Big Data Technologies

Agenda

- Spark Architecture
- Spark RDD operations

Java 8 Stream Tutorial

- https://winterbe.com/posts/2014/07/31/java8-stream-tutorial-examples/

Apache Spark

Spark cluster installation

- Spark cluster can be built with various cluster managers.
  - Standalone (built-in) ***
  - YARN
  - Mesos
  - Kubernetes
- Installation modes
  - Local mode
  - Single node cluster
  - Multi-node cluster

Single Node cluster (Standalone) on Linux

- set SPARK_HOME and PATH in ~/.bashrc
- set SPARK_MASTER_HOST and SPARK_LOCAL_IP in spark-env.sh
- set spark.master in SPARK_HOME/conf/spark-defaults.conf = spark://localhost:7077
- add "localhost" in SPARK_HOME/conf/workers

- terminal> spark-master.sh
- terminal> jps
- Browser: http://localhost:8080/
- terminal> spark-workers.sh
- terminal> jps
- Browser: http://localhost:8080/
- terminal> pyspark --master spark://localhost:7077
- terminal> jps

Word Count

- Run the commands in pyspark shell.

```
# read the file and load as rdd
rdd1 = sc.textFile("hdfs://master:9000/user/hduser/wordcount/input")

# convert each line in lower case
rdd2 = rdd1.map(lambda line: line.lower())

# split each line into multiple words
rdd3 = rdd2.flatMap(lambda line: line.split())

# filter out stop words
rdd4 = rdd3.filter(lambda word: word not in ["", "is", "are", "the", "in", "by", "of"])

# add one into each word
rdd5 = rdd4.map(lambda word: (word, 1))

# group by word and sum all occurrences of count
rdd6 = rdd5.reduceByKey(lambda a,b: a + b)

# capitalize the word
rdd7 = rdd6.map(lambda wc: (wc[0].upper(), wc[1]))
```

```
# collect the result
#result = rdd7.collect()

# save result into hdfs
rdd7.saveAsTextFile("hdfs://master:9000/user/hduser/wordcount/output1")

# print result
print(result)
```

Word Count Program -- Explanation

- Input -- rdd1

```
Red green blue
Red red green
green Blue black
green Green blue
```

- Lower Case -- rdd2

```
red green blue
red red green
green blue black
green green blue
```

- Split -- flatMap() -- rdd3

```
red
green
```

```
blue
red
red
green
green
blue
black
green
green
blue
```

- Filter Stop Words -- rdd4

```
red
green
blue
red
red
green
green
blue
black
green
green
blue
```

- Add one -- rdd5 -- Key-Value pair RDD (RDD of Tuple2)

```
(red,1)
(green,1)
(blue,1)
(red,1)
```

```
(red,1)
(green,1)
(green,1)
(blue,1)
(black,1)
(green,1)
(green,1)
(blue,1)
```

- Group by word and Sum ones -- reduceByKey -- rdd6
  - Grouping by Key

```
(red, [1,1,1])
(green, [1,1,1,1,1])
(blue, [1,1,1])
(black, [1])
```

  - Reduce on each group --> (a,b) => a + b
    - "1, 1," 1, 1, 1
      - 1 + 1 = 2
    - 1, 1, "1," 1, 1
      - 2 + 1 = 3
    - 1, 1, 1, "1," 1
      - 3 + 1 = 4
    - 1, 1, 1, 1, "1"
      - 4 + 1 = 5
  - Result of reduce

```
(red, 3)
(green, 5)
```

```
(blue, 3)
(black, 1)
```

- Capitalize the word --> rdd7

```
(RED, 3)
(GREEN, 5)
(BLUE, 3)
(BLACK, 1)
```

Spark Terminologies

- RDD
  - Resilient Distributed Dataset.
  - Typically loaded and processed in memory (RAM).
  - Divided into multiple partitions.
- Spark application
  - Set of Spark Jobs (one or more jobs).
  - Each application have single SparkContext.
  - UI is available on port 4040.
  - e.g. spark-shell -- is a spark application.
  - Application is submitted using spark-submit.
    - e.g. spark-submit ... demo01.py
    - e.g. spark-submit ... app.jar
- Spark job
  - Terminated with action operation.
  - Divided into one or more stages.
  - Stages are separated by shuffle/wide operations.
- Stage
  - Set of narrow RDD operations.

    - Executed in single JVM process (in machine).
    - Divided into multiple tasks.
- Tasks
    - The tasks are created for each partition in the RDD.
    - Each task is a thread of execution (in JVM process).

RDD

- RDD is Resilient.
    - If any RDD (or partitions in RDD) is failed (due to network, machine or operation failure), it can be recreated --> Resilient.
    - It is recreated from the source (textFile(), cached RDD, etc) by applying set of operations again.
    - Each RDD set of operations are maintained in its metadata -- RDD Lineage.
    - spark-shell> rdd5.toDebugString

```
res5: String =
(2) MapPartitionsRDD[5] at map
    |  MapPartitionsRDD[4] at filter
    |  MapPartitionsRDD[3] at flatMap
    |  MapPartitionsRDD[2] at map
    |  file:///D:/setup/spark-3.0.1-bin-hadoop3.2/LICENSE MapPartitionsRDD[1] at textFile
    |  file:///D:/setup/spark-3.0.1-bin-hadoop3.2/LICENSE HadoopRDD[0] at textFile
```

- RDD is Distributed.
    - RDD is logical entity -- no space occupied by RDD "directly".
    - RDD is divided into partitions. Partitions are stored in memory RAM of one or more nodes/machines.
    - Partition size depends on how RDD is created and source of RDD.
    - rdd = sc.textFile("hdfs://localhost:9000/user/sunbeam/ncdc/input")
        - When RDD is created from data in the HDFS, the number of Partitions = Number of Input splits = Number of HDFS blocks.
        - The partitioning is done by Hadoop TextInputFormat and RecordReader.
        - Usually one Partition = HDFS block size (128 MB).
    - val rdd = sc.parallelize(1 to 100) // scala

    - rdd = sc.parallelize(range(1, 100)) // python
        - sc.parallelize() convert scala/python collection into Spark RDD.
        - Number of Partitions depends on Cluster resources.
        - By default, "local" mode it depends on number of CPUs.
            - rdd.partitions.length
            - rdd.partitions
                - Array[org.apache.spark.Partition] = Array(org.apache.spark.rdd.ParallelCollectionPartition@802, ...()
    - val rdd = sc.parallelize(1 to 100, 2)
        - Programmer may decide numnber of partitions while reading from source (in few cases) and during shuffle operations.
        - rdd.partitions.length
- RDD is immutable.
    - RDD cannot be modified and operation will result into new RDD (similar to Java Strings).
- RDD Programming Guide
    - https://spark.apache.org/docs/latest/rdd-programming-guide.html

Assignment

- Load ratings dataset into a RDD. Sort ratings by count in descending order and get the first() movie. Load movies dataset into a RDD. Lookup movie with highest rating.
- Find number of movies for each genre using split() and explode().
- Try the commands in pyspark `--master local[4]`.

```
values = range(1,100)

rdd1 = sc.parallelize(values)
rdd1.getNumPartitions()
rdd1.saveAsTextFile("file:///tmp/output1")

rdd2 = sc.parallelize(values, 2)
rdd2.getNumPartitions()
rdd2.saveAsTextFile("file:///tmp/output2")
```