Informed Prefetching in I/O Bounded Distributed Deep Learning

Xiaojun Ruan[1] and Haiquan Chen[2]

[1] California State University, East Bay, xiaojun.ruan@csueastbay.edu
[2] California State University, Sacramento, haiquan.chen@csus.edu

3rd Workshop on Parallel AI and Systems for the Edge
co-conducted with 35th IEEE International Parallel & Distributed Processing Symposium

May 21, 2021

Outline

Outline

Deep Learning in Edge

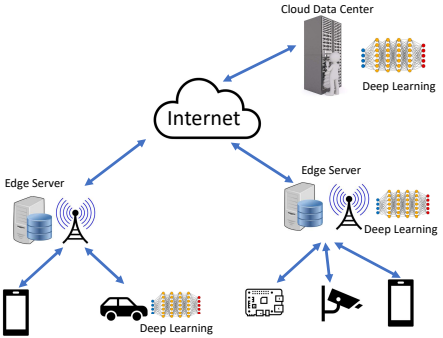


Figure: Deep Learning on Device, Edge Server, and Cloud Data Center

- Distributed deep learning is essential to support various edge computing-based applications which incur a significantly increased workload on cloud servers.
- Applications using Deep Neural Network have been widely deployed on mobile devices like smartphones and IoT sensors.
- A lot of training workloads will be assigned on Edge servers and cloud. So the deep training performance is critical on both edge and cloud.

Existing Solutions in Cloud and Edge Deep Learning

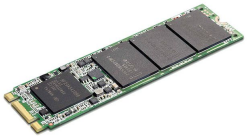
- GPU-accelerated deep learning on Edge and Cloud
- A cluster computing system with highperformance distributed file systems like Hadoop Distributed File System (HDFS)
  - accessing HDFS data involves local access and remote access.
  - remote data access includes network transfer delay.
  - Local data access time cost includes local Hard Drive Disk (HDD) or Solid State Drives (SSD) access time

Multi-Process and Distributed Training in PyTorch

- As a deep learning framework developed by Facebook, PyTorch offers multi-process data loading function that is able to load training dataset in parallel by using customizable number of processes as "DataLoader" workers.
- Although the multi-process data loading accelerates data I/O in training stage by creating multiple processes, it also increases the usage of CPU and memory on host computers.
- Furthermore, such multi-processing does not always improve performance when the overhead of context switching is high or the I/O bandwidth limit has been achieved.
- PyTorch also supports Distributed Deep Learning on a cluster of GPU servers.

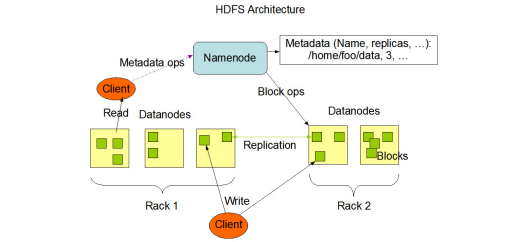
Outline

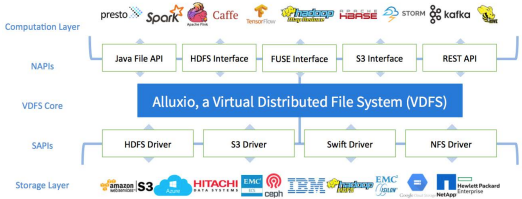
Storage Systems – Hard Drive Disks and Solid State Drives



- HDDs are still widely adopted in data centers for large capacity, low cost, and high sequential access performance.
- HDDs' performance is significantly affected by seeking and rotation delays, I/O performance could be a significant bottleneck when the training applications are accessing data on HDDs.
- SSDs are more capable of serving random accesses. Smaller capacity. Higher cost per GB. Frequent erasure rations reduce SSDs life, also cause write amplification which degrades I/O performance

Distributed File Systems and Virtual Distributed File Systems

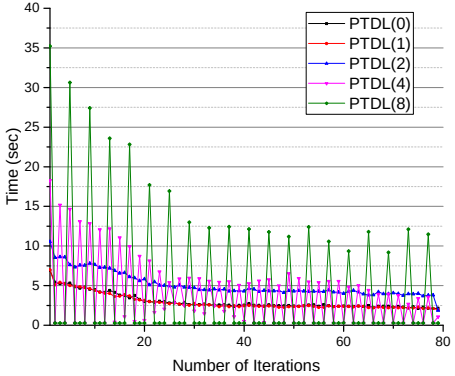




- Hadoop Distributed File System, or HDFS, is the most popular open-source distributed file system which offers high performance, reliability, and security.
- Alluxio, the first Virtual Distributed File System, was developed to use memory as the file system which significantly improves both local and remote data access for storing and caching data in local memory

Outline

I/O Bottleneck on Single GPU Server



No. of Extra Data Loader Worker

0 and 1: No noticeable difference.

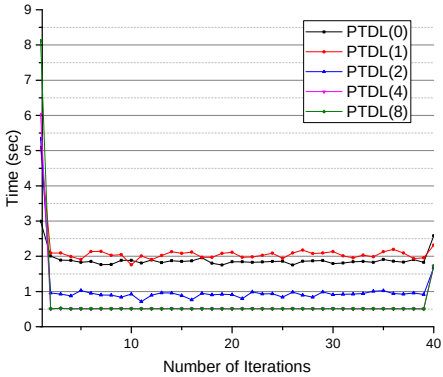
2: Worse Performance

4 and 8: improve some batches' performance

However, more data loader workers accessing one HDD in parallel may cause frequent head movements which causes frequent access time fluctuations for 4 and more workers.

The average response, training every 10 batches, original PyTorch Data Loader, Tiny ImageNet, Batch size 128, Resnet-18, Nvidia GTX 1070 Ti, 12GB main memory.

I/O Bottleneck in Deep Learning Cluster



No. of Extra Data Loader Worker

2 performs much better than 1 and 0
4 and 8 can achieve the best performance without performance fluctuation

Reasons

1) Distributed storage system has higher bandwidth
2) VDFS Alluxio is memory based so it handles parallel I/O requests much better than HDDs

The average response, training every 10 batches, original PyTorch Data Loader, Tiny ImageNet, Batch size 128, Resnet-18, 2 Training Node, 1 Nvidia RTX 2060/node, Alluxio DFS, 16GB, 10 Storage Nodes.

Outline

Memory Architecture

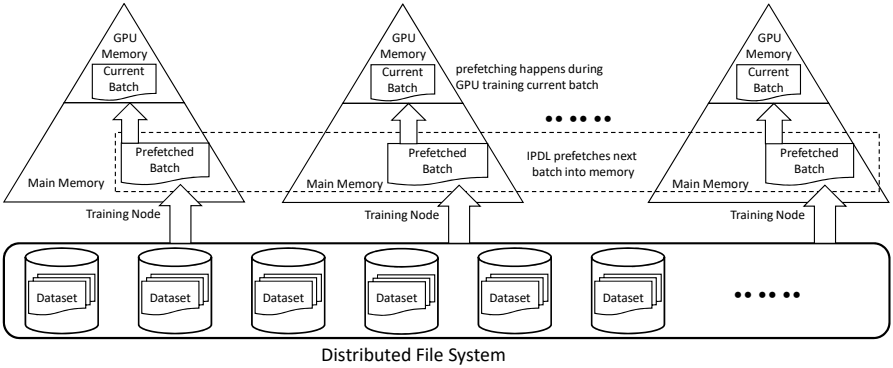
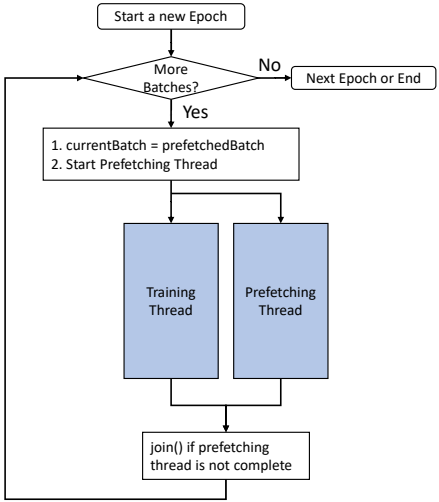


Figure: Memory Architecture of Informed Prefetching Data Loader in Distributed Deep Learning

IPDL fetches the batch for next iteration when GPU is training the current batch. Each training node has an individual IPDL for data prefetching from DFS to local main memory.

Informed Prefetching Data Loader Flow Chart



At each iteration of deep learning, a thread is created to fetch the next iteration's batch in parallel with training the current batch. If prefetching is not complete by the end of training, the program needs to pause until the data is completely loaded into main memory then start the next iteration.

Algorithm 1 Informed Prefetching Data Loader

```
1:  for each epoch do
2:      for each batch in current epoch do
3:          if is the first iteration of the epoch then
4:              fetch current batch
5:              create a new thread to prefetch next batch
6:          else
7:              current batch ← prefetched batch
8:              create a new thread to prefetch next batch
9:          end if
10:     end for
11:     Training batch...
12:     if Prefetching is not complete then
13:         wait till prefetching thread is complete
14:     end if
15: end for
```

System Implementation

- IPDL is implemented on PyTorch 1.4 by inheriting *DataLoader* and *_SingleProcessDataLoaderIter* classes.
- Since the file access pattern is decided at the beginning of each epoch, the prefetch of the next batch will start in parallel with the training of the current batch.
- A separate prefetching thread starts to prefetch the next batch of data at the beginning of each batch.
- If the prefetching is incomplete the iteration has to wait until the prefetching thread is complete.
- If the prefetching could not complete, it still reduces time cost since the data loading of the next batch was partially in parallel with the current training.

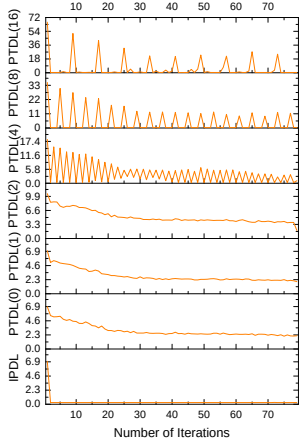
Testbeds

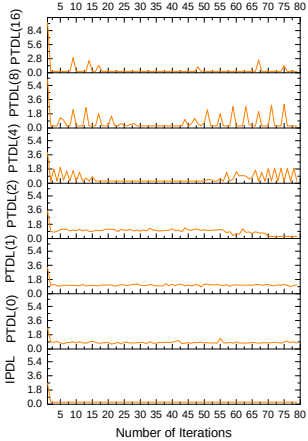
Table: Testbeds Configuration

1-Node Server	10-Node Cluster
Intel Xeon E3-1225 V3	Intel i7-9700
12GB Main Memory	16GB Main Memory
Nvidia GTX 1070 Ti with 8GB	Nvidia RTX 2060 with 6GB
1 HDD, NTFS	HDFS 2.7.3, Alluxio 2.2

Outline

One Training Node, PyTorch Dataloader vs. Informed Prefetching





Average Time cost of 10 iterations. Tiny ImageNet, Batch size 128 ResNet-18 **Left**) One Server, One HDD, Nvidia GTX 1070 Ti; **Right**) 1 Training Node, Nvidia RTX 2060, 10 Storage Nodes

Distributed Training Environment

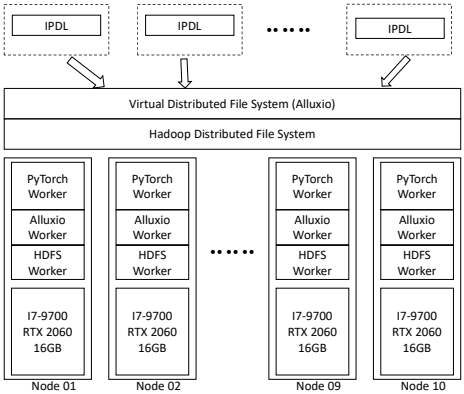
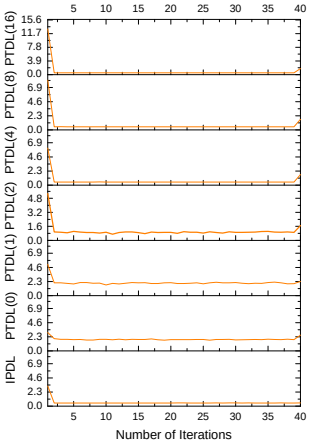
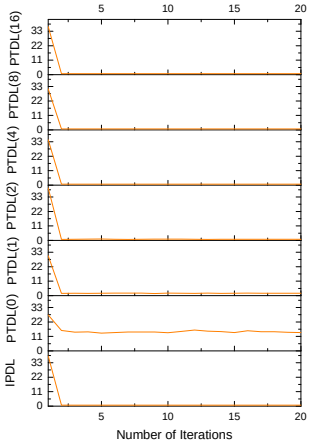


Figure: 10-node homogeneous cluster computing system, HDFS on all 10 nodes, Alluxio is built on top of HDFS. Each node can serve as both storage and computing node. Both the default PTDL and IPDL load batches via Alluxio.

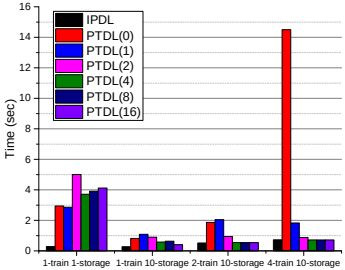
Distributed Training, PyTorch Dataloader vs. Informed Prefetching



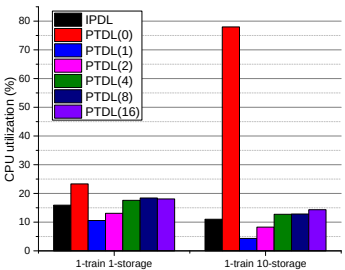


Average Time cost of 10 iterations. Distributed Training, Batch Size 128, Tiny ImageNet, ResNet-18. **Left**) 2 Training Nodes; **Right**) 4 Training Nodes, 10 Storage Nodes

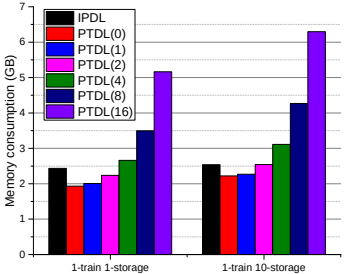
Performance Comparison of IPDL and PTDL



(a) Overall Average Time Cost.



(b) Median CPU Utilization.



(c) Median Memory Usage.

Batch Size 128, Tiny ImageNet, ResNet-18, PyTorch Data Loader number 0, 1, 2, 4, 8, 16.