BIG DATA ANALYTICS USING APACHE SPARK

IEEE IGARSS 2021 Tutorial on Scalable Machine Learning with High Performance and Cloud Computing

DR. SHAHBAZ MEMON
HIGH PRODUCTIVITY DATA PROCESSING RESEARCH GROUP
JÜLICH SUPERCOMPUTING CENTRE
STRUCTURE

• Introduction
• Apache Spark Basics
• Developing on Spark
• Machine Learning on Spark
• Conclusions
INTRODUCTION
HPC AND HTC

Introduction

- High Performance Computing (HPC) is based on computing resources that enable the efficient use of parallel computing techniques through specific support with dedicated hardware such as high performance CPU/core interconnections. These are compute-oriented systems.

- High Throughput Computing (HTC) is based on commonly available computing resources such as commodity PCs and small clusters that enable the execution of ‘farming jobs’ without providing a high performance interconnection between the CPU/cores. These are data-oriented systems.

Riedel [14]
END-TO-END IMAGE ANALYTICS

Motivational Scenario

Preprocessing
- Image denoise, segmentation
  - Embarrassingly Parallel (HTC)

Segmentation
- Extraction of interesting features
  - Embarrassingly Parallel (HTC)

Model (re)training
- DNN
  - Massively Parallel (HPC)

Prediction
- Classification output
  - Embarrassingly Parallel (HTC)

Generated Model

Raw data

Processed data
QUESTIONS TO PONDER

• Pre- and post-processing in the scope of application, rather than on one step

• Manage all workflow tasks within one framework e.g. end-to-end Deep Learning

• Data export and import from multiple kind of storage systems

• Data-intensive rather than compute-intensive processing
BIG DATA ANALYTICS

- Support of multiple algorithms and frameworks
  - Machine Learning and Deep Learning
  - Integrated processing with HTC, HPC and ML/DL frameworks
- Abstract parallelization complexity from user
  - Parallel processing, batch systems, environmental intricacies are abstracted
- Encapsulate distributed computing and storage infrastructure details
  - Operating systems, security, networks and security interfaces
BIG DATA USE CASES

- Web mining and search (e.g. Page Rank and Ad Analytics)
- Stream analytics (Twitter, Facebook and Trading)
- Graph processing
- IoT (Remote sensing, Automotive and Smart devices)
- Large scale image and video processing
- Time-series analysis
BIG DATA ANALYTIC FRAMEWORKS

• **Motto: Bring compute to data**
• Batch Processing
  • Manage job requests as batches
  • Map-reduce framework: Large problem space to many small tasks
  • E.g. Apache Hadoop
• In-Memory processing
  • Data processing in memory
  • Efficient map-reduce, filter and transform, Extract Transform and Load (ETL)
  • E.g. Apache Spark (Focus of this talk) and Apache Flink
IN-MEMORY: MORE I/O EFFICIENCY

Batch-Compute

In-Memory

MapOp1 → Reduce → Map → Reduce

HDFS / GFS / NoSQL

Operation 1 → Data Cached → Operation 2

HDFS / GFS / NoSQL
APACHE SPARK

In-Memory Computing Framework

- Open source, unified analytics [2] engine for distributed and parallel data processing
  - Data transformations + AI and ML

- Provides a set of extensible APIs for
  - SQL for interactive queries
  - Machine learning
  - Stream processing
  - Graph processing

Logistic regression in Hadoop and Spark

spark.apache.org [2]
ECOSYSTEM

• Apache Project and open source
• Databricks: main development driver
• Supported by major cloud computing providers
  • E.g. Amazon and Google
• Integrated with multiple schedulers, file systems, DBMS and data stores, a.k.a Connectors
• Hadoop (Batch processing) supported
  • Seamlessly complement / replace Map-Reduce layer
• HPC Supported
  • SLURM as a scheduler
APACHE SPARK COMPONENTS

Spark SQL and DataFrames + Datasets
Spark Streaming (Structured Streaming)
Machine Learning MLlib
Graph Processing Graph X

Spark Core and Spark SQL Engine
Scala  SQL  Python  Java  R

Learning Spark [2]
APACHE SPARK COMPONENTS

- Spark SQL (Dataframes):
  - Standard SQL and Hive QL, but parallel execution
  - Data sources: JSON, Images, Parquet, Binary and Hive tables

- Spark Streaming (Structured Streaming)
  - High throughput and low latency scenarios
  - Log inflow, sensor data (IOT), Twitter streams

- Machine Learning (ML Lib)
  - Clustering, classification and recommender systems
  - Support Deep learning frameworks – Keras, Tensorflow and Pytorch

- Graph Processing (GraphX)
  - Iterative and parallel Graph computations
  - Social computing, semantic networks and link data
SPARK APPLICATION AREAS

Commercial and Scientific Applications

• Data processing pipeline (load, preparation and transform) include machine learning (Alibaba [7])

• Netflix recommendation ML pipeline (Netflix [8])

• Remote sensing and image analytics [5] and [6]
ARCHITECTURE

Application Concepts

- Jobs: A parallel computation
- Stages: A job is divided into stages
- Tasks: A single unit of execution

Learning Spark [9]
RESILIENT DISTRIBUTED DATASETS (RDD)

Data Structure Representation

• Basic programming abstraction (not used by every user level)
  • Dependencies: DAG structure of tasks
  • Partitions: Data locality and parallel computation on partitions
  • Iterator [T]: Handle to multiple type of collections
• Less expressive and complex
  • Computations are opaque
  • Difficult to introspect and debug
• New releases (> 3.0) prefer Spark DataFrames
SPARK DATAFRAMES

Structured computing API – High level wrappers to RDDs

- Structured and the format is inspired by pandas DataFrames
- Distributed in-memory tables
  - Rows and columns, data types and schemas
  - E.g. integer, string, array and map
  - Simple and complex data types
- Scala (main implementation), Python, Java and R bindings
- Supports many formats as external data sources
  - Parquet, JSON, CSV, Images and Binary, etc
TRANSFORMATIONS AND ACTIONS

Operation Types

Transformations

• Transform Spark DataFrame into a new DataFrame without modifying the original data
• E.g. select, filter, groupBy, orderBy, join
• Lazy evaluation: not computed until action is called or any read / write occurs

Actions

• Compute operations
• E.g. show, take, count, collect, save

Narrow and Wide Transformations

• Wide transformations use multiple partitions

# In Python
```python
>>> strings = spark.read.text("../README.md")
>>> filtered = strings.filter(strings.value.contains("Spark"))
>>> filtered.count()
26
```
NARROW AND WIDE TRANSFORMATIONS

Transformation Types

Narrow

Input → Output
(e.g. filter or contains)

Wide

Input → Output
(e.g. groupBy or contains)
SPARK EXECUTION WORKFLOW

1. Create or load data
   - e.g. load data set in DataFrames or RDDs

2. Apply 1-n transformations data (narrow or wide)
   - e.g. select, filter, groupBy, orderBy

3. Perform actions that return or store data
   - E.g. reduce; count; collect
DEVELOPING ON SPARK
APACHE SPARK DEPLOYMENTS ON CLOUDS

- **Private Clouds**: in-premise deployments
  - E.g. OpenStack or Apache CloudStack or Containers

- **Public Clouds**: external provider, Pay-per use and Elastic scaling
  - Amazon EMR (Elastic Map-Reduce)
  - Microsoft Azure (Databricks or HDInsight)
  - Google (DataProc)
**JUPYTER-DOCKER STACK**

User Development Environment

---

**Host Machine**

- PySpark
- Scipy
- Scikit-learn
- Pandas
- matplotlib
- Ubuntu-base

**Container**

- User Code
- IGARSS-Notebooks

---

Image of Jupyter and IGARSS-Notebooks mount.
PYSPARK

Embedded Development Environment

- Python bindings for Apache Spark (implemented in Scala)
- Mostly every functionality is available in Python
- Easily developed on Jupyter-lab instances

PySpark Image Specifics
- PySpark v 3.1.2
  - Includes core Spark libraries
- PyArrow (for interoperability between Pandas and Spark Dataframes)
- JAVA (OpenJDK) 11 and Scala 2.12.10
LAUNCHING THE PYSPARK CONTAINER

Based on Jupyter-Docker Stack

• Enter the following URL:
  • https://labs.play-with-docker.com/
  • Press “Start”
• Login using your existing credentials or Sign up for a new Docker account (It is free)
• Account created
STARTING AN IMAGE

Hit “Start”

Click “Add New Instance”

Terminal is started
PREPARE THE WORKING DIRECTORY

• On the Terminal, write the following commands (step-wise)

```bash
$> mkdir igarss-nb (Press Enter)

$> chmod 777 igarss-nb (Press Enter)

$> cd igarss-nb (Press Enter)
```
$> docker run -p 8888:8888 -p 4040:4040 -p 4041:4041 -p 4042:4042 \
-v ~/igarss-nb:/home/jovyan/work jupyter/pyspark-notebook \
start-notebook.sh --NotebookApp.token='ig2021'

Server started.
Click 8888

New tab opens and there enter ig2021 and press Log in
OPEN THE JUPYTER TERMINAL
On the shell type:
$ cd work
$ wget https://fz-juelich.sciebo.de/s/xiNXrpfOfrmqLmMX/download

Untar the archive: $tar –xvf download

Switch to the previous browser tab and click IGARSS2021
Demonstration: Open “BasicDataFrame.ipynb”
USER DEFINED FUNCTIONS (UDF)

Explicit Customization

• Types: Simple and Pandas UDF
• Define new domain specific modules that extend the vocabulary of Spark’s built-in functions
• Useful for data normalization and cleaning (e.g. handling nulls, feature scaling)
• **Simple UDF**: Row-wise operation on a data frame – sequential processing, see example notebook *(next slide)*
• **Pandas UDF**: Vectorized operations (process entire array at once)
Demonstration: Open “SparkUDFExample.ipynb”
MACHINE LEARNING WITH SPARK

Spark SQL and DataFrames + Datasets
Spark Streaming (Structured Streaming)
Machine Learning MLlib
Graph Processing Graph X

Spark Core and Spark SQL Engine
Scala
SQL
Python
Java
R

Learning Spark [2]
SPARKML (MLLIB)

• Promises Machine Learning at Scale
• Parallel processing made easy
  • Develop locally (e.g. Jupyter Notebook) -> deploy on cluster
• MLLib features
  • Distributed with ML algorithms (clustering, classification..)
  • Parallel implementations
  • Processing data is cached in-memory (optimal for iterative algorithms)
  • Support of Python, Scala, Java, R
SPARK ML CONCEPTS

- **Transformer**: Data preparation and rule-based transformations. Input DataFrame and output a new DataFrame instance.

  ```python
  newDF = myDF.transform()
  ```

- **Estimators**: Learning or fitting parameters. Returns a Model (a transformer)

  ```python
  svmModel = svm.fit(newDF)
  ```

- **Pipeline**: A kind of estimator that orchestrates a series of transformers and estimators into a single model

  ```python
  pipeline = Pipeline(stages=[vec, svm])  # combine vectorization and classifier
  pipelineModel = pipeline.fit(trainData)  # model training
  preds = pipelineModel.transform(testData)  # model evaluation
  ```
SPARK ML IMPLEMENTATION

Taxonomy

Big Data Analytics Using Spark MLlib

Descriptive Analytics
- Statistics Summary
- Correlations
- Stratified Sampling
- K-means
- Bisecting K-means

Clustering
- Feature Extractor
  - TF-IDF
- Feature Transformer
  - Vector Slicer
  - Stop Words Remover
- Feature Selector
  - ChiSq Selector
- Binary Classification
  - Linear SVM
  - Decision Trees

Predictive Analytics
- Prediction Algorithms
  - Logistic Regression
  - Random Forests
  - Naive Bayes
  - Isotonic Regression
  - Linear Least Squares

Descriptive Analytical Data
- Internal Enterprise Data
- Event Log Data
- Sensing Data
- Social Media

Predictive Analytical Data
- Images and Videos
- Documents

Spark MLlib

[16]
K-MEANS CLUSTERING

Clustering Example

- Partition-based clustering
- Clusters are associated with respective centroids
- Number of clusters must be known

1: Select $K$ points as the initial centroids.
2: \textbf{repeat}
3: Form $K$ clusters by assigning all points to the closest centroid.
4: Recompute the centroid of each cluster.
5: \textbf{until} The centroids don’t change
K-MEANS: UBER PICKUP DATA

Clustering Example

- Dataset: Uber Pickups in the CSV format
- Problem: Cluster the dense pickup points
- Uber trip data of August 2014
- Attributes: Lat, Lon, Date and Time, Base (TLC)

Example

_Demonstration: Switch to Notebook “KMeansExample.ipynb”_
LOGISTIC REGRESSION

Classification Example

- Supervised machine learning algorithm
- Classification algorithm to deal with categorical response
- Predict binomial outcomes between 0 and 1
- Predictions are generated in the form of probabilities
- Uses Sigmoid function (a.k.a Logistic Function)

Example

Demonstration: Open “LogisticRegressionExample.ipynb”
ONEVSREST CLASSIFICATION

Multi-class classification

- Multiple class labels dataset
- Resolve multi-class as binary-class problem
- Apply N-binary classifiers for N-classes
- Example: The shape is triangle, square or cross

Example

Demonstration: Open “OneVsRestExample.ipynb”
**ML PIPELINE – LINEAR REGRESSION EXAMPLE**

**Refresher**

**Pipeline** is a kind of estimator that orchestrates a series of transformers and estimators into a single model.
Demonstration: Open Notebook “PipelineExample.ipynb”
DOWNLOAD THE NOTEBOOKS TO THE LOCAL FILESYSTEM

- Open the Jupyter Terminal
- CD to the /home/jovyuan/work directory ($> cd ~/work)
- Create a Tar archive ($> tar –czvf igarss21.tar.gz IGARSS2021/)
- Switch to the file browser view and download the created (igarss21.tar.gz) archive
Introduction

- Combine ETL/ELT, model training and hyper-parameter tuning in one workflow
- Train TensorFlow / PyTorch models integrated with the Spark ecosystem
DISTRIBUTED DEEP LEARNING

Options

Distributed Inference

• Pandas UDF (User Defined Functions) and Apache Arrow

Distributed Training

1) Spark-TensorFlow-Distributor

2) HorovodRunner (only available for Databricks Runtime ML users)
from spark_tensorflow_distributor import MirroredStrategyRunner

# Adapted from https://www.tensorflow.org/tutorials/distribute/multi_worker_with_keras

def train():
    import tensorflow as tf
    import uuid

    BUFFER_SIZE = 10000
    BATCH_SIZE = 64

    def make_datasets():
        (mnist_images, mnist_labels), _ = 
        tf.keras.datasets.mnist.load_data(path=str(uuid.uuid4()) + 'mnist.npz')
        dataset = tf.data.Dataset.from_tensor_slices((
            tf.cast(mnist_images[..., tf.newaxis] / 255.0, tf.float32),
            tf.cast(mnist_labels, tf.int64))
        )
        dataset = dataset.repeat().shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
        return dataset

    def build_and_compile_cnn_model():
        model = tf.keras.Sequential([
            tf.keras.layers.Conv2D(32, 3, activation='relu', input_shape=(28, 28, 1)),
            tf.keras.layers.MaxPooling2D(),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(64, activation='relu'),
            tf.keras.layers.Dense(10, activation='softmax'),
        ])
        model.compile(
            loss=tf.keras.losses.sparse_categorical_crossentropy,
            optimizer=tf.keras.optimizers.SGD(learning_rate=0.001),
            metrics=['accuracy'],
        )
        return model

    train_datasets = make_datasets()
    options = tf.data.Options()
    options.experimental_distribute.auto_shard_policy = tf.data.experimental.AutoShardPolicy.DATA
    train_datasets = train_datasets.with_options(options)
    multi_worker_model = build_and_compile_cnn_model()
    multi_worker_model.fit(x=train_datasets, epochs=3, steps_per_epoch=5)

    MirroredStrategyRunner(num_slots=8).run(train)
HOROVOD RUNNER

Distributed Training

- A generic API to manage distributed DL workloads
- Implemented through Spark’s barrier execution mode scheduling (to support the MPI execution model)

Development workflow
1) Write single node DL code (e.g. TF/Keras)
2) Horovod-ify your code
3) Invoke HorovodRunner <hr.run(hvd_tr,..)>

```
hr = HorovodRunner(np=2)

def train():
    import tensorflow as tf
    hvd.init()

hr.run(train)
```
CONCLUSIONS

• Big data analytics frameworks such as Apache Spark allows end-to-end ML/DL pipelines

• A viable direction for remote sensing and image analysis applications where whole processing workflow runs HPC and HTC simultaneously

• Harness public clouds (e.g. Amazon or Google) that provides stable deployments; integrated with state-of-the-art data analysis and DL frameworks (e.g. TF or PyTorch)
THANKS FOR LISTENING
REFERENCES

[3] Tan, Pan-Ning and Steinbach, Introduction to Data Mining
[10] Databricks documentation: HorovodRunner: distributed deep learning with Horovod
REFERENCES

[10] Databricks documentation: HorovodRunner: distributed deep learning with Horovod