BigDL: A Distributed Deep Learning Framework for Big Data

Jason (Jinquan) Dai¹, Yiheng Wang² †, Xin Qiu¹, Ding Ding¹, Yao Zhang³ †, Yanzhang Wang¹, Xianyan Jia⁴ †, Cherry (Li) Zhang¹, Yan Wan⁴ †, Zhichao Li¹, Jiao Wang¹, Shengsheng Huang¹, Zhongyuan Wu¹, Yang Wang¹, Yuhao Yang¹, Bowen She¹, Dongjie Shi¹, Qi Lu¹, Kai Huang¹, Guoqiong Song¹

¹Intel, ²Tencent, ³Sequoia Capital, ⁴Alibaba, †Work was done when the author worked at Intel
Agenda

• Motivation
• BigDL Execution Model
• Experimental Evaluation
• Real-World Applications
• Future Work
Real-World ML/DL Systems Are Complex Big Data Analytics Pipelines

Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

Big Data Analysis Challenges

Real-World data analytics and deep learning pipelines are challenging

- 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
BigDL Execution Model
Distributed Training in BigDL

Data Parallel, Synchronous Mini-Batch SGD

Prepare training data as an RDD of Samples
Construct an RDD of models (each being a replica of the original model)

for (i <- 1 to N) {
   //“model forward-backward” job
   for each task in the Spark job:
      read the latest weights
      get a random batch of data from local Sample partition
      compute errors (forward on local model replica)
      compute gradients (backward on local model replica)

   //“parameter synchronization” job
   aggregate (sum) all the gradients
   update the weights per specified optimization method
}


Data Parallel Training

Task 1

Worker 1
  Partition 1
  Partition 1

Worker 2
  Partition 2
  Partition 2

Worker n
  Partition n
  Partition n

Task 2

Sample RDD

Model RDD

Task n: zip Sample and model RDDs, and compute gradient on co-located Sample and model partitions

“Model Forward-Backward” Job
Parameter Synchronization

“Parameter Synchronization” Job

1. **Task 1**
   - Local gradient: 1
   - Weight 1: 1
   - Update: 

2. **Task 2**
   - Local gradient: 2
   - Weight 2: 2
   - Update: 

3. **Task n**
   - Local gradient: n
   - Weight n: n
   - Update: 

\[
\sum \text{gradient 1, gradient 2, \ldots, gradient n} \\
\sum \text{weight 1, weight 2, \ldots, weight n} \\
\text{update}
\]
For each task $n$ in the "parameter synchronization" job {
    \textbf{shuffle} the $n^{th}$ partition of all gradients to this task \\
    aggregate (sum) the gradients \\
    updates the $n^{th}$ partition of the weights \\
    \textbf{broadcast} the $n^{th}$ partition of the updated weights \\}

"Parameter Synchronization" Job
\textit{(managing $n^{th}$ partition of the parameters - similar to a parameter server)}

\textbf{AllReduce Operation (directly on top of primitives in Spark)}
\begin{itemize}
    \item Gradient aggregation: \textit{shuffle}
    \item Weight sync: \textit{task-side broadcast}
    \item In-memory persistence
\end{itemize}
### Difference vs. Classical AllReduce

#### Classical AllReduce architecture
- Multiple long-running, potentially stateful tasks
- Interact with each other (in a blocking fashion for synchronization)
- Require fine-grained data access and in-place data mutation
- Not directly supported by existing big data systems

#### BigDL implementation
- Run a series of short-lived Spark jobs (e.g., two jobs per mini-batch)
- Each task in the job is stateless and non-blocking
- Automatically adapt to the dynamic resource changes (e.g., preemption, failures, resource sharing, etc.)
- Built on top of existing primitives in Spark (e.g., shuffle, broadcast, and in-memory data persistence)
Experimental Evaluation
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.
MLPerf 0.5 training results URL: https://mlperf.org/training-results-0-5

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.
Training Scalability

*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).*

Real-World Applications
Object Detection and Image Feature Extraction at JD.com

Problem with previous “connector approach” (similar to CaffeOnSpark)
• Very complex and error-prone in managing large-scale distributed systems
• Impedance mismatch
  • Mismatch in the parallelism for data processing and for model compute
Object Detection and Image Feature Extraction at JD.com

- 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 (3.83x speed-up vs. GPU servers) as benchmarked by JD

[Source](https://software.intel.com/en-us/articles/building-large-scale-image-feature-extraction-with-bigdl-at-jdcom)
And Many More

TECHNOLOGY

bluedata
cloudera
CRAY
the supercomputer company
databricks
DELL EMC
GIGASPACES
innovate with confidence
Lightbend
Qu bole

CLOUD SERVICE PROVIDERS

Alibaba Cloud
aliyun.com

aws

Cloud Dataproc

Azure

IBM Cloud

Telefonica

KINGSOFT

END USERS

cdhi

CDH

CHINA TELECOM

JD.COM

Midea

UnionPay

*Other names and brands may be claimed as the property of others.

http://software.intel.com/bigdl/build

Not a full list

SOCC 2019
Future Work
### Analytics Zoo: Unified Data Analytics + AI Platform

**Distributed TensorFlow, Keras, PyTorch and BigDL on Apache Spark**

<table>
<thead>
<tr>
<th>Use case</th>
<th>Recommendation</th>
<th>Anomaly Detection</th>
<th>Text Classification</th>
<th>Text Matching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>Image Classification</td>
<td>Object Detection</td>
<td>Seq2Seq</td>
<td>Transformer</td>
</tr>
<tr>
<td>Feature Engineering</td>
<td>image</td>
<td>3D image</td>
<td>text</td>
<td>Time series</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>High Level Pipelines</th>
<th>tfpark: Distributed TF on Spark</th>
<th>Distributed Keras w/ autograd on Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>pipelines for Deep Learning</td>
<td>nnframes: Spark Dataframes &amp; ML</td>
<td>Distributed Model Serving (batch, streaming &amp; online)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Backend/Library</th>
<th>TensorFlow</th>
<th>Keras</th>
<th>PyTorch</th>
<th>BigDL</th>
<th>NLP Architect</th>
<th>Apache Spark</th>
<th>Apache Flink</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ray</td>
<td>MKLDNN</td>
<td>OpenVINO</td>
<td>NLP Architect</td>
<td>Apache Spark</td>
<td>Apache Flink</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Intel® Optane™ DCPMM</td>
<td>DL Boost (VNNI)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[https://github.com/intel-analytics/analytics-zoo](https://github.com/intel-analytics/analytics-zoo)
AI on Apache Spark

BigDL

Distributed, High-Performance
Deep Learning Framework
for Apache Spark*

https://github.com/intel-analytics/bigdl

Analytics + AI Platform
Distributed TensorFlow*, Keras*,
PyTorch* and BigDL on Apache Spark*

https://github.com/intel-analytics/analytics-zoo

Accelerating Data Analytics + AI Solutions At Scale

*Other names and brands may be claimed as the property of others.
Appendix
Chasm b/w Deep Learning and Big Data Communities

The Chasm

Deep learning experts

Average users (big data users, data scientists, analysts, etc.)
Apache Spark

Low Latency, Distributed Data Processing Framework

• A Spark cluster consists of a single driver node and multiple worker nodes

• A Spark job contains many Spark tasks, each working on a data partition

• Driver is responsible for scheduling and dispatching the tasks to workers, which runs the actual Spark tasks

https://spark.apache.org
Training Scalability

Overheads of parameter synchronization (as a fraction of average model computation time) of ImageNet Inception-v1 training in BigDL


For more complete information about performance and benchmark results, visit www.intel.com/benchmarks.
Reducing Scheduling Overheads Using Drizzle

Scaling to even larger (>500) workers

• Iterative model training
  • Same operations run repeatedly

• Drizzle
  • A low latency execution engine for Spark
  • Group scheduling for multiple iterations of computations at once

Precipitation nowcasting using sequence-to-sequence models in Cray

- Running data processing on a Spark cluster, and deep learning training on GPU cluster not only brings high data movement overheads, but hurts the development productivity due to the fragmented workflow.
- Using a single unified data analysis and deep learning pipeline on Spark and BigDL improves the efficiency of development and deployment.
Real-time streaming speech classification in GigaSpaces

The end-to-end workflow of real-time streaming speech classification on Kafka, Spark Streaming and BigDL in GigaSpaces.

- BigDL allows neural network models to be directly applied in standard distributed streaming architecture for Big Data (using Apache Kafka and Spark Streaming), and efficiently scales out to a large number of nodes in a transparent fashion.

LEGAL DISCLAIMERS

• Intel technologies’ features and benefits depend on system configuration and may require enabled hardware, software or service activation. Learn more at intel.com, or from the OEM or retailer.

• No computer system can be absolutely secure.

• Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. For more complete information about performance and benchmark results, visit http://www.intel.com/performance.

Intel, the Intel logo, Xeon, Xeon phi, Lake Crest, etc. are trademarks of Intel Corporation in the U.S. and/or other countries.

*Other names and brands may be claimed as the property of others.

© 2019 Intel Corporation