Analyzing and Mitigating Data Stalls in DNN Training

Jayashree Mohan, Amar Phanishayee, Ashish Raniwala, Vijay Chidambaram
Deep Neural Networks (DNNs)

- Widely used for a variety of tasks

- Image Classification

- Object detection

- Language Translation

- Text To Speech
DNN Data Pipeline

- Training happens in epochs
- Each epoch processes the **entire dataset** in a **random order** with **random data augmentations**
- Each epoch is split into iterations (smaller minibatches of data)
- Fetched, pre-processed, and computed upon in a pipelined manner.
DNN Data Pipeline

- Training happens in epochs
- Each epoch processes the entire dataset in a random order with random data augmentations
- Each epoch is split into iterations (smaller minibatches of data)
- Fetched, pre-processed, and computed upon in a pipelined manner.

Data Load Per Iteration

GPU Time Per Iteration
Analyzing and Mitigating Data Stalls

Analyze the impact of the ingest pipeline (storage, memory and CPU) on DNN training in a variety of training scenarios and propose solutions to mitigate data stalls
Outline

• **Data Stalls**
  • Analyzing Data Stalls
  • CoorDL : Mitigating Data Stalls
  • Evaluation
Data Stall

Iteration n

Compute batch n-1
Compute batch n

Data Load batch n

Iteration time = compute time

GPU

Data Loader

Compute batch n-1
Data Load batch n

Compute batch n
Data Stall

Iteration time = compute time + data stall
Fetch Stalls

Training pipeline is stalled on data fetch
Training is I/O bound
Prep Stalls

Training pipeline is stalled on **data prep**
Training is **CPU bound**
Outline

• Data Stalls

• Analyzing Data Stalls

• CoorDL : Mitigating Data Stalls

• Evaluation
Analyzing data stalls

Configurations

Scalability
- 1node, single-GPU
- 1node, multi-GPU
- Multi-node, multi-GPU

Workloads
- Image classification
- Object detection
- Audio classification
- Language

Datastore
- Local:SSD
- Local:HDD
- Remote: Blobfuse
- Remote: GFS

GPU gen
- Volta – V100
- Pascal-1080Ti

#CPUs
- 1 per GPU
- 24 per GPU

Dataset Size
- Fits in memory
- Does not fit in memory

Frameworks
- Pytorch
- Tensorflow
Data Stall Analysis

1. DNNs need anywhere between 3 – 24 CPU cores per GPU for data pre-processing.

![Graph showing the speed of different models (AlexNet, MobileNet, ResNet18, ResNet50) as a function of the number of CPU cores per GPU.]
Data Stall Analysis

1. DNNs need anywhere between 3 – 24 CPU cores per GPU for data pre-processing

![Graph showing data stall analysis for different models and setups](image)

**Setup**
- V100 GPU
- 100% Cached
- 8GPU Training
- Use all CPU cores
Data Stall Analysis

1. DNNs need anywhere between 3 – 24 CPU cores per GPU for data pre-processing

2. Fetch stalls exist across models with large datasets
   - OS Page Cache is inefficient for DNN training due to thrashing

![Graph showing fetch stall (% of epoch time) for different models and setups.]

**Setup**
- V100 GPU
- 35% Cached
- 8GPU Training
Data Stall Analysis

1. DNNs need anywhere between 3 – 24 CPU cores per GPU for data pre-processing

2. Fetch stalls exist across models with large datasets
   • OS Page Cache is inefficient for DNN training due to thrashing

3. Redundancy in data fetch and pre-processing
Outline

• Data Stalls
• Analyzing Data Stalls
• CoorDL: Mitigating Data Stalls
• Evaluation
CoorDL: Insights

<table>
<thead>
<tr>
<th>Finding</th>
<th>Insight</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Page Cache is inefficient for DNN training due to thrashing</td>
<td>Optimize DNN cache to eliminate thrashing across epochs <em>(MinIO Cache)</em></td>
</tr>
<tr>
<td>Redundant data fetch in distributed training</td>
<td>Local caches of servers can be coordinated to fetch data from the remote cache to overcome storage I/O bottlenecks <em>(Partitioned Caching)</em></td>
</tr>
<tr>
<td>Redundant data fetch and prep in HP search</td>
<td>HP search jobs must coordinate data fetch &amp; prep <em>(Coordinated Prep)</em></td>
</tr>
</tbody>
</table>
## CoorDL: Insights

<table>
<thead>
<tr>
<th>Finding</th>
<th>Insight</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS Page Cache is inefficient for DNN training due to thrashing</td>
<td>Optimize DNN cache to eliminate thrashing across epochs <em>(MinIO Cache)</em></td>
</tr>
<tr>
<td>Redundant data fetch in distributed training</td>
<td>Local caches of servers can be coordinated to fetch data from the remote cache to overcome storage I/O bottlenecks <em>(Partitioned Caching)</em></td>
</tr>
<tr>
<td>Redundant data fetch and prep in HP search</td>
<td>HP search jobs must coordinate data fetch &amp; prep <em>(Coordinated Prep)</em></td>
</tr>
</tbody>
</table>
OS Page Cache is ineffective across epochs!

- Uses OS Page cache to cache the prefetched data items for subsequent epochs
- Unaware of DNN access pattern
- ResNet18 on OpenImages Dataset (Server – 8V100 GPUs, 500GB DRAM)

<table>
<thead>
<tr>
<th>Dataset size</th>
<th>645GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache size</td>
<td>65% (420GB)</td>
</tr>
<tr>
<td>Expected disk access</td>
<td>225GB (35%)</td>
</tr>
<tr>
<td>(stable state)</td>
<td></td>
</tr>
</tbody>
</table>
OS Page Cache is ineffective across epochs!

- ResNet18 on OpenImages Dataset (Server – 8V100 GPUs, 500GB DRAM)

<table>
<thead>
<tr>
<th>Dataset size</th>
<th>645GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache size</td>
<td>65% (420GB)</td>
</tr>
<tr>
<td>Expected disk access</td>
<td>225GB (35%)</td>
</tr>
<tr>
<td>(stable state)</td>
<td></td>
</tr>
<tr>
<td>DALI-Seq</td>
<td>422GB (87%)</td>
</tr>
<tr>
<td>DALI-Shuffle</td>
<td>340GB (53%)</td>
</tr>
</tbody>
</table>

Increased disk access makes training I/O bound => Fetch stalls
OS Page Cache is ineffective across epochs!

Across epochs, the items in OS Page Cache are not used effectively!
- Prefetched items replace existing, unused items in Page cache (LRU)
- These evicted items are prefetched from storage later in the epoch
- Models like ShuffleNet spend 40% of epoch time in blocking I/O
MinIO cache

- Given a cache capacity, fill it up with random data items when first accessed
- Once cache is full, unlike traditional caching, there is **no cache replacement**
- Disk accesses per epoch = capacity misses
Outline

• Data Stalls
• Analyzing Data Stalls
• CoorDL: Mitigating Data Stalls

• Evaluation
  • Setup
    • Single-Node Training
    • Multi-Node Training
    • Hyperparameter Search
## Evaluation : Setup

<table>
<thead>
<tr>
<th>Task</th>
<th>Model</th>
<th>Dataset (Size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image Classification</td>
<td>AlexNet, ShuffleNetv2, ResNet18, SqueezeNet, MobileNetv2, ResNet50, VGG11</td>
<td>ImageNet-1K (146GB), Imagenet-22K (1.3TB), OpenImages-Extended (645GB)</td>
</tr>
<tr>
<td>Object Detection</td>
<td>SSD + ResNet18</td>
<td>OpenImages (561GB)</td>
</tr>
<tr>
<td>Audio Classification</td>
<td>M5</td>
<td>Free Music Arxiv (950GB)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Servers</th>
<th>GPU Config</th>
<th>GPU Mem (GB)</th>
<th>Storage Media</th>
<th>Rand Read (MBps)</th>
<th>DRAM (GB)</th>
<th>CPU cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD-V100</td>
<td>8 x V100</td>
<td>32</td>
<td>SSD</td>
<td>530</td>
<td>500</td>
<td>24</td>
</tr>
<tr>
<td>HDD-1080Ti</td>
<td>8 x 1080Ti</td>
<td>11</td>
<td>HDD</td>
<td>15-100</td>
<td>500</td>
<td>24</td>
</tr>
</tbody>
</table>
Outline

• Data Stalls
• Analyzing Data Stalls
• CoorDL: Mitigating Data Stalls

• Evaluation
  • Setup
  • Single-Node Training
  • Multi-Node Training
  • Hyperparameter Search
1. Single-server training

Upto 1.8x faster training on SSD-V100 over DALI by reducing cache misses (minIO)
1. Single-server training

Upto 1.8x faster training on SSD-V100 over DALI by reducing cache misses (minIO)
Outline

• Data Stalls
• Analyzing Data Stalls
• CoorDL : Mitigating Data Stalls

• Evaluation
  • Setup
  • Single-Node Training
  • Multi-Node Training
  • Hyperparameter Search
2. Multi-server training

![Graph showing speedup with DALI-shuffle and CoordDL.](image)

Speedup wrt DALI

<table>
<thead>
<tr>
<th></th>
<th>1-node</th>
<th>2-nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DALI-shuffle</td>
<td><img src="image" alt="1-node" /></td>
<td><img src="image" alt="2-nodes" /></td>
</tr>
<tr>
<td>CoordDL</td>
<td><img src="image" alt="1-node" /></td>
<td><img src="image" alt="2-nodes" /></td>
</tr>
</tbody>
</table>

Alexnet – HDD-1080Ti
2. Multi-server training

minIO + Partitioned caching minimizes disk IO and accelerates training by upto 15x
2. Multi-server training

minIO + Partitioned caching minimizes disk IO and accelerates training by up to 15x on HDD and up to 3x on SSD.
Outline

• Data Stalls
• Analyzing Data Stalls
• CoorDL : Mitigating Data Stalls
• **Evaluation**
  • Setup
  • Single-Node Training
  • Multi-Node Training
  • **Hyperparameter Search**
3. HP search
3. HP search

Coordinated prep is able to speed up training by up to 5.5x by eliminating redundant pre-processing and disk IO.
Summary

• Data stalls exist in DNN training on commodity servers
  • Squander away benefits from fast GPUs
• Analyzed causes for data stalls
• Built CoorDL to mitigate I/O and CPU bottlenecks in some scenarios

Analysis → Data Stalls → Mitigation using CoorDL

DS-Analyzer
• Precisely measure data stalls
• Predictive what-if analysis

Challenges
• Decoded data cache
• Cost-performance tradeoff
Thank you!

Source code: https://github.com/msr-fiddle/DS-Analyzer

Contact: jaya@cs.utexas.edu