraphX [19] introduce vertex-cut, which splits a vertex into multiple replicas across machines. However, this splitting also comes at a cost, including extra replication and synchronization across replicas. PowerLyra [12] and Libra [44] adopt hybrid-cut algorithms to partition different types of vertices according to their degrees. Although these algorithms consider edges but also vertices and general skewness, HET-GMP reaps the full performance potential considering the following unique features of embedding models.

First, existing algorithms for graph processing emphasize partitioning nodes in a one-pass manner, to minimize the pre-processing overhead given small graph computation time. By contrast, the training time of embedding models is much larger, which allows considering more pre-processing capabilities. Therefore, we improve graph partition performance in an iterative manner, which is in essence a different design principle from the traditional graph systems. Second, the commonly used distributed graph processing algorithms assume homogeneous graphs and homogeneous network bandwidth for every pair of workers. But the graphs that model embedding models often naturally have different types of nodes playing distinct roles (e.g., parameter and user nodes). Moreover, embedding models are typically trained on a GPU cluster exhibiting heterogeneous connectivity (e.g., NVLink, PCIe, QPI, and Ethernet), leading to uneven bandwidth between different pairs of workers. To capture these characteristics of embedding models, HET-GMP also adopts new principles to a heterogeneity aware load-balancer design considering both computation and communications.

Consistency Protocols. The bulk synchronous parallel (BSP) model is a default option for popular training frameworks including TensorFlow, which forces strict synchronization barriers. Several relaxed consistency protocols have been proposed to reduce the model synchronization costs of BSP. Asynchronous protocol [29] allows the workers to proceed without coordinating with each other but may incur excessive inconsistency. Bounded-asynchronous models such as SSP [21] provide deterministic convergence guarantees by specifying a bounded staleness among workers. However, HET-GMP faces a unique challenge: the *update dependencies* across embeddings revealed by their co-access relations for a specific data sample, which has not been investigated in conventional consistency protocols and learning systems (e.g., [38, 39]).

By explicitly capturing such dependencies with a graph view, HET-GMP proposes a new design principle of enforcing the bounded staleness at two synchronization points where staleness can be tolerated: (1) replicas of a specific embedding when reading it, and (2) replicas of multiple co-accessed embeddings when used by the same data sample. The intuition is to ensure that if a worker conducts an update on a specific embedding, then the replicas of associated embeddings it gathered from other workers cannot be too obsolete (bounded by s). Otherwise, the embedding quality would be adversely affected by the high staleness of those associated embeddings it depends on.

Graph Learning Systems. ML on graphs has attracted immense attention in recent years. These systems also adopt graph-oriented optimization to improve learning performance. For example, both distributed graph neural network (GNN) systems (e.g., DistDGL [51]) and knowledge graph (KG) training systems (e.g., DGL-KE [52]) partition the graph inputs to improve the scalability. However, the graph partition algorithms used in these systems are similar to those commonly used in traditional distributed graph processing systems. By contrast, HET-GMP provides a new partition design that exploits the new characteristics of embedding models. In addition, the model is not partitioned in GNN training systems (e.g., DistDGL), which do not face the unique challenge of write consistency like us. For KG training systems, our graph-based replication (vertex-cut)

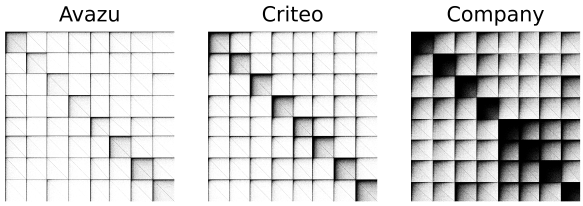


Figure 3: Embedding co-occurrence graph partition results. The graph of each dataset is partitioned into 8 clusters with METIS [22] (8 is only for illustrative purposes, e.g., trained on an 8-GPU server).

and consistency principles (algorithms) could be naturally applied for a larger communication optimization space.

4 OBSERVATION AND OPPORTUNITIES

As illustrated in Figure 2, we represent the interaction of data sample $\xi \in \mathbb{R}^{n \times m}$ and embedding parameters $X \in \mathbb{R}^{m \times d}$ using a graph: each vertex represents an individual embedding $x_i \in \mathbf{X}$ or data sample vertex $\xi_i \in \xi$, with the edges connecting a data sample with the embedding vertices it accessed. Note an embedding is to represent an element (high-dimensional one-hot vector) in a data sample with a low-dimensional vector of parameters. For a data sample (multi-hot vector or sparse vector input), we represent it by the sample vertex in our graph representation (see Figure 2), which connects to multiple embedding vertices and corresponds to a set of such one-hot vectors in the same space. The graph reflects the co-access pattern among different categorical features, e.g., the association rules [9] in data mining. With the graph, we uncover the following characteristics of embedding access patterns:

Locality. We find that the graph representation of deep embedding models exhibits the graph locality property. For illustrative purpose, we transform the data-embedding graph representation into an embedding co-occurrence graph, regarding embeddings as nodes and co-occurrence relations in a data sample as edges. The co-occurrence count of a pair of embeddings that appeared in the same data samples is defined to be the corresponding edge weight. We then perform a clustering method (e.g., METIS[22]) on the co-occurrence graph, and Figure 3 shows the graph clustering results over various datasets, including two public datasets (Avazu and Criteo) and one private dataset. We see that the co-occurrence relations are well clustered into dense diagonal regions. This implies that the training data and parameters can be clustered such that we can store co-occurrence embeddings (e.g., those in dense regions) and corresponding data samples into the same worker to improve embedding access locality. This property motivates us to apply locality-aware graph partition over the data-embedding graph.

Skewness. We notice that some cross-cluster edges exist in the partitioned graph and could be the potential communication bottleneck. This is because the embeddings also follow another important graph property — the embedding nodes have highly skewed power-law degree distributions (i.e., access frequency). This motivates us to locally cache embeddings with large degrees (i.e., high frequency), to further reduce the remote access costs. Introducing replication raises the problem of ensuring consistency in the presence of writes. Fortunately, existing embedding models fall into the category of iterative convergent algorithms which have been shown robust to a bounded amount of inconsistency and still converge correctly [30].

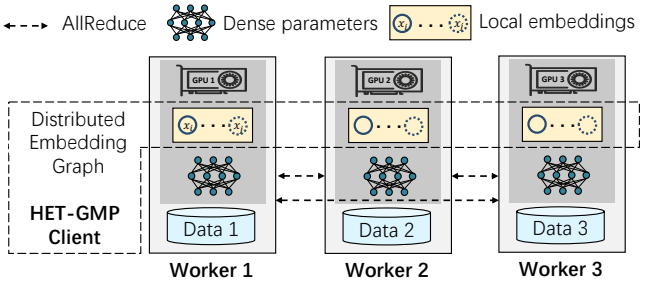


Figure 4: Overview of HET-GMP.

5 HET-GMP DESIGN

In this section, we introduce HET-GMP, a novel large-scale embedding model training system, which utilizes the locality and skewness of the embedding access pattern to improve efficiency and scalability. Our system design is illustrated in Figure 4. We apply the hybrid communication architecture where each worker holds a replica of the dense model parameters and uses All-Reduce for synchronization during training. Since the embedding parameters cause most of the communication costs, we focus on accelerating the communication of the embedding parameters.

We distribute the embedding parameters and the input datasets into different workers. Note that each worker maintains a HET-GMP client, and all the local model parameters are directly stored in the GPU memory. The system design can be treated as a GPU-based model-parallelism approach. But the key novelty here is that HET-GMP organizes the embedding vectors into a graph abstraction rather than randomly partitioned parameter blocks. From this new perspective, we can perform more efficient model-parallelism training by utilizing graph locality and skewness. Specifically, we introduce graph-based partitioning and synchronization considering these graph properties, which makes our system fundamentally different from existing solutions.

5.1 Graph Representation

We propose a bigraph representation to manage the data layout for the deep CTR embedding models with large and sparse embedding tables. As illustrated in Figure 5, there are two types of vertices in the graph $G = (V_x, V_\xi, E)$, including embedding vertices x and sample vertices ξ. In this bigraph model, each sample from the input dataset is denoted by a *sample vertex* and each embedding vector from the embedding table is denoted by an *embedding vertex*. An edge (x_i, ξ_j) between a sample vertex and an embedding vertex indicates that the current sample ξ_j has the corresponding categorical feature x_i.

Through modeling the input features from a graph perspective, the bigraph abstraction can support many existing embedding models (e.g., Wide & Deep [13], Deep & Cross [41], DeepFM [20], xDeepFM [28] and Deep Interest Network [53]). Specifically, given a mini-batch of data samples, the embedding layer of these models performs a lookup operation to access the corresponding adjacent embedding vectors x_i. During the forward stage, these models perform the deep neural networks computation and make predictions on the target objective. After the following backward stage, they compute the gradients of these embeddings used in the forward stage. At last, the embedding gradients Δx_i are applied to the embedding vertices, and the next iteration of training starts.

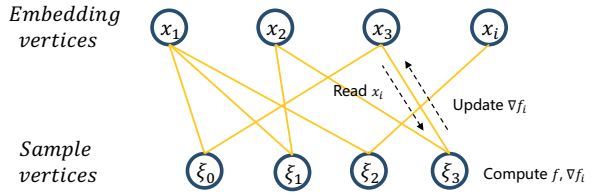


Figure 5: Bigraph abstraction of embedding models.

5.2 Hybrid Graph Partitioning

Given such a bigraph representation, a key problem here is how to partition the graph to reduce embeddings/gradients communication among different workers (i.e., partitions) while achieving optimally balanced workloads. Our algorithm leverages the observations on the embedding model such as the skewed distribution and locality of vertices, their different roles, and imbalanced data sizes of embedding and data vertices to derive a good partition. We design a hybrid iterative graph partitioning framework to improve the distributed training performance of the embedding model. As shown in Algorithm 1, our proposed graph partitioning algorithm consists of two steps in each iteration:

Step 1: Edge-cut Partitioning (1D Partitioning). Considering the graph locality, we first perform an edge-cut partitioning to distribute both embedding and sample vertices across different GPU workers in a balanced manner. First, given the partitions $\hat{G} = \{G_1, G_2, ..., G_N\}$, a vertex v is assigned to partition G_i such that the global score $\delta_g(G_i) \leq \delta_g(G_j), \forall j \in \{1, 2, ..., N\}$. Here, $\delta_g(G_i)$ is defined as:

$$\delta_g(G_i) = \delta_c(G_i) - \delta_b(G_i), \tag{2}$$

where $\delta_c(G_i)$ denotes the inter-GPU communication costs on the i-th GPU, and $\delta_b(G_i)$ is the balance formula.

For a homogeneous communication architecture, $\delta_c(G_i)$ is the same as the edge-cut for vertices in partition G_i:

$$\delta_c(G_i) = \sum_{\forall x \in V_x}^{x \notin G_i} count(x, i), \tag{3}$$

where the function $count(x, i)$ represents the number of times embedding x is used by the data samples in the i-th partition. To reflect network bandwidth unevenness, we propose to capture heterogeneous connections by using weighted edge-cuts. Specifically, we profile the communication speeds for all GPU-GPU pairs and formulate them into a weight matrix. When counting the edge-cuts, we multiply the corresponding weight value from the matrix to the *count* function results, so that the number of cross-partition edges among data graph partitions is gracefully adapted to the uneven bandwidth across workers.

The balance formula $\delta_b(G_i)$ can be interpreted as the marginal cost of adding vertex v to partition G_i, which is used to balance workloads among different partitions. We have

$$\begin{aligned} \delta_b(G_i) &= \alpha\delta_{\xi}(G_i) + \beta\delta_x(G_i) + \gamma\delta_d(G_i), \\ \delta_{\xi}(G_i) &= |G_{i,\xi}| - |G_{\xi}|/N, \\ \delta_x(G_i) &= |G_{i,x}| - |G_x|/N, \end{aligned} \tag{4}$$

where α, β, and γ are hyper-parameters. $\delta_{\xi}(G_i)$ and $\delta_x(G_i)$ are the balance formulas to balance the number of sample and embedding vertices for each partition, respectively. They describes the gap between the number of samples and embedding vertices in the partition (i.e., $|G_{i,\xi}|$ and $|G_{i,x}|$) and the averaged number for all partitions (i.e., $|G_{\xi}|/N$ and $|G_x|/N$). The third term $\delta_d(G_i)$ is used to balance the inter-GPU communications among the partitions:

$$\delta_d(G_i) = \delta_c(G_i) - \sum_{j=1}^{N} \delta_c(G_j)/N, \tag{5}$$

which describes the gap between the unbalanced communication on the i-th GPU $\delta_c(G_i)$ and the average communication of all GPUs.

Algorithm 1: Balanced hybrid graph partitioning

Input: max iterations T, embeddings and samples graph $G = (V, E)$, number of partitions N
Output: Partitions $\hat{G} = G_1, G_2, ..., G_N$.

```
Initialize Ĝ with random graph partitions;
for t ∈ range(T) do
    /* 1D partitioning                                   */
    for v ∈ V_x ∪ V_ξ do
        j ← arg min_{∀j∈{1,2,...,N}} δ_g(G_j);
        Assign vertex v to G_j;
    /* 2D partitioning                                   */
    for i ∈ range(N) do
        while True do
            v ← arg max_{∀v∈V_x, v∉G_i} δ_p(v, G_i);
            if G_i reaches GPU i's memory budget then
                break
            Replicate vertex v to G_i;
```

Step 2: Vertex-Cut Partitioning (2D Partitioning). Vertex-Cut replicates the high-degree embedding vertices to further reduce communications. Since high-degree vertices inevitably access neighbors on most of the machines, we further apply 2D partitioning (i.e., vertex-cut) by allowing edges of a single vertex to be split over multiple workers. However, randomly 2D partitioning might not be efficient. Considering the limited GPU memory, there is a trade-off between the benefit of reduced remote accesses and the overhead of redundant replication. Our key insight here is that highly skewed power-law degree distributions of embedding nodes mitigate this trade-off: a few replicas of popular embedding nodes across workers can effectively enhance data locality.

Based on the insight, we provide a sequential greedy heuristic to replicate embedding nodes on the partition (i.e., worker) that maximizes the expected cross-edges. We define the score formula in E.q. (6). Given the 1D partitions of $P = \{G_1, G_2, ..., G_N\}$, an embedding vertex $x \notin G_i$ has higher priority to be replicated to partition G_i than the other embeddings when $\delta_p(x, G_i) \geq \delta_p(v, G_i), \forall v \in V_x$ and $v \notin G_i$.

$$\delta_p(x, G_i) = \frac{count(x, i)}{\sum_{\forall v \in V_x}^{v \notin G_i} count(v, i)}. \tag{6}$$

As illustrated in Figure 6, the local embeddings of each worker contain two types of vertices. The 1D partitioning results in non-overlapping partition results and these vertices are **primary** vertices on their respecting partitions. The replicas of embedding vertices on the other partitions are **secondary** vertices. For example,

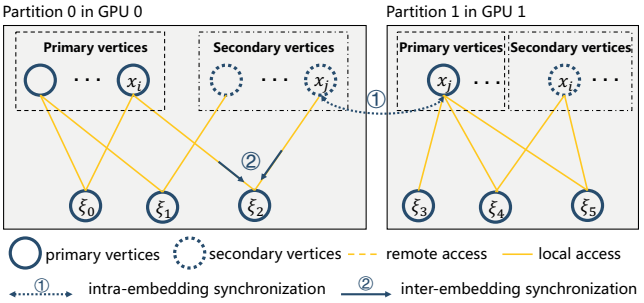


Figure 6: Illustration of graph partition and two synchronization points in the Read operation HET-GMP.

x_i is a primary vertex in GPU 0 and it is replicated to GPU 1. Similarly, x_j is a primary vertex in GPU 1 and it is replicated to GPU 0. Each embedding only has one corresponding primary vertex and might have multiple secondary vertices on different workers. The balanced graph partitioning minimizes the remote access costs and the replication further improves the local access.

5.3 Graph-based Bounded Asynchrony

The vertex-cut replications of high-degree embeddings reduce the remote access costs, but also bring consistency problems for model synchronization across replicas. To unleash the full power of replication, HET-GMP introduces bounded asynchronous training so that the local embeddings do not need to wait for all updates on their replicas to proceed in most cases.

The distributed training of the embedding model has two synchronization points: (1) replicas of a specific embedding when reading it, and (2) replicas of multiple embeddings when used by the same data vertex. HET-GMP uses bounded asynchrony in a novel way at these two synchronization points where staleness can be tolerated. As illustrated in Figure 6, when performing a Read operation, HET-GMP first checks if the requested embeddings exist locally. Primary vertices can be directly accessed locally since we keep them always up-to-date, i.e., every update on a secondary replica is written back to the primary replica. For the secondary vertices, we perform two synchronizations to ensure the bounded staleness:

Bounded Asynchrony at Intra-embedding. When a data sample reads a set of embeddings X, for each embedding $x_j \in X$, we first check whether the version of the local secondary embedding is within s updates away from its primary replica (see ① in Figure 6). If not, we perform synchronization between the local secondary embedding and its remote primary.

Bounded Asynchrony at Inter-embedding. After collecting the set of embeddings X, we check whether the versions of each pair of embeddings $x_i, x_j \in X$ is bounded by s, i.e., the secondary embedding is not too far behind the other related local embeddings (i.e., E) used by the same data sample node (see ② in Figure 6). If not, we perform synchronization between the local secondary embedding and its remote primary or make the worker wait when replicas are still too stale. This avoids the quality of an embedding to be adversely affected by the excessive staleness of those associated embeddings it depends on.

To implement both intra and inter-bounded asynchrony, we maintain a clock c_i^k for embedding vertex x_i at worker k to record the accumulated number of updates on that replica. Once the embedding fails to pass the bounded staleness checking (i.e., the clocks gap exceeds the threshold s), remote access occurs to synchronize with the primary. After the current iteration of training, all embeddings perform gradient updating and the secondary embeddings directly write back to the corresponding primary vertices (i.e., Update). Note that to eliminate the effect of uneven access frequencies of different embeddings [43] for staleness validation, we perform a clock normalization based on the access frequency p_i for each embedding x_i. Specifically, when checking the bounded staleness for a given pair of embeddings $(x_i^{k_1}, x_j^{k_2})$ from workers k_1 and k_2 with clocks $(c_i^{k_1}, c_j^{k_2})$, and assuming that $p_i \geq p_j$, then the normalized clock gap between the two embeddings is $|c_i^{k_1} * p_j/p_i - c_j^{k_2}|$. In the special case $i = j$ (e.g., intra-embedding synchronization), the normalized clock gap is still measured by the number of updates on the replicas of those embeddings, i.e., $|c_i^{k_1} - c_j^{k_2}|$.

5.4 Convergence Analysis

We define the global model $\mathbf{x}$ as the combination of all latest embeddings (i.e., primary) for our analysis. We denote x_i and $\nabla_i f(\mathbf{x})$ as the i-th embedding and its gradient on $\nabla f(\mathbf{x})$, respectively. Clearly, $\mathbf{x} = (x_0, x_1, \ldots, x_m)$ and $\nabla f = (\nabla_0 f, \nabla_1 f, \cdots, \nabla_m f)$. Considering a logical global clock t shared by all workers, we use $\mathbf{x}(t)$ to represent the combination of all latest embeddings among all workers, and $\mathbf{x}^i(t)$ to represent the embeddings that could be accessed by i-th worker when it performs the t-th update. The convergence of the bounded staleness training is guaranteed by the following theorem:

Theorem 1. *Suppose (1) f is bounded below, differentiable and the gradient ∇f of f is L-Lipschitz continuous; (2) the embedding is updated with bounded delay s; (3) the objective function could sufficiently decrease for all large t. Let F satisfy the KŁ property in [10, Lemma 6]. Then, with step size $\eta \in \left(0, \frac{1}{L(1+2\sqrt{ps})}\right)$, every sequence $\{\mathbf{x}(t)\}$ generated by HET-GMP satisfies*

$$\sum_{t=0}^{\infty} \|\mathbf{x}(t+1) - \mathbf{x}(t)\| < \infty, \tag{7}$$

$$\forall i = 1, \ldots, p, \sum_{t=0}^{\infty} \|\mathbf{x}^i(t+1) - \mathbf{x}^i(t)\| < \infty. \tag{8}$$

Furthermore, $\{\mathbf{x}(t)\}$ and $\{\mathbf{x}^i(t)\}_{i=1}^p$ converge to the same critical point of F. Recall from [40], we have the following convergence rate:

$$F\left(\frac{1}{t}\sum_{k=1}^{t}\mathbf{x}(k)\right) - F_{inf} \leq O\left(\frac{1}{t}\right). \tag{9}$$

Proof Sketch. We obtain the usual $O(1/t)$ rate of convergence of the objective value. The theorem is proven by extending the proximal gradient descent method in [54]. The detailed theorem proofs and assumption justifications can be found in our extra material [5]. The key step is to formulate the delay of secondaries and we define the active clocks T_i as the updated clocks set during the past s clocks. Based on the bounded delay s from assumption (2), we can prove $\|\mathbf{x}^i(t+1) - \mathbf{x}^i(t)\| \leq \sum_{k=(t-s)_+}^{t} \|\mathbf{x}(k+1) - \mathbf{x}(k)\|$, which bounds the inconsistency between the global model and the local models.

Table 1: Overview of the three datasets

Dataset	#Samples	#Features	#Fields
Avazu	40,428,967	9,449,445	22
Criteo	45,840,617	33,762,577	26
Company	35,682,429	66,102,027	43

Then we can prove that $\sum_{t=0}^{\infty} \|\mathbf{x}(t+1) - \mathbf{x}(t)\|^2 < \infty$ by utilizing assumption (1), which further implies $\lim_{t\to\infty} \|\mathbf{x}(t+1) - \mathbf{x}(t)\| = 0$ and $\lim_{t\to\infty} \|\mathbf{x}(t) - \mathbf{x}^i(t)\| = 0$. Since the inconsistency diminishes as the number of iterations grows, we can also prove the existence of critical points for the sequences $\{\mathbf{x}(t)\}$ and $\{\mathbf{x}^i(t)\}$. With such guarantees and the sufficiently decreasing and KL property assumptions of F, we can further obtain $\sum_{t=0}^{\infty} \|\mathbf{x}(t+1) - \mathbf{x}(t)\| < \infty$ and complete the proof.

6 IMPLEMENTATION

HET-GMP is built on Hetu [5], a DL system consists of 14.5K LOC in C/C++/CUDA with a Python datafow front-end (20.7K LOC). It consists of a large number of computation kernels with cuDNN [14] and communication methods with NCCL 2.7 [6].

GPU Embedding Table. We implement the embedding table in CUDA and store the local embedding table in the limited GPU memory (e.g., 32 GB) of each worker in our experiments. When the embedding table is created, space is allocated for both primary and secondary embeddings guided by the partition result. Secondary embeddings require extra space for stale gradients.

Decentralized Communication. Our implementation for the distributed embedding table is based on NCCL peer-to-peer communication interfaces. Upon each read operation, each embedding table instance in the communication group will first send sparse indexes and clocks to the others. This step only takes a short time to complete since these sparse indexes and clocks are small compared with the embedding. Each instance then processes the message received and sends back embeddings that are considered outdated according to the staleness bound. For each update operation, each embedding table will first perform a local reduction and then write to primaries without conflicts.

Asynchronous Execution. We also take efforts to overlap communication with computation to boost overall training throughput. There are two crucial time-points in the training loop, which are the embedding lookup operation in the forward propagation (T1) and the time when gradients for the embedding table are ready in the backward propagation (T2). The time between T2 for the current training batch and T1 for the next training batch can be used for asynchronous execution, overlapping communication incurred by embedding lookups and embedding updates with computation. For the time between T1 and T2, when gradients are not ready, we cannot start embedding lookup for the next batch. We utilize this duration for loading the next batch of data from host to device.

7 EXPERIMENTS

Baselines. In this section, we compare our prototype system with three state-of-the-art systems: TensorFlow 1.15 (TF) [8], Parallax [23] and HugeCTR v2.3. Although TF and Parallax manage embeddings on CPU servers, they still perform the computation on GPUs. Including these baselines allows us to show to what extent increasing parameter access locality via graph-based approach is beneficial. HugeCTR uses GPU memory to manage the embedding parameters, has been elaborately optimized by Nvidia, and is one of the state-of-the-art systems for deep recommendation models. To alleviate the concerns on the difference between the system backbones and implementations, we implement an auxiliary baseline over our system named by HET-MP, which only performs model-parallelism with random partitioning.

Datasets and Models. We select two representative CTR embedding model workloads, including Wide & Deep (WDL) [13] and Deep & Cross (DCN) [41]. They are evaluated on the two popular public recommendation datasets Avazu [3] and Criteo [2] and one production advertising dataset Company from our industrial partner. Avazu was released in the CTR prediction contest on Kaggle. Criteo contains one month of click logs with billions of data samples. Criteo is also the largest standard benchmark in MLPerf [4]. Company dataset is collected from a recommendation scenario in Tencent Inc. containing ad features (e.g., ID, category). We list the metrics of these datasets in Table 1.

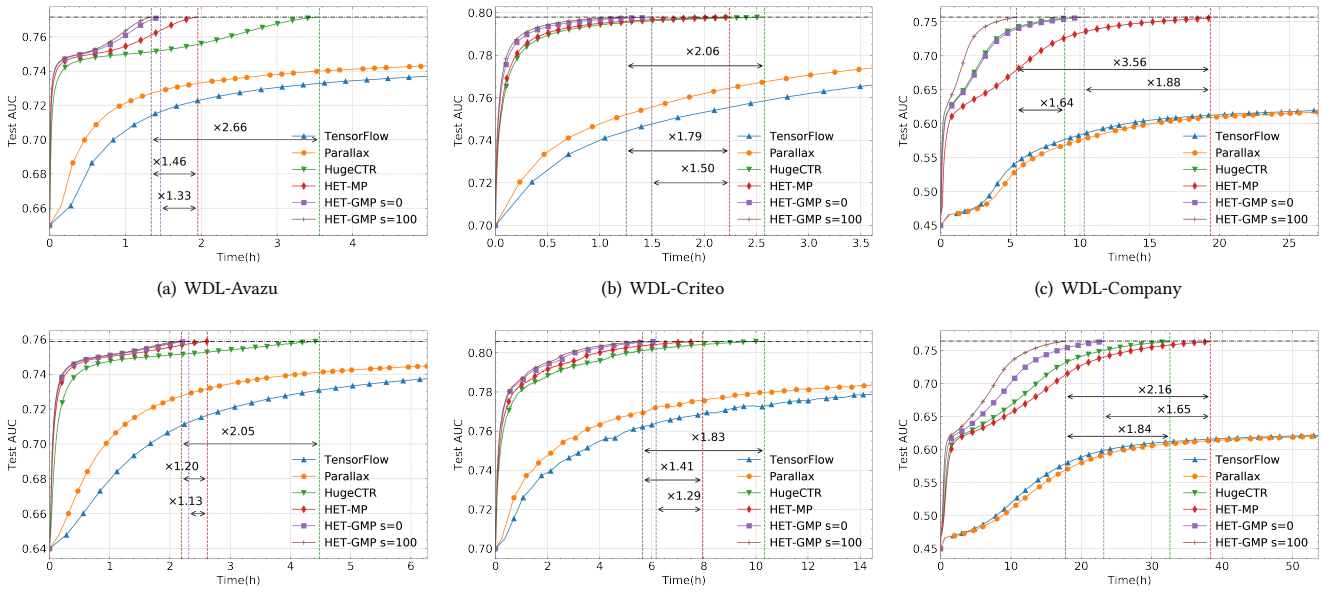
Experimental Setting. We have conducted our evaluations on two GPU clusters. Most are conducted on cluster A where each node is equipped with 8 Nvidia RTX TITAN 24 GB cards supporting PCIe 3.0, and 1 Gb Ethernet. Cluster B is used to perform the scalability study and each node has 8 Nvidia Tesla V100 32 GB cards supporting NVLink, and 10 Gb Ethernet. The testing AUC thresholds of convergence are set to be around 76%, 80% for Avazu and Criteo datasets, as reported in [13, 27]. For 2-D partitioning, we select top 1% embeddings as secondaries using E.q. (6). All experiments are executed five times, and the averaged results are reported.

7.1 End-to-end Comparison

In this section, we first provide end-to-end comparison experiments with the baselines. These experiments are evaluated on one node with 8 GPUs on cluster A. Figure 7 shows the convergence curves on six different workloads. TF-PS and Parallax follow the ASP algorithm and cannot converge to the target thresholds in these workloads. We provide HET-GMP with different staleness thresholds $s = 10$ and 100. As we can see, our system always outperforms the other baselines on all tasks. Compared to HET-MP, we can achieve around 1.2-3.56× speed up when $s = 100$. Even if we remove the staleness tolerance (i.e., $s = 0$), the hybrid graph partitioning still makes HET-GMP outperform the others in most cases. Besides, HugeCTR significantly outperforms CPU-based approaches (i.e., TF and Parallax) and achieves similar performance to HET-MP since they select the same system design. But the illustrated end-to-end convergence time shows that HET-GMP still achieves 1.64-2.66× speedup compared to HugeCTR. Parallax performs better than TF due to the hybrid communication architecture, but both of them cannot converge within the given time thresholds.

7.2 Detailed Analysis

Communication Comparison. Figure 8 breaks down the total communication costs of HET-GMP with different settings for one



(a) WDL-Avazu (b) WDL-Criteo (c) WDL-Company

(d) DCN-Avazu (e) DCN-Criteo (f) DCN-Company

Figure 7: Convergence performance comparison.

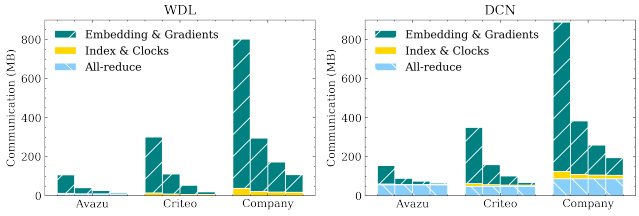


Figure 8: Communication details for HET-GMP. The four columns from left to right represent random/1-D /2-D ($s = 10$)/2-D ($s = 100$) partitioning respectively.

iteration into three categories. As we can see, using random partitioning, the majority of the communication costs are due to transferring embeddings and their gradients. With the 2-D partitioning ($s = 100$), we can achieve around 87.5% embedding communications reduction on the Company dataset. The HET-GMP client also needs to frequently send and receive the embedding index (i.e., keys) and clocks information, but they are small compared to the first category. The last term is for All-Reduce. Since the DCN network has more dense parameters in its cross layers, it requires more model synchronization costs than WDL. However, the embedding parameters still dominate the communication costs.

Bounded Asynchrony. We evaluate HET-GMP with different bounded staleness s on three datasets with the WDL model. The convergent test AUC results are shown in Table 2. As we can see, HET-GMP is robust to s and even with $s = 10k$, the final model performance is still competitive. We also note that continuing to increase s might hurt the model quality, which implies us to limit the asynchrony during the training process.

Hybrid Graph Partitioning. We now demonstrate the effectiveness of our partitioning method by comparing the overall throughput on three datasets with the WDL model in Figure 9(a). The

Table 2: Final test AUC (%) with different s on WDL

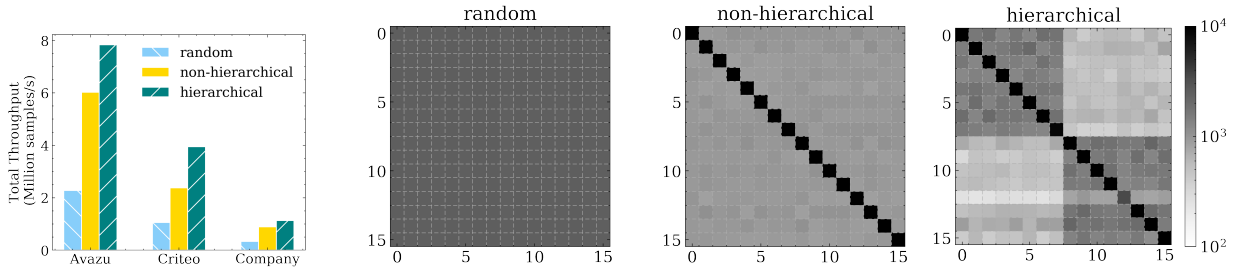
Dataset	$s = 0$	$s = 100$	$s = 10k$	$s = \infty$
Avazu	77.19	77.19	76.9	76.21
Criteo	79.79	79.79	79.77	78.70
Company	76.09	76.10	76.11	73.27

following experiment is carried out with 16 GPUs placed on 2 machines connected with 10 Gb Ethernet on cluster B. We adopt three different partition policies: random, non-hierarchical, and hierarchical. The hierarchical and non-hierarchical partitioning corresponds to w or w/o topology-aware partitioning. For the random policy, we use the initial random partitioning. For the non-hierarchical policy, we treat all pair-to-pair communication costs as a fixed value. For the hierarchical policy, we set inter-machine communication costs 10 times higher than intra-machine communication. For a fair comparison, we do not introduce replication in this experiment. The result in 9(a) shows that hierarchical partitioning outperforms non-hierarchical and random partitioning on all three datasets.

Figure 9(b) further explains the reason for such speedup. Each block represents the amount of embeddings fetched in one iteration from one worker to another worker, deeper colors represent more communications. For random partitioning, each worker fetches embeddings equally from the other workers. For non-hierarchical partitioning, most embeddings can be found on the same worker. When using hierarchical partitioning, embedding are clustered on both machine level and worker level. This significantly reduces communication overhead on each level and yields the highest throughput.

7.3 Graph Partitioning Comparison

We now compare our hybrid iterative graph partitioning algorithm against existing algorithms to show its superiority for the new workloads of embedding models. Although PowerLyra [12] is a hybrid graph partitioning algorithm considering edges but also



(a) Throughput comparison (b) Illustration of GPU-GPU embedding communication comparison.

Figure 9: (a) Throughput comparison on WDL when using different partitioning methods under different datasets. (b) The amount of sparse embedding communication between each pair of workers on Criteo.

Table 3: Graph partitioning algorithms performance comparison on three datasets.

Algorithms	Company			Criteo			Avazu		
	Communication	Reduction	Time (s)	Communication	Reduction	Time (s)	Communication	Reduction	Time (s)
Random	1,052,231	0%	0	277,660	0%	0	94,044	0%	0
BiCut	910,696	13.5%	268	231,851	16.5%	124	76,431	18.7%	52
Ours (1 round)	659,387	37.3%	358	135,942	51.0%	202	34,677	63.1%	115
Ours (3 rounds)	424,611	59.7%	589	103,807	62.6%	324	30,995	67.0%	192
Ours (5 rounds)	380,576	63.8%	799	101,425	63.5%	435	30,427	67.7%	265

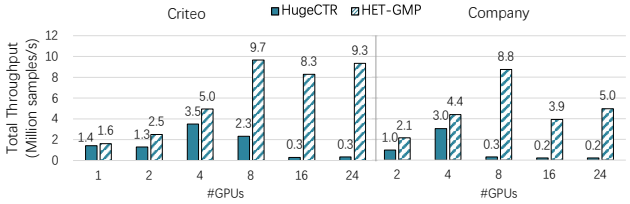


Figure 10: Total throughput comparison for WDL under different number of GPUs. The company dataset is too large to be stored on a single GPU.

vertices and general skewness, it is not suitable for bipartite graphs, because the general graph partitioning algorithms are oblivious to the unique features of bipartite graphs resulting in suboptimal graph placement and high replication factor. Therefore, we select a customized optimized version of PowerLyra over bipartite graphs, BiCut [11], as a strong baseline. BiCut also leverages the skewed distribution of vertices, and distinguishes computation load between the two subsets of vertices.

Table 3 reports the total number of remote embedding communications for different algorithms for an epoch. As we can see, the results on the Company dataset with 8 partitions illustrate that BiCut only reduces the communication compared to random partitioning by a factor of 13.5%, whereas our method can reduce it by up to 63.8%, with negligible costs compared to the training time (< 2%). On Criteo and Avazu, our method still outperforms BiCut in regard to communication reduction easily with only 3 rounds. These results verify the effectiveness of our new design principle and graph partitioning algorithm for embedding models.

7.4 Scalability Study

We conduct a scalability study of HET-GMP and HugeCTR for WDL on Criteo and Company datasets under different numbers of GPUs on cluster B. As shown in Figure 10, when the number of GPUs increases from 4 to 8, the total throughput of HugeCTR decreases immediately and continues decreasing as it grows to 16. Because the inter-GPU connections change from NVLink to QPI and Ethernet with lower bandwidth when involving more GPUs. We also find that HET-GMP is more robust to the environments and always outperforms HugeCTR. Our system is up to 27.5× and 24.8× faster than HugeCTR on Criteo and Company, respectively. Currently, with 24 GPUs (32 GB), we support around 10^{11} float parameters in the embedding table. We are looking forward to improving the training efficiency with much higher network bandwidth (e.g., RDMA) and involving more GPUs to enlarge the embedding model capacity in our future works.

8 CONCLUSION

Through a graph-based approach, HET-GMP opens up a new optimization perspective and provides a scalable solution that supports efficient large-scale embedding model training. The novelty of HET-GMP lies in 1) introducing graph-based replication and partitioning for balanced and locality-conscious distributed training, and 2) presenting novel graph-based asynchrony at both intra and inter embeddings where bounded staleness can be tolerated. The results show that HET-GMP supports 10^{11} parameters scale embedding model training, achieving up-to 87.5% communication reduction and up-to 27.5× speedup over the state-of-the-art baselines.

9 ACKNOWLEDGEMENT

This work is supported by the National Key Research and Development Program of China (No. 2018YFB1004403), the National Natural Science Foundation of China (No. 61832001), PKU-Tencent joint research Lab, Beijing Academy of Artificial Intelligence (BAAI). Bin Cui is the corresponding author.

REFERENCES

[1] 2014. The Apache Giraph Project. http://giraph.apache.org/.
[2] 2014. Criteo Kaggle Ad. https://www.kaggle.com/c/criteo-display-ad-challenge.
[3] 2015. Avazu Ad. http://www.kaggle.com/c/avazu-ctr-prediction.
[4] 2020. MLPerf Benchmark. https://mlperf.org.
[5] 2021. Hetu. https://github.com/PKU-DAIR/Hetu/.
[6] 2021. NVIDIA collective communications library (NCCL). https://developer.nvidia.com/nccl.
[7] 2021. NVIDIA HugeCTR. https://github.com/NVIDIA/HugeCTR.
[8] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *OSDI*. USENIX Association, 265–283.
[9] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. 1993. Mining Association Rules between Sets of Items in Large Databases. In *SIGMOD*. ACM Press, 207–216. https://doi.org/10.1145/170035.170072
[10] Jérôme Bolte, Shoham Sabach, and Marc Teboulle. 2014. Proximal alternating linearized minimization for nonconvex and nonsmooth problems. *Math. Program.* 146, 1-2 (2014), 459–494. https://doi.org/10.1007/s10107-013-0701-9
[11] Rong Chen, Jiaxin Shi, Haibo Chen, and Binyu Zang. 2015. Bipartite-Oriented Distributed Graph Partitioning for Big Learning. *J. Comput. Sci. Technol.* 30, 1 (2015), 20–29. https://doi.org/10.1007/s11390-015-1501-x
[12] Rong Chen, Jiaxin Shi, Yanzhe Chen, and Haibo Chen. 2015. PowerLyra: differentiated graph computation and partitioning on skewed graphs. In *EuroSys*. ACM, 1:1–1:15. https://doi.org/10.1145/2741948.2741970
[13] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. 2016. Wide & Deep Learning for Recommender Systems. In *DLRS@RecSys*. 7–10.
[14] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759* (2014).
[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*. 4171–4186. https://doi.org/10.18653/v1/n19-1423
[16] Wenfei Fan, Kun He, Qian Li, and Yue Wang. 2020. Graph algorithms: parallelization and scalability. *Sci. China Inf. Sci.* 63, 10 (2020), 1–21.
[17] Andrew Gibiansky. 2017. Bringing HPC techniques to deep learning. *Baidu Research, Tech. Rep.* (2017).
[18] Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. 2012. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs. In *OSDI*. USENIX Association, 17–30.
[19] Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. 2014. GraphX: Graph Processing in a Distributed Dataflow Framework. In *OSDI*. USENIX Association, 599–613.
[20] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction. In *IJCAI*. 1725–1731.
[21] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A. Gibson, Gregory R. Ganger, and Eric P. Xing. 2013. More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. In *NeurIPS*. 1223–1231.
[22] George Karypis and Vipin Kumar. 1998. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. Sci. Comput.* 20, 1 (1998), 359–392. https://doi.org/10.1137/S1064827595287997
[23] Soojeong Kim, Gyeong-In Yu, Hojin Park, Sungwoo Cho, Eunji Jeong, Hyeonmin Ha, Sanha Lee, Joo Seong Jeong, and Byung-Gon Chun. 2019. Parallax: Sparsity-aware Data Parallel Training of Deep Neural Networks. In *EuroSys*. ACM, 43:1–43:15. https://doi.org/10.1145/3302424.3303957
[24] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *ICLR*. OpenReview.net.
[25] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. 2014. Scaling Distributed Machine Learning with the Parameter Server. In *OSDI*. USENIX Association, 583–598.
[26] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. 2020. PyTorch Distributed: Experiences on Accelerating Data Parallel Training. *Proc. VLDB Endow.* 13, 12 (2020), 3005–3018. https://doi.org/10.14778/3415478.3415530
[27] Zeyu Li, Wei Cheng, Yang Chen, Haifeng Chen, and Wei Wang. 2020. Interpretable Click-Through Rate Prediction through Hierarchical Attention. In *WSDM*. ACM, 313–321. https://doi.org/10.1145/3336191.3371785
[28] Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. 2018. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. In *SIGKDD*. 1754–1763.
[29] Xiangru Lian, Yijun Huang, Yuncheng Li, and Ji Liu. 2015. Asynchronous Parallel Stochastic Gradient for Nonconvex Optimization. In *NeurIPS*. 2737–2745.
[30] Xiangru Lian, Wei Zhang, Ce Zhang, and Ji Liu. 2018. Asynchronous Decentralized Parallel Stochastic Gradient Descent. In *ICML*, Vol. 80. PMLR, 3049–3058.
[31] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. 2012. Distributed GraphLab: A Framework for Machine Learning in the Cloud. *Proc. VLDB Endow.* 5, 8 (2012), 716–727.
[32] Xupeng Miao, Nezihe Merve Gürel, Wentao Zhang, Zhichao Han, Bo Li, Wei Min, Susie Xi Rao, Hansheng Ren, Yinan Shan, Yingxia Shao, Yujie Wang, Fan Wu, Hui Xue, Yaming Yang, Zitao Zhang, Yang Zhao, Shuai Zhang, Yujing Wang, Bin Cui, and Ce Zhang. 2021. DeGNN: Improving Graph Neural Networks with Graph Decomposition. In *SIGKDD*. ACM, 1223–1233. https://doi.org/10.1145/3447548.3467312
[33] 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*, Guoliang Li, Zhanhuai Li, Stratos Idreos, and Divesh Srivastava (Eds.). ACM, 2262–2270. https://doi.org/10.1145/3448016.3452773
[34] Xupeng Miao, Hailin Zhang, Yining Shi, Xiaonan Nie, Zhi Yang, Yangyu Tao, and Bin Cui. 2022. HET: Scaling out Huge Embedding Model Training via Cache-enabled Distributed Framework. *Proc. VLDB Endow.* 15, 312–320. https://doi.org/10.14778/3489496.3489511
[35] Xupeng Miao, Wentao Zhang, Yingxia Shao, Bin Cui, Lei Chen, Ce Zhang, and Jiawei Jiang. 2021. Lasagne: A Multi-Layer Graph Convolutional Network Framework via Node-aware Deep Architecture. *TKDE* (2021), 1–1. https://doi.org/10.1109/TKDE.2021.3103984
[36] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, Liang Luo, Jie Amy Yang, Leon Gao, Dmytro Ivchenko, Aarti Basant, Yuxi Hu, Jiyan Yang, Ehsan K. Ardestani, Xiaodong Wang, Rakesh Komuravelli, Ching-Hsiang Chu, Serhat Yilmaz, Huayu Li, Jiyuan Qian, Zhuobo Feng, Yinbin Ma, Junjie Yang, Ellie Wen, Hong Li, Lin Yang, Chonglin Sun, Whitney Zhao, Dimitry Melts, Krishna Dhulipala, K. R. Kishore, Tyler Graf, Assaf Eisenman, Kiran Kumar Matam, Adi Gangidi, Guoqiang Jerry Chen, Manoj Krishnan, Avinash Nayak, Krishnakumar Nair, Bharath Muthiah, Mahmoud khorashadi, Pallab Bhattacharya, Petr Lapukhov, Maxim Naumov, Lin Qiao, Mikhail Smelyanskiy, Bill Jia, and Vijay Rao. 2021. High-performance, Distributed Training of Large-scale Deep Learning Recommendation Models. *CoRR* abs/2104.05158 (2021). arXiv:2104.05158
[37] Xuehai Qian. 2021. Graph processing and machine learning architectures with emerging memory technologies: a survey. *Sci. China Inf. Sci.* 64, 6 (2021). https://doi.org/10.1007/s11432-020-3219-6
[38] Alexander Renz-Wieland, Tobias Drobisch, Zoi Kaoudi, Rainer Gemulla, and Volker Markl. 2021. Just Move It! Dynamic Parameter Allocation in Action. *Proc. VLDB Endow.* 14, 12 (2021), 2707–2710.
[39] Alexander Renz-Wieland, Rainer Gemulla, Steffen Zeuch, and Volker Markl. 2020. Dynamic Parameter Allocation in Parameter Servers. *Proc. VLDB Endow.* 13, 11 (2020), 1877–1890.
[40] Mark Schmidt, Nicolas Le Roux, and Francis R. Bach. 2011. Convergence Rates of Inexact Proximal-Gradient Methods for Convex Optimization. In *NeurIPS*. 1458–1466.
[41] Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. 2017. Deep & Cross Network for Ad Click Predictions. In *ADKDD*. 12:1–12:7.
[42] Xuhong Wang, Ding Lyu, Mengjian Li, Yang Xia, Qi Yang, Xinwen Wang, Xinguang Wang, Ping Cui, Yupu Yang, Bowen Sun, and Zhenyu Guo. 2021. APAN: Asynchronous Propagation Attention Network for Real-time Temporal Graph Embedding. In *SIGMOD*. ACM, 2628–2638. https://doi.org/10.1145/3448016.3457564
[43] Qiyu Wu, Chen Xing, Yatao Li, Guolin Ke, Di He, and Tie-Yan Liu. 2021. Taking Notes on the Fly Helps Language Pre-Training. In *ICLR*. OpenReview.net.
[44] Cong Xie, Ling Yan, Wu-Jun Li, and Zhihua Zhang. 2014. Distributed Power-law Graph Computing: Theoretical and Empirical Analysis. In *NeurIPS*. 1673–1681.
[45] Minhui Xie, Kai Ren, Youyou Lu, Guangxu Yang, Qingxing Xu, Bihai Wu, Jiazhen Lin, Hongbo Ao, Wanhong Xu, and Jiwu Shu. 2020. Kraken: memory-efficient continual learning for large-scale real-time recommendations. In *SC*. 21.
[46] Zhiqiang Xu, Dong Li, Weijie Zhao, Xing Shen, Tianbo Huang, Xiaoyun Li, and Ping Li. 2021. Agile and Accurate CTR Prediction Model Training for Massive-Scale Online Advertising Systems. In *SIGMOD*. ACM, 2404–2409. https://doi.org/10.1145/3448016.3457236
[47] Lele Yu, Bin Cui, Ce Zhang, and Yingxia Shao. 2017. LDA*: A Robust and Large-scale Topic Modeling System. *Proc. VLDB Endow.* 10, 11 (2017), 1406–1417. https://doi.org/10.14778/3137628.3137649
[48] Ce Zhang. 2015. DeepDive: A Data Management System for Automatic Knowledge Base Construction. *Doctoral dissertation, The University of Wisconsin-Madison* (2015).
[49] Wentao Zhang, Xupeng Miao, Yingxia Shao, Jiawei Jiang, Lei Chen, Olivier Ruas, and Bin Cui. 2020. Reliable data distillation on graph convolutional network. In *SIGMOD*. 1399–1414.

[50] Weijie Zhao, Deping Xie, Ronglai Jia, Yulei Qian, Ruiquan Ding, Mingming Sun, and Ping Li. 2020. Distributed Hierarchical GPU Parameter Server for Massive Scale Deep Learning Ads Systems. In *MLSys*. mlsys.org.
[51] 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. In *IA3 Workshop*. 36–44.
[52] Da Zheng, Xiang Song, Chao Ma, Zeyuan Tan, Zihao Ye, Jin Dong, Hao Xiong, Zheng Zhang, and George Karypis. 2020. DGL-KE: Training Knowledge Graph Embeddings at Scale. In *SIGIR*. ACM, 739–748.
[53] Guorui Zhou, Xiaoqiang Zhu, Chengru Song, Ying Fan, Han Zhu, Xiao Ma, Yanghui Yan, Junqi Jin, Han Li, and Kun Gai. 2018. Deep Interest Network for Click-Through Rate Prediction. In *SIGKDD*. 1059–1068.
[54] Yi Zhou, Yaoliang Yu, Wei Dai, Yingbin Liang, and Eric P. Xing. 2016. On Convergence of Model Parallel Proximal Gradient Algorithm for Stale Synchronous Parallel System. In *AISTATS*, Arthur Gretton and Christian C. Robert (Eds.), Vol. 51. JMLR.org, 713–722.