HET-KG: Communication-Efficient Knowledge Graph Embedding Training via Hotness-Aware Cache

Sicong Dong[†,*], Xupeng Miao[‡,*], Pengkai Liu[†], Xin Wang[†,§], Bin Cui[‡], Jianxin Li[¶]
[†]College of Intelligence and Computing, Tianjin University
[‡]School of Computer Science & Key Lab of High Confidence Software Technologies (MOE), Peking University
[¶]School of Information Technology, Deakin University
[†]{sicongdong, liupengkai, wangx}@tju.edu.cn [‡]{xupeng.miao, bin.cui}@pku.edu.cn [¶]jianxin.li@deakin.edu.au

2022 IEEE 38th International Conference on Data Engineering (ICDE) | 978-1-6654-0883-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDE53745.2022.00177

***Abstract*—With the popularization and application of Artificial Intelligence technology, knowledge graph embedding methods are widely used for a variety of machine learning tasks. However, most of the current knowledge graph embedding models are trained with a large number of parameters and high computational time complexity. This becomes a main obstacle to apply these existing models to large-scale knowledge graphs. To address this challenge, we propose HET-KG, a distributed system for training knowledge graph embedding efficiently. HET-KG can reduce the communication overheads by introducing a cache embedding table structure to maintain *hot-embeddings* at each worker. To improve the effectiveness of the cache mechanism, we design a prefetching algorithm and a filtering algorithm for adaptively selecting *hot-embeddings*, and provide two kinds of hot-embedding table construction strategies. To address the issue of inconsistency between the local cached *hot-embeddings* and the global embeddings, we also develop a hot-embedding synchronization algorithm for dynamically updating the cache embedding table, which can guarantee the inconsistency bounded within a given threshold. Finally, extensive experiments are conducted on three knowledge graph datasets FB15k, WN18, and Freebase-86m. The experimental results show that HET-KG achieves 3.7x and 1.1x speedup over the state-of-the-art systems PyTorch-BigGraph and DGL-KE, respectively.**

***Index Terms*—knowledge graphs, parameter server, distributed knowledge graph embedding, cache**

I. Introduction

In recent years, Knowledge Graphs (KGs) [24], [33], consisting of entities and their relations, have been used to store structured knowledge information and successfully applied to many real-world applications, such as information extraction [2], question answering [4], recommendation [16], [29], and drug discovery [6]. To leverage the structured knowledge of KGs in downstream applications, Knowledge Graph Embedding (KGE) [1], [21], [30] has been proposed to embed the entities and relations of a KG into continuous vector spaces. Since KGE could capture the semantics of information in knowledge graphs, it has demonstrated great success in many machine learning tasks over KGs. However, the computational cost of KGE training is expensive since KGE methods need to calculate relevance scores between embeddings, and iteratively optimize objective functions until convergence, which thus is a computationally intensive task.

To reduce the computation time, several distributed KGE training approaches and systems have been proposed by leveraging multiple computing units (e.g., cores, GPUs, machines). For instance, the block-based approaches, such as PyTorch-BigGraph (PBG) [13] and GraphVite [36], split the KG into disjoint blocks and parallelize the computation of a subset of the blocks across workers. However, these solutions lead to high block swapping overheads. In addition, another line of approaches have been exploited by utilizing the architecture of Parameter Server (PS) [10]. For example, DGL-KE [34] stores the embeddings on the parameter servers and uses synchronous data parallelism training on the workers. To reduce the communication cost during training process, DGL-KE further enables co-located PS and min-cut-based graph partition algorithms (e.g., METIS [12]) to distribute a knowledge graph across workers. However, DGL-KE still suffers from high network communication cost when the cluster size increases, e.g., when DGL-KE with TransE model is used to train a large-scale knowledge graph like Freebase-86m, the network communication dominates more than 70% of the end-to-end training time under our experimental environment, which is shown in Table I.

The communication cost may become even more expensive when the knowledge graphs contain a large number of entities with high degrees [5], [23], this is because these entities may have lots of related entities located at other workers. To alleviate this issue, in this work, we will exploit the skew distribution of node frequencies in KGs, by which we intend to cache the most frequently accessed embedding (i.e., *hot-embeddings*) at workers. Although the caching mechanism can help to reduce the remote communication overheads, it faces several technical challenges. Firstly, each worker has only limited memory space. Thus, it requires to have a reasonable

∗ These authors contribute equally to this work and should be considered co-first authors.
§ Xin Wang is the corresponding author.

2375-026X/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDE53745.2022.00177

TABLE I
WORKLOAD FOR TRAINING TRANSE ON FREEBASE-86M.

# Edges	Dimension	Storage Space	Communication Ratio
3×10^5	400	275 MB	82.6%
3×10^6	400	2.7 GB	85.4%
3×10^7	400	27.5 GB	88.3%
3×10^8	400	275.2 GB	88.7%
3×10^8	50	34.4 GB	70.7%
3×10^8	100	68.8 GB	73.5%
3×10^8	200	137.6 GB	84.9%
3×10^8	400	275.2 GB	88.7%

selection strategy to select and cache entities and relations on each worker, so that the communication reduction can be achieved. Secondly, it requires a well-designed approach to handle the inconsistency between local embeddings cached on workers and the global embeddings on the parameter server. Otherwise, the local embeddings on the workers can only be updated based on the caches on the local workers without resorting to the remote updates from the other workers. The inconsistency will become divergent after several iterations during training, which can easily lead to the failure of convergence. This is because the embeddings of some entities should be jointly updated by different workers during training.

To tackle the above challenging problems, in this paper, we propose a distributed knowledge graph embedding training system, named HET-KG, by introducing cache embedding tables at workers. To the best of our knowledge, this is the first work to reduce the communication cost by utilizing cache embedding table for distributed KGE training. Specifically, for a given knowledge graph, we design a prefetching algorithm that can sample positive and negative triples, construct mini-batches, and obtain entities and relations used in each mini-batch. To deal with the cache embedding table, we further develop a filtering algorithm to dynamically select the top-k entities and relations, pull these entity and relation embeddings from the parameter server, and store them in the cache embedding table. In the meanwhile, we devise a partial stale synchronous algorithm by dynamically updating the cache embedding table, which guarantees that the inconsistency between the local cached embeddings and the global embeddings can be bounded within a given threshold. We also provide the theoretical analysis about the convergence. Furthermore, we present implementation details of the prototype system HET-KG. Caching hot items have been extensively studied in the context of graph embedding [19], feature stores [35], and embedding-based parameter servers [32]. While the caching technique in our work significantly defers from that in the existing systems [17], [19], [32], [35] due to the differences in the targeted research problems. Unlike these studies, both cached and non-cached embeddings will be updated during the training process in our problem. It not only requires to find the most frequently accessed embeddings to utilize the limited memory space, but also has to handle the embedding inconsistency problem due to the updates.

Our contributions can be summarized as follows:

- In knowledge graph embedding, HET-KG is the first work to reduce the communication overheads by introducing a cache embedding table structure to maintain hot-embeddings.
- We design a prefetching algorithm and a filtering algorithm for adaptively selecting the hot-embeddings, and provide two kinds of hot-embedding table construction strategies for HET-KG.
- We develop a partial stale synchronous algorithm for dynamically updating the cache embedding table, and provide the theoretical analysis to guarantee that the inconsistency between the local cached hot-embeddings and the global embeddings can be bounded within a given threshold.
- Extensive experiments on large-scale knowledge graphs verify the efficiency and scalability of the proposed system. The evaluation shows that the training time of HET-KG is 3.7x and 1.1x times faster than that of PyTorch-BigGraph and DGL-KE, respectively.

The rest of the paper is organized as follows. Section II summarizes the related work. Section III introduces the preliminaries, problem statement, and our motivation. Section IV describes our method in detail, followed by the system implementation in Section V. We show the experimental results in Section VI, and conclude in Section VII.

II. RELATED WORK

Knowledge graph embedding has been well investigated in recent years, which includes the different types of knowledge graph embedding models, and the different kinds of knowledge graph embedding systems.

Translational Distance Based Embedding Models. TransE [1] is the initial work in this category, which is extended by a series of following works, e.g., TransH [30], TransR [15], and TransD [9]. TransE represents entities and relations as vectors in the same space. TransH embeds entities and relations into different hyperplanes, thus solving the problem that the vectors of entities with different types have similar distances during multi-relational embedding process, which cannot be solved in TransE. TransR employs a hyperspace corresponding to each relation rather than a hyperplane, which makes TransR particularly successful in modeling complex realations but sacrifices simplicity and efficiency of TransH. TransD uses projection vectors to replace the projection matrix in TransR, which significantly reduces time complexity while achieving the same effect as TransR.

Semantic Matching Based Embedding Models. There are several representative works in this line of research, including RESCAL [21], DistMult [31], HolE [20], and ComplEx [27]. RESCAL associates each entity with a vector to capture its latent semantics, and represents each relation as a matrix that models the pairwise interactions between factors. DistMult restricts the matrices mentioned in RESCAL to diagonal

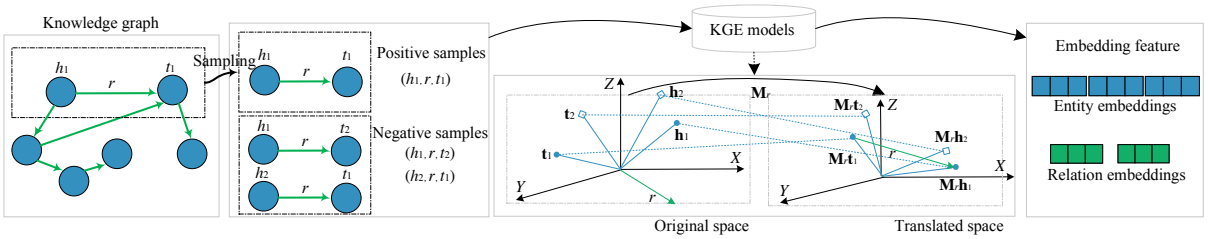


Fig. 1. The training process of KGE.

matrices to simplify RESCAL, which allows DistMult to handle only symmetric relations. HolE uses a circular correlation operation to compress pairwise interactions in RESCAL, which makes it have both the expressive power of RESCAL and the simplicity of DistMult. ComplEx extends DistMult by providing complex-valued embeddings, allowing it to model asymmetric relations more accurately.

Standalone Embedding Systems. KB2E [15] is a graph embedding toolkit that integrates a unified implementation of various KGE models. OpenKE [7] applies GPU acceleration and parallelization mechanisms to speed up the training process based on KB2E. However, OpenKE remains a memory-based framework on a single machine and cannot support representation learning of large-scale knowledge graphs. GraphVite [36] is a hybrid CPU-GPU system for training node embedding that reduces the cost of synchronization between CPUs and GPUs through an effective collaboration strategy, enabling it to be up to 50 times faster than previous systems without sacrificing performance. In order to support large-scale knowledge graphs, Mohoney [19] et al. proposed Marius, a single-machine system for efficiently training of graph embeddings. Marius implements the graph embedding training on a single machine, which suffers from the PCIe bandwidth bottleneck between CPU and GPU. It treats a graph as a partitioned matrix and swaps partitions based on the proposed traversal ordering algorithm.

Distributed Embedding Systems. PyTorch-BigGraph [13] (PBG) uses graph partitioning to train large embeddings on either a single machine or a cluster, which allows it to scale to graphs with billions of edges. However, these approaches suffer from high data-transfer overheads and low computational efficiency. DGL-KE [34] proposes various novel optimization strategies to increase data locality, reduce network transmission, and achieve high operation efficiency. DGL-KE applies multi-processing, multi-GPU, and distributed parallelism to accelerate the training of knowledge graphs with millions of nodes and billions of edges, providing a 2-5x speedup compared to GraphVite and PBG.

Remarks. The above embedding models or standalone systems tend to reach the bottleneck due to the expensive cost when the scale of knowledge graph grows. Although the above distributed embedding systems can alleviate the computational challenge and improve the efficiency of embedding big knowledge graphs, they still face challenges in high network communication overhead. This is because the existing distributed systems simply combine the knowledge embedding models and parameter servers. Thus, the communication cost will become expensive with the increase of number of workers, especially in a low bandwidth network environment. To the best of our knowledge, there is a lack of an optimization method for distributed knowledge graph embedding to improve the training efficiency without sacrificing the training accuracy.

III. Preliminaries and Problem Statement

In this section, we will present the preliminaries, revisit two existing studies of distributed embedding systems, and describe the targeted problem to solve in this work.

To make the presentation clear, in this paper we use a bold lower-case letter $\mathbf{x}$ to represent an embedding vector, and $[\mathbf{x}]_i$ denotes the i-th entry of $\mathbf{x}$. $\|\mathbf{x}\|_{l_j}$ denotes l_j norm of vector $\mathbf{x}$, e.g., $\|\mathbf{x}\|_{l_1}$ is the l_1 norm of $\mathbf{x}$. $diag(\mathbf{x})$ represents diagonal matrix with its i-th diagonal entry denoted as $[\mathbf{x}]_i$. $\mathbf{M}$ represents a matrix where its ij-th entry is denoted as $[\mathbf{M}]_{ij}$.

A. Preliminaries

Knowledge Graph. A knowledge graph can be represented as $G = \{(h, r, t) \mid h, t \in E, r \in R\}$, where E is the entity set and R is the relation set. The number of entities and relations are represented as n_e and n_r.

Knowledge Graph Embedding. The goal of knowledge graph embedding (KGE) is to learn vector embedding $\mathbf{h}, \mathbf{t} \in \mathbb{R}^d$ and $\mathbf{r} \in \mathbb{R}^d$, where d is the dimension. In general, KGE models assign a score to each triple (h, r, t) based on the embedding $(\mathbf{h}, \mathbf{r}, \mathbf{t})$. KGE models target to maximize the score of triple $(h, r, t) \in G$ and minimize it for $(h, r, t) \notin G$.

In addition, a loss function is often designed to encourage discrimination between positive triples and negative triples in knowledge graphs. There are two commonly used loss functions:

- the logistic loss

$$\mathcal{L} = \sum_{(h,r,t)\in\xi\cup\xi'} \log(1 + \exp(-y \cdot f_r(\mathbf{h}, \mathbf{t})))$$

- the ranking loss

$$\mathcal{L} = \sum_{(h,r,t)\in\xi} \sum_{(h',r',t')\in\xi'} \max(0, \gamma - f_r(\mathbf{h}, \mathbf{t}) + f_r(\mathbf{h}', \mathbf{t}'))$$

Here, ξ and ξ' are the set of positive triples and negative triples, respectively. y is the label of a triple where y is assigned as $+1$ when (h, r, t) appears in ξ, otherwise it is set as -1. A common strategy to obtain negative samples from real triples is to corrupt a triple by changing its head entity or tail entity. For a triple (h, r, t), replacing its head entity h and tail entity t yields negative samples (h', r, t) and (h, r, t'), respectively, where h' and t' are randomly sampled entities. Potentially, negative samples (h, r', t) can also be obtained by replacing the original relation r with randomly sampled relation r'. As shown in Fig. 1, A KGE model takes a knowledge graph as input, generates positive and negative samples for training, and presents entities and relations to embedding vectors.

B. Revisit PBG and DGL-KE

Traditional knowledge graph embedding methods tend to reach the bottleneck of a single machine system when the scale of a knowledge graph becomes large. To address this challenge, there are two existing distributed knowledge graph embedding systems, which will be revisited in this section.

PyTorch-BigGraph (PBG). The development of PBG focuses on scalability and distributed training on clusters, and its implementation steps include:

(1) divide the adjacency matrix of the knowledge graph into disjoint partitions and store them on the shared file system, then start a lock server for scheduling workers to avoid conflicts;
(2) each worker requests an edge partition from the lock server, load node partitions and edge partitions from the shared file system;
(3) perform training process using multiple threads without inter-thread synchronization, update the entity embeddings locally, and push the gradients of the relation embeddings to a shared parameter server;
(4) save the partitions to the shared file system that are no longer in use.

PBG treats entity embeddings and relation embeddings as the sparse model parameters and the dense model parameters, respectively. However, it will suffer from both high communication overhead and low computational efficiency if the dense relation embeddings are processed by using the above partitioning methods, especially for knowledge graphs with many relations.

DGL-KE. DGL-KE adopts a parameter server for distributed training. The training process starts with a preprocessing step to partition a knowledge graph and follows with mini-batch training, whose specific training steps include:

(1) each worker samples from a local partition to obtain a mini-batch and corrupts positive triples to construct negative triples;
(2) pull embeddings required in the mini-batch from the parameter servers;

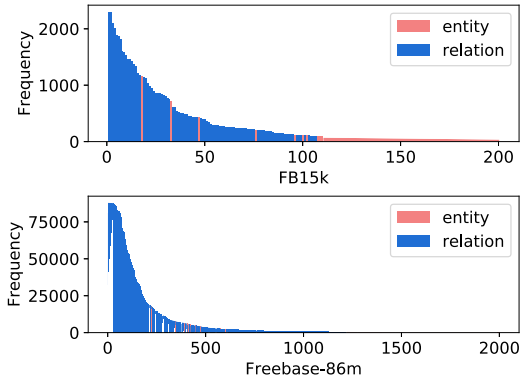


Fig. 2. Distribution of the number of entity and relation embeddings used for KGE in an epoch.

(3) perform forward computation and back propagation to compute the gradients of the embeddings;
(4) push the gradients to the parameter servers, which uses the gradients to update the embddings involved in the mini-batch.

Compared with PBG, DGL-KE performs sparse embedding reads and sparse gradient updates on relation embeddings. This strategy significantly reduces the amount of parameters transferred in distributed training. Nevertheless, DGL-KE only alleviates the problem of high network overhead to a certain extent. As the scale of the knowledge graph and the embedding dimension increase, the proportion of time taken by network communication gradually grows, which is shown in Table I. Specifically, DGL-KE simply uses the parameter server for KGE, maintains all embeddings on the parameter server and conducts computation on the workers. This mechanism may incur high communication cost because in an each iteration the workers need to extract all embeddings used in the training phases from the parameter server. It becomes a major bottleneck for DGL-KE if there is a large number of parameters to be transferred.

C. Our Problem and Motivation

In order to efficiently obtain the embeddings of large-scale knowledge graphs, in this work, our goal is to optimize the distributed training efficiency of KGE without sacrificing accuracy. By so doing, we expect to develop HET-KG that reduces the communication overhead during distributed KGE training.

In order to achieve this goal, we conduct a micro-benchmark to analyze the features of several knowledge graph datasets, and statistics on the frequency of remote communications for different embeddings in an epoch. As shown in Fig. 2, our finding is that only a few critical embeddings are employed regularly in training phases. As a result, our motivation is to reduce communication overhead by preserving these critical embeddings on each worker. However, maintaining crucial embeddings may incur additional issues on how to select and

utilize crucial embeddings in KGE training, and how to manage crucial embeddings to reduce the impact of inconsistency across different replicas.

IV. Methodologies

In this section, we first provide an overview of the HET-KG architecture in Section IV-A. And then, we present the approaches of utilizing "hot-embeddings" to train a KGE model in Section IV-B, and bound the cache staleness in Section IV-C.

A. System Overview

Large-scale knowledge graphs usually have millions of entities and billions of edges. However, as illustrated in Fig. 3, our proposed HET-KG system basically follows the traditional PS architecture. In fact, some of existing knowledge embedding systems also prefer to use the PS because of its excellent concurrency and scalability. Due to the centralized parallel training paradigm of PS, the network communication is apt to become the major bottleneck of the entire training process. In our approach, we adopt the co-located PS architecture to improve the communication efficiency of PS, which has been analyzed in [11]. In co-located PS, each server is co-located with the workers physically to maximize the utilization of network bandwidth. Before the training iterations start, we also partition KG with the METIS [12] algorithm, which has been widely used in other distributed graph learning systems in order to reduce the amount of edges between nodes at different workers.

However, the communication overheads still dominate the KGE training process due to the large amounts of remote embedding access even after the graph partition step. To address this issue, unlike the traditional PS architecture, HET-KG first involves the *hot-embedding table* structure to cache the most frequently used entities and relations embeddings at each worker. Since the embedding access in KGs follows the skew distribution as shown in Fig. 2, HET-KG proposes the prefetching and filtering techniques to adaptively select these hot-embeddings during the training process. After that, HET-KG enables the workers to read these hot-embeddings locally and avoid the majority of remote embedding communication. Furthermore, HET-KG provides a hot-embedding synchronization algorithm, which guarantees the convergence performance.

B. System Workflow

We first introduce the detailed workflow of HET-KG and then discuss the main differences with traditional PS-based KGE training systems. Before the training starts, the KG has been partitioned by the graph partition algorithm (e.g., METIS) and each worker accommodates a subgraph of triples, including both entities and relations. According to the graph partitioning results, the knowledge embeddings are initialized by the corresponding parameter servers on these workers. At this time, the hot-embedding tables are empty and will be dynamically constructed during the training iterations.

Algorithm 1: prefetch

Input : worker id i, subgraph G_i, iteration t, prefetch threshold D
Output: sample list L_s, entity and relation list L_{er}

```
1  Initialize empty lists L_er, L_s;
2  for iteration j ← t, ..., t + D do
3      ξ_j^{i'} ← ∅;
4      ξ_j^i ← sample(G_i);
5      for (h, r, t) ∈ ξ_j^i do
6          ξ_j^{i'} ← ξ_j^{i'} ∪ {(h', r, t), (h, r, t')};
7      for (h, r, t) ∈ ξ_j^i ∪ ξ_j^{i'} do
8          L_er.append(h, r, t);
9      L_s.append((ξ_j^i, ξ_j^{i'}));
10 return L_s, L_er
```

Hot-Embedding Table Construction. In HET-KG, we construct and adaptively adjust the hot-embedding table according to the training workloads. The construction consists of two main processes: *prefetching* and *filtering*. Algorithm 1 presents the details of prefetching. In each iteration, the worker i samples positive triples ξ_j^i from the subgraph G_i (line 4), and generates negative triples $\xi_j^{i'}$ (line 5-6). Then, the worker de-duplicates the entities and relations contained inside each mini-batch and stores the preload results l (line 7-9).

In the filtering algorithm, we use the frequency of entities and relations in the preloaded list as the indicator of their importance. Algorithm 2 presents the process of filtering, the worker counts the frequency of entities and relations in the preloaded list (line 3-8). After that, the entities and relations are sorted in descending order by their frequencies in list L_{num} (line 9). Finally, the hot-embedding identifier table stores the top-k entities and relations (line 10-12), and pull embeddings from the parameter servers (line 13).

A unique challenge in KGE comes from the node heterogeneity of the knowledge graphs. As shown in Fig. 2, the frequency of relations is usually higher than that of entities. Ignoring such node heterogeneity when constructing the hot-embedding table could lead to caching preference on relational embeddings. To address this issue, HET-KG fixes the percentage of entity and relation embeddings in the hot-embedding table to avoid the uneven update frequencies. We study the effect of node heterogeneity in Section VI.

Specifically, we manage to select the most important embeddings to cache and provide two kinds of hot-embedding table construction strategies.

1) Constant partial stale: In order to obtain high-quality hot embeddings, we analyze the sampling process on a knowledge graph. When a uniform sampler generates the samples from knowledge graphs, the entities with more connected edges and relations that appear more frequently would be more likely to be fetched. Taking the FB15k dataset as an example,

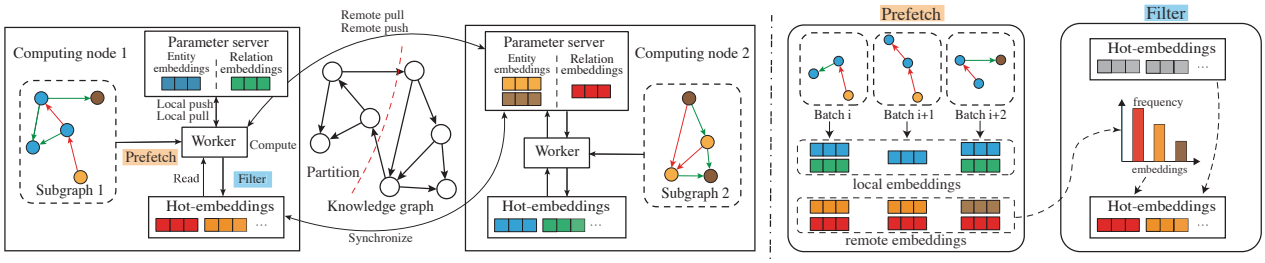


Fig. 3. The overview of HET-KG.

Algorithm 2: filter

Input : entity and relation list L_{er}, cache table size k
Output: hot-embedding id set S_{er}, hot-embedding set S_{emb}

$S_{er}, S_{emb} \leftarrow \emptyset$;
Initial an empty list L_{num};
foreach id *in* L_{er} **do**
  if $id \notin S_{er}$ **then**
    $L_{num}[id] \leftarrow 1$;
    $S_{er} \leftarrow S_{er} \cup \{id\}$;
  else
    $L_{num}[id] \leftarrow L_{num}[id] + 1$;
// Count the frequencies
$L_{er} \leftarrow$descSort(L_{num}, L_{er});
// Sort L_{er} by the frequencies in L_{num} in descending order
for $j \leftarrow 1, \ldots, k$ **do**
  $id \leftarrow L_{er}[j]$;
  $S_{er} \leftarrow S_{er} \cup \{id\}$;
$S_{emb} \leftarrow$ pull(S_{er}) ; // in Algorithm 4
return S_{er}, S_{emb}

the top 1% of entities and relations with the highest access frequency occupy 6% and 36% of the embedding usage, respectively. Based on this observation, HET-KG prefetches the entire subgraph and counts the frequency of all entity and relation embeddings. Since each worker only has limited memory space, we filter the top-k hot-embeddings to construct the cache embedding table. It is intuitive that caching high-frequency embedding will yield greater benefits in terms of reducing more network communications. With constant partial stale (CPS), hot-embeddings in the cache are fixed and will not be replaced during the training process.

2) Dynamic partial stale: In constant partial stale, we assume that the distribution of embedding visits in each mini-batch is similar to the global distribution. Therefore, the top-k hot-embeddings are identified before training. In fact, due to the randomness of sampling, hot-embeddings in some mini-batches may be those that appear less frequently in global embeddings, which makes the lower bound of the CPS strategy lower. To solve this problem, dynamic partial stale (DPS) prefetches D consecutive iterations of input samples in advance, filters the top-k embeddings based on the access frequency, and updates the hot-embedding table. Consequently, the DPS strategy could improve the cache hit ratio as it guarantees that the hot-embeddings reflect the embedding accessing pattern in a shorter term and further improves the reduction of network communications.

Hot-Embedding Oriented Training. With the hot-embedding table, each worker could avoid large amounts of remote communications for the hot-embeddings. During the training iterations, each worker iterates on the following workload:

(1) sample from a local partitioned subgraph to obtain a mini-batch of data samples and corrupt positive triples to construct negative samples;
(2) load the entity and relation embeddings from the hot-embedding table and pull the remaining required embeddings from the parameter server for the current mini-batch;
(3) perform forward computation and back propagation to compute the gradients of the embeddings;
(4) update the corresponding gradients to the involved hot-embeddings and push all the embedding gradients of this iteration back to the parameter server.

Note that, for CPS, the hot-embedding table is fixed during the training iterations. While for DPS, the hot-embedding table will be periodically updated, i.e., every D iterations, to achieve higher cache hit ratio. Fig. 4 presents an example to show the workflow of CPS and DPS ($D = 3$). For a KGE training containing 6 iterations, CPS constructs the hot-embedding table before training and uses the fixed table for all 6 iterations. For DPS with $D = 3$, the worker reconstructs the hot-embedding table every 3 iterations. Compared to CPS, the dynamic hot-embedding construction mechanism results in a higher average cache hit ratio for DPS.

The key difference between the existing KGE training systems (e.g., DGL-KE) and HET-KG is the stage of pulling embeddings. HET-KG enables each worker to read hot-embeddings in the table that could be stale compared to their replicas on the other workers. Although using the hot-embeddings could significantly reduce the network communication, it might affect the final model quality. In the worst case, the worker may miss all the related embedding information from the other workers. To balance the trade-off between the communication efficiency and the model quality, we propose a hot-embedding

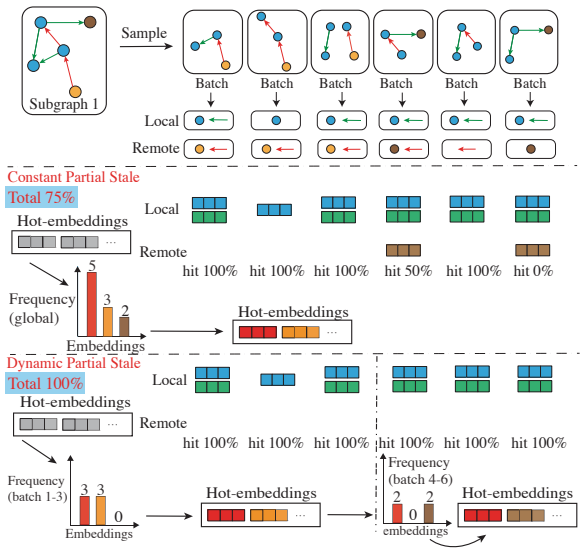


Fig. 4. Example of CPS and DPS ($D = 3$).

synchronization algorithm to guarantee the convergence performance.

Hot-Embedding Synchronization Algorithm. We manage to maximize the local access of the hot-embeddings and minimize the inconsistency between the local hot-embeddings and their replicas on the other workers. Based on this target, HET-KG proposes a hot-embedding synchronization algorithm for the distributed training process. Our intuition is to periodically synchronize hot-embeddings with the parameter server to prevent the hot-embedding table from the staleness extension over a given threshold.

Algorithm 3 and 4 present the details of how our method works. In partial stale, workers undertake distributed training tasks using the ASP protocol. Before training, the worker divides knowledge graph G into n partitions. When the iteration reaches the fetch threshold D, the worker prefetches samples, entities, and relations from the subgraph G_i, and then constructs the hot-embedding table (line 5-7). When the iteration reaches the specified synchronization threshold, the latest version of the hot-embeddings are pulled from the parameter server and are used to update the caches (line 8-9).

Subsequently, the worker loads samples from preload list L_{er}, fetches embeddings from the cache and the parameter server (line 10-13), and then computes the loss function (line 14-16). When the loss $\mathcal{L} > 0$, i.e., the update condition is satisfied, the workers begin to compute gradients of the embeddings using back propagation, and then push back gradients to the parameter server (line 17-19). The parameter server continuously fetches the elements of the message queue and employs the AdaGrad [3] optimizer to update the embedding using gradients (line 1-4), or pushes the embedding to the target worker (line 5-7).

Algorithm 3: Hot-embedding Synchronization (Worker)

Input : knowlege graph G, max number of training iterations T, number of workers n, learning rate ℓ, margin γ, batch size m, hot-embedding table size k, prefetch threshold D, synchronization threshold K

Output: Embedding set S

1 $G_1, \ldots, G_n \leftarrow$ partition(G);
2 **foreach** *Worker* $i \leftarrow 1, \ldots, n$ **do**
3 $S_{ca}, S_{rem} \leftarrow \emptyset$;
4 **for** *iteration* $j \leftarrow 0, \ldots, T-1$ **do**
5 **if** $t \mod D = 0$ **then**
6 $L_s, L_{er} \leftarrow$ prefetch(i, G_i, t, D);
7 $S_{er}, S_{emb} \leftarrow$ filter(L_{er}, k);
8 **if** $t \mod K = 0$ **then**
9 $S_{emb} \leftarrow$ pull(S_{er}); `// in Algorithm 4`
10 $\xi_j^i, \xi_j^{i'} \leftarrow L_s[t \mod D]$;
11 **foreach** $(h, r, t) \in \xi_j^i \cup \xi_j^{i'}$ **do**
12 $S_{ca} \leftarrow S_{ca} \cup (S_{emb} \cap \{\mathbf{h}, \mathbf{r}, \mathbf{t}\})$;
 `// Load hot-embeddings`
13 $S_{rem} \leftarrow$ pull$(S \backslash S_{ca})$;
14 $\mathcal{L} \leftarrow 0$;
15 **for** $j \leftarrow 1, \ldots, m$ **do**
16 $\mathcal{L} \leftarrow \mathcal{L} + \gamma + f_r(\mathbf{h}, \mathbf{t}) - f_r(\mathbf{h}', \mathbf{t}')$;
17 **if** $\mathcal{L} > 0$ **then**
18 $g_t \leftarrow \nabla \mathcal{L}$;
19 push(g_t) ; `// in Algorithm 4`
20 $S \leftarrow$ pull$(E \cup R)$;
21 **return** S

Besides the embedding updates, we highlight another unique challenge in knowledge graph embedding training as follows. It comes from the node heterogeneity of the knowledge graph. As shown in Fig. 2 of Section III, the frequency of relations is usually higher than that of entities. Ignoring such node heterogeneity when designing the caching algorithm could lead to caching preference on relational embeddings. Then it results in uneven update frequencies and affects the model convergence performance. In contrast, HET-KG takes both entities and relations in the knowledge graph into account and caches them using separate mechanisms.

Discussion on existing deep learning frameworks. HET-KG applies the hot-embedding caching to accelerate knowledge graph embedding training. Existing works have explored similar optimizations in deep learning frameworks. For example, Morteza et al. [22] proposed to cache the input data into GPU over PyTorch to accelerate GCN training and perform sampling inside the GPU, based on the original training pipeline. Meanwhile, existing works [32], [35] only cache the frequently accessed raw data (e.g., photos, features), which are quite straightforward and cannot meet the requirement for our

Algorithm 4: Hot-embedding Synchronization (Server)

Function push (g_t) **:**
  foreach $\mathbf{x} \in S$ **do**
    $w_{t+1} \leftarrow w_t + g_t \odot g_t$;
    $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \frac{\ell}{\sqrt{w_t+\varepsilon}} \odot g_t$;
    // Update embeddings with AdaGrad
Function pull (S_{er}) **:**
  $S_{emb} \leftarrow$ getEmb(S_{er});
  return S_{emb};

KGE settings. Unlike these studies, in HET-KG, both cached embeddings and non-cached embeddings will be continuously updated during the training process. It not only requires to find the most frequently accessed embeddings to utilize the limited memory space, but also has to handle the embedding inconsistency problem due to the updates, which have been successfully addressed in our approach.

Discussion on related systems. HET [18] also applies the embedding caching technique in the domain of embedding and graph neural networks. However, HET is different from HET-KG in three aspects: 1) Different research problems: HET targets recommendation systems while HET-KG is for knowledge graph embedding training. As aforementioned, the node heterogeneity makes existing methods designed for general graphs fail on knowledge graphs. 2) Different caching strategies: HET adopts the standard LFU and LRU strategies while HET-KG introduces a novel prefetching and filtering step to dynamically determine the optimal caching embeddings during the training process. Such design makes our cache hit ratio much higher than those strategies, as shown in Table VI. 3) Different cache consistency protocols: Unlike relying on the complex fine-grained embedding clocks in HET, we devise a novel hot-embedding synchronization algorithm in HET-KG, which is simple but effective. It only relies on coarse-grained execution steps, which has shown significant performance improvements and could be easily integrated into the existing systems like DGL-KE.

C. Convergence Analysis

The main challenge of using partial stale synchronous algorithm is the sacrifice of accuracy due to inconsistency between local cache embeddings and the global embeddings on accuracy during asynchronous training [8]. In each training stride, the worker pulls the latest version of hot-embeddings in the first iteration, using them until the next training stride. At this stage, even if these hot-embeddings are updated by other workers, the current node will still use the stale version of the hot-embeddings for training. HET-KG brings such asynchrony into the training process and bounds the degree of hot-embeddings asynchrony with a fully synchronization step per P iterations. Bounded staleness is an effective technique for mitigating the convergence problem by employing lightweight synchronization and has been widely used in many distributed training algorithms [8], [14], [17] and systems [18], [19], [26].

The theoretical guarantee and analysis on the staleness bound is the most crucial part in asynchronous algorithms and systems (e.g., SSP [8]). With such a staleness bound, it is easy to analyze the impacts on the variance of the gradients, which can be used to measure the difference between stale model and the fully synchronous model during the convergence process. Given the bounded staleness assumption, the ergodic convergence rate for Algorithm 3 and 4 achieves $O(1/\sqrt{mT})$, where m is the batch size and T is the iteration number.

Proof Sketch. Before our analysis, we make the following assumptions, which are commonly used in the existing work [8]:

(1) the stochastic gradient g is unbiased;
(2) the variance of stochastic gradient is bounded by σ;
(3) the gradient function $\triangledown(\cdot)$ is L-Lipschitzian;
(4) the delayed model versions are bounded by K.

Under the above assumptions, we could prove that the upper bound of the ergodic convergence rate [14] is $\mathcal{O}(\frac{2(f(x_1)-f(x^*))+\ell_2}{m\ell_1})$ (ℓ_1 and ℓ_2 are functions of learning rate ℓ), when the staleness bound K has certain relationship with ℓ. After setting the learning rate ℓ to a constant, we could further prove that the upper bound of the convergence rate is $4\sqrt{\frac{(f(x_1)-f(x^*))L}{mT}}\sigma$, when $T \geq \frac{4(f(x_1)-f(x^*))mL}{\sigma^2}(K+1)^2$. Therefore, when the total iteration number T is greater than $\mathcal{O}(K^2)$, the convergence rate achieves $\mathcal{O}(1/\sqrt{mT})$.

V. Implementation

In this section, we describe the implementation details of our prototype system, which is built on top of PyTorch [13] and DGL-KE [34], and relies on basic operations in DGL [28], such as communication, sampling, shared memory, etc., and deep learning libraries such as PyTorch for tensor operations.

Cache Implementation. Our proposed approach relies on caches to maintain hot-embeddings. Distributed KGE in this paper is executed using an asynchronous parallel algorithm, so we maintain a cache at each worker independently, which is updated every P iterations. P is defined as a hyperparameter, and the hot-embeddings stored in the cache will be determined by the partial stale synchronous algorithm. In constant partial stale, before the training starts, the worker first prefetches the mini-batch in all subsequent iterations and uses the top k embeddings with the highest number of occurrences as the fixed hot-embeddings. During the training process, the workers pull the latest version of the hot-embeddings from the parameter server every P iterations and update the caches. In dynamic partially stale, the workers prefetch the samples of the following D iterations and use the top k embeddings with the highest number of occurrences among them as the hot-embeddings for the subsequent D training iterations. The workers pull hot-embeddings from the parameter server every

P iterations and reconstruct the cache embedding table after D iterations.

Parameter Server. We use the traditional parameter server architecture for distributed training. The parameter server is implemented by the C++-based distributed key-value store (KVStore) provided in DGL, where the complete embeddings are automatically divided into multiple parts and stored in the training cluster. HET-KG starts multiple server processes on each machine for load balancing, and these servers collectively access the entity and relation embeddings stored locally through shared memory.

The communication between parameter server and workers can be divided into two categories. Local communication is used for workers to pull parameters and push gradients from the local server via localPull() and localPush() interfaces, respectively, which are implemented by shared memory. The purpose of remote communication is to pull parameters and push gradients across machines via remotePull() and remotePush() implemented by C++-based interfaces provided by DGL. Workers adopt the asynchronous parallel algorithm for training, and communicate with the parameter server asynchronously without waiting for each other. In order to get a consistent accuracy, workers are fully synchronized after every few thousand mini-batches, the detailed discussion is in [34].

Graph Partitioning. In distributed training, knowledge graph triples are partitioned and stored on different machines, and the triples stored on one machine are classified as local triples and cross triples. For local triples, their head and tail entity embeddings are stored locally, while for cross triples, their head and tail entities may be stored in other machines. To reduce the communication required for the mini-batch training process due to cross-machine pulling of entity embeddings, we use the METIS [12] algorithm to partition the knowledge graph. In a cluster of n machines, the knowledge graph is divided into n partitions by the METIS method, which are stored in the cluster. Compared with random partitioning, METIS significantly reduces the network communication for pulling entity embeddings across machines, and more details are discussed in [34].

Negative Sampling. KGE requires both positive and negative triples for training. Positive triples are sampled directly from the knowledge graph, while negative triples are obtained by randomly corrupting head or tail entities of positive triples. In [1], [15], each positive triple is corrupted b_n times independently. For a d-dimensional embedding, each mini-batch of size b_p, the time complexity of sampling is $O(b_p d(b_n + 1))$. To reduce the complexity, we use a batch negative sampling strategy similar to that in PBG and DGL-KE. Positive mini-batch is divided into k sets with size of b_c, and the triples in the same set are corrupted together. This sampling method reduces the complexity to $O(b_p d + b_p k d/b_c)$.

VI. Experiments

In this section, we conduct experiments on three knowledge graphs, compare HET-KG with the state-of-the-art distributed KGE systems, i.e., PBG and DGL-KE. Then, we show how the configurations of our hot-embedding synchronization algorithm affect the staleness, and discuss the trade-off between accuracy and performance.

A. Experimental Settings

Hardware and Software Setup. The experiments were conducted on a cluster of 4 machines, each with 32 Intel(R) Xeon(R) cores and 512 GB of RAM. The distributed environment consists of 4 machines with a network bandwidth of 1Gbps. The operating system used for the experiments is CentOS 7.6, where the Python version is 3.6.8 and PyTorch version is 1.9.1. When comparing the performance with baselines, the version of systems we use are PBG (v1.0.0) and DGL-KE (v0.1.0).

Datasets. We use three datasets to evaluate the performance of HET-KG against that of DGL-KE and PBG. Table 3 shows various statistics for these datasets. The FB15k and WN18 datasets are standard benchmarks for evaluating KGE methods. FB15k and WN18 are derived from Freebase and WordNet, respectivly. The Freebase-86m (Freebase-86m) dataset is a large-scale knowledge graph, which contains general facts extracted from Wikipedia. We use the same splits by following the evaluation in [25] to deal with FB15k and WN18, and use a 90/5/5 train, validation, and test split in Freebase-86m. All datasets are downloaded from [34].

Baselines. To the best of our knowledge, PBG and DGL-KE are the state-of-the-art knowledge graph embedding systems which support distributed training. Our method relies solely on the CPU for computation. To ensure fair comparison, we do not compare with GPU-based systems, such as GraphVite and Marius.

Knowledge Embedding Models. TransE and DistMult are two of the most prominent works in the translational distance models and semantic matching models, respectively. On the benchmark datasets FB15k and WN18, we use TransE and DistMult, and on the large-scale dataset Freebase-86m we use TransE. These models are chosen to match the evaluation of Lerer et al. [13] and Zheng et al. [34].

Hyperparameters. In order to ensure a fair comparison, we utilize the same hyperparameters in each system rather than modifying them individually. Table II shows the hyperparameters utilized in the experiments. The optimizers of the system are all AdaGrad [3] optimizers. Based on past experience, it can get embeddings of greater quality than SGD. The problem with the AdaGrad optimizer is that it needs to save the historical gradients of each parameter separately, which increases the memory usage.

Evaluation Metrics. We evaluate the performance of the different KGE models using a link prediction task. Datasets

TABLE II
KNOWLEDGE GRAPHS USED FOR EVALUATION.

Dataset	# Vertices	# Edges	# Relations	Size	Hyperparameters
FB15k	14,951	592,213	1345	52 MB	$d = 400, lr = 0.1, b = 32, n_t = 8$, FilteredMRR
WN18	40,943	151,442	18	7.8 MB	$d = 400, lr = 0.1, b = 32, n_t = 8$, FilteredMRR
Freebase-86m	86,054,151	338,586,276	14,824	275.2 GB	$d = 400, lr = 0.1, b = 512, n_t = 128, n_e = 1000$

are splited into training, validation, and test subsets. We assess the performance by using the standard metrics [9] of Hit@k ($k \in \{1, 3, 10\}$), Mean Rank (MR), and Mean Reciprocal Rank (MRR). All these metrics are derived by comparing how the score of a positive triple relates to the scores of its associated negative triples. Formally, they are defined as $Hits@k = \frac{1}{|G|}\sum_{i=1}^{|G|}\mathbb{R}_{rank_i \leq k}$, $MR = \frac{1}{|G|}\sum_{i=1}^{|G|} rank_i$, and $MRR = \frac{1}{|G|}\sum_{i=1}^{|G|}\frac{1}{rank_i}$, where n_e is the total number of positive triples and $\mathbb{R}_{rank_i \leq k}$ is 1 if $rank_i \leq k$, otherwise it is 0. Hits@k and MRR are between 0 and 1, whereas MR ranges from 1 to $\Sigma_i^{|G|}|S_i|$.

B. Evaluation on accuracy

1) Results on FB15k: As shown in Table III, PBG achieves the highest training accuracy and also takes the longest time. HET-KG has a similar implementation as DGL-KE, so it obtains a comparable accuracy to DGL-KE with less training time. Since HET-KG is not designed for small knowledge graphs, the optimization technique is less effective. Even so, HET-KG is able to obtain embeddings of comparable quality to the existing state-of-the-art systems in much less time.

TABLE III
LINK PREDICTION RESULTS ON FB15K.

System	Model	MRR	Hits@1	Hits@10	Time(s)
PBG	TransE	0.582	0.429	0.818	1074.1
DGL-KE	TransE	0.570	0.433	0.799	483.7
HET-KG-C	TransE	0.569	0.429	0.804	465.9
HET-KG-D	TransE	0.564	0.422	0.803	418.6
PBG	DistMult	0.681	0.544	0.849	1147.0
DGL-KE	DistMult	0.673	0.560	0.850	1160.2
HET-KG-C	DistMult	0.642	0.510	0.839	731.9
HET-KG-D	DistMult	0.662	0.550	0.836	742.1

2) Results on WN18: Similarly, on the benchmark WN18 dataset, the TransE and DistMult models were trained for 60 epochs on the systems using the same configuration as DGL-KE. Table IV gives the results of link prediction, it can be seen that HET-KG can achieve a higher quality of model training. The proposed methods (both CPS and DPS) have better training efficiency than PBG and DGL-KE for both TransE and DistMult models. Since the WN18 dataset has a fewer number of types of relations, in the cache utilized by HET-KG, the relations appear more densely, and the use of partial stale synchronous algorithm can save more communication overhead and thus reduce the training time while ensuring the accuracy. However, since DPS needs to dynamically prefetch and count the entities and relations in the later iterations of training and modify the contents of the caches, the cost of prefetching is higher than the cost of computing on a small dataset, so the training time using DPS is slightly higher than that using CPS. It is further shown that by prefetching and caching, HET-KG can obtain better model training results when training specific datasets with certain characteristics, such as the number of different relations is much less than that of entities.

TABLE IV
LINK PREDICTION RESULTS ON WN18.

System	Model	MRR	Hits@1	Hits@10	Time(s)
PBG	TransE	0.722	0.545	0.956	477.4
DGL-KE	TransE	0.715	0.548	0.954	184.3
HET-KG-C	TransE	0.720	0.552	0.955	163.0
HET-KG-D	TransE	0.719	0.552	0.954	167.7
PBG	DistMult	0.889	0.840	0.954	1177.6
DGL-KE	DistMult	0.881	0.840	0.931	238.3
HET-KG-C	DistMult	0.877	0.835	0.927	232.1
HET-KG-D	DistMult	0.885	0.845	0.933	251.4

3) Results on Freebase-86m: To verify the applicability and superiority of HET-KG for large-scale knowledge graph training, we train the TransE model on Freebase-86m dataset for 10 epochs. As shown in Table V, efficiency and accuracy of model training using HET-KG are both improved, further demonstrating that using partial stale, HET-KG is able to guarantee performance while thus improving the efficiency of processing large-scale data. For a large-scale knowledge graph dataset like Freebase-86m, HET-KG can guarantee the accuracy of the model training while reducing the time overhead of communication and further improving the efficiency of model training by setting the top-k value larger to cache more hot-embedding with higher frequency ranking at one time, i.e., DPS strategy.

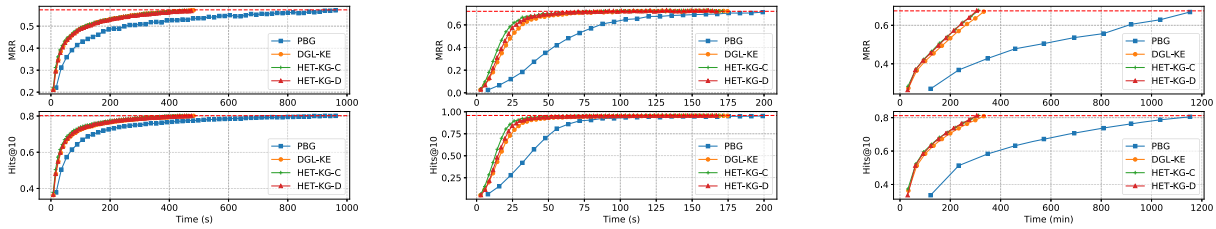
TABLE V
LINK PREDICTION RESULTS ON FREEBASE-86M.

System	Model	MRR	Hits@1	Hits@10	Time (min)
PBG	TransE	0.669	0.602	0.805	1152.7
DGL-KE	TransE	0.671	0.599	0.809	332.9
HET-KG-C	TransE	0.678	0.608	0.813	312.7
HET-KG-D	TransE	0.677	0.605	0.813	305.2

C. Evaluation of training efficiency

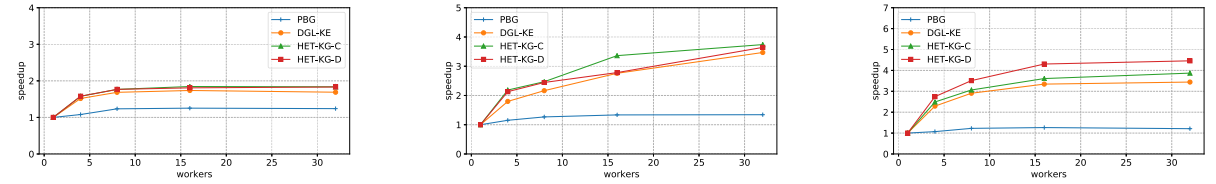
In this section, we verify the effectiveness of the proposed methods (HET-KG-C and HET-KG-D) with the goal of answering the following questions. 1) How does the convergence of HET-KG as compared with the state-of-the-art baselines? 2) How scalable is HET-KG compared with existing baseline methods? 3) Is the proposed Hot-Embedding Oriented Training efficient in distributed knowledge graph embedding?

1) Training convergence: In this subsection, we report the training convergence of baselines and the proposed methods to answer the first question and present our findings. Fig. 5



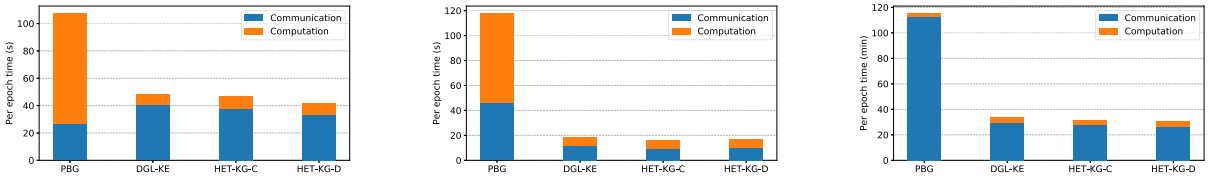
(a) FB15k (b) WN18 (c) Freebase-86m

Fig. 5. Training convergence comparison with PBG and DGL-KE.



(a) FB15k (b) WN18 (c) Freebase-86m

Fig. 6. Scalability comparison with PBG and DGL-KE.



(a) FB15k (b) WN18 (c) Freebase-86m

Fig. 7. Communication and computation cost comparison with PBG and DGL-KE.

depicts the convergence over time for model training and the time required for each epoch, all systems can achieve similar accuracy. The experimental results have exhibit that, compared with the baseline methods, HET-KG requires less time to achieve a comparable accuracy. On the Freebase-86m, HET-KG-D outperforms other methods, which verifies the dynamic partial stale algorithm can achieve better performance on large-scale KGs.

2) Scalability: In this subsection, we will answer the second question. The scalability study is conducted in terms of run-time speedup on Freebase-86m with different number of workers. As shown in Fig. 6, PBG has limited scalability, which suffers from the unnecessary data transfer overhead caused by treating relation embeddings as dense model weights. By contrast, both HET-KG and DGL-KE have more significant changes in the acceleration ratio with the increase of the number of workers, and the average acceleration ratio of HET-KG can be 30% higher than that of DGL-KE.

3) Communication and computation cost: In this subsection, we further break down the per epoch time into computation and communication parts to answer the third question. The computation time, communication time, and total time of the evaluated systems for model training with three datasets are given separately, as shown in Fig. 7. It can be seen that the overheads of DGL-KE and HET-KG are close in terms of computation time, which proves that HET-KG does not adversely affect the efficiency of the model computation while improving the efficiency by prefetching and caching entities and relations. The reduction of communication time also proves that HET-KG can effectively reduce the number of entity and relation embedding transmission during the training process, and thus reduce the communication time. Futhermore, the communication time of PBG far exceeds that of other systems, mainly due to its use of relation embeddings as dense model weights, which increases the amount of parameter transfer.

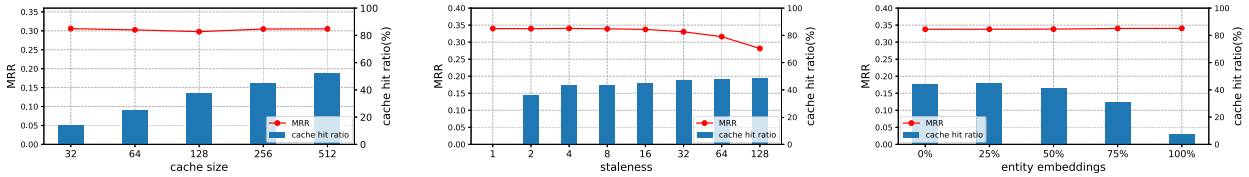
D. Evaluation of HET-KG Solutions

In this subsection, we present the impact of cache size, bouded staleness, and the selecion of hot-embeddings on the Freebase-86m dataset.

1) Impact of cache size: As shown in Fig. 8(a), the cache hit ratio first increases with the increase of cache size, which indicates that caching hot-embeddings in descending order of entities and relations frequency can effectively reduce the amount of remote embedding communication.

In addition, the MRR values do not change significantly, as the percentage of embedding that is delayed remains small relative to the total training size and does not have a significant impact on accuracy, which further demonstrates that the cumulative error caused by the cache of hot-embeddings does not significantly affect the model training accuracy.

2) Impact of bounded staleness: As shown in Fig. 8(b), we evaluate how the performance and MRR vary as we change



(a) Cache size (b) Staleness (c) hot-embedding selection

Fig. 8. Impact of settings on Freebase-86m

the staleness. Experimental results have exhibited that when the staleness $P \leq 8$, MRR is not significantly affected by inconsistency between hot-embeddings and global embeddings. However, the MRR decreases with further increase of staleness P, which ensures that the inconsistency can be bounded within a given threshold. Meanwhile, the cache hit ratio improves while staleness increasing, which needs a trade-off between training efficiency and model quality.

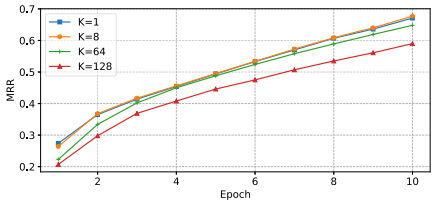


Fig. 9. Impact of synchronization threshold K

We have analyzed the impact of asynchronous training in Fig. 5 in Section VI.C. In order to further motivate the importance of guaranteeing consistency, the epoch-MRR training curves are shown in Fig. 9. When the consistency is guaranteed (e.g., staleness=1), the MRR of the model is 0.67. However, if we relax the consistency guarantee (e.g., staleness=128), the MRR decreases to 0.59, which verifies that the consistency guarantee significantly affects the training convergence.

3) Impact of hot-embedding selection: We further analyze the selection of hot-embeddings, as shown in Fig. 8(c). It shows that the cache hit ratio increases and then decreases as the entity ratio increases, with the highest hit ratio at 25% of the entity ratio. This is mainly due to the fact that relation embeddings are more dense compared to entity embeddings, which coincides with our microbenchmark results in Fig. 2. Therefore, storing more relation embeddings in the hot-embedding table yields better performance optimization.

TABLE VI
CACHE HIT RATIO COMPARISON WITH SIMPLE CACHING TECHNIQUES.

Dataset	FIFO	LRU	Importance cache	HET-KG
FB15k	7.4%	11.6%	15.2%	25.2%
WN18	16.5%	17.6%	32.1%	35.5%
Freebase-86m	6.6%	8.6%	34.5%	43.1%

4) Comparison with simple caching techniques: We show the comparison of the cache hit ratio of HET-KG and several simple caching techniques on three datasets in Table VI. The experimental results demonstrate that the cache hit ratio of HET-KG is much higher than that of simple caching techniques, reflecting the effectiveness of the hot embedding optimization in HET-KG.

5) Impact of node heterogeneity in knowledge graphs: A unique challenge in knowledge graph embedding training comes from the heterogeneity of nodes. As shown in Table VII, we tested HET-KG and HET-KG-N after 30 epochs of training. In HET-KG-N, we build the cache based on the frequency of entities and relations. While in HET-KG, we additionally consider the heterogeneity of nodes and fix the cache content as 25% entity embeddings and 75% relation embeddings. Compared with HET-KG, HET-KG-N achieves better performance but lower accuracy. The experimental results illustrate that although simple caching techniques can also achieve optimization, neglecting the heterogeneity of nodes would result in uneven update frequencies and affect the model convergence performance.

TABLE VII
THE PERFORMANCE OF HET-KG WITH AND WITHOUT HETEROGENEITY OPTIMIZATION.

Dataset	System	MRR	Hits@1	Hits@10	Time(s)
FB15k	HET-KG	0.343	0.249	0.518	236.8
FB15k	HET-KG-N	0.304	0.214	0.472	227.2
WN18	HET-KG	0.629	0.444	0.907	86.0
WN18	HET-KG-N	0.606	0.426	0.870	77.1

VII. CONCLUSION

In this paper, we have extended the traditional PS-based architecture by introducing a cache embedding table structure to reduce the remote communication overheads, and proposed HET-KG, a distributed system for training knowledge graph embeddings. We employed a prefetching mechanism to adaptively select hot-embeddings and dynamically update the cache embedding table. A hot-embedding synchronization algorithm is designed to reduce the number of high-frequency parameter transfers by retaining hot-embeddings, and to constrain the inconsistency between hot-embeddings and global embeddings. Experimental results have verified that HET-KG outperforms the state-of-the-art systems in terms of both efficiency and scalability.

ACKNOWLEDGEMENT

This work is supported by the National Key Research and Development Program of China (No. 2019YFE0198600), the National Natural Science Foundation of China (No. 61972275 and 61832001), PKU-Tencent joint research Lab, and Australian Research Council Linkage Project (LP180100750).

REFERENCES

[1] Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: NeurIPS. pp. 2787–2795 (2013)

[2] Cowie, J., Lehnert, W.: Information extraction. Commun. ACM 39(1), 80–91 (1996)

[3] Duchi, J.C., Hazan, E., Singer, Y.: Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res. 12, 2121–2159 (2011)

[4] Fader, A., Zettlemoyer, L., Etzioni, O.: Open question answering over curated and extracted knowledge bases. In: ACM SIGKDD. pp. 1156–1165 (2014)

[5] Fan, W., He, K., Li, Q., Wang, Y.: Graph algorithms: parallelization and scalability. Sci. China Inf. Sci. 63(10), 1–21 (2020)

[6] Ghose, A.K., Herbertz, T., Salvino, J.M., Mallamo, J.P.: Knowledge-based chemoinformatic approaches to drug discovery. Drug discovery today 11(23-24), 1107–1114 (2006)

[7] Han, X., Cao, S., Lv, X., Lin, Y., Liu, Z., Sun, M., Li, J.: Openke: An open toolkit for knowledge embedding. In: EMNLP. pp. 139–144 (2018)

[8] Ho, Q., Cipar, J., Cui, H., Lee, S., Kim, J.K., Gibbons, P.B., Gibson, G.A., Ganger, G.R., Xing, E.P.: More effective distributed ml via a stale synchronous parallel parallel server. In: NeurIPS. pp. 1223–1231 (2013)

[9] Ji, G., He, S., Xu, L., Liu, K., Zhao, J.: Knowledge graph embedding via dynamic mapping matrix. In: ACL-IJCNLP. pp. 687–696 (2015)

[10] Jiang, J., Xiao, P., Yu, L., Li, X., Cheng, J., Miao, X., Zhang, Z., Cui, B.: Psgraph: How tencent trains extremely large-scale graphs with spark? In: IEEE ICDE. pp. 1549–1557 (2020)

[11] Jiang, Y., Zhu, Y., Lan, C., Yi, B., Cui, Y., Guo, C.: A unified architecture for accelerating distributed dnn training in heterogeneous gpu/cpu clusters. In: USENIX OSDI. pp. 463–479 (2020)

[12] Karypis, G., Kumar, V.: Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices (1998)

[13] Lerer, A., Wu, L., Shen, J., Lacroix, T., Wehrstedt, L., Bose, A., Peysakhovich, A.: Pytorch-biggraph: A large scale graph embedding system. In: MLSys (2019)

[14] Lian, X., Huang, Y., Li, Y., Liu, J.: Asynchronous parallel stochastic gradient for nonconvex optimization. In: NeurIPS. pp. 2737–2745 (2015)

[15] Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: AAAI. pp. 2181–2187 (2015)

[16] Miao, X., Gürel, N.M., Zhang, W., Han, Z., Li, B., Min, W., Rao, S.X., Ren, H., Shan, Y., Shao, Y., et al.: Degnn: Improving graph neural networks with graph decomposition. In: ACM SIGKDD. pp. 1223–1233 (2021)

[17] Miao, X., Nie, X., Shao, Y., Yang, Z., Jiang, J., Ma, L., Cui, B.: Heterogeneity-aware distributed machine learning training via partial reduce. In: ACM SIGMOD. pp. 2262–2270 (2021)

[18] Miao, X., Zhang, H., Shi, Y., Nie, X., Yang, Z., Tao, Y., Cui, B.: HET: scaling out huge embedding model training via cache-enabled distributed framework. Proc. VLDB Endow. 15(2), 312–320 (2021)

[19] Mohoney, J., Waleffe, R., Xu, H., Rekatsinas, T., Venkataraman, S.: Marius: Learning massive graph embeddings on a single machine. In: USENIX OSDI. pp. 533–549 (2021)

[20] Nickel, M., Rosasco, L., Poggio, T.: Holographic embeddings of knowledge graphs. In: AAAI. pp. 1955–1961 (2016)

[21] Nickel, M., Tresp, V., Kriegel, H.P.: A three-way model for collective learning on multi-relational data. In: ICML. pp. 809–816 (2011)

[22] Ramezani, M., Cong, W., Mahdavi, M., Sivasubramaniam, A., Kandemir, M.T.: GCN meets GPU: decoupling "when to sample" from "how to sample". In: NeurIPS (2020)

[23] Song, D., Zhang, F., Lu, M., Yang, S., Huang, H.: Dtranse: Distributed translating embedding for knowledge graph. IEEE TPDS 32(10), 2509–2523 (2021)

[24] Stokman, F.N., de Vries, P.H.: Structuring knowledge in a graph. In: Human-computer interaction, pp. 186–206 (1988)

[25] Sun, Z., Deng, Z.H., Nie, J.Y., Tang, J.: Rotate: Knowledge graph embedding by relational rotation in complex space. arXiv abs/1902.10197 (2019)

[26] Thorpe, J., Qiao, Y., Eyolfson, J., Teng, S., Hu, G., Jia, Z., Wei, J., Vora, K., Netravali, R., Kim, M., et al.: Dorylus: Affordable, scalable, and accurate gnn training with distributed cpu servers and serverless threads. In: USENIX OSDI. pp. 495–514 (2021)

[27] Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., Bouchard, G.: Complex embeddings for simple link prediction. In: ICML. pp. 2071–2080 (2016)

[28] Wang, M., Yu, L., Zheng, D., Gan, Q., Gai, Y., Ye, Z., Li, M., Zhou, J., Huang, Q., Ma, C., et al.: Deep graph library: Towards efficient and scalable deep learning on graphs. CoRR abs/1909.01315 (2019)

[29] Wang, X., He, X., Cao, Y., Liu, M., Chua, T.S.: Kgat: Knowledge graph attention network for recommendation. In: ACM SIGKDD. pp. 950–958 (2019)

[30] Wang, Z., Zhang, J., Feng, J., Chen, Z.: Knowledge graph and text jointly embedding. In: EMNLP. pp. 1591–1601 (2014)

[31] Yang, B., Yih, W., He, X., Gao, J., Deng, L.: Embedding entities and relations for learning and inference in knowledge bases. In: ICLR (2015)

[32] Yang, H.: Aligraph: A comprehensive graph neural network platform. In: ACM SIGKDD. pp. 3165–3166 (2019)

[33] Zhang, L., Li, D., Xi, Y., Jia, S.: Reinforcement learning with actor-critic for knowledge graph reasoning. Sci. China Inf. Sci. 63(6) (2020)

[34] Zheng, D., Song, X., Ma, C., Tan, Z., Ye, Z., Dong, J., Xiong, H., Zhang, Z., Karypis, G.: Dgl-ke: Training knowledge graph embeddings at scale. In: ACM SIGIR. pp. 739–748 (2020)

[35] Zhou, K., Sun, S., Wang, H., Huang, P., He, X., Lan, R., Li, W., Liu, W., Yang, T.: Demystifying cache policies for photo stores at scale: A tencent case study. In: ICS. pp. 284–294 (2018)

[36] Zhu, Z., Xu, S., Tang, J., Qu, M.: Graphvite: A high-performance cpu-gpu hybrid system for node embedding. In: WWW. pp. 2494–2504 (2019)