Spark and Deep Learning frameworks with distributed workloads
Agenda

• Machine Learning and Deep Learning on a scalable infrastructure
• Comparative look - CPU vs CPU+GPU vs GPU
• Challenges with distributing ML pipelines and algorithms
• Frameworks that support distributed scaling
• Spark/YARN on GPUs vs CPUs
Machine Learning and Deep Learning on a scalable infrastructure
Machine Learning on a scalable infrastructure

- Single machine multiple CPU cores
- Training distributed across machines - still mostly using CPUs
- For Deep Learning - Sometimes requires GPUs
- Distributed across machines
- Mixed setup (CPU+GPU)
Deep Learning on a scalable infrastructure

Model parallelization, Data parallelization

- Single machine - multiple GPUs
- Distributed deep learning across machines - sometimes inevitable
Deep Learning - why resource hungry

Model parallelization challenges

- Memory in neural networks - to store input data, weight parameters and activations as an input propagates through the network.
- GPUs' reliance on dense vectors - fill SIMD compute engines
- CPU/GPU intensive - matrix multiplications - weights x activations.

Example: 50-Layer ResNet

Using 32-bit floating-point - parallelise training data
Mini-batch of 32
7.5 GB of local DRAM
CPU vs CPU+GPU vs GPU
CPU vs GPU

**CPU**
- Few very complex cores
- Single-thread performance optimization
- Transistor space dedicated to complex ILP
- Few die surface for integer and fp units
- Has other memory types but they are provisioned only as caches, not directly accessible to the programmer

**GPU**
- Hundreds of simpler cores
- Thousands of concurrent hardware threads
- Maximize floating-point throughput
- Most die surface for integer and fp units
- Has several forms of memory of varying speeds and capacities - memory types are exposed to the programmer
Why is CPU the new bottleneck?

- Hardware configurations offer large aggregate IO bandwidth
- Spark’s optimizer allows many workloads - avoiding significant disk IO
- Spark’s shuffle subsystem, serialization & hashing key - bottlenecks
- Spark constrained by CPU efficiency and memory pressure rather than IO.

Challenges with distributing ML & DL pipelines, algorithms & processes
Training

- Compute heavy
- Requires synchronization across network of machines
- GPUs often necessary/beneficial
Inference

• Embarrassingly parallel
• Varying latency requirements
• Can be mostly done on CPUs - GPUs often unnecessary
Hyper-parameter optimization

Parameters that define the model architecture are referred to as hyperparameters.

HPO (hyperparameter tuning) is a process of searching for the ideal model architecture.

- HPO is a necessary part of all model training
- It is embarrassingly parallel
- Can avoid distributed training

Scenario:
- Run 4 HP configurations, 1/gpu, in parallel vs. 4 HP configurations, 1/4gpu, in serial
Transfer learning

*Transfer learning - machine learning where “knowledge” learned on one task is applied to another, related task.*

- Take publicly available models and re-purpose them for your task
  - Leverage the work from hundreds of GPUs that is already baked in
- Train a small percentage of the model for your task
- Greatly reduced computational requirements mean you may not need GPUs or may not need distributed architecture
Current Landscape

- Machine Learning Frameworks
- Deep learning Frameworks
- Distributed Frameworks
- Spark based Frameworks
Distributed Machine Learning Frameworks
Machine learning on Spark

ML Frameworks

• Spark stores the model parameters in the driver
• Workers communicate with the driver to update the parameters after each iteration.
• For large scale machine learning deployments, the model parameters may not fit into the driver node and they would need to be maintained as an RDD.
• Drawback -
  • This introduces a lot of overhead because a new RDD will need to be created in each iteration to hold the updated model parameters.
  • Since updating the model usually involves shuffling data across machines, this limits the scalability of Spark.
Using PMLS Parameter-Server framework

ML Frameworks

• Both data and workloads are distributed over worker nodes
  • server nodes maintain globally shared parameters
  • represented as dense or sparse vectors and matrices
• The framework manages asynchronous data communication between nodes
• Flexible consistency models
• Elastic scalability
• Continuous fault tolerance.

Image Source: https://cse.buffalo.edu/~demirbas/publications/DistMLplat.pdf
Distributed Deep Learning Frameworks
Distributed TensorFlow

DL Frameworks

- Tensorflow distributed relies on master, worker, parameter server processes
- Provides fine-grained control, you can place individual tensors and operations
- Relies on a cluster specification to be passed to each process
- Need to take care of booting each process and syncing them up somehow
- Uses Google RPC protocol

Source - https://www.tensorflow.org/deploy/distributed
Distributed TensorFlow

```
# In task 0:
cluster = tf.train.ClusterSpec({"local": ["localhost:2222",
  "localhost:2223"]})
server = tf.train.Server(cluster, job_name="local", task_index=0)

# In task 1:
cluster = tf.train.ClusterSpec({"local": ["localhost:2222",
  "localhost:2223"]})
server = tf.train.Server(cluster, job_name="local", task_index=1)
```
PyTorch (torch.distributed)

DL Frameworks

• Has a distributed package that provides MPI style primitives for distributing work
• Has interface for exchanging tensor data across multi-machine networks
• Currently supports four backends (tcp, gloo, mpi, nccl - CPU/GPU)
• Only recently incorporated
• Not a lot of documentation

import torch
dist.init_process_group(backend="nccl",
                        init_method="file:///distributed_test",
                        world_size=2,
                        rank=0)
tensor_list = []
for dev_idx in range(torch.cuda.device_count()):
    tensor_list.append(torch.FloatTensor([1]).cuda(dev_idx))

tensor_list.append(torch.FloatTensor([1]).cuda(dev_idx))

dist.all_reduce_multigpu(tensor_list)
Apache MXNet

DL Frameworks

• Parameter server/worker architecture
• Must compile from source to use
• Provide built-in “launchers”, e.g. YARN, Kubernetes, but still cumbersome
• Data Loading(IO) - Efficient distributed data loading and augmentation.
• Can specify the context of the function to be executed within - that tells if it should be run on CPU or GPU

import mxnet.ndarray as nd

X = nd.zeros((10000, 40000), mx.cpu(0))
#Allocate an array to store 1000 datapoints (of 40k dimensions) that lives on the CPU
W1 = nd.zeros(shape=(40000, 1024), mx.gpu(0))
#Allocate a 40k x 1024 weight matrix on GPU for the 1st layer of the net
W2 = nd.zeros(shape=(1024, 10), mx.gpu(0))
#Allocate a 1024 x 1024 weight matrix on GPU for the 2nd layer of the net
Horovod is a distributed training framework for TensorFlow, Keras, and PyTorch.

- Released by Uber to make data parallel deep learning using TF easier
- Introduces a ring all-reduce pattern to eliminate need for parameter servers
- Uses MPI
- Require less code changes than the Distributed TensorFlow with parameter servers

To run on 4 machines with 4 GPUs each

```
$ mpirun -np 16 \
  -H server1:4,server2:4,server3:4,server4:4 \n  -bind-to none -map-by slot \n  -x NCCL_DEBUG=INFO -x LD_LIBRARY_PATH -x PATH \n  -mca pml ob1 -mca btl ^openib \
  python train.py
```
Kubeflow

DL Frameworks

Machine Learning Toolkit for Kubernetes

- Custom resources for deploying ML tasks on Kubernetes
  Currently basic support for TF only
- Boots up your TF processes
- Can place on GPUs, automatic restarts, etc...
- Configures Kubernetes services for you
- Still need to know lots of Kubernetes, plus KSonnet!

Image source: https://github.com/kubeflow/kubeflow/issues/33
DL4J

DL Frameworks

- Deep learning in Java
- Comes with Spark support for data parallel architecture
- Also takes advantage of Hadoop
- Works with multi-GPUs
- Also has feature engineering/preprocessing tools, model serving tools, etc...

```java
ParallelWrapper wrapper = new ParallelWrapper.Builder(model)
    .prefetchBuffer(24)
    .workers(2)
    .averagingFrequency(3)
    .reportScoreAfterAveraging(true)
    .build();
```
BigDL

DL Frameworks

• Built for Spark, only works with Spark
• No GPUs!! CPUs are fast too, Intel says
• Good option if no GPU and tons of commodity CPUs
• Whole team of devs from Intel working on it
• If you have a Spark cluster and want to do DL and are not super concerned with performance - then consider BigDL

BigDL on Kubernetes

```bash
$SPARK_HOME/bin/spark-submit \
  --deploy-mode cluster --class \
  com.intel.analytics.bigdl.models.lenet.Train \
  --master \
  k8s://https://<k8s-apiserver-host>:<k8s-apiserver-port> --kubernetes-namespace default \
  --conf spark.executor.instances=4 \
  ...
$BIGDL_HOME/lib/bigdl-0.4.0-SNAPSHOT-jar-with-dependencies.jar -f \
  hdfs://master:9000/mnist \
  -b 128 -e 2 --checkpoint /tmp
```
TensorflowOnSpark

DL Frameworks

• Lightweight wrapper that boots up distributed Tensorflow processes in Spark executors
• Potentially high serialization costs
• Not super active development
• Supports training (CPU/GPU) and inference
• Easy to integrate with other Spark stages/algos

ML Pipeline with multiple programs on separate clusters

Image source: https://goo.gl/CX8N9B
Spark DL Pipelines

Tensorframes

- Released by Databricks for doing Transfer Learning and Inference as a Spark pipeline stage
- Good for simple use cases
- Relies on Databricks’ Tensorframes

```python
featurizer = DeepImageFeaturizer(inputCol="image", outputCol="features", modelName="InceptionV3")
lr = LogisticRegression(maxIter=20, regParam=0.05, elasticNetParam=0.3, labelCol="label")
p = Pipeline(stages=[featurizer, lr])
```

For technical preview only
On YARN

- GPU on YARN
- Spark on YARN
- TensorFlow on YARN
Hadoop YARN

Support for GPUs

• YARN is the resource management layer for the Apache Hadoop ecosystem.
• Pre Hadoop 3.1 had CPU and memory hard-coded as the only available types of consumable resources.
• With Hadoop 3.1 YARN is declaratively configurable - can create GPU type resources for which YARN will track consumption.
GPU On YARN
Hadoop 3.1.x

- As of now, only Nvidia GPUs are supported by YARN
- YARN node managers have to be pre-installed with Nvidia drivers.
- When Docker is used as container runtime context, nvidia-docker 1.0 needs to be installed
- One can set additional configurations to allow admins leverage specialized requirements.

```
<configuration>
  <property>
    <name>yarn.resource-types</name>
    <value>yarn.io/gpu</value>
  </property>
</configuration>
```

```
<property>
  <name>yarn.nodemanager.resource-plugins</name>
  <value>yarn.io/gpu</value>
</property>
```

Deep Learning on Hadoop

HDL

• A new layer in Hadoop for launching, distributing and executing Deep Learning workloads
• Leverage and support existing Deep Learning engines (TensorFlow, Caffe, MXNet)
• Extend and enhance YARN to support the desired scheduling capabilities (for FPGA, GPU)

Source - https://github.com/Intel-bigdata/HDL
TensorFlow on YARN

Toolkit to enable Hadoop users an easy way to run TensorFlow applications in distributed pattern and accomplish tasks including model management and serving inference.

- One YARN cluster can run multiple TensorFlow clusters
- The tasks from the same and different sessions can run in the same node
Spark on YARN (with GPU)
On Hadoop 3.1.0

- First class GPU support
- YARN clusters can schedule and use GPU resources
- To get GPU isolation - and to pool GPUs Hadoop YARN cluster should be Docker enabled.

Source - https://issues.apache.org/jira/browse/YARN-6223
Other frameworks on YARN

Work in progress

• **CaffeOnYARN**
  - Caffe on YARN is a project to support running Caffe on YARN, based on CaffeOnSpark from yahoo to rebase on YARN by removing Spark dependency. It's a part of Deep Learning on Hadoop (HDL).
  - **Note** - *Current project is a prototype with limitation and is still under development.*

• **MXNetOnYARN**
  - MXNet on YARN is a project based on dmlc-core and MXNet, aiming at running MXNet on YARN with high efficiency and flexibility. It's an important part of Deep Learning on Hadoop (HDL).
  - **Note** - *both the codebase and documentation are work in progress. They may not be the final version.*

Sources
- [CaffeOnYARN](https://github.com/Intel-bigdata/CaffeOnYARN)
- [MXNetOnYARN](https://github.com/Intel-bigdata/MXNetOnYARN)
CDH 6.x

- YARN Improvements
CDH 6.0

Work in progress

- Ability to add arbitrary consumable resources (via YARN configuration files) and have the FairScheduler schedule based on those consumable resources.
- Some examples of consumable resources that can be used are GPU and FPGA.
- Support for MapReduce
CDH 6.1

Upcoming features

• Boolean (i.e. non-consumable) Resource Types that can be used for labeling nodes and scheduling based on those.
• Some examples are nodes that have a specific license installed or nodes that have a specific OS version.
• Support for Spark
• CM UI to be able to configure Resource Types
• Preemption capabilities with arbitrary Resource Types
THANK YOU

Machine Learning Presentations: cloudera.com/ml