AISTORE OVERVIEW
WHY BOTHER?

- GPUs are getting faster and faster
  - Hard to keep feeding them data
  - Training data needs to be moved from SSD/HDD/Network -> CPU -> GPU
- Need a distributed system for training ML models -> distributed way to access training data
  - Modern SotA models use billions of parameters and train on terabytes of data
  - GPT-3 has 175 billion parameters and was trained on 45TB of text data
  - Imagine how much data Tesla gets/uses to train self-driving models
- Deep Learning workloads would benefit from features not in other distributed systems
AISTORE

AIStore is a lightweight object storage system with a focus on AI workloads.

- Linear scale-out with each added storage node
- Supports public clouds
- Bare-metal or containerized deployment
- Linear scale-out with each added storage node
- Fully open-source

But why build another distributed storage service?
BASIC ARCHITECTURE

- Nodes can be **proxies** or **targets** and can run on commodity hardware
- Proxies are stateless and take care of client requests
  - Cluster information
  - Control plane
  - 1 primary proxy keeps track of the cluster map
- Targets are stateful and provide disk space to the cluster
  - Store datasets (or chunks of them)
  - Serve/store data
- Data is stored in “buckets”, just like AWS S3
  - Buckets can be replicated, erasure-coded
MAIN DATA SOURCES

• Cluster Map
  • Information about the cluster - node information, uptime, statuses, etc.

• Cluster Configuration
  • Various configuration knobs that can be set by the user

• Bucket Metadata
  • Information regarding buckets - bucket contents, bucket list, bucket location, etc.

• Volume Metadata
  • Information about volumes mounted to each node

All this information must be replicated uniformly throughout the cluster.
WEBDATASET

- Tries to solve issues with using large datasets
  - You want to get subsets and shuffles of your large datasets
- Datasets are tar files, and each shard gets its own tar file
- Don’t have to manually untar to use dataset (WebDataset library does it?)
- Don’t have to have whole dataset to start training
- Can use web server/cloud storage as source for data (don’t need to store locally)

For example, ImageNet is stored in 1282 separate 100 Mbyte shards with names `pythonimagenet-train-000000.tar` to `imagenet-train-001281.tar`, the contents of the first shard are:

```
-1-1-1-1- bigdata/bigdata 3 2020-05-08 21:23 n03991062_24866.cls
-1-1-1-1- bigdata/bigdata 108611 2020-05-08 21:23 n03991062_24866.jpg
-1-1-1-1- bigdata/bigdata 3 2020-05-08 21:23 n07749582_9506.cls
-1-1-1-1- bigdata/bigdata 129044 2020-05-08 21:23 n07749582_9506.jpg
-1-1-1-1- bigdata/bigdata 3 2020-05-08 21:23 n03425413_23604.cls
-1-1-1-1- bigdata/bigdata 106255 2020-05-08 21:23 n03425413_23604.jpg
-1-1-1-1- bigdata/bigdata 3 2020-05-08 21:23 n02795169_27274.cls
```
SMALL FILE PROBLEM

• Storing millions of small files can cause issues
• Every file, directory and block in HDFS is represented as an object in memory
  • Objects are usually 150 bytes
  • 10 million files -> 3 GB of memory
• DL datasets are almost always lots of small files, especially image datasets
  • Image augmentation and other techniques only make this problem larger
• Small files will cause random reads and not sequential reads
SHUFFLING

Client → Proxy → Map → Shuffle → Reduce

- Extract
- Sort
- Create

Shards are numbered consecutively.
Advantages:

- Popular solution
- Works on commodity hardware
- Compatible with other Apache projects

Disadvantages:

- Not always linearly scalable thanks to JVM
- Not purpose-built for deep learning workloads
  - No cloud provider support
  - Small file issue
CEPH

Advantages:

• Built for object storage
• Self-healing
• Amazon S3 compatibility layer

Disadvantages:

• Time consuming setup
• Need many nodes for full performance
• No cloud provider support
Advantages:

- Highly Scalable
- POSIX compatibility layer

Disadvantages:

- Time consuming setup
- Not performant with commodity networking
- Built for file storage
AIS Cluster

DL Client

Proxy

Target

HDFS Cluster

DL Client

HDFS Client

NameNode

Datanode

Datanode

Any client will receive data directly from the storage source
EXTRACT TRANSFORM LOAD (ETL)

• “Moving computation is easier than moving data”
• Users can provide a docker image to AIStore and run inline/offline ETL
  • Inline: ETL runs on data as clients want to read it
  • Offline: ETL runs on data, output becomes another dataset
• Use cases
  • Preprocess data before training
  • Distributed training
  • Distributed inferencing
OTHER FEATURES

- Can easily import data from AWS, GCP, Azure, and even HDFS or HTTP
- Small-file problem from HDFS solved
  - Many DL training datasets are comprised of small files
  - Use dSort to reshard small files into larger shards
- Easy to consume data
  - Pytorch and Tensorflow compatibility through WebDataset and AisDataset, respectively
  - POSIX/REST APIs provided for data access
- Easily deployable
  - Developed with Kubernetes in mind (operators support, deployment examples)
PERFORMANCE

Fig. 6. ResNet-50 training on a variety of inflated ImageNet datasets and storage backends (for labels see Table I).

Fig. 7. ResNet-50 training: throughput per-GPU.

Graphs from the AIStore whitepaper
QUESTIONS
TAKEAWAYS

• Paper isn’t well written and focuses on wrong things
• Feature set of Aistore is large and useful
  - ETL can help with preprocessing
  - Support for erasure-coding, replication
  - Lots of ease-of-use features
• Opposite of other papers- code is available and documented, paper doesn’t make it seem exciting or notable
EXAMPLES OF DISTRIBUTED TRANSACTIONS
1. Client sends request to cluster
2. Forward to primary if needed
3. Mark node as “in maintenance”
   1. Broadcast cluster map
   2. Rebalance if needed
4. Tell node to switch off
SAMPLE TRANSACTION

Shutdown a Node

1. Client sends request to cluster
2. Forward to primary if needed
3. **Mark node as “in maintenance”**
   1. Broadcast cluster map
   2. Rebalance if needed
4. Tell node to switch off
1. Client sends request to cluster
2. Forward to primary if needed
3. Mark node as “in maintenance”
   1. Broadcast cluster map
   2. Rebalance if needed
4. Tell node to switch off