Ekko: A Large-Scale Deep Learning Recommender System with Low-Latency Model Update

Chijun Sima¹*, Yao Fu²*, Man-Kit Sit², Liyi Guo¹, Xuri Gong¹, Feng Lin¹, Junyu Wu¹, Yongsheng Li¹, Haidong Rong¹, Pierre-Louis Aublin³, Luo Mai²

Tencent¹, University of Edinburgh², IIJ Innovation Institute³

*Co-primary authors
Deep learning recommender systems

Characteristics of Deep Learning Recommender Systems (DLRSs):

- Billions of global online users
- Service-Level-Objectives (SLOs)

Data source: How NVIDIA Supports Recommender Systems feat. Even Oldridge, Stanford MLSys Seminar
The DLRS architecture

Models in a DLRS

- Embedding Table (EMB)
  - User & item embeddings
- Deep Neural Network (DNN)
  - Multi-layer perceptrons, transformers

Model access characteristics

- Massive parameters
  - EMB (GB - TB), DNN (MB - GB)
- Sparse & intensive reads
  - Hot content, hot users

Parameter servers are replicated in geo-distributed inference DCs
Updating models in a DLRS

Reasons for updating models

- New content
- New users
- New events

Update process:

1. **Collecting** training data
2. **Training** models online
3. **Disseminating** updated models
4. **Serving** user requests
Low-latency model update

Content freshness
- WeChat, TikTok, Instagram

Anonymous users
- Guest login
- Data protection rules (e.g., GDPR [1])

Online training methods [2]
- Capture user's current interests

**Question**: How to support updating TB–PB model parameters in seconds

Existing model update techniques

SOTA PS [1] – **Checkpoint broadcast**
- Multiple long-latency model update steps

- SLO loss

1. Checkpoint (seconds - minutes)
2. Validation (minutes - hours)
3. Broadcast (seconds – minutes)

**Issue:** Existing techniques compromise latency or SLO performance

---

[1] Check-N-Run: a checkpointing system for training deep learning recommendation models, NSDI 2022
Our idea and its associated challenges

**Idea**: Let the training DC directly send updates to inference DCs

**Coordinating massive model updates**
- Leader bottlenecks

**Sending model updates over WANs**
- Network congestions

**Biased model updates**
- SLO loss
Efficient peer-to-peer (P2P) model update
- Utilise more network paths
- Reduce dissemination cost

SLO-aware model update scheduler
- Prioritise significant updates

Inference model state manager
- Monitor and recover model state online
Contribution 1
Efficient P2P Model Update Mechanism
P2P complexity in model synchronisation

Format:

- A parameter is a *key-value pair*
- Parameters are distributed into shards

How can PSs find **newer** parameter update from each other?

- **P2P synchronisation** [1, 2]: parameter *version*, version vector (*VV*, shard knowledge)

![Diagram](image)

Details in §4.2, §4.3

[1] P2P replica synchronization with vector sets, SIGOPS 2007

$N$ comparisons are not acceptable

$N$ # of parameters per shard $O(100,000)$
Observation: Only a small portion of “hot” parameters are touched (<1% per minute)

Idea: Cache which parameters have been updated recently

Key design: Dominator Version Vector (DVV), which summarises all parameters NOT in the cache

<table>
<thead>
<tr>
<th>Server A</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard 1</td>
<td></td>
</tr>
<tr>
<td>VVA</td>
<td>Pointer to den_1</td>
</tr>
<tr>
<td>emb_1</td>
<td>version</td>
</tr>
<tr>
<td>den_1</td>
<td>version</td>
</tr>
</tbody>
</table>

Case 1: DVV ≤ VV_B
Case 2: DVV ≤ VV_B

$O(N)$ comparison → $O(N_r)$ comparison

$N_r$ is # of recently updated parameters

$N_r \ll N$

Cache maintenance algorithms in § 4.4.1

More optimisations: i.e., shard version, WAN-optimisation in paper

In production,
- 0.13-0.2% parameters are kept in caches
- 99.4% cache hit ratio
Contribution 2
SLO-aware Model Update Scheduler
SLO-critical updates:

- Newly created embedding items are critical
- Large gradients have significant impact [1, 2]
- Popular models can influence large number of users

Problem: Congested networks delay SLO-critical model updates
i.e., O(100) GB/s updates vs. O(100) Mbps network

[1] Gradient Compression Supercharged High-Performance Data Parallel DNN Training, SOSP 2021
**Idea**: Prioritising SLO-critical updates

**Key priority metrics**:

- Freshness: $p_u = +\infty$ when parameter created; otherwise $p_u = 0$
- Update significance: $p_g = |g|/|\bar{g}|$, i.e., normalised gradients of a model
- Model priority: $p_m = c_m/\sum_{i=1}^{M} c_i$, i.e., percentage of requests to a model $m$

Ekko supports UDF for custom metrics (e.g., update count, update interval, parameter positions)
Contribution 3
Inference Model State Manager
Biased model updates

 Causes of biased updates

• Gradient overflow
• Data outlier

**Problem**: biased updates can be detrimental to SLOs
**Inference model state manager**

**Idea:** using baseline models as a reference for model healthiness

- **Inference servers**
  - Quality SLOs of online/baseline models
  - <1% inference requests
  - >99% inference requests

- **Baseline models**

- **Model State Manager**
  - Anomaly detection algorithm
  - Healthy
  - Corrupted
  - Uncertain

  1. Redirect requests to healthy models
  2. Rollback corrupted model states

---

16th USENIX Symposium on Operating Systems Design and Implementation
Evaluation
End-to-end model update latency

- **Setup**
  - 30 servers, 3 servers per DC

- **Adam**
  - A PS with high-performance model update synchronisation

- **7x faster**
  - Avoiding leader bottleneck
  - Utilising more network paths
  - Accelerated P2P

![Graphs showing latency comparison between Ekko and Adam](image)

(a) Production workload
(b) Criteo workload

*Lower is better*
Model update latency breakdown

• Setup
  • 10 DC
  • Production workload

• 29.3x lower latency
  • Model update cache
  • Shard version (details in §4.4.2)
  • WAN-optimisation (details in §4.5)
Improve the model update latency by up to 100x.

<table>
<thead>
<tr>
<th># data centres</th>
<th># servers</th>
<th># models</th>
<th>Size of parameters</th>
<th>Model update per second</th>
<th>Avg. latency (inter-DC)</th>
<th>Avg. latency (intra-DC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>4,600</td>
<td>100s</td>
<td>40 TB</td>
<td>212 GB/s</td>
<td>2.4 s</td>
<td>0.7 s</td>
</tr>
</tbody>
</table>

Latency of prior systems (similar to [1]): 10 minutes

Benefits of our designs

- Low-latency model updates: 1.30 – 3.82% SLO improvement
- SLO-aware model update scheduler: avoiding 2.32% SLO dropping
- Fast recovery: 6.4 seconds to rollback 113 GB parameters

[1] https://www.tensorflow.org/api_docs/python/tf/train/Checkpoint
• Ekko: support low-latency model updates without compromising SLOs
  • Efficient P2P model update
  • SLO-aware model update scheduler
  • Inference model state manager

• Many potential applications beyond DLRSs

Summary

Chijun Sima*
chijunsima@tencent.com
*Co-primary authors

Yao Fu*
Y.Fu@ed.ac.uk