ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

CO3067 - Parallel Computing

Assignment Report

Performance evaluation of BigDL in big data analysis problems

---

Lecturer Thoại Nam
Student Phan Khánh Thịnh - 1814182
Võ Công Thành - 1814038
Nguyễn Phi Long - 1812915

Hồ Chí Minh, 2021

Contents

1 Introduction

1.1 Apache Spark

1.1.1 What is Apache Spark?

Apache Spark is an open-source, distributed processing system used for big data workloads. It utilizes in-memory caching, and optimized query execution for fast analytic queries against data of any size.

1.1.2 Features of Apache Spark

Speed: Run workloads 1000x faster than Hadoop

Apache Spark achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.

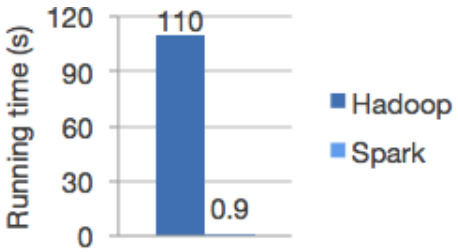

Figure 1: Regression in Hadoop and Spark

Ease of Use: Write applications quickly in Java, Scala, Python, R, and SQL.
Spark offers over 80 high-level operators that make it easy to build parallel apps. And you can use it interactively from the Scala, Python, R, and SQL shells.

```
df = spark.read.json("logs.json")
df.where("age > 21")
  .select("name.first").show()
```

Spark's Python DataFrame API
Read JSON files with automatic schema inference

Generality: Combine SQL, streaming, and complex analytics.
Spark powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.

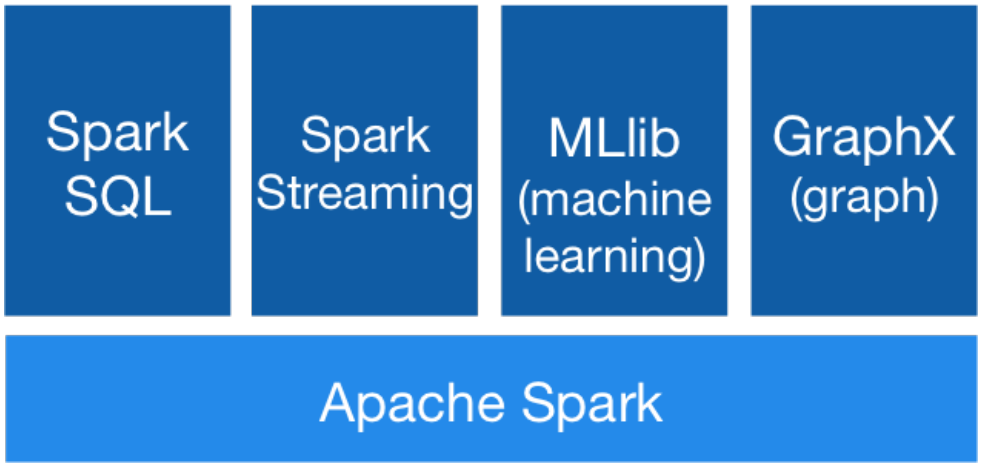

Figure 2: Apache Spark Overview

Runs everywhere: Spark runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud. It can access diverse data sources.
You can run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, on Mesos, or on Kubernetes. Access data in HDFS, Alluxio, Apache Cassandra, Apache HBase, Apache Hive, and hundreds of other data sources.

Figure 3: Popular Application run on Spark

1.2 BigDL

1.2.1 What's BigDL?

BigDL is a distributed deep learning library for Apache Spark; with BigDL, users can write their deep learning applications as standard Spark programs, which can directly run on top of existing Spark or Hadoop clusters.

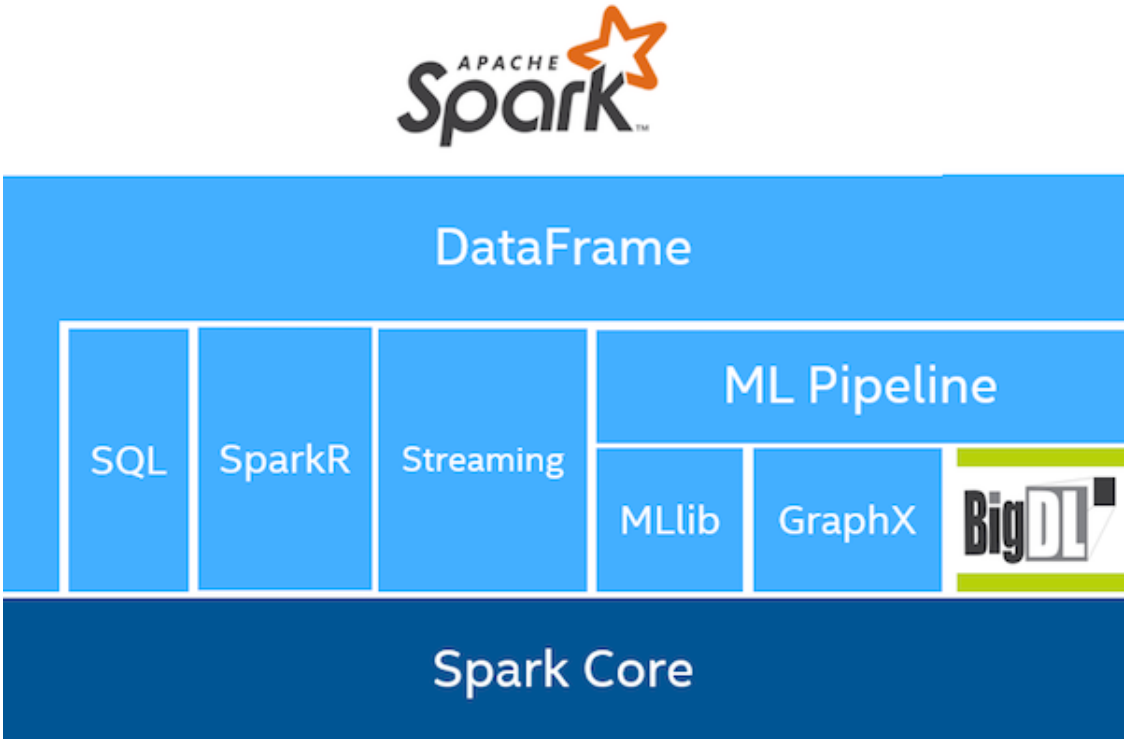

Figure 4: Bigdl architecture running on spark

- Rich deep learning support. Modeled after Torch, BigDL provides comprehensive support for deep learning, including numeric computing (via Tensor) and high level neural networks; in addition, users can load pre-trained Caffe or Torch models into Spark programs using BigDL.
- Extremely high performance. To achieve high performance, BigDL uses Intel oneMKL, oneDNN and multi-threaded programming in each Spark task. Consequently, it is orders of magnitude faster than out-of-box open source Caffe or Torch on a single-node Xeon (i.e., comparable with mainstream GPU).
- Efficiently scale-out. BigDL can efficiently scale out to perform data analytics at "Big Data scale", by leveraging Apache Spark (a lightning fast distributed data processing framework), as well as efficient implementations of synchronous SGD and all-reduce communications on Spark.

1.2.2 Why BigDL?

You may want to write your deep learning programs using BigDL if:

- You want to analyze a large amount of data on the same Big Data (Hadoop/Spark) cluster where the data are stored (in, say, HDFS, HBase, Hive, Parquet, etc.).
- You want to add deep learning functionalities (either training or prediction) to your Big Data (Spark) programs and/or workflow.
- You want to leverage existing Hadoop/Spark clusters to run your deep learning applications, which can be then dynamically shared with other workloads (e.g., ETL, data warehouse, feature engineering, classical machine learning, graph analytics, etc.)

1.2.3 Utility of BigDL

BigDL helps us in balancing our needs:

- Big Compute: Fast Linear Algebra, Intel MKL library
- Big Data: I/O parallelized to run on many CPUs

BigDL allows Massive Scalability

- Runs on Spark
- Works with Hadoop eco system (via Spark)
- Hadoop is The Big Data platform for on-premise deployments

Plays nicely with other BigDL frameworks

- Use existing Tensorflow or Caffe at scale in BigDL
- Train new models based on existing TF/Caffe models

2 Implementation

2.1 Problem introduction

The MNIST handwritten digit classification problem is a standard dataset used in computer vision and deep learning. The MNIST database, which stands for the Modified National Institute of Standards and Technology database, is a very large dataset containing several thousands of handwritten digits.

This dataset was created by mixing different sets inside the original National Institute of Standards and Technology (NIST) sets, so as to have a training set containing several types and shapes of handwritten digits.

2.1.1 Dataset Description

It is a dataset of 60,000 small square 28×28 pixel grayscale images of handwritten single digits between 0 and 9. The MNIST data is split into three parts: 60,000 data points of training data, 10,000 points of test data.

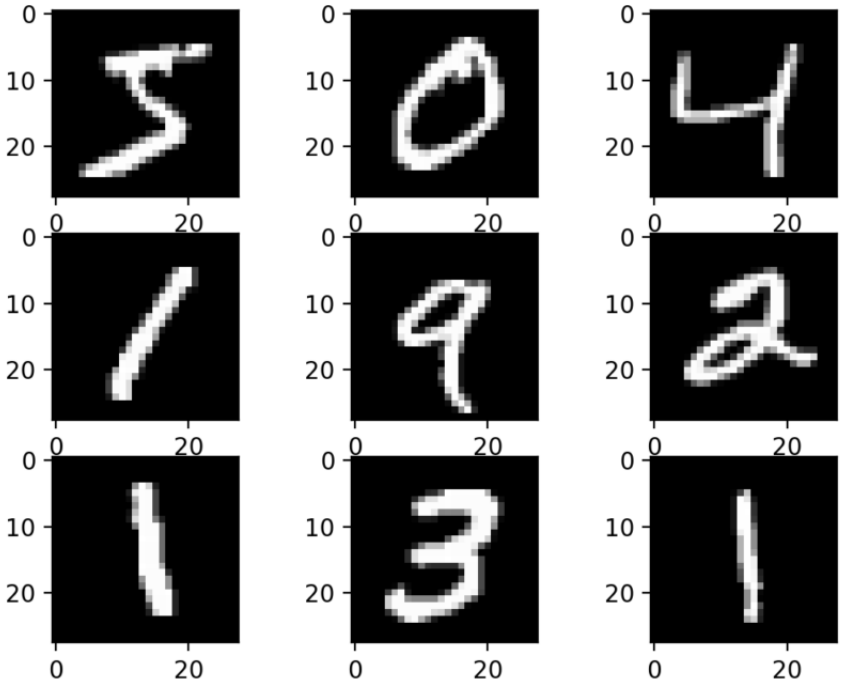

Figure 5: Illustration of the shape of numbers

2.1.2 Problem Goal

The task is to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9, inclusively.

The MNIST dataset has been the target of so many research done in recognizing handwritten digits. This allowed the development and improvements of many different algorithms with a very high performance, such as machine learning classifiers, Deep learning using Big Data.

2.1.3 Dataset as an RDD

BigDL provides 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 (e.g., RDD [10], Dataframe [11], Spark SQL, ML pipeline, etc.).

Resilient Distributed Datasets (RDD) is an underlying data structure of Spark. It is a distributed immutable collection of an object. Each dataset in an RDD is divided into several logical regions. Can be computed on different nodes of a cluster.

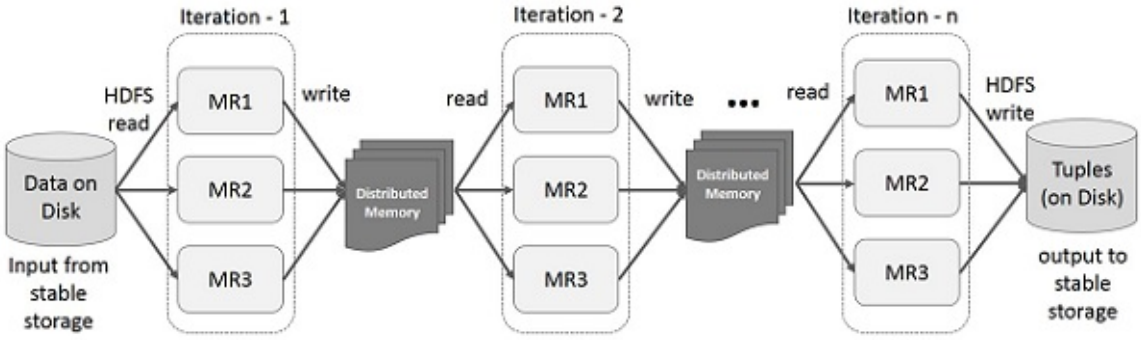

Figure 6: Interactive Operations on Spark RDD

2.2 BigDL Execution

2.2.1 Spark exucution model

Similar to other Big Data systems (such as MapReduce [28] and Dryad [29]), a Spark cluster consists of a single driver node and multiple worker nodes, as shown in Figure 2. The driver is responsible for coordinating tasks in a Spark job (e.g., task scheduling and dispatching), while the workers are responsible for the actual computation. To automatically parallelize the data processing across the cluster in a fault-tolerant fashion, Spark provides a data-parallel, functional compute model.

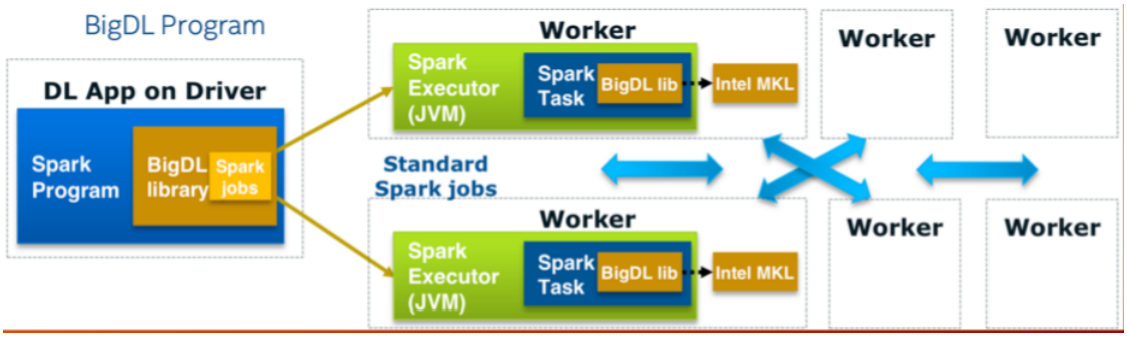

Figure 7: BigDL Program Proccess

2.2.2 BigDL Training Model

2.2.2.a Model Creation and Parameter

With this BigDL Problem, We use LSTM model for MNIST digit classification problem.

Model Parameter:

- batchsize = 64
- ninput = 28 MNIST data input (img shape: 28*28)
- nhidden = 128 hidden layer num of features
- nclasses = 10 MNIST total classes (0-9 digits)

2.2.2.b BigDL Parameter run on Spark

Usage: spark-submit-with-bigdl.sh + [options] + file.py

Options:

- –master MASTER URL: spark, yarn, k8s, local.
- local[k]: Run Spark locally with k worker threads as logical cores on your machine.
- File.py: File for executing program.

2.2.3 System configuration

Program run on system includes:

- System/Host Processor: Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz
- CPU(s): 48
- Core(s) per socket: 12
- Socket(s): 2
- Memory: 183 G (free)

3 BigDL performance evaluation of system

3.1 Execution running time

Image Size	1 cores	2 cores	4 cores	8 cores	16 cores	32 cores
1000	6.11	4.87	4.37	**4.28**	4.54	4.94
5000	22.97	16.22	13.50	**12.27**	12.55	13.69
10000	42.30	28.86	23.18	21.19	**20.73**	22.68
30000	120.62	79.16	62.29	**54.37**	54.71	59.41
60000	243.05	157.56	119.71	**103.45**	106.21	108.63

Figure 8: Table show Runtime of cores on each relative image size. Bold's the best

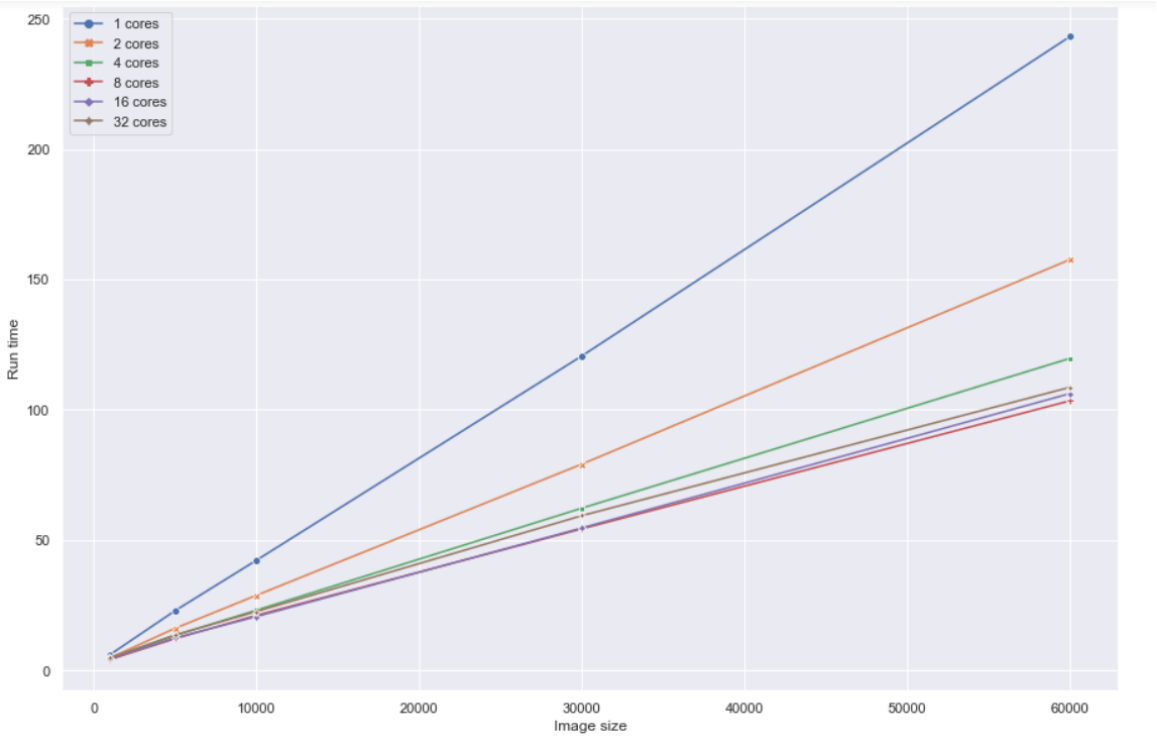

Figure 9: Line graph show RUNTIME of cores on each relative image size

3.2 Computation Evaluation(SPEED UP)

With the BigDL problem running on a parallel system, when we run one processor (cores), it shows that the running time is the longest compared to running many cores. Therefore, using multiple cores on a parallel system will show that the running time can be shortened accordingly.

However, for each type of corresponding problem, we use the appropriate number of processor to achieve the most optimal speed up.

When we use multiple cores with parallel system, starting from one core shows an improvement in performance and continuing to increase until between 8 and 16 cores will be achieved the best speed up. Because, depending on the size of the problem, we will choose the optimal number of cores. With this problem, when running with an image size of 10000 rows, 16 processors achieve the best speed up. The rest, 8 processors will be more optimal.

When achieving the optimal speed up with this problem 8-16 processors, if we increase the processor to 32 as the graph. Speed up will decrease because of:

- Software overhead processing time required by system software too much.
- Communication overhead can dramatically affect the performance of parallel computations that repeatedly access remote data can easily spend most of their time communicating rather than performing useful computation.

Image Size	1 cores	2 cores	4 cores	8 cores	16 cores	32 cores
1000	1	1.25	1.40	**1.43**	1.35	1.24
5000	1	1.42	1.70	**1.87**	1.83	1.68
10000	1	1.47	1.82	2.00	**2.04**	1.87
30000	1	1.52	1.94	**2.22**	2.20	2.03
60000	1	1.54	2.03	**2.35**	2.29	2.24
Average SPEED UP	**1**	**1.44**	**1.78**	**1.97**	**1.94**	**1.81**

Figure 10: Line graph show RUNTIME of cores on each relative image size

- Load balancing set of tasks over a set of resources.

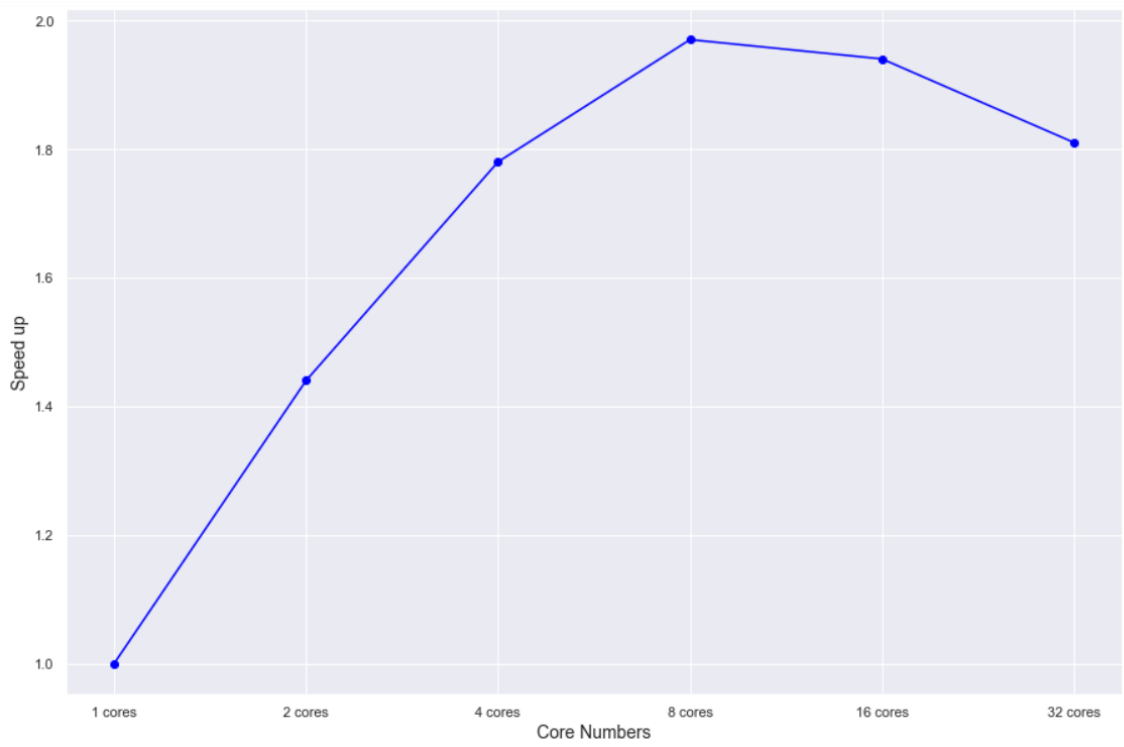

Figure 11: Line Graph show SPEED UP of Image Training of cores

4 Conclusions and future work

We have described BigDL, including its distributed execution model,data integration computation performance. It allows users to build deep learning applications for big data using a single unified data pipeline on Spark.

Unlike existing deep learning frameworks, it provides efficient and scalable distributed training directly on top of the functional compute model of Spark. BigDL is a work in progress.

Continue to improve the BigDL model as well as dig deeper into running on spark for better performance.

5 Task assignment

Member	Task	Completion level
Võ Công Thành	Learn theory and run experiments	100%
Phan Khánh Thịnh	Learn theory and run experiments	100%
Nguyễn Phi Long		0%

References

[1] BigDL: https://github.com/intel-analytics/BigDL

[2] Spark: https://spark.apache.org

[3] Spark and Hadoop: Spark and Hadoop

[4] Apache Spark RD:https://laptrinh.vn/books/apache-spark/page/apache-spark-rdd

[5] Deep Learning on Apache Spark* Cluster: https://www.intel.com/content/www/us/en/developer/articles/technical/bigdl-scale-out-deep-learning-on-apache-spark-cluster.html

[6] Intel's Digital Marketing Technology and Capabilities: https://conferences.oreilly.com/artificial-intelligence/ai-ny-2018/cdn.oreillystatic.com/en/assets/1/event/280/Build%20deep%20learning-powered%20big%20data%20solutions%20with%20BigDL%20Presentation.pdf

[7] MNIST Dataset: http://yann.lecun.com/exdb/mnist/