THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

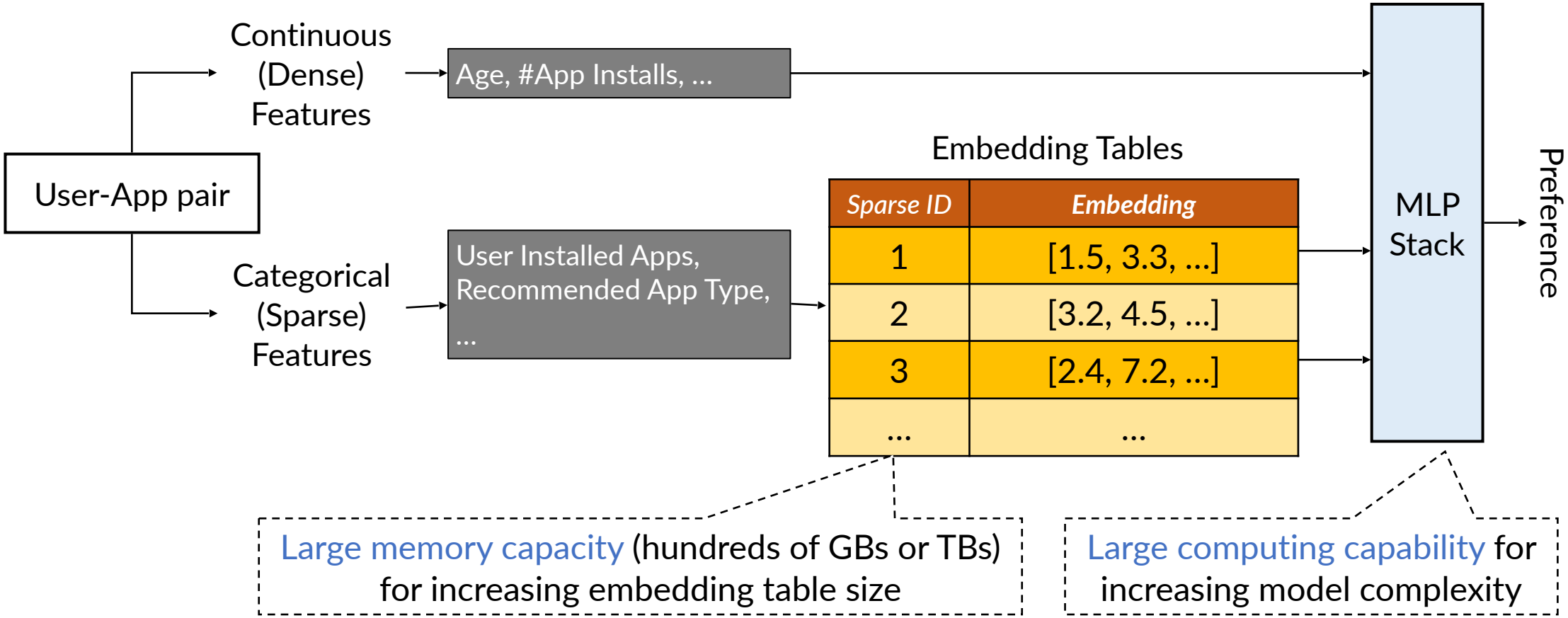
Accelerating Neural Recommendation Training with Embedding Scheduling

Chaoliang Zeng*, **Xudong Liao***, Xiaodian Cheng, Han Tian,
Xinchen Wan, Hao Wang, Kai Chen

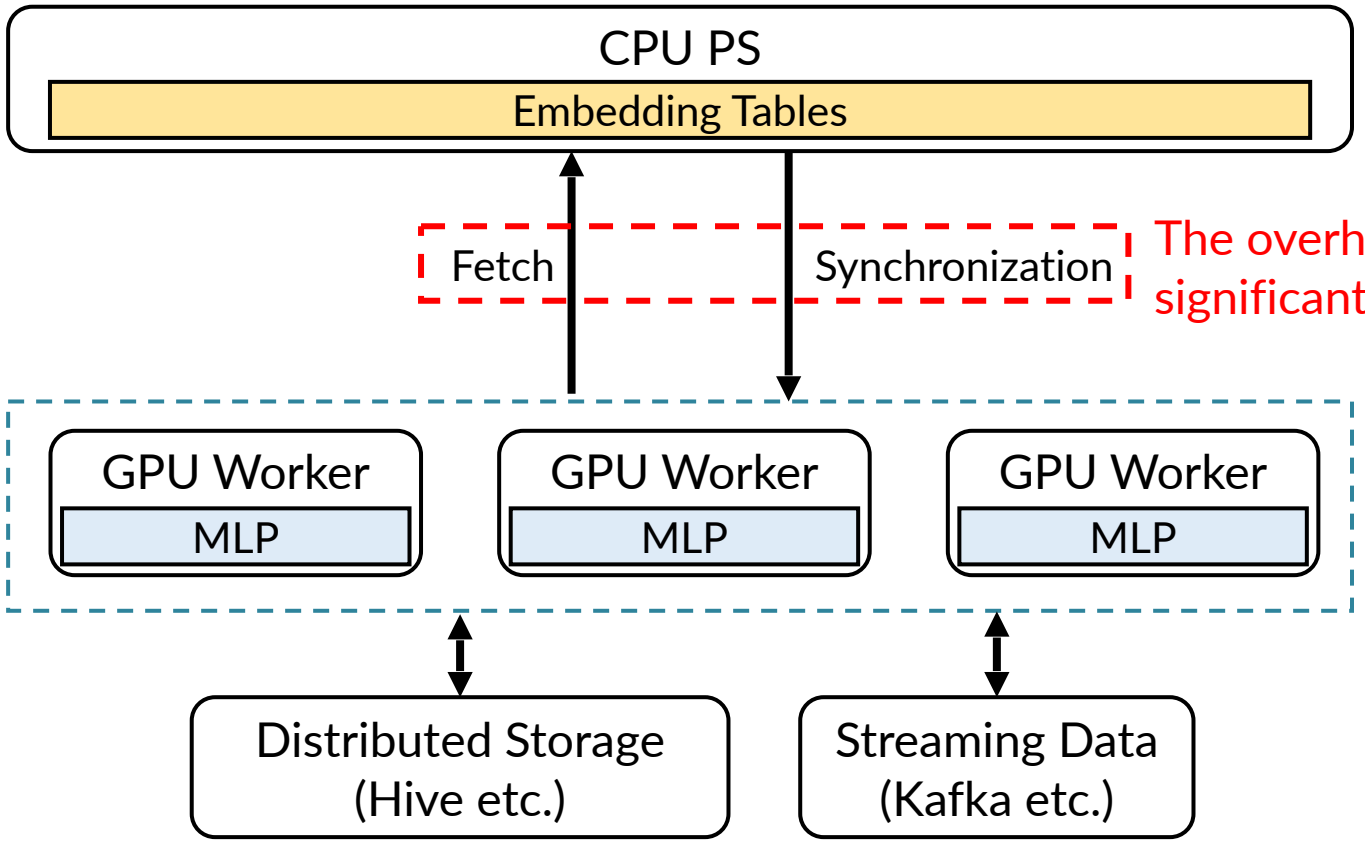
iSingLab HKUST

Neural Recommendation

A high-level architecture of a typical deep-learning recommendation model (DLRM) with the example of apps recommendation



Distributed DLRM Training System



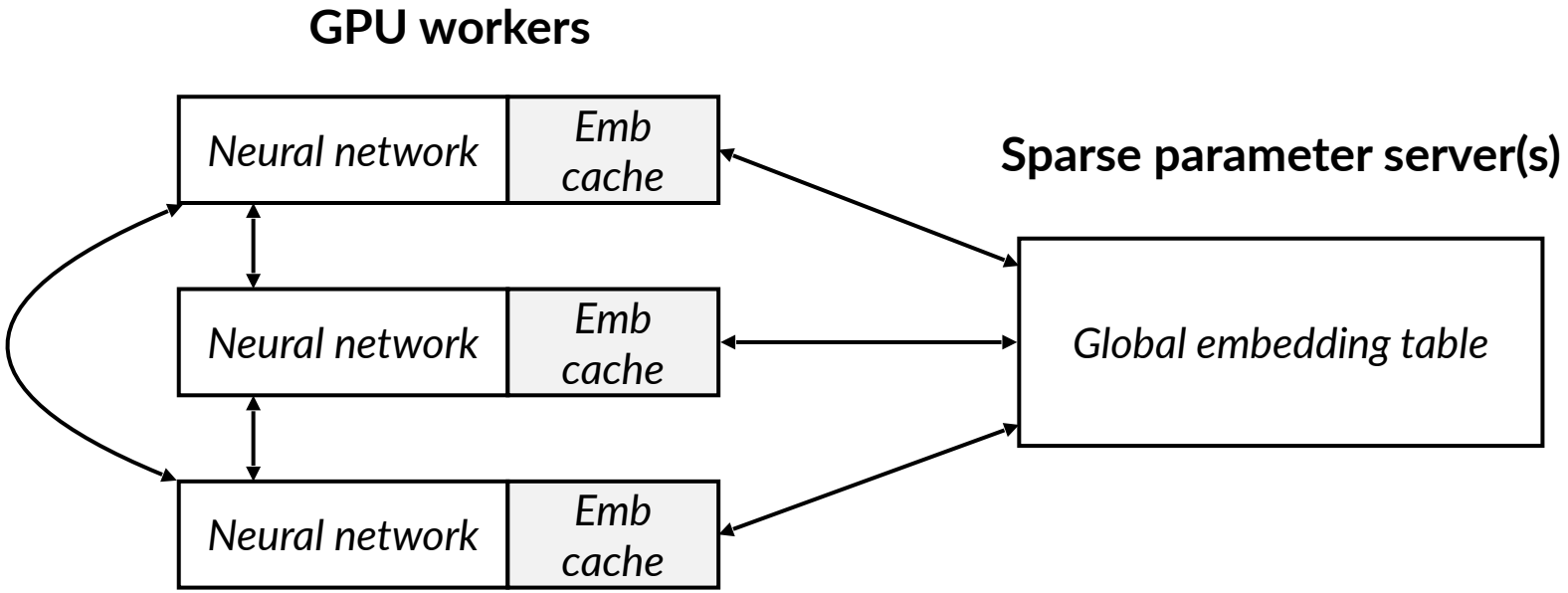
Embedding tables -> Memory-optimized CPU PS

The overhead of embedding transmissions is significant in BSP!

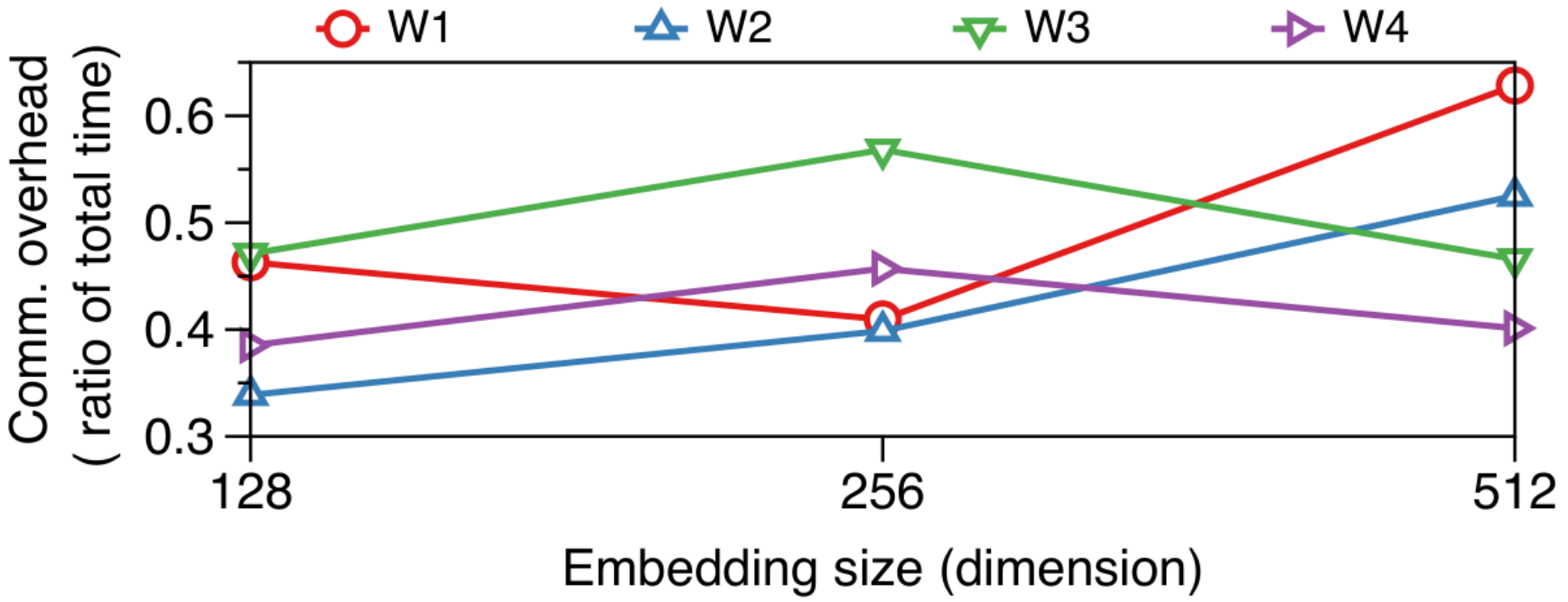
MLP stack -> Compute-optimized GPU

Embedding Cache as a Remedy

Reduce embedding communication with worker local cache



Embedding Communication Matters



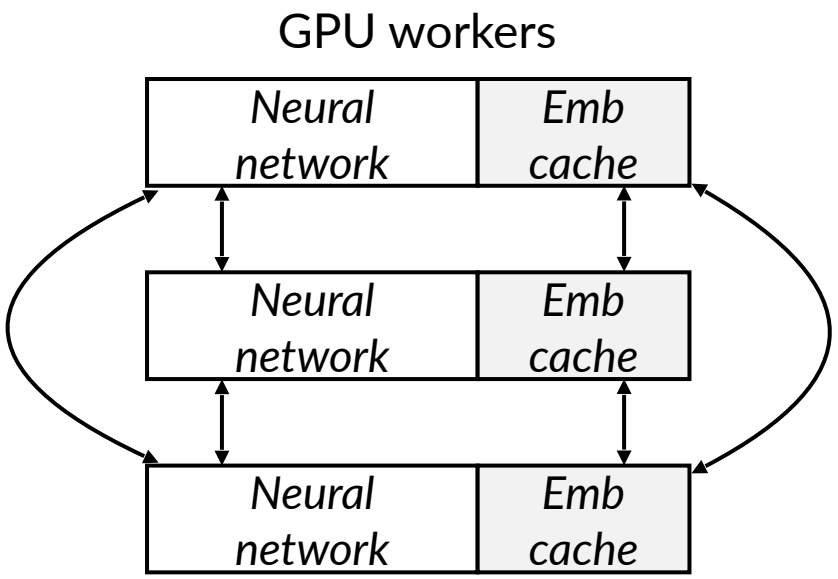
Embedding communication can contribute to up to 63% of training time

Configuration:

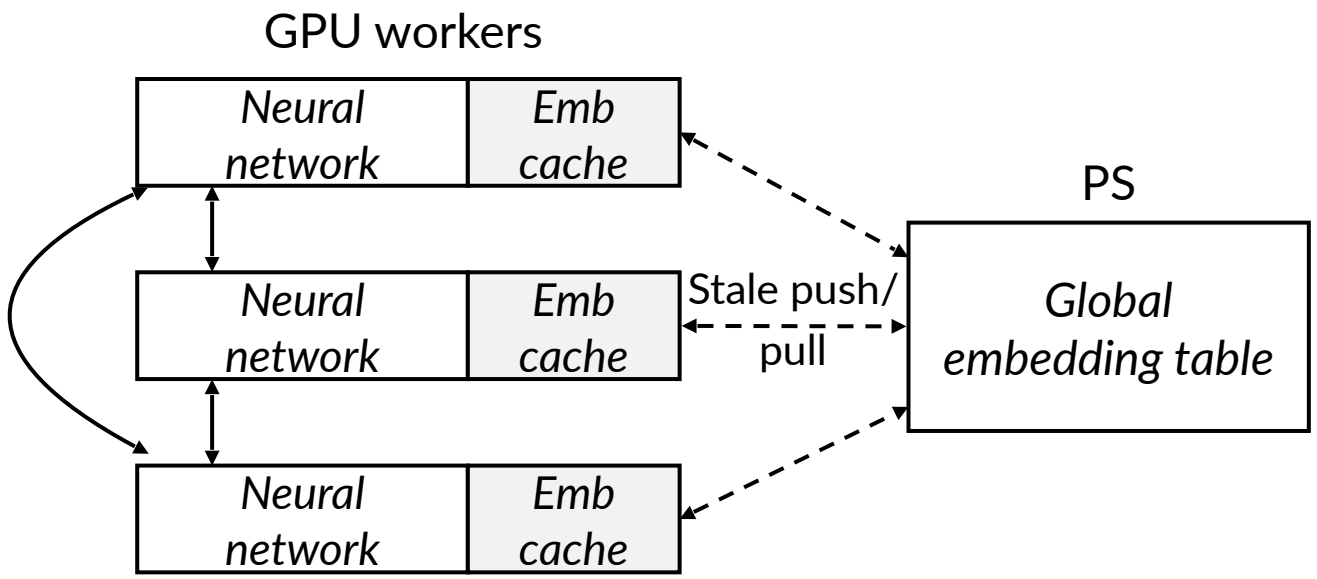
- 8 NVIDIA 3090 GPU workers
- 1 CPU PS
- 100 Gbps Ethernet with TCP

	Model	Dataset
W1	Wide & Deep [DLRS'16]	Criteo AD
W2	Neural Collaborative Filtering [WWW'17]	MovieLens 25M
W3	DeepFM [IJCAI'17]	Avazu
W4	Deep & Cross [ADKDD'17]	Criteo Sponsored Search

Existing Optimizations with Embedding Cache



FAE [1] oversample training data containing only hot embeddings



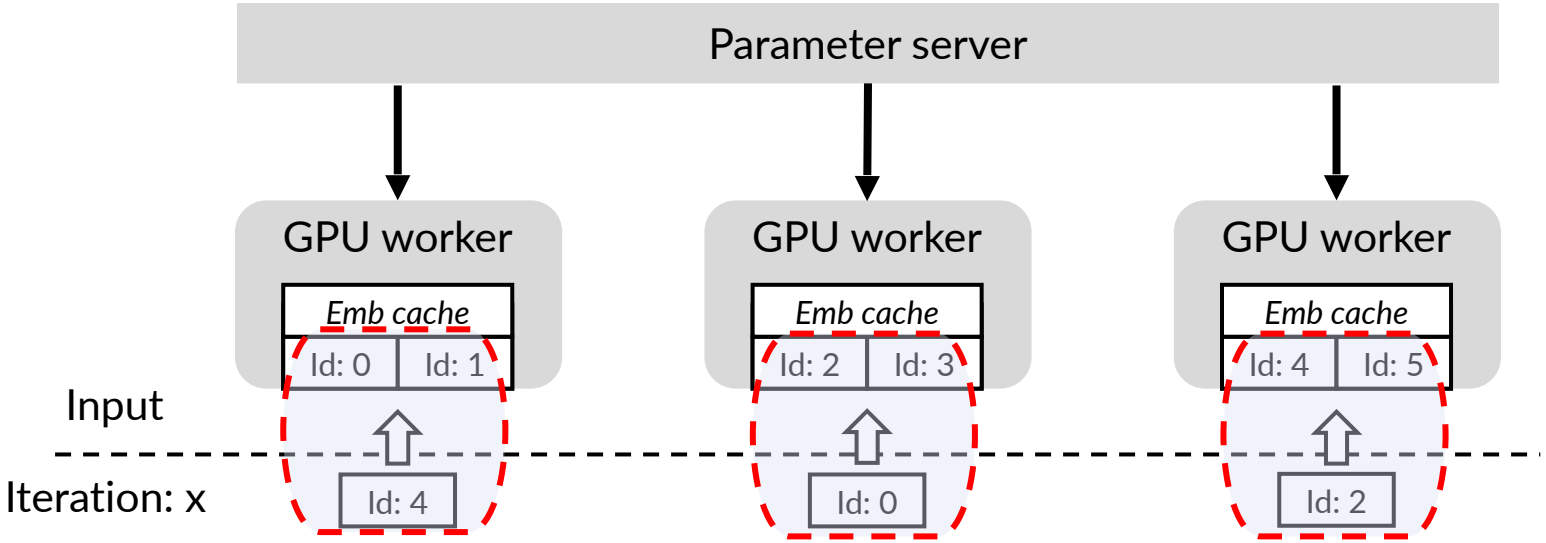
HET [2] applies a staleness-tolerant embedding update method

Existing solutions may hurt model accuracy

[1] M. Adnan, et al. Accelerating Recommendation System Training by Leveraging Popular Choices. VLDB, 2021.
[2] X. Miao, et al. HET: Scaling out Huge Embedding Model Training via Cache-enabled Distributed Framework. VLDB, 2021

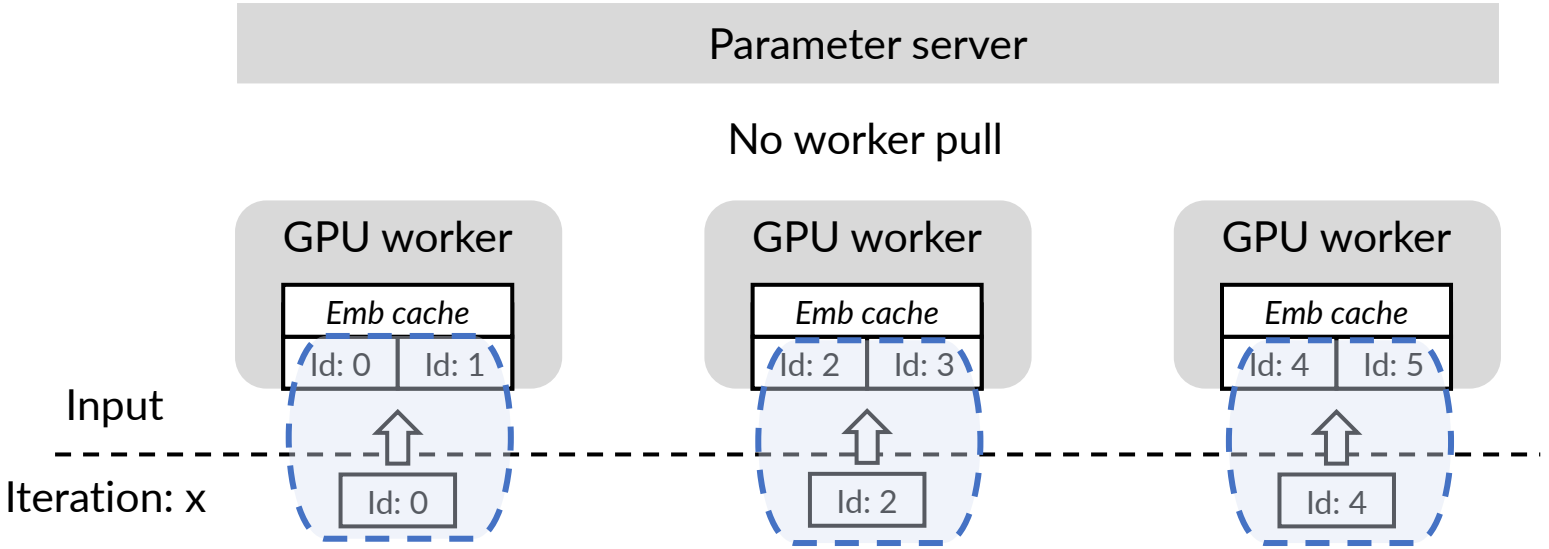
Can we reduce the communication overhead
without compromising the model accuracy?

Revisit Cache Accessing Behavior: Pull



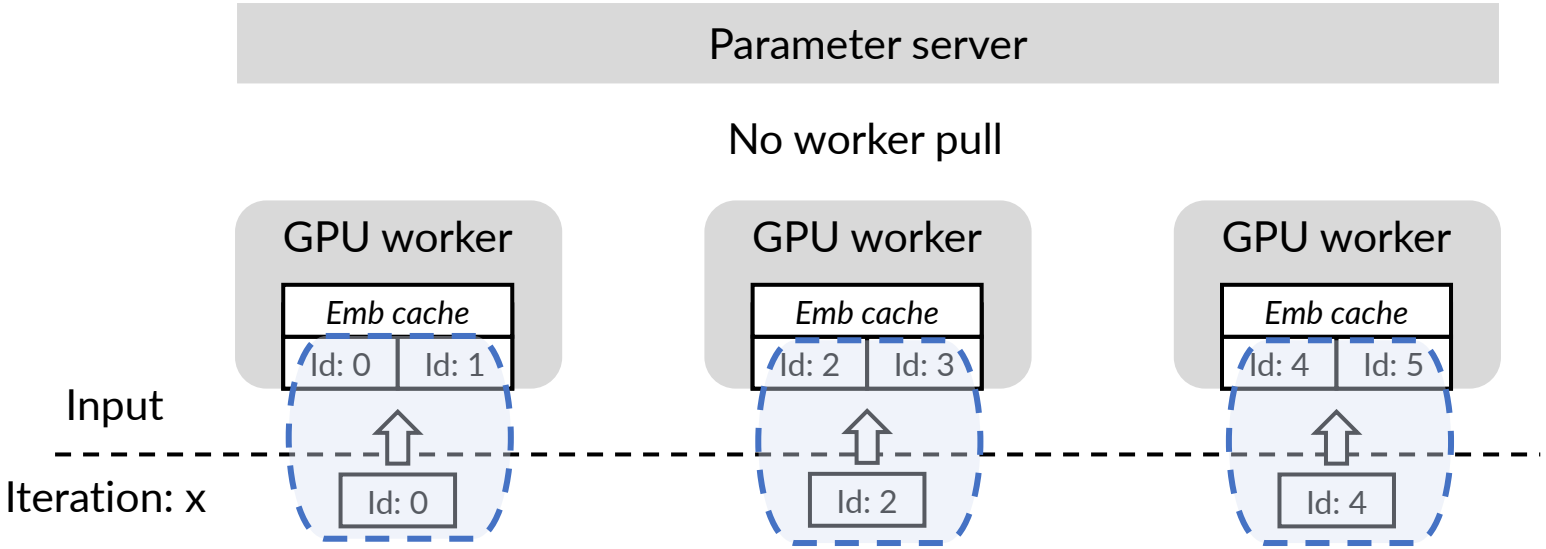
Bad cache accessing leads to worker pull

Opportunity: Input Permutation



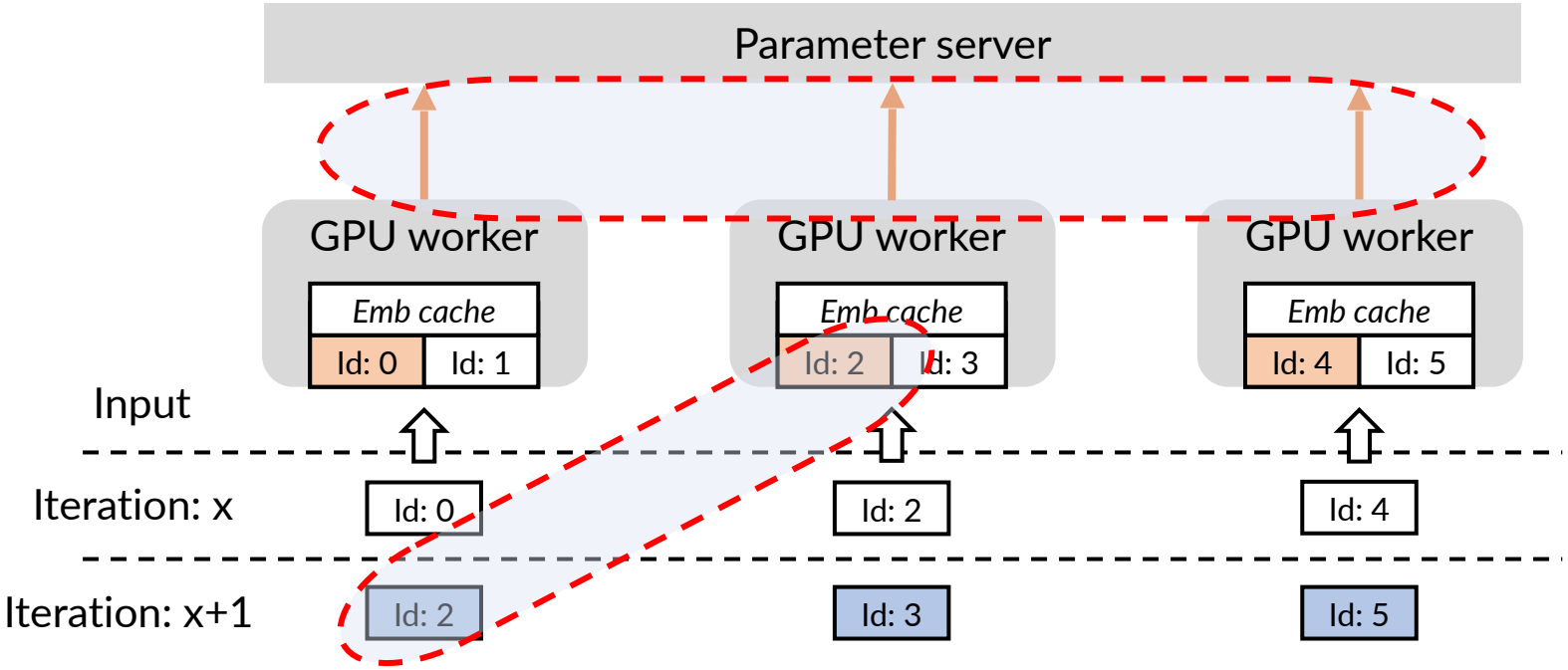
After permutating the input IDs, all samples are trained with in-cache embeddings.

Opportunity: Input Permutation



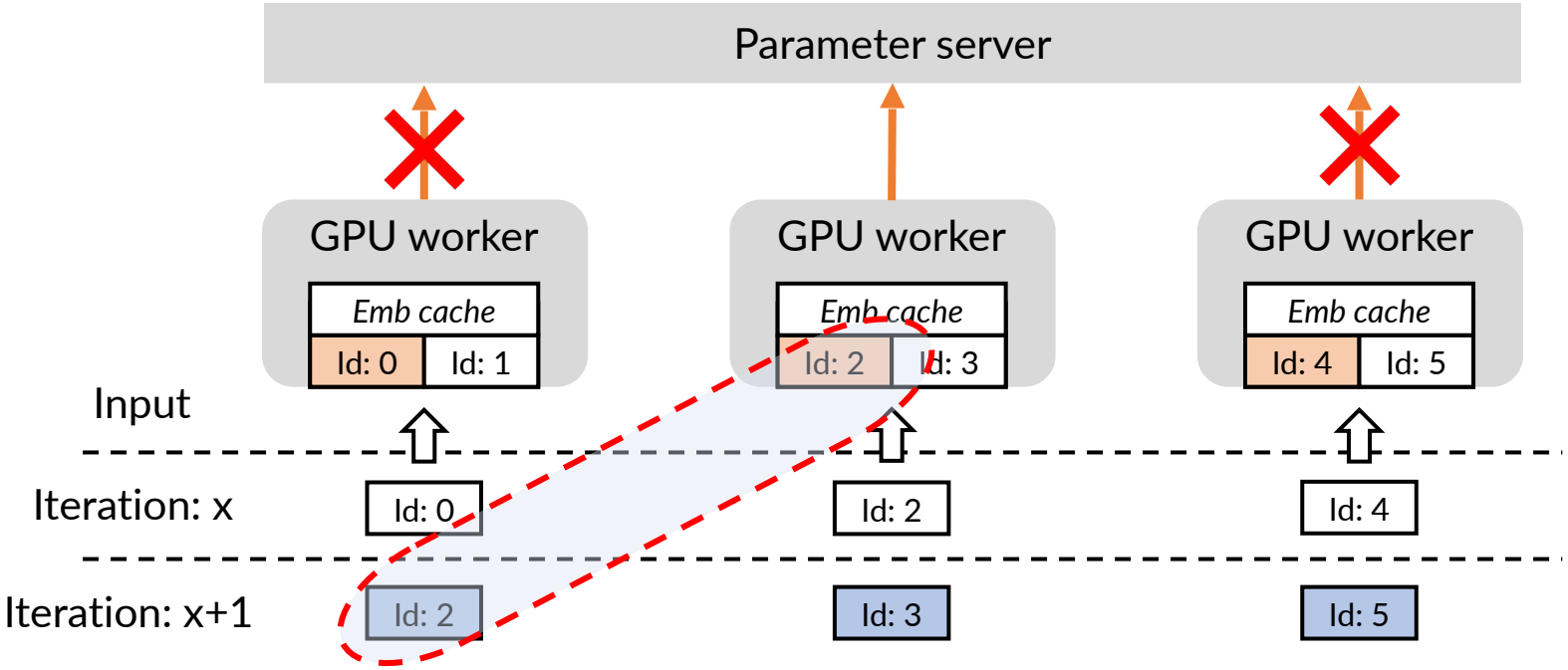
Theoretical analysis proves that this can preserve the model consistency and thus deliver the same model accuracy as vanilla BSP

Revisit Cache Accessing Behavior: Push



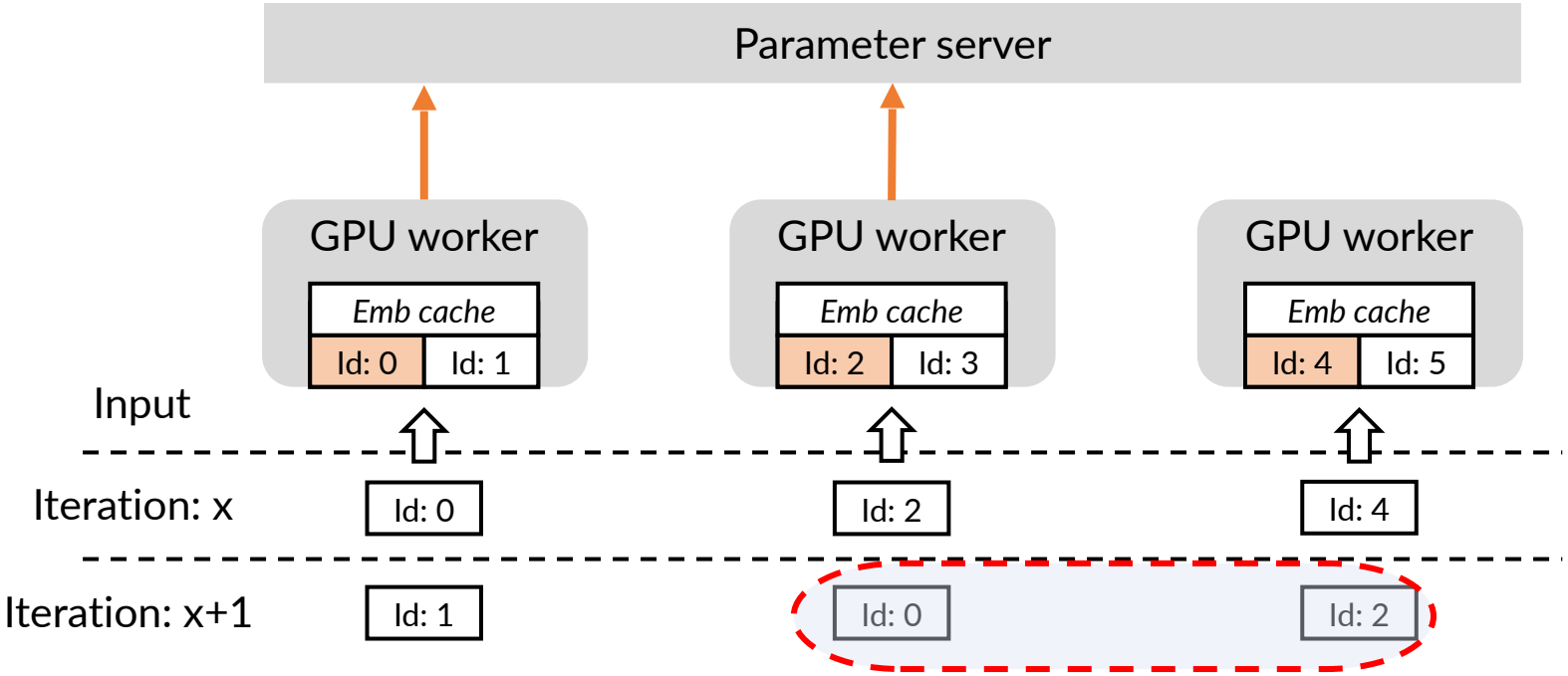
Naïve worker push may introduce useless transfers.

Revisit Cache Accessing Behavior: Push



Naïve worker push may introduce useless transfers.

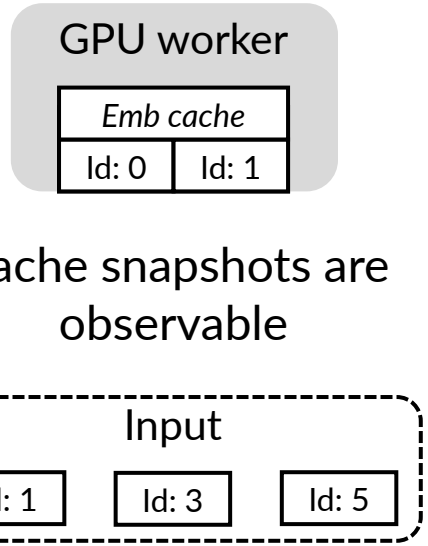
Opportunity: On-demand Synchronization



We only need to do necessary synchronizations by considering future batches.

Observations of In-cache Embedding Accesses

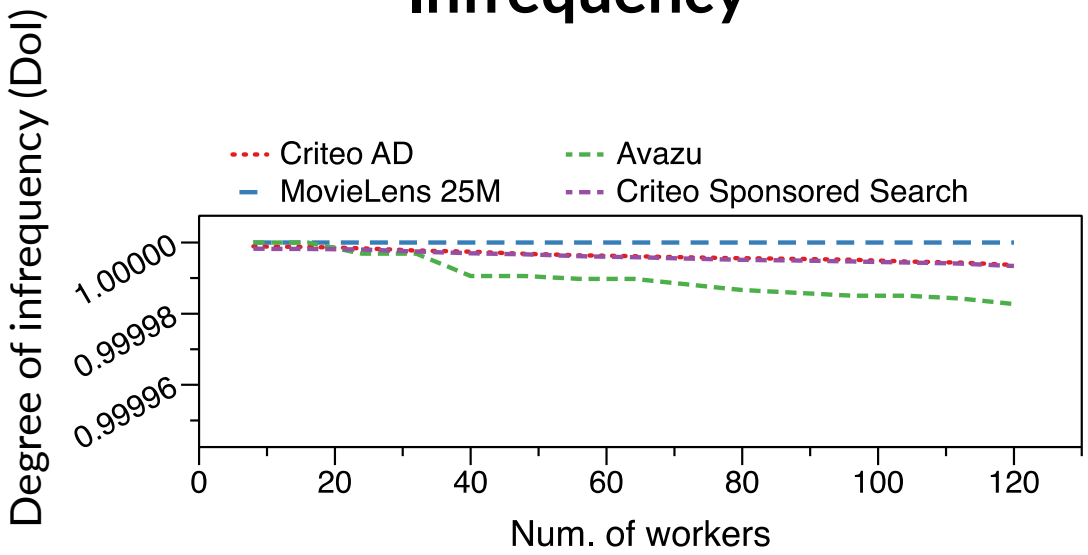
Predictability



Cache snapshots are observable

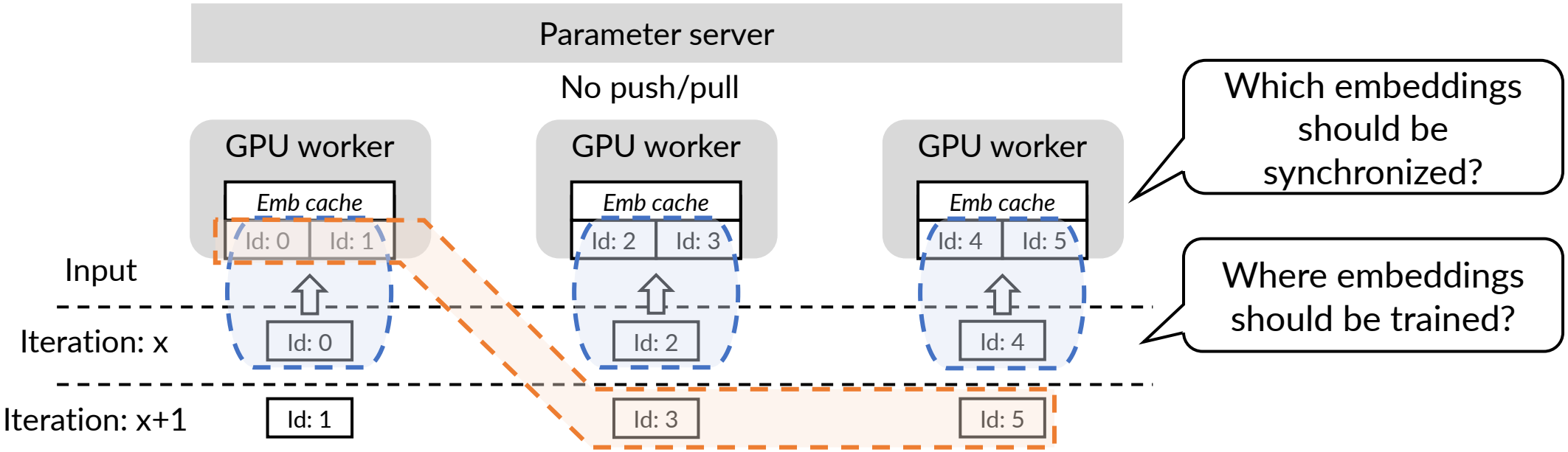
An inputs partition determines future embedding accesses and cache behaviors

Infrequency



The physical accesses to most in-cache embeddings are infrequent enough, so that they can be trained by ***a consistent worker***

Key Idea: Embedding Scheduling



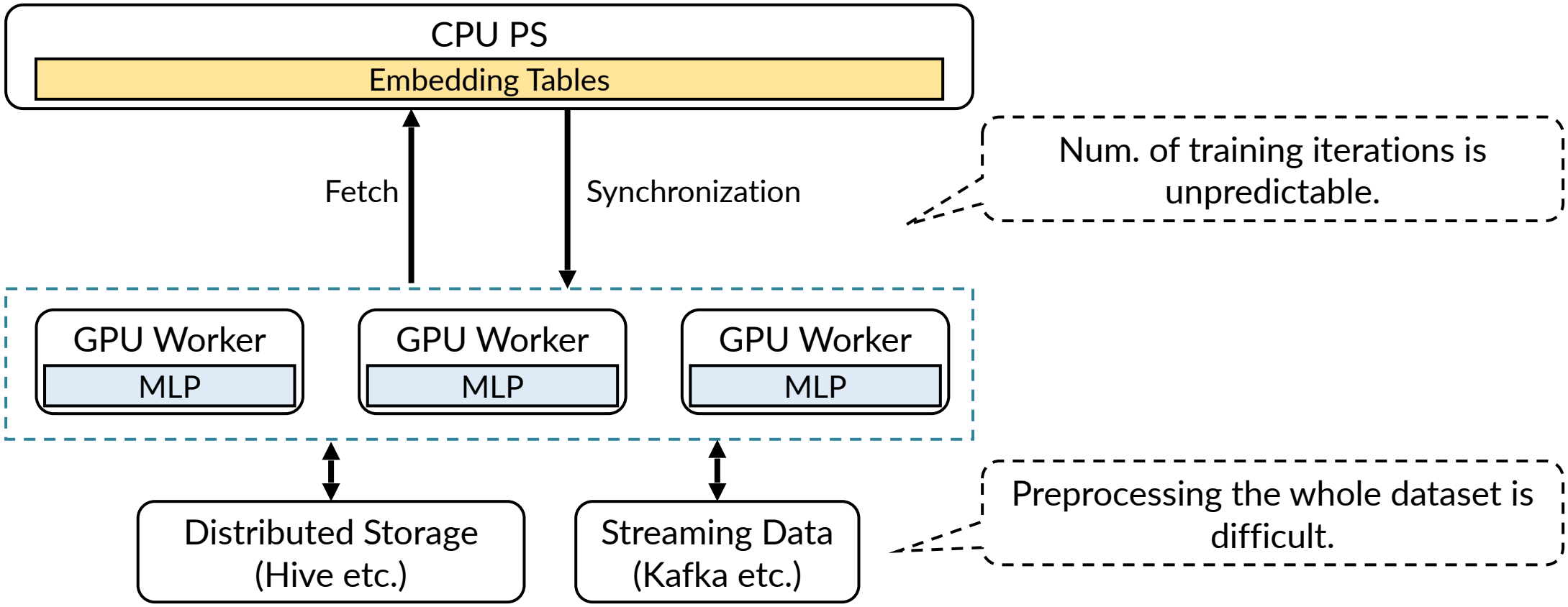
Cache hitting can avoid **worker pull**

Updated embeddings that are not required by other workers later can avoid **worker push**

Scheduling philosophies:

P1. Training in-cache embeddings as much as possible
P2. Performing synchronizations when necessary, i.e., on-demand synchronizations

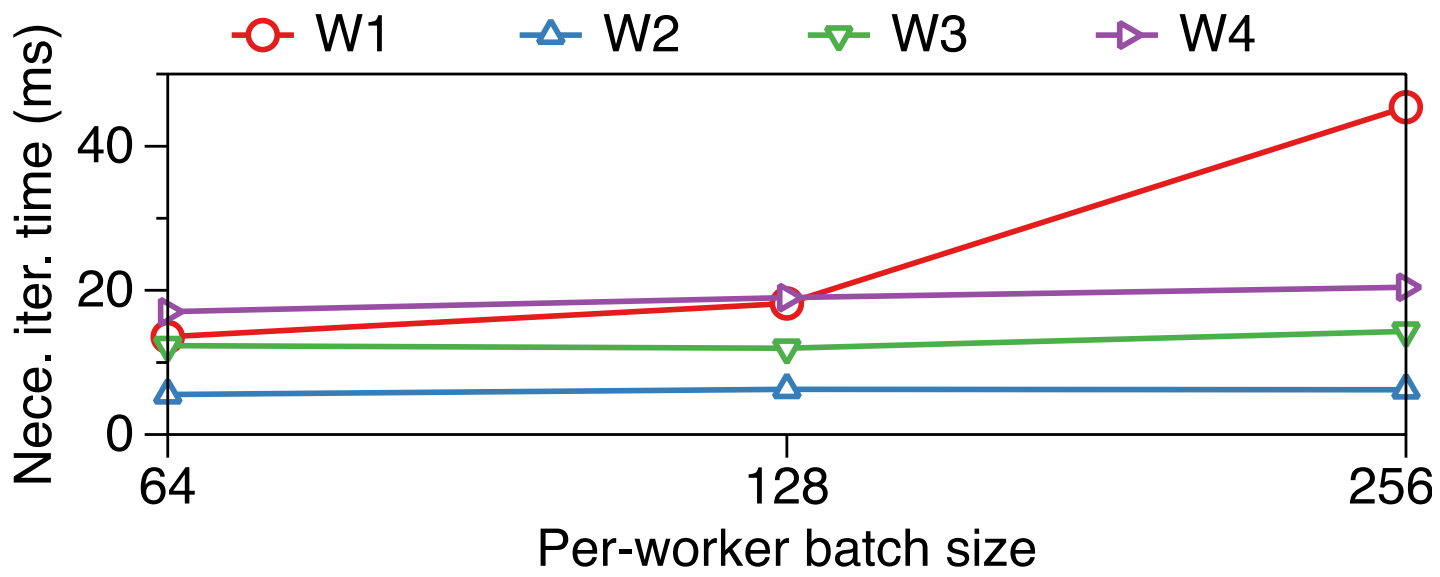
Real-time Scheduling



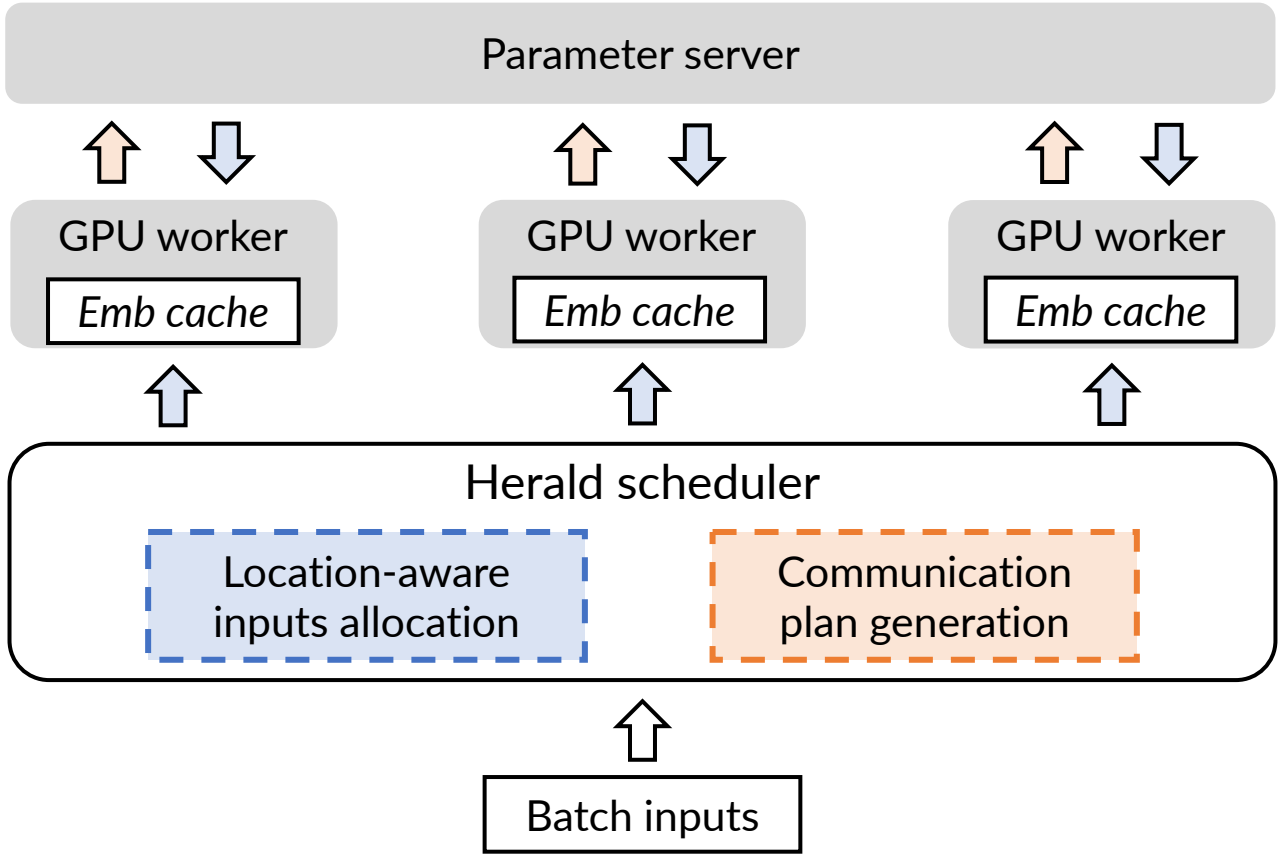
Challenges of Real-time Scheduling

C1: *Limited scheduling budget*:
High scheduling quality vs. ***long*** scheduling time

C2: *Varied scheduling budgets across different workloads and training settings.*



Herald Scheduler



Which embeddings should be synchronized?

Where embeddings should be trained?

Location-aware Inputs Allocation (LAIA)

Parameter server

GPU worker 1

Embedding cache	
Table	Cached IDs
0	0, 1, 2
1	1000, 1001, 1002

GPU worker 2

Embedding cache	
Table	Cached IDs
0	7, 8, 10
1	1006, 1007, 1008

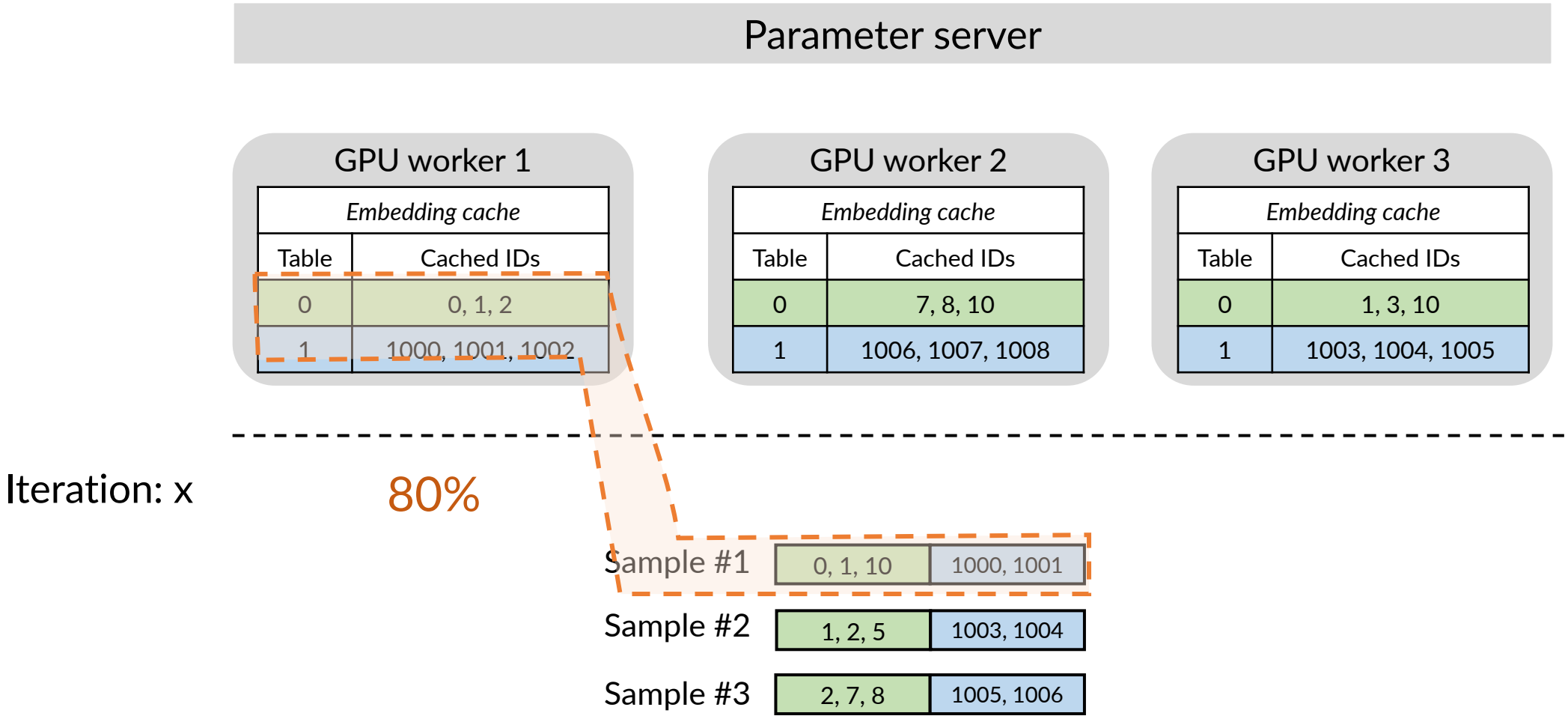
GPU worker 3

Embedding cache	
Table	Cached IDs
0	1, 3, 10
1	1003, 1004, 1005

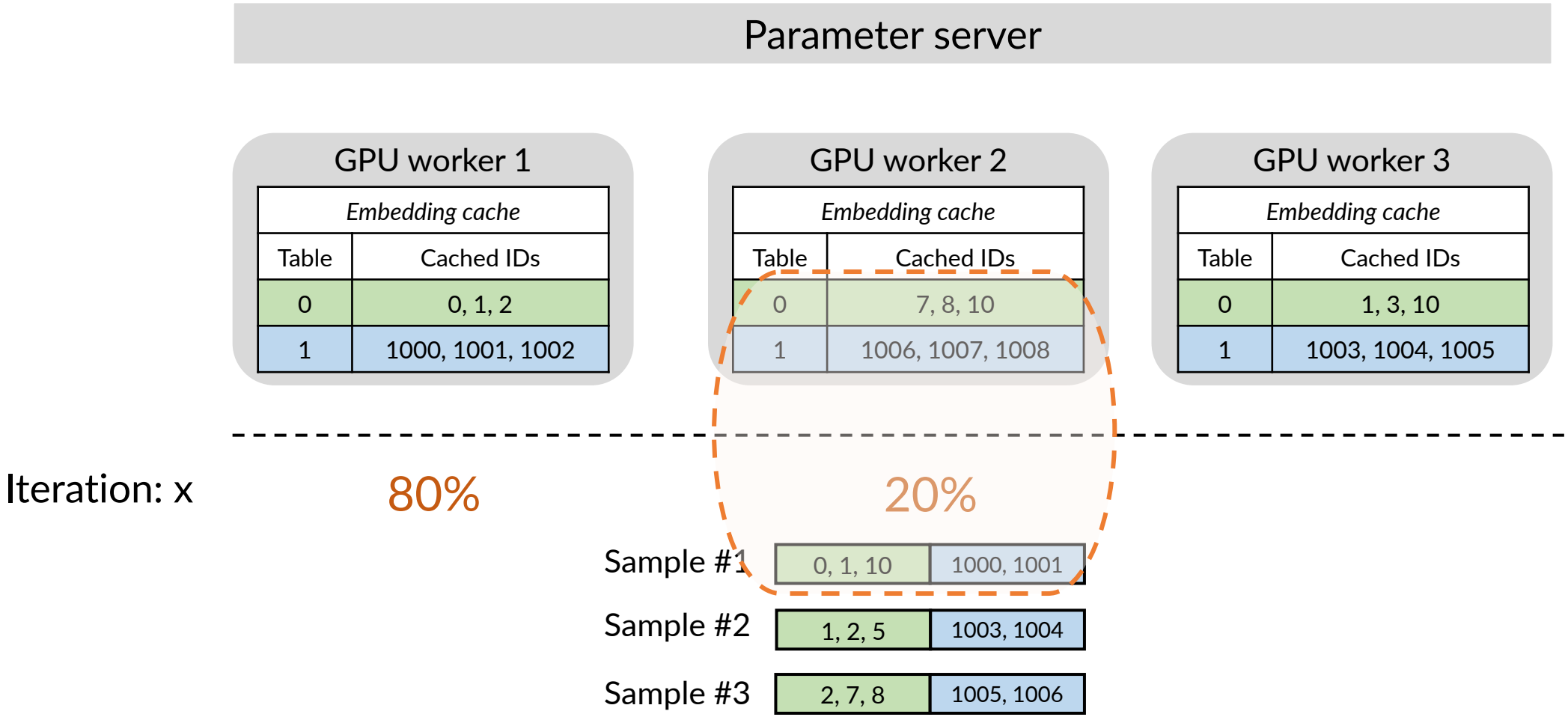
Iteration: x

Sample #1	0, 1, 10	1000, 1001
Sample #2	1, 2, 5	1003, 1004
Sample #3	2, 7, 8	1005, 1006

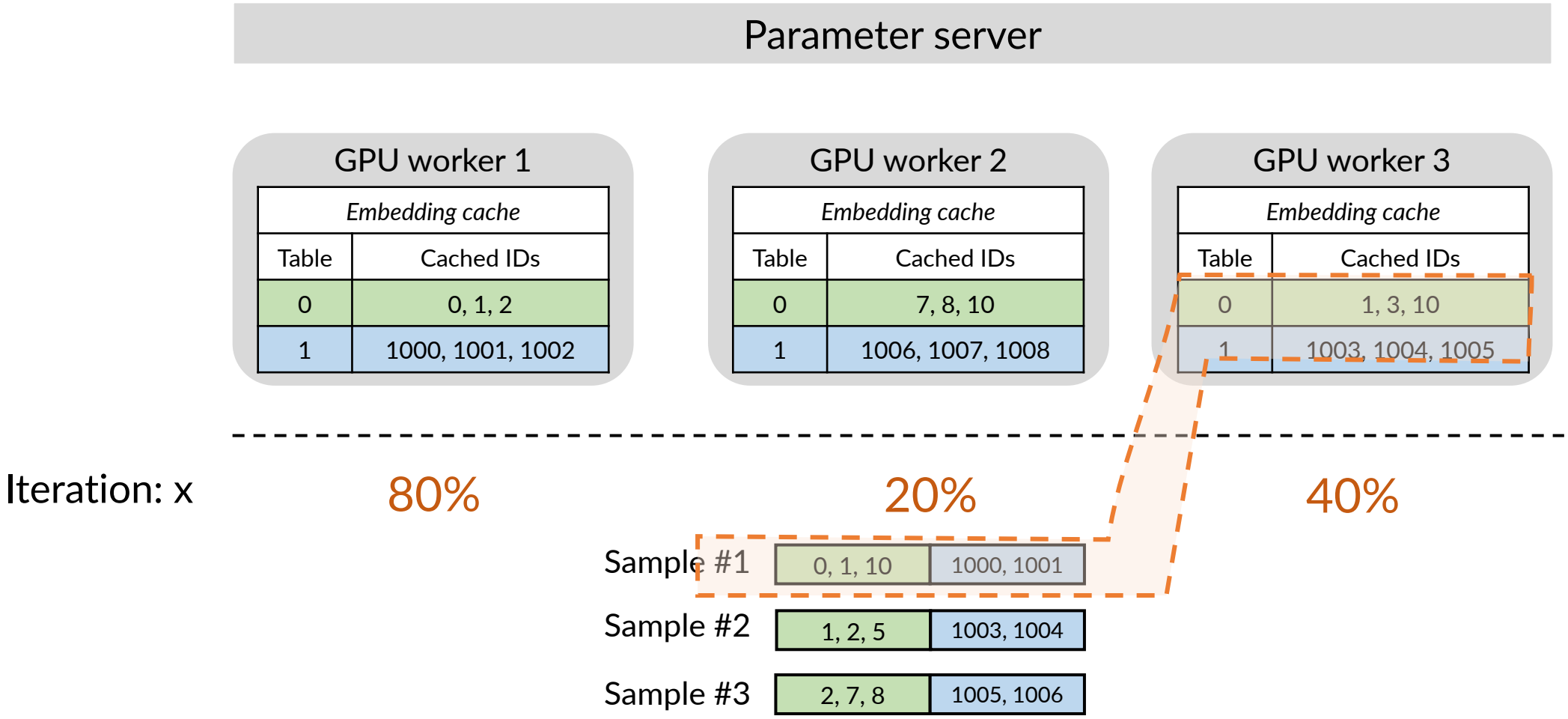
Location-aware Inputs Allocation (LAIA)



Location-aware Inputs Allocation (LAIA)



Location-aware Inputs Allocation (LAIA)



Location-aware Inputs Allocation (LAIA)

Parameter server

GPU worker 1

Embedding cache	
Table	Cached IDs
0	0, 1, 2
1	1000, 1001, 1002

GPU worker 2

Embedding cache	
Table	Cached IDs
0	7, 8, 10
1	1006, 1007, 1008

GPU worker 3

Embedding cache	
Table	Cached IDs
0	1, 3, 10
1	1003, 1004, 1005

Iteration: x

80% 20% 40%

Sample #1 | 0, 1, 10 | 1000, 1001 |

Sample #2 | 1, 2, 5 | 1003, 1004 |

Sample #3 | 2, 7, 8 | 1005, 1006 |

Location-aware Inputs Allocation (LAIA)

Parameter server

GPU worker 1

Embedding cache	
Table	Cached IDs
0	0, 1, 2
1	1000, 1001, 1002

GPU worker 2

Embedding cache	
Table	Cached IDs
0	7, 8, 10
1	1006, 1007, 1008

GPU worker 3

Embedding cache	
Table	Cached IDs
0	1, 3, 10
1	1003, 1004, 1005

Iteration: x

	40%	0%	60%
Sample #1	0, 1, 10 | 1000, 1001		
Sample #2			1, 2, 5 | 1003, 1004
Sample #3		2, 7, 8 | 1005, 1006	

Location-aware Inputs Allocation (LAIA)

Parameter server

GPU worker 1

Embedding cache	
Table	Cached IDs
0	0, 1, 2
1	1000, 1001, 1002

GPU worker 2

Embedding cache	
Table	Cached IDs
0	7, 8, 10
1	1006, 1007, 1008

GPU worker 3

Embedding cache	
Table	Cached IDs
0	1, 3, 10
1	1003, 1004, 1005

Iteration: x

0%

60%

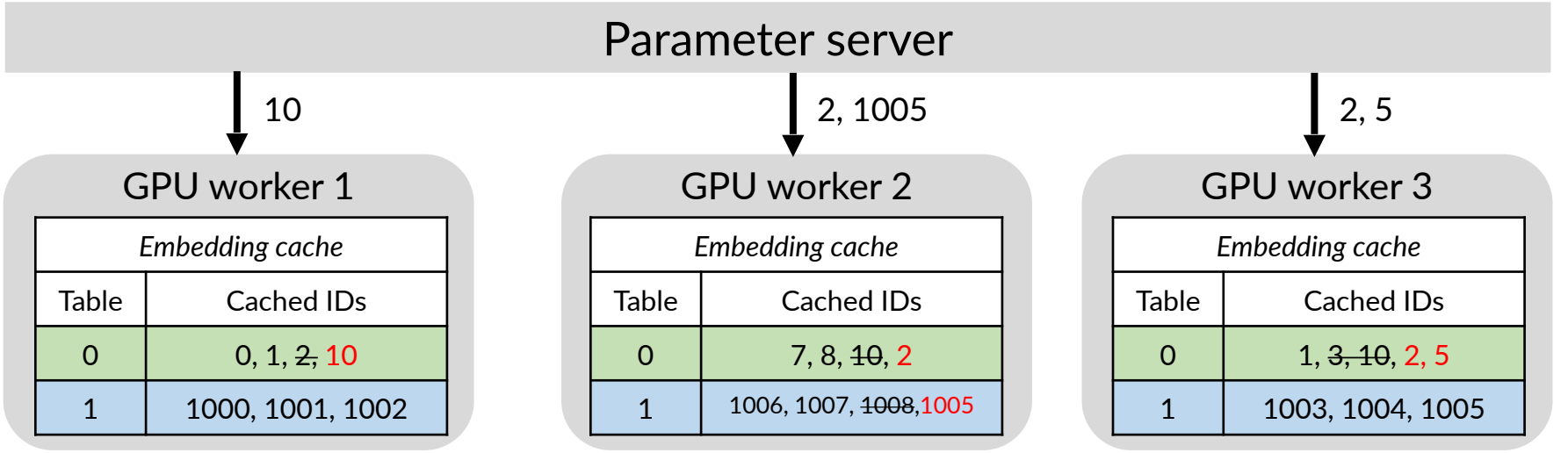
20%

Sample #1: 0, 1, 10 | 1000, 1001

Sample #2: 1, 2, 5 | 1003, 1004

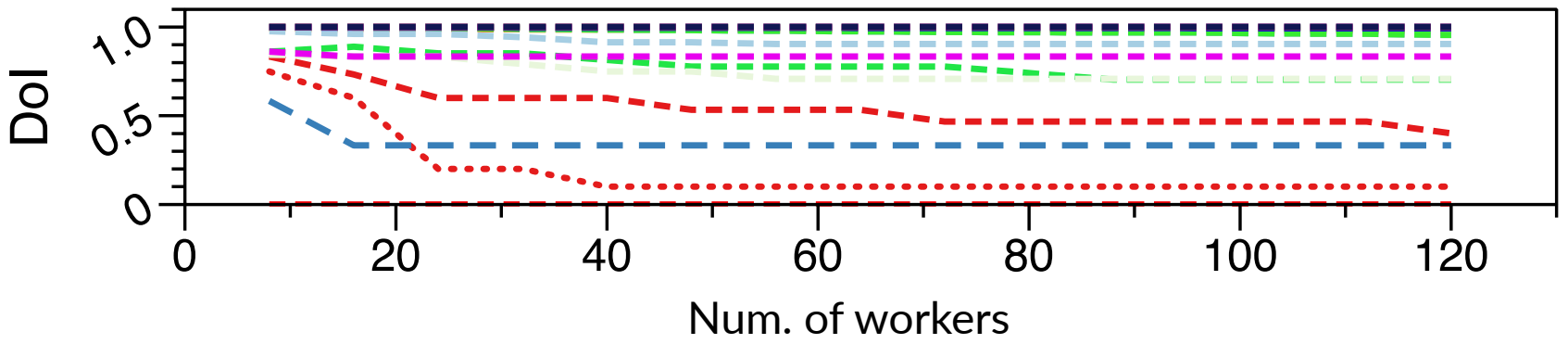
Sample #3: 2, 7, 8 | 1005, 1006

Location-aware Inputs Allocation (LAIA)



Iteration: x

Adaptive Location-aware Inputs Allocation



Higher Degree-of-infrequency indicates higher gains.

The DoI of different embedding tables exhibits a large variance. Each line represents an embedding table.

Adaptively Inputs Allocation

1. Profile the tables' DoI offline
2. Only choose the ***scheduling-worthy*** tables

Embedding cache	
Table	Cached IDs
0	0, 1, 2
1	1000, 1001, 1002

On-demand Synchronization

Parameter server

GPU worker 1

Embedding cache	
Table	Cached IDs
0	0, 1, ~~2,~~ 10
1	1000, 1001, 1002

GPU worker 2

Embedding cache	
Table	Cached IDs
0	7, 8, ~~10~~, 2
1	1006, 1007, ~~1008~~,1005

GPU worker 3

Embedding cache	
Table	Cached IDs
0	1, ~~3,~~ ~~10~~, 2, 5
1	1003, 1004, 1005

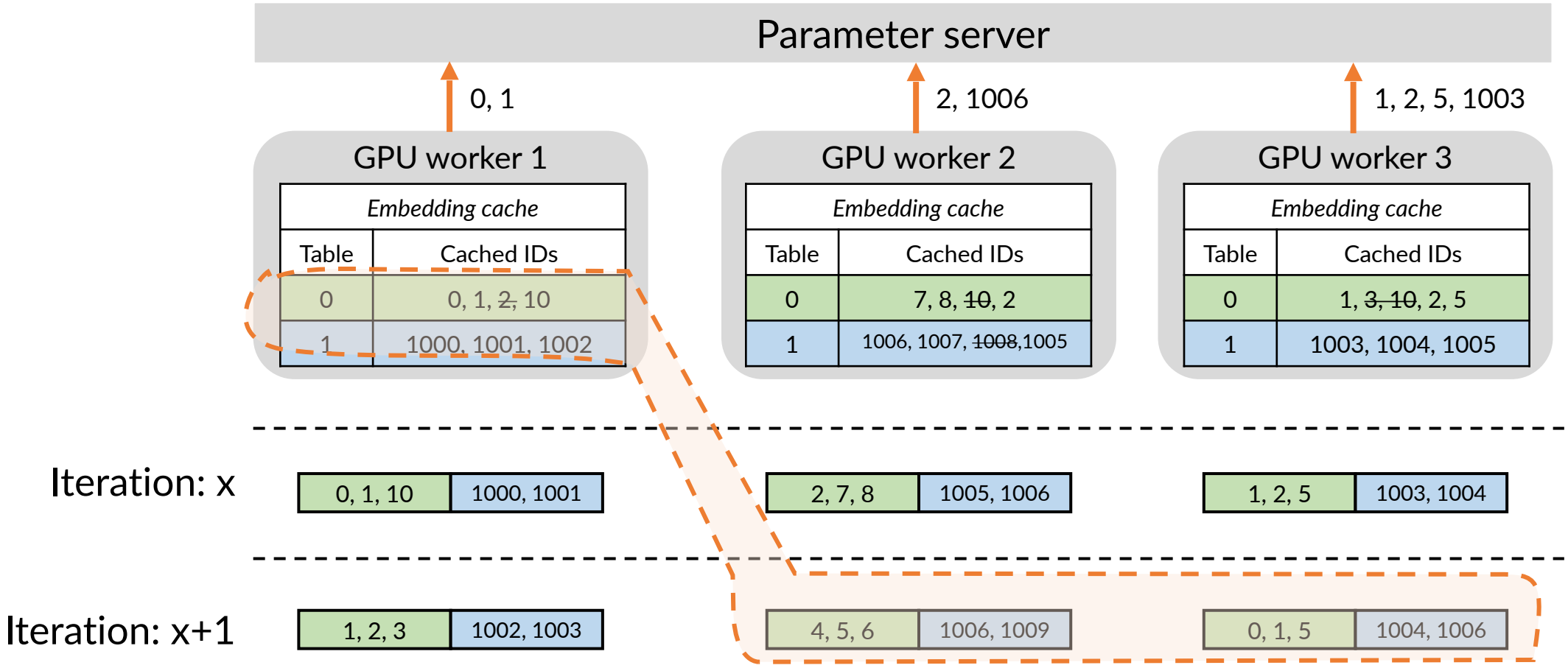
Iteration: x

0, 1, 10	1000, 1001	2, 7, 8	1005, 1006	1, 2, 5	1003, 1004

Iteration: x+1

1, 2, 3	1002, 1003	4, 5, 6	1006, 1009	0, 1, 5	1004, 1006

On-demand Synchronization



Evaluation Setup

Basic configurations:

- 8 NVIDIA 3090 GPU workers
- 1 CPU PS
- 100 Gbps Ethernet (default TCP)
- Caching 10% embeddings

Baseline:

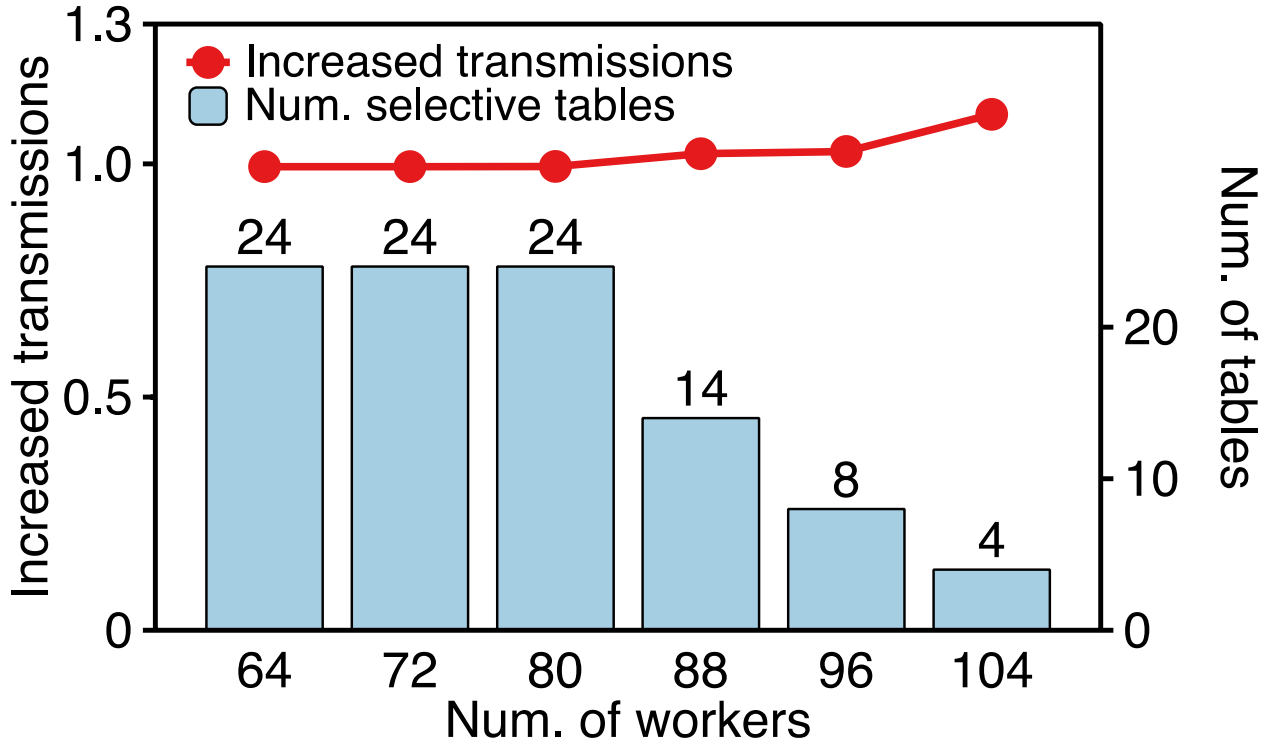
- HET with vanilla BSP, where MLP parameters are synchronized in ring-based all-reduce and embedding parameters are synchronized with PS

Workloads:

- Batch size = 256, embedding size = 512

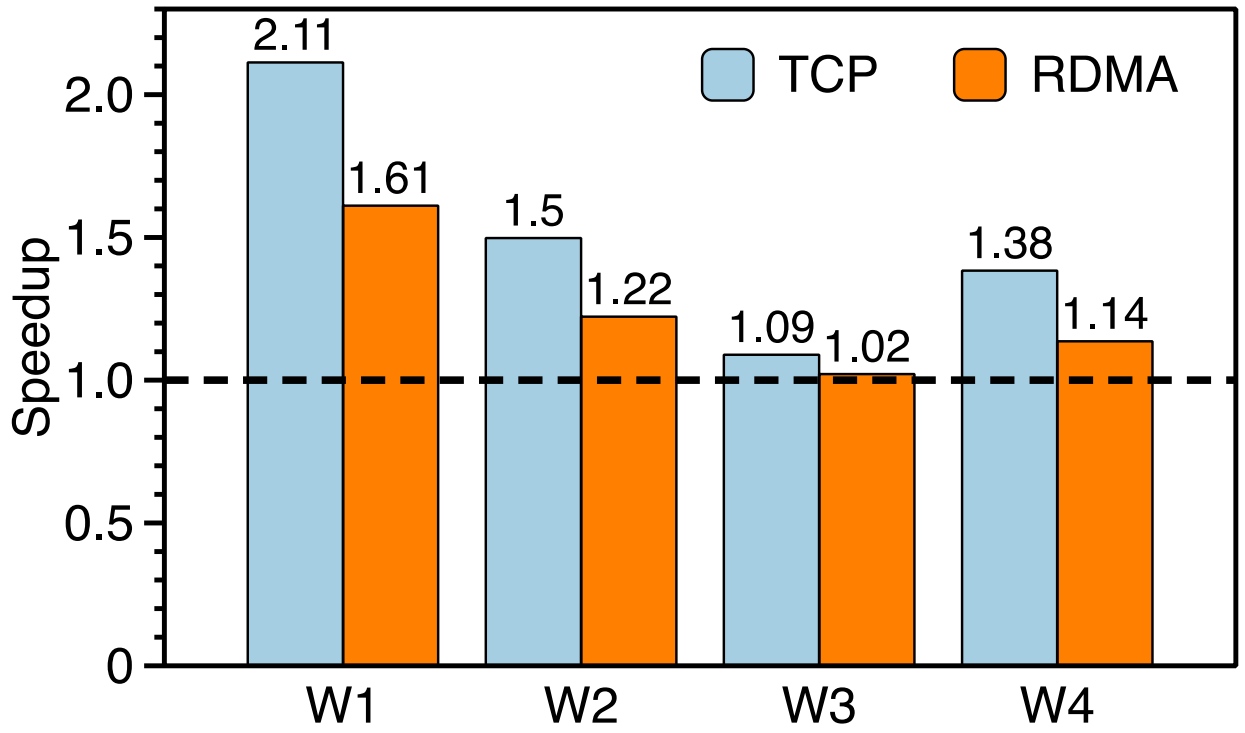
	Model	Dataset
W1	Wide & Deep [DLRS'16]	Criteo AD
W2	Neural Collaborative Filtering [WWW'17]	MovieLens 25M
W3	DeepFM [IJCAI'17]	Avazu
W4	Deep & Cross [ADKDD'17]	Criteo Sponsored Search

Scalability of Adaptive LAIA



Adaptive LAIA is scalable with less than 1.11× transmissions increase.

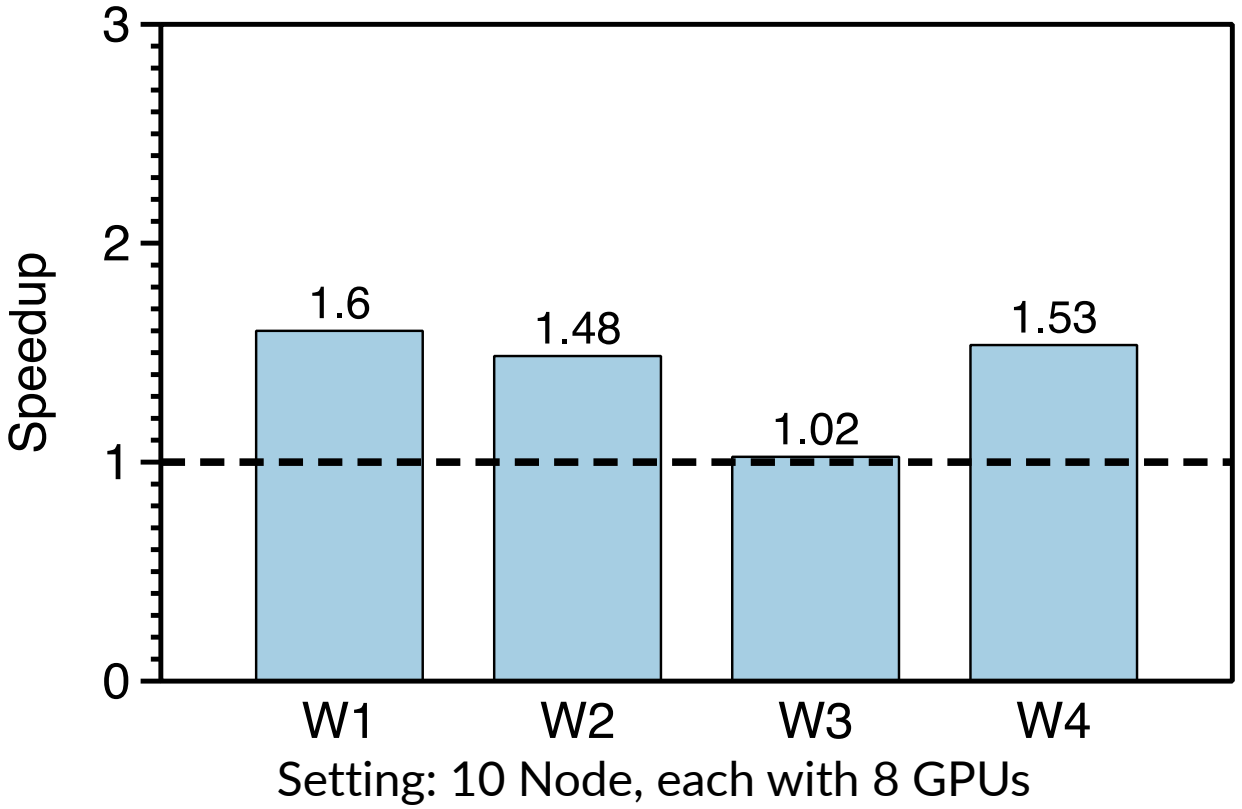
End-to-end Speedup: Herald vs. HET



1.09x-2.11x speedup over TCP

1.02x-1.61x speedup over RDMA

Multi-GPU and Multi-node Experiment



Setting: 10 Node, each with 8 GPUs

Herald achieves up to 1.6x speedup over HET.

See more results in the paper.

Herald Recap

- We explore embedding scheduling as a new acceleration direction for neural recommendation training by leveraging two key characteristics of in-cache embedding accesses, i.e., predictability and infrequency
- We design Herald, a runtime embedding scheduler, to reduce embedding transmissions with location-aware inputs allocation and on-demand synchronization
- Herald can be extended to accelerate generic embedding model training in future

Herald is available at https://github.com/HKUST-SING/herald.

Thank You!

Q&A

Contact email: xudong.liao.cs@gmail.com