Distributed Hierarchical GPU Parameter Server for Massive Scale Deep Learning Ads Systems

MLSys 2020 article

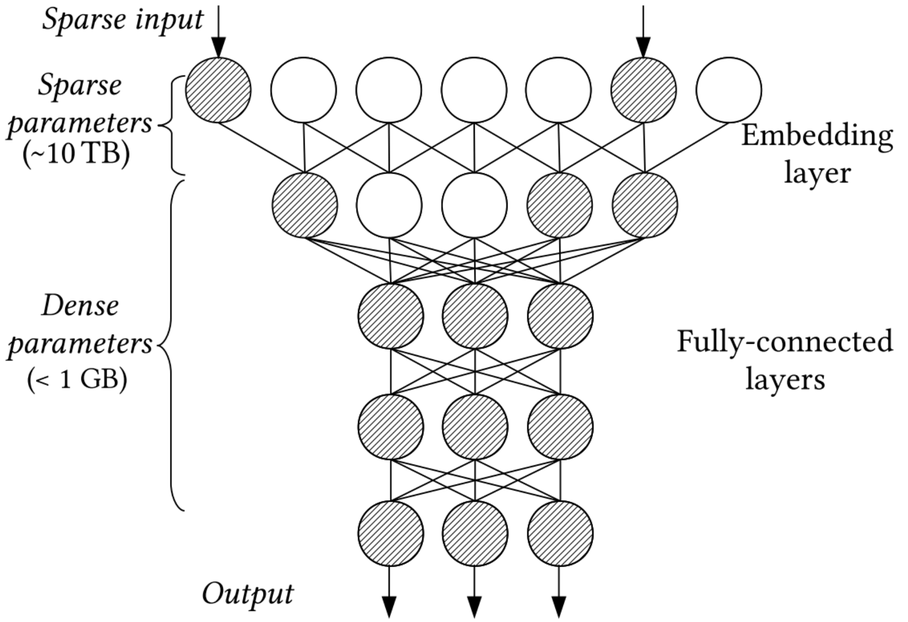
Ye Tian

Content

- Background
- Hierarchical parameter server
- HBM-PS
- MEM-PS
- SSD-PS
- Experimental evaluation

Background

- Training ultra-large scale deep neural network to predict Click Through Rate (CTR)
- Model contains around 10TB parameters
- 2010~2013: distributed logistic regression model
- 2013~2020: MPI based distributed parameter server and OP+OSRP reduced DNN model



Background

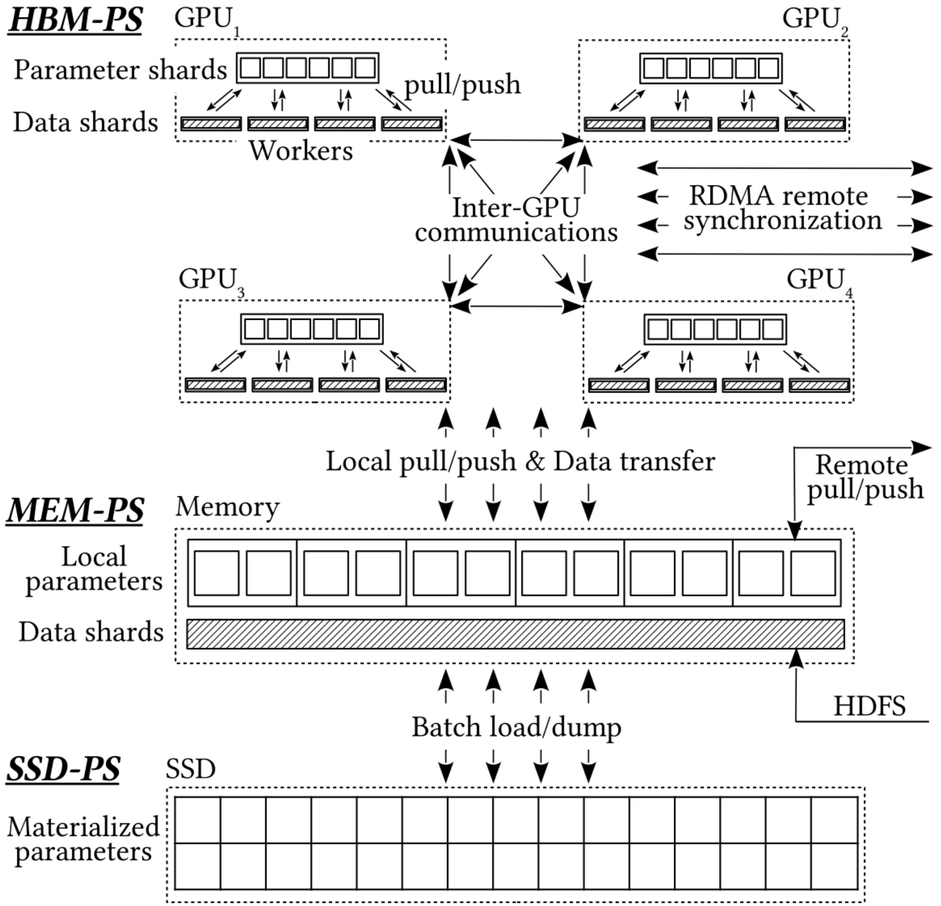
- MPI solution drawbacks
  - Pricey hardware maintenance and energy cost of large scale cluster
  - Communication and synchronization cost restrict scaling up training speed
- GPU parameter server for CTR prediction
  - Highly sparse inputs: only a subset of parameters is used and updated
  - Few millions dense parameters: can be stored in GPU memory

Background

- Targets
  - Efficient distributed GPU hash table
  - Efficient data transferring and caching mechanism
  - Effective materialized parameter organization

Hierarchical parameter server

- HBM-PS: High-Bandwidth Memory across multiple GPUs
- MEM-PS: pulls parameters from remote and materializes updated parameters in SSDs
- SSD-PS: stores and organizes parameters in files to achieve better bandwidth



Hierarchical parameter server architecture

Workflow

- Fetch training data to memory from network file system (HDFS)
- Identify referenced parameter, load from MEM-PS, SSD-PS and remote, partition and transfer to HBM-PS
- Performs forwards and backwards propagation
- Collect and dumps updated parameter to SSD-PS

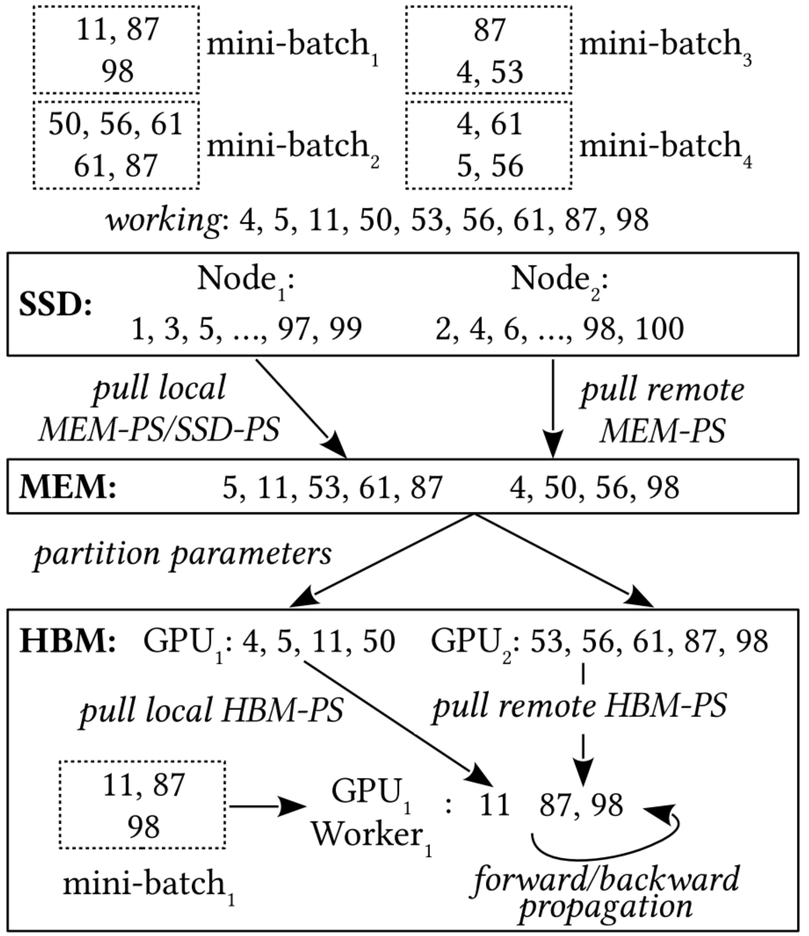
Algorithm 1 Distributed Hierarchical Parameter Server Training Workflow.

1. **while** *not converged* **do**
2. *batch* $\leftarrow$ *get_batch_from_HDFS*()
3. *working*$\leftarrow$*pull_local_MEM-PS_and_SSD-PS*(*batch*)
4. *working* $\leftarrow$ *working* $\cup$ *pull_remote_MEM-PS*(*batch*)
5. *minibatches* $\leftarrow$ *shard_batch*(*batch*,*#GPU*,*#minibatch*
6. *partitions* $\leftarrow$ *map_parameters_to_GPUs*(*working*,#GP
7. **for** $i \leftarrow 1$ **to** *#GPU* **do**
8. *transfer_to_GPU*(i,*minibatches*$_i$)
9. *insert_into_hashtable*(i,*partitions*$_i$)
10. **end for**
11. **for** $j \leftarrow 1$ **to** *#minibatch* **do**
12. *pull_HBM-PS_kernel*(*minibatch*$_j$)
13. $\Delta_j \leftarrow$ *train_mini-batch_kernel*(j)
14. *push_parameters_updates_back_kernel*(Δ_j)
15. **end for**
16. $\Delta \leftarrow$ *pull_updates_HBM-PS*()
17. *parameters_to_dump* $\leftarrow$ *update_local_cache*()
18. *push_local_SSD-PS*(*parameters_to_dump*)
19. **end while**

Distributed hierarchical parameter server training workflow

Workflow example

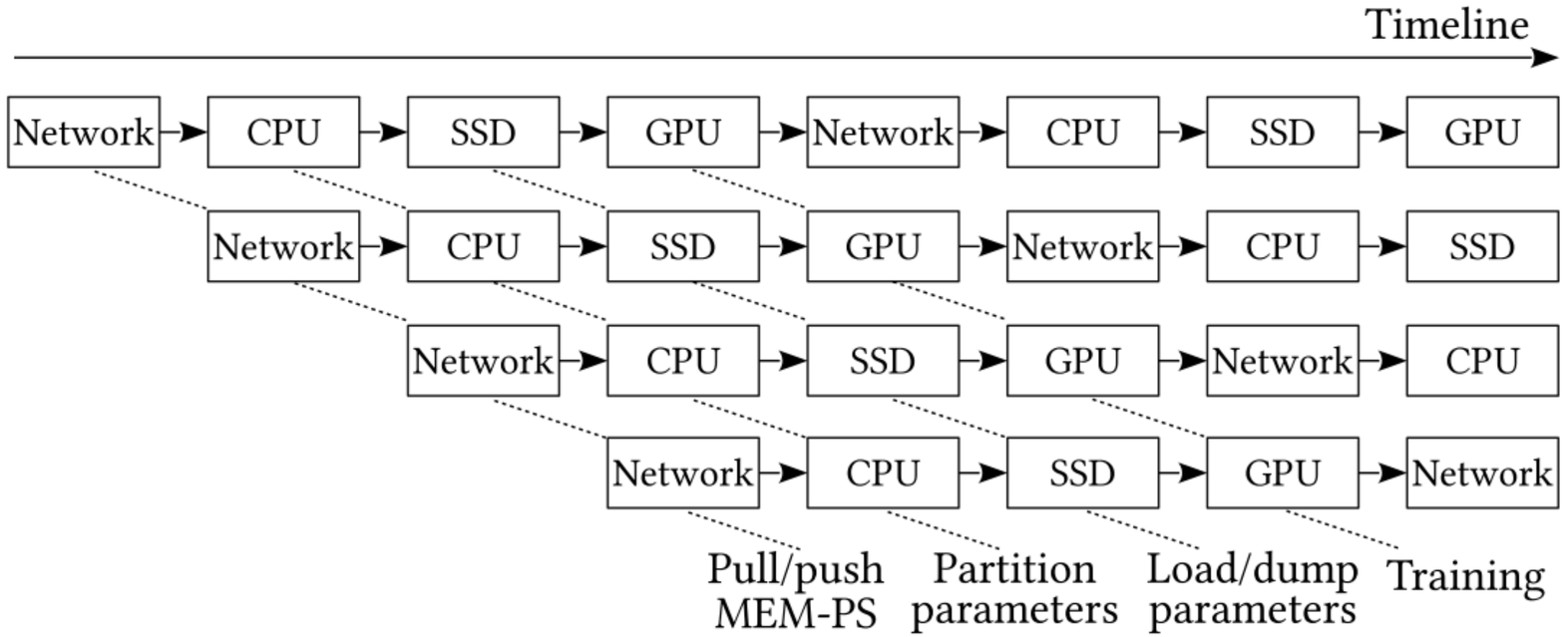
- Working parameters: 4, 5, 11, 50, 53, 56, 61, 87, 98
- Pull from node1 (local MEM-PS/SSD-PS) and node2 (remote MEM-PS)
- Use hashing function to map parameter to GPU
- Pull parameter from remote HBM-PS through NVLink



Hierarchical parameter server workflow example

Training workflow

- 4-stage pipeline: data transferring, parameter partitioning, materialized data loading/dumping and neural network training



The 4-stage pipeline

HBM-PS

- Local GPU hash table: cuDF library
- Accumulate operation: input multiple key-value pairs
- Switch to GPU that owns the memory of inputs
- Performs partitioning
- Send key-value pairs to its corresponding GPUs and apply accumulation

Algorithm 2 Distributed GPU hash table accumulate.

Input: A collections of key-value pairs: (*keys*, *vals*).

1. *switch_to_GPU*(*get_resource_owner*(*keys*, *vals*))
2. *partitioned* $\leftarrow$ *parallel_partition*(*keys*, *vals*)
3. **for** $i \leftarrow 1$ *to* $\#GPU$ **do**
4. **if** $partitioned_i \neq \emptyset$ **then**
5. *async_send*(i, $partitioned_i$)
6. **end if**
7. **end for**
8. *wait_send_destination*()
9. **for** $i \leftarrow 1$ *to* $\#GPU$ **do**
10. *switch_to_GPU*(i)
11. $hash\_tables_i$.*async_accum*($partitioned_i$)
12. **end for**

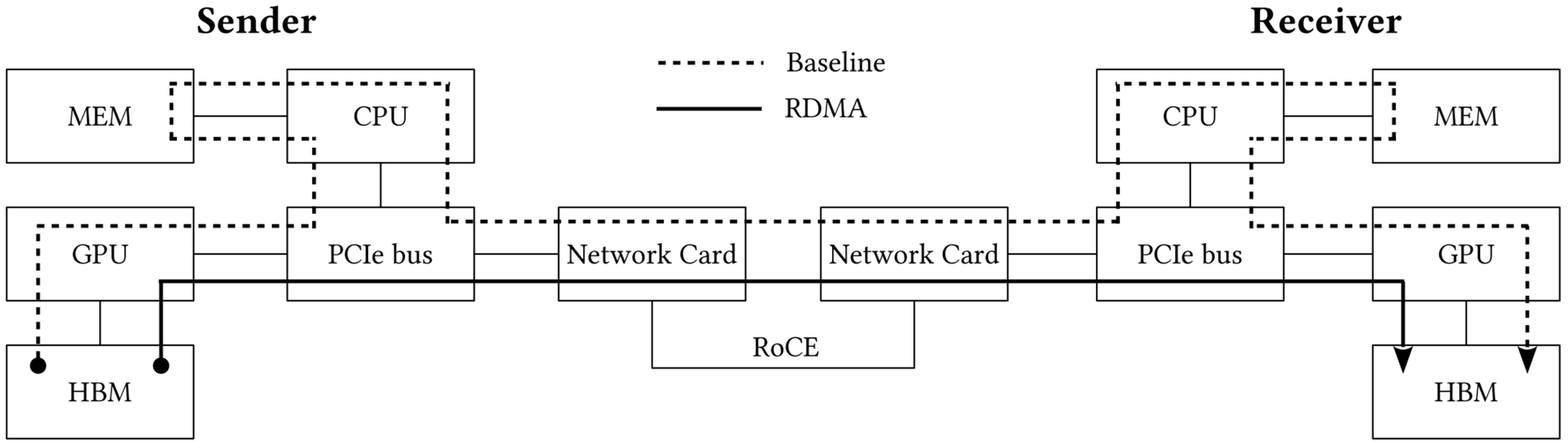
Distributed GPU hash table accumulate

HBM-PS

- Multi-GPU distributed hash table: parameter partitioning
- Maps a parameter key to a GPU id
- Simple modulo hash function
- Group parameters with high co-ocurrence together
  - Pre-train a learned hash function
  - Axis-vertical partitioning

HBM-PS

- GPU RDMA communication: Remote Direct Memory Access



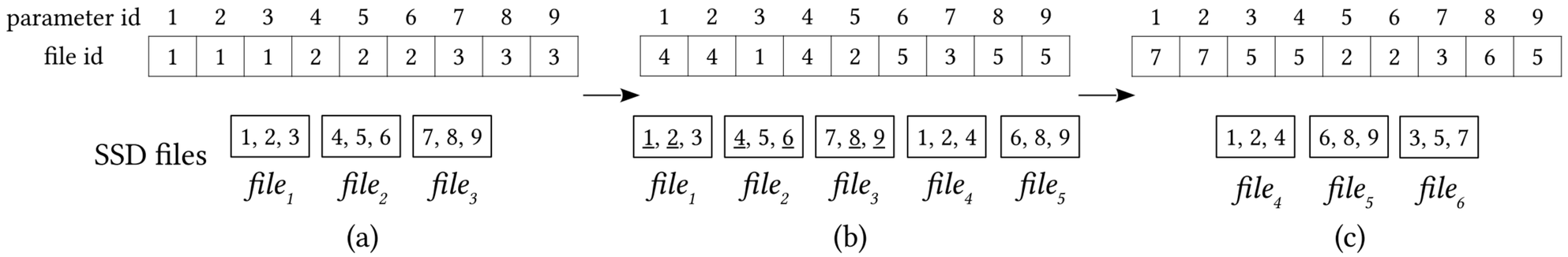
Inter-node RDMA communication

MEM-PS

- Use modulo hashing as partition scheme
- For local parameters, reads from SSDs
- For remote parameters, pulls from other MEM-PS through network
- Maintains an in-memory cache to store recently/frequently used parameters (using LRU and LFU cache) and avoid excessive SSD I/Os
- Pulls updated parameter from local GPUs

SSD-PS

- Considers a parameter file an SSD I/O unit
- Writes updated parameter to new file sequentially
- Updates parameter to file mapping of updated parameters
- Performs file compaction to reduce SSD usage



SSD-PS examples: (a) parameter-to-file mapping and parameter files; (b) 1, 2, 4, 8, 9 are updated; (c) a compaction operation.

Experimental evaluation

- Set up: 4 GPU nodes, each nodes has 8 32GB GPUs, 48 core CPUs, 1TB memory, 20 TB RAID-0 NVMe SSDs and 100 Gb RDMA net-work adaptor
- 5 CTR prediction models

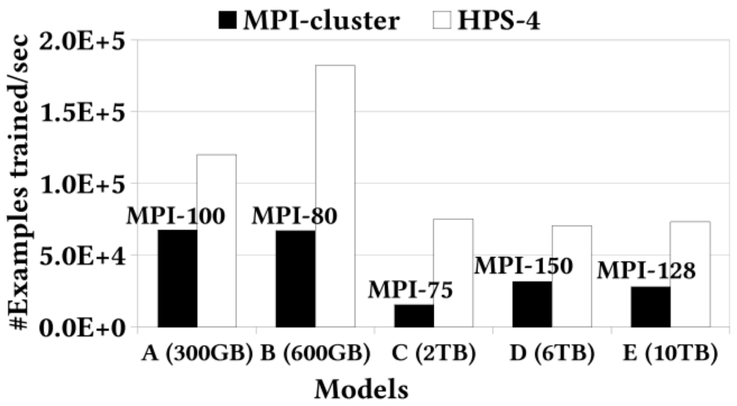
	#Non-zeros	#Sparse	#Dense	Size (GB)	MPI
A	100	8×10^9	7×10^5	300	100
B	100	2×10^{10}	2×10^4	600	80
C	500	6×10^{10}	2×10^6	2,000	75
D	500	1×10^{11}	4×10^6	6,000	150
E	500	2×10^{11}	7×10^6	10,000	128

Model specifications

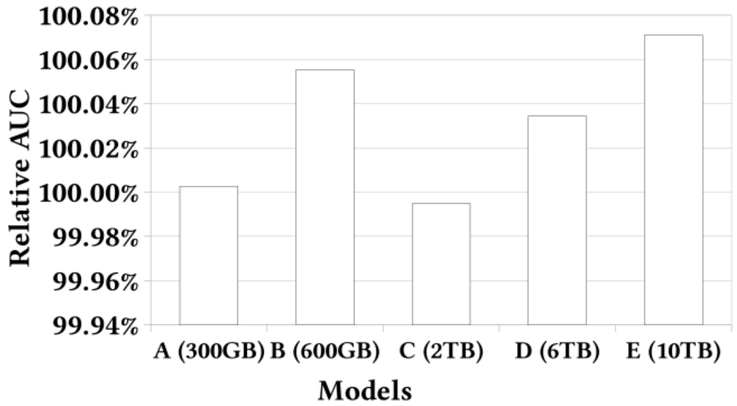
Experimental evaluation

- Performance of 4-node hierarchical parameter server

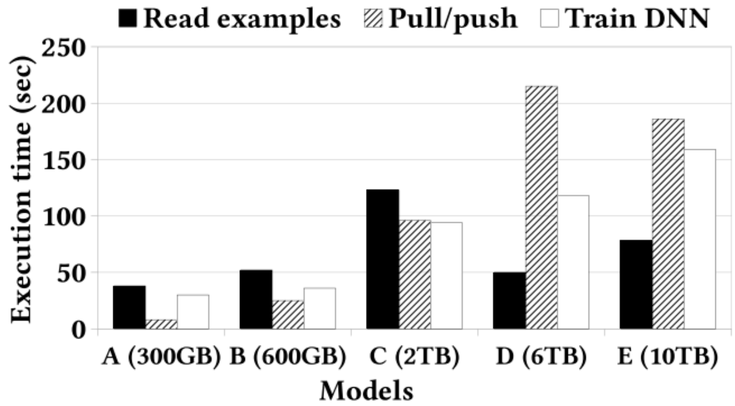
	A	B	C	D	E
Speedup over MPI-cluster	1.8	2.7	4.8	2.2	2.6
Cost-normalized speedup	4.4	5.4	9.0	8.4	8.3



(a) Total execution time



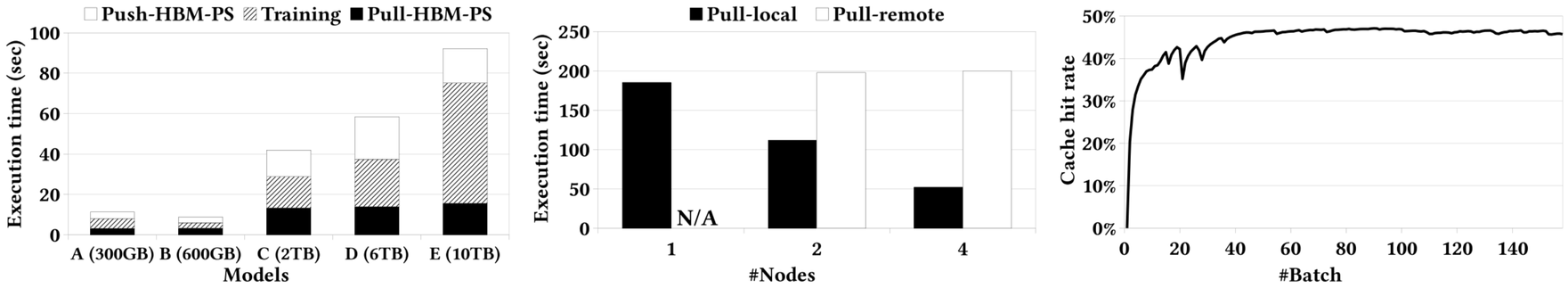
(b) AUC accuracy



(c) Execution time distribution

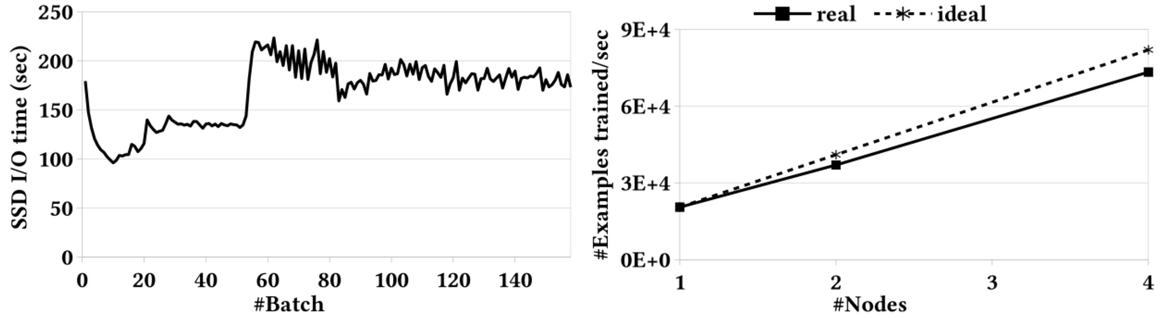
Performance of HPS-4

Experimental evaluation



(a) Time distribution in HBM-PS (b) Time distribution in MEM-PS (c) Cache hit rate

Time distribution in HBM-PS/MEM-PS and the cache hit rate on Model E.



(a) SSD-PS I/O time (b) Speedup

SSD-PS I/O time and speedup on Model E