Pre-train and Search: Efficient Embedding Table Sharding with Pre-trained Neural Cost Models

Daochen Zha, Louis Feng, Liang Luo, Bhargav Bhushanam, Zirui Liu, Yusuo Hu, Jade Nie, Yuzhen Huang, Yuandong Tian, Arun Kejariwal, Xia Hu

Rice University
Meta Platforms
Background
Background
Embedding Table Sharding Problem

- **Problem Setting**
  - Given N number of embedding tables, output a sharding plan that decides 1) partitioning which tables, and 2) how to place them on GPU devices.
Our Proposal “Pre-train, and Search”

- Why neural networks?
  - The computation and communication costs have a nonlinear correlation with the sum of the costs of the individual tables.
Key Challenges

- **Challenges**
  - How to collect data and pre-train?
  - How to search (NP-hard problem).

- **Solution**
  - Neural cost models
  - Nested search process
Neural Cost Models

Input Generation
Original Table Pool → Augment → Augmented Table Pool → Sample → Random Table Combination, Random Table Placement

Pre-training
Hardware → Collect → Computation Cost Data → Train → Computation Cost Model
Hardware → Collect → Communication Cost Data → Train → Forward Communication Cost Model, Backward Communication Cost Model
• **Key observations**
  • When partitioning a table into two halves column-wisely, the computation cost of each shard is larger than half the cost of the original table (left figure).
  • The max forward/backward communication cost among all the GPUs positively correlates with the max device dimension among all the GPUs (right figure).

![Computation cost vs. Table Dimension](image1)

![Communication cost vs. Max device dimension](image2)
Nested Search Process

- **Key ideas**
  - In the outer loop, use beam search to perform column-wise sharding.
  - In the inner loop, use “greedy grid-search”, i.e., 1) use max dimension as the proxy of communication cost and do grid search, and 2) use greedy algorithm (with max dimension as constraint) to assign tables.
• Settings
  • 800+ tables sharded on 128 GPUs.

<table>
<thead>
<tr>
<th>Sharding Algorithm</th>
<th>Embedding Cost (Milliseconds)</th>
<th>Training Throughput Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>118.3</td>
<td>-</td>
</tr>
<tr>
<td>Size-based</td>
<td>107.6</td>
<td>+4.0%</td>
</tr>
<tr>
<td>Dim-based</td>
<td>90.8</td>
<td>+13.9%</td>
</tr>
<tr>
<td>Lookup-based</td>
<td>102.4</td>
<td>+11.9%</td>
</tr>
<tr>
<td>Size-lookup-based</td>
<td>109.2</td>
<td>+12.8%</td>
</tr>
<tr>
<td>AutoShard</td>
<td>86.6</td>
<td>+32.4%</td>
</tr>
<tr>
<td>DreamShard</td>
<td>61.6</td>
<td>+45.3%</td>
</tr>
<tr>
<td>TorchRec</td>
<td>86.4</td>
<td>+34.6%</td>
</tr>
<tr>
<td>NeuroShard</td>
<td>55.2</td>
<td>+54.9%</td>
</tr>
</tbody>
</table>
Summary and Takeaways

- **Embedding table sharding problem**
  - Placing a large number of embedding tables on hundreds of (GPU) devices.
  - Challenges: cost estimation, NP-hardness.

- **Our contributions**
  - NeuroShard with neural cost models and a nested search process.
  - Validated its effectiveness on both open-sourced and production data.

[QR Code for Paper]

[QR Code for Code]