DL over Big Data & BigDL

Wei Liu
Outline

DLoBD: A comprehensive study of deep learning over big data stacks on HPC clusters.

BigDL: A distributed deep learning framework for big data.
In Proceedings of the ACM Symposium on Cloud Computing (pp. 50-60).
DLoBD: A comprehensive study of deep learning over big data stacks on HPC clusters.

Deep Learning over Big Data (DLoBD)

• Deep Learning over Big Data (DLoBD) is one of the most efficient analyzing paradigms

• More and more deep learning tools or libraries (e.g., Caffe, TensorFlow) start running over big data stacks, such as Apache Hadoop and Spark

• Major benefits of DLoBD:
  1) Easily build a powerful data analytics pipeline
  2) Better data locality
  3) Efficient resource sharing and cost effective
Examples of DLoBD stacks

• CaffeOnSpark
• SparkNet
• TensorflowOnSpark
• DL4J
• BigDL
• CNTKOnSpark
5 Major layers of DLoBD stacks

- Deep Learning model (application layer)
- Deep Learning library layer
- Big Data analytics framework layer
- Resource scheduler layer
- Distributed file system layer
Overview of DLoBD Stacks - CaffeOnSpark

- Spark Driver: Job Launching and Job Control
- Spark Executor: For data feeding and task control
- Model Synchronizer: Communicates across nodes with RDMA\TCP, and output model file on HDFS
- Scalable and Communication intensive
  - Server-to-server direct communication (Ethernet or InfiniBand) achieves faster learning and eliminates scalability bottleneck
  - Out-of-band communication
Overview of DLoBD Stacks - TensorFlowOnSpark

- Spark Executors acting as containers used to run TensorFlow code

- Two different modes to ingesting data
  - Read data directly from HDFS using built-in TensorFlow modules
  - Feeding data from Spark RDDs to Spark executors (TensorFlow core)

- Scalable and Communication intensive
  - Parameter Server-based approach
  - Embedded inside one Spark executor and talk to other workers over gRPC or gPRC with RDMA
  - Out-of-band communication
Overview of DLoBD Stacks - CNTKOnSpark

- Microsoft Cognitive Toolkit (CNTK) and OpenCV into Spark Machine Learning pipelines without data transfer overhead

- Feeding data for CNTK Core (e.g. images or texts) can be directly read from HDFS by Spark Executors

- Scalable and Communication intensive
  - Embedded inside one Spark executor and talk to other workers over MPI (RDMA, TCP)
  - Out-of-band communication
Overview of DLoBD Stacks - BigDL

- Users can write deep learning applications as Spark programs
- Users can load pre-trained Caffe or Torch models into Spark programs using BigDL
- Feed data to BigDL core by Spark Executor which can directly load data from HDFS
- Support Intel MKL
- Scalable and Communication intensive
  - Spark block manager as parameter server
  - Organically designed and integrated with Spark architecture
  - in-band Communication
Overview of Characterization Methodology

- Choose proper DL workloads, models and datasets
- Choose representative DLoBD stacks
- Define characterization dimensions
- Generate evaluation reports
### Selected Datasets and Models

#### TABLE 2

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers (Convolutional/Full-connected)</th>
<th>Dataset</th>
<th>Description</th>
<th>Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeNet</td>
<td>2 / 2</td>
<td>MNIST</td>
<td>A CNN designed for handwritten and machine-printed character recognition</td>
<td>CaffeOnSpark, TensorFlowOnSpark</td>
</tr>
<tr>
<td>SoftMax Regression</td>
<td>NA / NA</td>
<td>MNIST</td>
<td>A logistic function that compresses a vector to another vector of real values in the range (0, 1) that add up to 1</td>
<td>TensorFlowOnSpark</td>
</tr>
<tr>
<td>CIFAR-10 Quick</td>
<td>3 / 1</td>
<td>CIFAR-10</td>
<td>A model reproduced from Alex Krizhevsky’s cuda-convnet</td>
<td>CaffeOnSpark, TensorFlowOnSpark, MMSpark</td>
</tr>
<tr>
<td>VGG-16</td>
<td>13 / 3</td>
<td>CIFAR-10</td>
<td>A deep convolutional network for object recognition</td>
<td>BigDL</td>
</tr>
<tr>
<td>AlexNet</td>
<td>5 / 3</td>
<td>ImageNet</td>
<td>A CNN architecture designed to deal with complex object classification task, won ILSVRC 2012</td>
<td>CaffeOnSpark</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>22 / 0</td>
<td>ImageNet</td>
<td>A CNN architecture with an Inception module, won ILSVRC 2014</td>
<td>CaffeOnSpark</td>
</tr>
<tr>
<td>ResNet</td>
<td>53 / 1</td>
<td>Synthetic</td>
<td>A deep convolutional network based on residual learning framework</td>
<td>TensorFlow</td>
</tr>
</tbody>
</table>

#### TABLE 1

<table>
<thead>
<tr>
<th>Dataset</th>
<th>MNIST</th>
<th>CIFAR-10</th>
<th>ImageNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>Digit Classification</td>
<td>Object Classification</td>
<td>Object Classification</td>
</tr>
<tr>
<td>Resolution</td>
<td>28 × 28 B&amp;W</td>
<td>32 × 32 Color</td>
<td>256 × 256 Color</td>
</tr>
<tr>
<td>Classes</td>
<td>10</td>
<td>10</td>
<td>1,000</td>
</tr>
<tr>
<td>Training Images</td>
<td>60 K</td>
<td>50 K</td>
<td>1.2 M</td>
</tr>
<tr>
<td>Testing Images</td>
<td>10 K</td>
<td>10 K</td>
<td>100 K</td>
</tr>
</tbody>
</table>
Performance Evaluation on CPU versus GPU

- DL workloads can benefit from the high performance of the DLoBD stacks.
- Network will become a bottleneck at some point if the sub-optimal IPoIB network protocol is used.
- GPU/ GPU+cuDNN can get the best performance. GPU + cuDNN is degraded at a large scale (e.g., 16 nodes).
- For some models, solutions with CPU + MKL may outperform GPU-based solutions.
Performance Evaluation on IPoIB versus RDMA

- CaffeOnSpark benefits from the high performance of RDMA compared to IPoIB once communication overhead becomes significant.
- Experiments show that the default RDMA design in TensorFlowOnSpark is not fully optimized yet. For MNIST tests, RDMA is not showing obvious benefits.
- OpenMPI-based communication over IPoIB and RDMA; Similar performance; The latency and bandwidth of IPoIB in experiment cluster are sufficient for small models.
Evaluation on Performance and Accuracy

- Performance Evaluation of CaffeOnSpark (training time to achieve a 70% accuracy)
  - RDMA reduces the overall time cost by 22% in training AlexNet on ImageNet
  - RDMA reduces the overall time cost by 15% in training GoogleLeNet on ImageNet
- Performance Evaluation of BigDL (training time to achieve a 70% accuracy)
  - RDMA reduces the overall time cost by 48% in training VGG on CIFAR-10
Epoch-Level and Scalability Evaluation

- Epoch-level evaluation of training VGG model using BigDL on default Spark with IPoIB or RDMA
- RDMA version takes constantly less time than the IPoIB version to finish every epoch.
  RDMA finishes epoch 18 in 2.6x time faster than IPoIB

- For VGG model trained with BigDL, RDMA-based Spark scales better than default IPoIB Spark
- A speedup of 2.7x using RDMA for the epoch-level training time
Evaluation on Resource Utilization

- CIFAR-10 Quick Model and CIFAR-10 Dataset
- GPU-based solutions use less memory than CPU-based ones as they mostly use GPU memory.
- CPU + MKL solution uses host memory more efficiently and has better performance than CPU + OpenBLAS.
- RDMA utilizes the network resources more efficiently than the IPoIB in CaffeOnSpark.
- CaffeOnSpark still does not fully utilize the high throughput characteristic of RDMA and memory resource.
Performance Overhead across Layers of TensorFlowOnSpark

• SoftMax Regression model, over MNIST Dataset

• Up to 15.5% time in Apache Hadoop YARN scheduler layer
• Up to 18.1% execution time in Spark job execution layer

• Data size is small, so we do not count the time spent on accessing HDFS layer.

• Need more effort to reduce the overhead across different layers of DLoBD stacks
• Maybe amortized in long-running deep learning jobs
• Use gRPC, gRPC+Verbs, and gRPC+MPI based channels for Process-to-Process tensor communication.

• Verbs-based RDMA communication channel could use the RDMA-capable networks in a much more efficient manner than the IPoIB protocol.
Performance of Different Communication Channels

- Train ResNet50 using 2, 4 and 8 nodes (2 GPUs per node) TensorFlow cluster deployed in parameter server (1 PS and rest workers) mode
- Process around 8 percent (2 nodes), 15 percent (4 nodes), and 21 percent (8 nodes) more images when RDMA channel is used compared to the scheme of using IPoIB
- MPI-based channel performance is similar to the IPoIB channel.
- Current design of the MPI-based channel in TensorFlow could not use MPI in the best manner.
Observations from experiments

• For complex deep neural network models, RDMA contributes clear benefit for TensorFlow performance.

• Increase the number of worker nodes, RDMA can deliver more performance benefit than IPoIB.

• For complex deep neural network models deployed in a larger scale, where the Process-to-Process communication is more frequent during the training time, RDMA can improve TensorFlow performance.
Conclusions

• RDMA scheme can benefit Deep Learning workloads. The RDMA scheme can also scale better and utilize resources more efficiently than the IPoIB scheme over InfiniBand clusters.

• Both GPU and CPU can compute Deep Learning workloads faster with their co-designed efficient Deep Learning oriented libraries, such as cuDNN and Intel MKL.

• The current generation DLoBD stacks still can not utilize all the available cluster resources efficiently. There are still large rooms for them to be further improved.
BigDL: A distributed deep learning framework for big data.

Big Data Analytics Challenges

- Deep learning benchmarks (ImageNet, SQuAD, etc.)
  Curated and explicitly labelled Dataset
  Suitable for dedicated DL systems

- Real-world production data pipeline
  Dynamic, messy (and possibly implicitly labeled) dataset
  Suitable for integrated data analytics and DL pipelines using BigDL

- Problems with “connector approaches”
  TFX, TensorFlowOnSpark, Project Hydrogen, etc.
  Adaptation overheads, impedance mismatch
Introduction to BigDL

• A distributed deep learning framework for big data platforms and workflows
  - A library on top of Apache Spark
  - Support an API similar to Torch and Keras
  - Support both large-scale distributed training and inference, leveraging the scale-out architecture of the underlying Spark framework

• Provide an expressive, "data-analytics integrated" deep learning programming model
  - Within a single, unified data analysis pipeline, users can efficiently process very large dataset using Spark APIs
  - Allow new deep learning algorithms to be seamless integrated into production data pipelines, which can then be easily deployed, monitored and managed in a single unified big data platform.
The end-to-end text classification pipeline on Spark and BigDL

```python
#distributed data processing
spark = SparkContext(appName="text_classifier", ...)
input_rdd = spark.textFile("hdfs://...")
train_rdd = input_rdd.map(lambda x: read_text_and_label(x))
    .map(lambda data: decode_to_ndarrays(data))
    .map(lambda arrays: to_sample(arrays))

#distributed training
model = Sequential().add(Recurrent()).add(LSTM(...))
    .add(Linear(...)).add(LogSoftMax())
optimizer = Optimizer(model=model, training_rdd=train_rdd,
    criterion=ClassNLLCriterion(),
    optim_method=Adagrad(), ...)
trained_model = optimizer.optimize()

#distributed inference
test_rdd = ...
prediction_rdd = trained_model.predict(test_rdd)
```
Spark Execution Model

- The driver is responsible for coordinating tasks in a Spark job
- The workers are responsible for the actual computation
Distributed Training in BigDL

Algorithm 1 Data-parallel training in BigDL

1: for $i = 1$ to $M$ do
2: //"model forward-backward" job
3: for each task in the Spark job do
4: read the latest weights;
5: get a random batch of data from local Sample partition;
6: compute local gradients (forward-backward on local model replica);
7: end for
8: //"parameter synchronization" job
9: aggregate (sum) all the gradients;
10: update the weights per specified optimization method;
11: end for
• A single "model forward-backward" Spark job can simply apply the functional zip operator to the co-located partitions of the two RDDs and compute the local gradients in parallel for each model.
Parameter Synchronization Algorithm

**Algorithm 2** "Parameter synchronization" job

1: for each task $n$ in the "parameter synchronization" job do
2:   shuffle the $n^{th}$ partition of all gradients to this task;
3:   aggregate (sum) these gradients;
4:   updates the $n^{th}$ partition of the weights;
5:   broadcast the $n^{th}$ partition of the updated weights;
6: end for
Parameter Synchronization in BigDL

- Each local gradient (computed by a task in the "model forward-backward" job) is evenly divided into N partitions
- Each task n in the "parameter synchronization" job aggregates these local gradients and updates the weights for the nth partition.
Comparison between existing deep learning frameworks and BigDL

<table>
<thead>
<tr>
<th>Existing deep learning frameworks</th>
<th>BigDL</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Multiple long-running, potentially stateful tasks</td>
<td>• Run a series of short-lived Spark jobs</td>
</tr>
<tr>
<td>• Interact with each other (in a blocking fashion for synchronization)</td>
<td>• Each task in the job is stateless and non-blocking</td>
</tr>
<tr>
<td>• Require fine-grained data access and in place data mutation</td>
<td>• Automatically adapt to the dynamic resource changes</td>
</tr>
<tr>
<td>• Not directly supported by existing big data systems</td>
<td>• Built on top of existing primitives in Spark (e.g., shuffle, broadcast, and in memory data persistence)</td>
</tr>
</tbody>
</table>
Evaluation of Computing Performance

- NCF training on single node:
  - PyTorch 0.4 on Nvidia P100 GPU
  - BigDL 0.7.0 and Spark 2.1.0 on a dual-socket Intel Skylake 8180 server (56 cores and 384GB)

- The training performance of NCF using the BigDL implementation is 1.6x faster than the reference PyTorch implementation, as reported by MLPerf[1].

Evaluation of scalability of distributed training

- Throughput of ImageNet Inception v1 training (w/ BigDL 0.3.0 and dual-socket Intel Broadwell 2.1 GHz)

- The throughput scales almost linear up to 128 nodes (and continue to scale reasonably up to 256 nodes).
Efficiency of task scheduling

• The overhead of launching tasks grows to over 10% when there are close to 500 tasks per iteration

• Group scheduling introduced by Drizzle can help schedule multiple iterations of computations at once

• Greatly reduce scheduling overheads even if there are many tasks in each iteration
Application of BigDL - Image feature extraction

- Problem with previous “connector approach” (similar to CaffeOnSpark)
- Very complex and error-prone in managing large-scale distributed systems
- Impedance mismatch in the parallelism for data processing and for model compute
Application of BigDL - Image feature extraction

- Implement the entire data analysis and deep learning pipeline under a unified programming paradigm on Spark
- Greatly improves the efficiency of development and deployment
- Efficiently scale out on Spark with superior performance
Application of BigDL - Precipitation nowcasting using Seq2Seq models

- Read over a terabyte of raw radar scan data into Spark and convert it into an RDD of NumPy ndarrays.
- Train a sequence-to-sequence model, using a sequence of images leading up to the current time as the input, and a sequence of predicted images in the future as the output.
- After the model is trained, it can be used to predict the precipitation of the next hour
Conclusions

• BigDL allows users to build deep learning applications for big data using a single unified data pipeline.

• The entire pipeline of BigDL can directly run on top of existing big data systems in a distributed fashion.

• Provide efficient and scalable distributed training directly on top of the functional compute model (with copy-on-write and coarse-grained operations) of Spark.
Thank You!