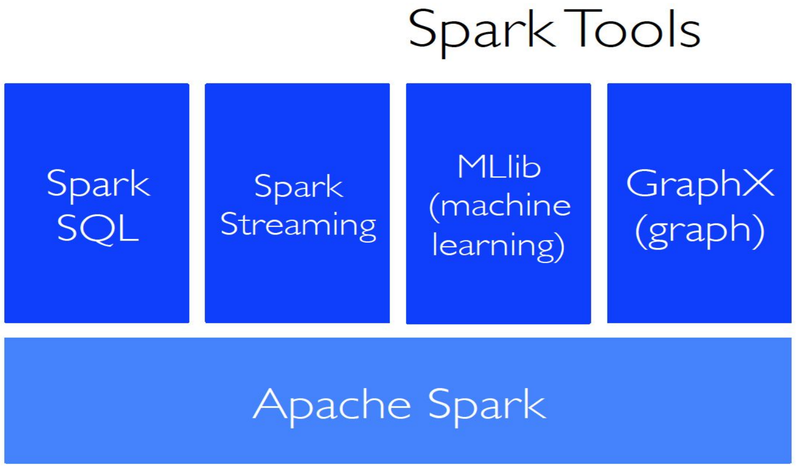
Spark

Introduction and Demo

Ref : https://github.com/sukriti10/sparkdemo

Introduction

1. Apache Spark is an open-source, distributed processing system commonly used for big data workloads.
2. Apache Spark utilizes in-memory caching and optimized execution for fast performance..
3. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs.
4. It provides higher-level tools like:
   a. Spark SQL: for SQL and structured data proc
   b. MLlib : For Machine Learning
   c. GraphX : For Graph Processing
   d. Spark Streaming



Spark and Map Reduce Differences

	Hadoop Map Reduce	Spark
Storage	Disk only	In-memory or on disk
Operations	Map and Reduce	Map, Reduce, Join, Sample, etc…
Execution model	Batch	Batch, interactive, streaming
Programming environments	Java	Scala, Java, R, and Python

Other Spark and Map Reduce Differences

- Generalized patterns
  ⇒ unified engine for many use cases
- Lazy evaluation of the lineage graph
  ⇒ reduces wait states, better pipelining
- Lower overhead for starting jobs
- Less expensive shuffles

In-Memory Can Make a Big Difference

- Two iterative Machine Learning algorithms:

Spark MLlib

What : Apache Spark's scalable machine learning library.

Why use MLlib when there are other packages like sklearn ? : It is built on Apache Spark, a fast and general engine for large-scale data processing. This means that it can run programs up to 100x faster than Hadoop MapReduce in memory, or 10x faster on disk.

Ref : https://spark.apache.org/docs/latest/mllib-guide.html

Example 1 : Gradient Descent

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^{n} g(w; x_i, y_i)$$

```
val points = spark.textFile(...).map(parsePoint).cache()
var w = Vector.zeros(d)
for (i <- 1 to numIterations) {
  val gradient = points.map { p =>
    (1 / (1 + exp(-p.y * w.dot(p.x)) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= alpha * gradient
}
```

Example 2 : Linear Regression

```python
from pyspark.ml.regression import LinearRegression

# Load training data
training = spark.read.format("libsvm")\
    .load("data/mllib/sample_linear_regression_data.txt")

lr = LinearRegression(maxIter=10, regParam=0.3, elasticNetParam=0.8)

# Fit the model
lrModel = lr.fit(training)

# Print the coefficients and intercept for linear regression
print("Coefficients: %s" % str(lrModel.coefficients))
print("Intercept: %s" % str(lrModel.intercept))

# Summarize the model over the training set and print out some metrics
trainingSummary = lrModel.summary
print("numIterations: %d" % trainingSummary.totalIterations)
print("objectiveHistory: %s" % str(trainingSummary.objectiveHistory))
trainingSummary.residuals.show()
print("RMSE: %f" % trainingSummary.rootMeanSquaredError)
print("r2: %f" % trainingSummary.r2)
```

Example 3 : KMeans clustering

```
# Load and parse the data
data = sc.textFile("kmeans_data.txt")
parsedData = data.map(lambda line:
               array([float(x) for x in line.split(' ')])).cache()

# Build the model (cluster the data)
clusters = KMeans.train(parsedData, 2, maxIterations = 10,
             runs = 1, initialization_mode = "kmeans||")

# Evaluate clustering by computing the sum of squared errors
def error(point):
    center = clusters.centers[clusters.predict(point)]
    return sqrt(sum([x**2 for x in (point - center)]))

cost = parsedData.map(lambda point: error(point))
         .reduce(lambda x, y: x + y)
print("Sum of squared error = " + str(cost))
```

Spark SQL

Integrates relational processing with Spark's functional programming.

Goals :

1. Support relational processing both within spark processing and on external data sources with a friendly API.
2. High performance
3. Easily support new data sources such as semi-structured data, and external databases.

What :

1. Spark module for structured data processing
2. Implemented as a library on top of Spark

Spark SQL

Three main APIs:

1. SQL Literal syntax
2. Dataset API
3. Dataframe API

Dataset

Dataset is a distributed collection of data. It provides the benefits of RDDs (strong typing, ability to use powerful lambda functions) with the benefits of Spark SQL's optimized execution engine.

The dataset API is available in Java and Scala. Python does not have the support for the Dataset API. But due to Python's dynamic nature, many of the benefits of the Dataset API are already available.

DataFrame is a *Dataset* organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood.

Apache Spark with AWS EMR

Amazon Elastic MapReduce(EMR) : Developers and analysts can use Jupyter-based EMR Notebooks for iterative development, collaboration, and access to data stored across AWS data products to reduce time to insight and quickly operationalize analytics.

You can install Spark on an EMR cluster along with other Hadoop applications, and it can also leverage the EMR file system (EMRFS) to directly access data in Amazon S3.

How to :

Boto3 Client :

```
import boto3

client = boto3.client('emr')
```

Amazon EMR Notebook : Based on Jupyter Notebook that allows you to quickly create Jupyter notebooks, attach them to Spark clusters, and then open the Jupyter Notebook editor in the console to remotely run queries and code.

An EMR notebook is saved in Amazon S3 independently from clusters. You can have multiple notebooks open, attach multiple notebooks to a single cluster, and re-use a notebook on different clusters.

Pyspark

```
import sys
from random import random
from operator import add

from pyspark import SparkContext


if __name__ == "__main__":
    """
        Usage: pi [partitions]
    """
    sc = SparkContext(appName="PythonPi")
    partitions = int(sys.argv[1]) if len(sys.argv) > 1 else 2
    n = 100000 * partitions

    def f(_):
        x = random() * 2 - 1
        y = random() * 2 - 1
        return 1 if x ** 2 + y ** 2 < 1 else 0

    count = sc.parallelize(xrange(1, n + 1), partitions).map(f).reduce(add)
    print "Pi is roughly %f" % (4.0 * count / n)

    sc.stop()
```

Demo: Wordcount using AWS EMR with Spark

Create emr cluster configured for Spark

We will now go through three ways to calculate word count using spark.

- Way 1 : SSH and run script

```
# Secure copy your python code to the cluster
scp -i <SECRET-KEY> wordcount.py <CLUSTER>.us-east-2.compute.amazonaws.com

# SSH into your cluster
ssh -i <SECRET-KEY> <CLUSTER>.compute.amazonaws.com

# Run the script on the spark
spark-submit wordcount.py | tee output.txt
```

- Way 2: Use lambda to run spark script
- Way 3 : Using EMR, go to notebook on the emr page , create notebook and attach the cluster you created. [Sample link to word count using emr notebook](#).

References

https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-spark-application.html

https://docs.aws.amazon.com/code-samples/latest/catalog/python-emr-jupyterhub-install-libraries.py.html

https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/emr.html

https://docs.amazonaws.cn/en_us/emr/latest/ManagementGuide/emr-managed-notebooks-working-with.html