Week3_Intro_Apache_Spark

What is?

Spark is an **open** source **in-memory application** framework for **distributed** data processing and **iterative** analysis on **massive** data volumes.

Spark is written predominantly in **Scala** and runs on **Java virtual machines (JVMs)**

Distributed Computing is a group, or **cluster** of computers working together to appear as one system to the end user. The term distributed computing often used interchangeably with parallel computing as both are similar. However there are some differences.

- Parallel computing processors access shared memory
- Distributed computing processors usually have their own private or distributed memory

Distributed Computing benefits

- Scalability and modular growth;
- Fault tolerance and redundancy;

Spark Benefits

- Supports a computing framework for large scale data processing and analysis
- Provides parallel and distributed processing, scalability and fault tolerance on commodity hardware
- Provides speed due to in-memory processing
- Creates a comprehensive, unified framework to manage big data processing;
- Enables programming flexibility with easy-to-use Python, Scala, and Java APIs

Apache Spark & MapReduce Compared

Traditional approach:

- Create MapReduce jobs for complex jobs, interactive query, and online event-hub processing involves lots of slow disk i/o

Solution:

- Keep more data in-memory with a new distributed execution engine

Spark and Data Engineering

- Core Spark Engine
- Clusters and executors

- Cluster management
- SparkSQL
- Catalyst
- Tungsten DataFrames

Spark and Data Science and Machine Learning

- SparkML
- DataFrames
- Streaming

Functional Programming Basics

- A mathematical "function" programming style
- Follows a declarative programming model
- Emphasizes "what" instead of "how-to"
- Uses "expressions" instead of "statements"
- First implementation was LISt Programming Language (LISP) in the 1950s
- Scala most recent representative (Python, R and Java also provide rudimentary support for functional programming)
- Functions are first class citizens in Scala
- It makes easier to perform parallelization

What are lambda functions

Lambda calculus functions, or operators, are anonymous functions that enable functional programming.

- Scala

```
// an example lambda operator in scala to add 2 numbers

val add = (x:Int, y:Int) => x + y
println(add(1,2))
```

- Python

```
// an example lambda function in Python to add 2 numbers
add = lambda x, y: x + y
print( add(1,2))
```

Lambda functions and Spark

- Spark parallelizes computations using the lambda calculus

- All functional Spark programs are inhently parallelizable

Parallel Programming using Resilient Distributed Datasets

What are RDDs
A resilient distributed dataset is:

- Spark's primary data abstraction
- A fault-tolerant collection of elements
- Partitioned across the nodes of the cluster
- Capable of accepting parellel operations
- Immutable once created

Spark applications consist of a **driver program** that runs the user's main functions and multiple **parallel operations** on a cluster

RDD supported files

- Supported File Types
  - Text
  - SequenceFiles
  - Avro
  - Parquet
  - Hadoop input formats
- Supported file formats
  - Local
  - Cassandra
  - HBase
  - HDFS
  - Amazon s3
  - and others

Creating an RDD in Spark

1. Use an external or local file from Hadoop-supported file system such as : HDFS, Cassandra, Hbase, Amazon s3 or;
2. Use a programmin language through the spark driver or;

```
//Scala
val data = Array(1,2,3,4,5)
val_distData = sc.parallelize(data)
```

```
//Python
data = [1,2,3,4,5]
distData = sc.parallelize(data)
```

3. Apply a transformation on an existing RDD to create a new RDD

What is Parallel Programming

It is the simultaneous use of multiple compute resources to solve a computational problem;
Breaks problems into discrete parts that can be solved concurrently;
Runs simultaneous instructions on multiple processors;
Employs an overall control/coordination mechanism;

RDDs & Parallel Programming

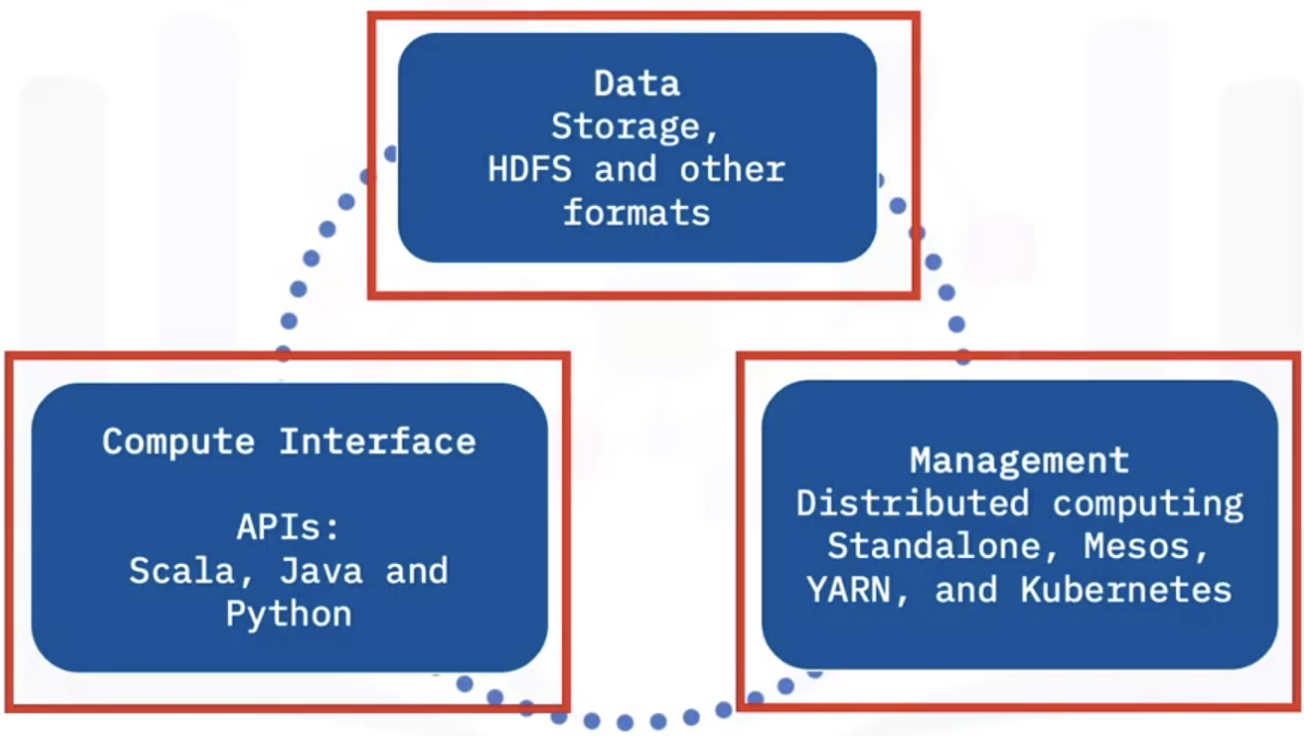
- You can create an RDD by parallelizing an array of objects, or by splitting a dataset into partitions
- Spark runs one task for each partition of the cluster

Resilience and Spark
Resilient Distributed Datasets

- Are always recoverable as they are immutable
- Can persist or cache datasets in memory across operations, which speeds iterative operations

Scale out/ Data Parallelism in Apache Spark

Apache Spark Components



Spark Core

- Is a base engine
- Is fault-tolerant
- Performs large scale parallel and distributed data processing
- Manages memory
- Schedules tasks
- Houses APIs that define RDDs
- Contains a distributed collection of elements that are parallelized across the cluster

Dataframes and SparkSQL

- Is a Spark module for structured data processing
- Used to query structured data inside Spark program, using either SQL or a familiar DataFrame API
- Usabe in Java, Scala, Python and R
- Runs SQL queries over imported data and existing RDDs independently of api or programming language

e.g

```
results = spark.sql(
"SELECT * FROM people")
names = results.map(lambda p: p.name)
```

Spark SQL - Benefits

- Includes a cost-based optimizer, columnar storage, and code generation to make queries fast
- Scales to thousands of nodes and multi-hour queries using the Spark engine, which provides full mid-query fault tolerance
- Provides a programming abstraction called DataFrames and can also act as a distributed SQL query engine

Dataframes

- Distributed collection of data organized into named columns
- Conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations
- Built on top of the RDD API
- Uses RDDs
- Perform relational queries

```
df = spark.read.json("people.json")
df.show()
df.printSchema()
// Register the dataframe as a sql temporary view
df.createTempView("people")
```

Dataframe Benefits

- Ability to scale from kilobytes of data on a single laptop to petabytes on a large cluster;
- Support for a wide array of data formats and storage systems
- State-of-the-art optimization and code generation through the Spark SQL Catalyst optimizer
- Seamless integration with all big data tooling and infrastructure via Spark
- APIs for Python, Java, Scala, and R, which is in development via Spark R

Summary

Spark is an open source in-memory application framework for distributed data processing and iterative analysis on massive data volumes. Both distributed systems and Apache Spark are inherently scalable and fault tolerant. Apache Spark solves the problems encountered with MapReduce by keeping a substantial portion of the data required in-memory, avoiding expensive and time-consuming disk I/O.

Functional programming follows a declarative programming model that emphasizes “what” instead of “how to” and uses expressions.

Lambda functions or operators are anonymous functions that enable functional programming. Spark parallelizes computations using the lambda calculus and all functional Spark programs are inherently parallel.

Resilient distributed datasets, or RDDs, are Spark’s primary data abstraction consisting of a fault-tolerant collection of elements partitioned across the nodes of the cluster, capable of accepting parallel operations.You can create an RDD using an external or local Hadoop-supported file, from a collection, or from another RDD. RDDs are immutable and always recoverable, providing resilience in Apache Spark RDDs can persist or cache datasets in memory across operations, which speeds iterative operations in Spark.

Apache Spark architecture consists of components data, compute input, and management. The fault-tolerant Spark Core base engine performs large-scale Big Data worthy parallel and distributed data processing jobs, manages memory, schedules tasks, and houses APIs that define RDDs.

Spark SQL provides a programming abstraction called DataFrames and can also act as a distributed SQL query engine. Spark DataFrames are conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations.