Big Data and Spark

* Spark needs a cloud service such as AWS.

# Overview

* Using your local computer alone restricts you to max 8gb data (RAM) which can be increased by using SQL to move some of these to the hard drive but the best option is to move it to a distributed system of computers which is controlled by one machine known as master node.
* Distributed machines have the advantages of easy scaling, fault tolerant, etc.
* Hadoop allows distribution of very large files across machines using Hadoop Distributed File System (HDFS) allowing users to work with large data sets.
* For fault tolerance, Hadoop duplicates blocks of data and then uses MapReduce to allow computation on the data.
* HDFS usually has a main name node along with along with data nodes (other distributed machines) that hold data.
* HDFS uses a block size of 128 mb by default and is replicated thrice and distributed across the data nodes in such a way that if one of the nodes went offline, the data can be got from the other nodes and still be used irrespective.
* MapReduce is a way of splitting a computation task to a distributes set of files.
* MapReduce consists of a Job tracker which send code to run in the task trackers and multiple task trackers which allocate space to run the code while monitoring the tasks on the worker nodes.
* Summary, use HDFS to distribute files to a distributed group of machines, then, use MapReduce to distribute a computational task to a distributed data set.

# Spark Overview

* Its an open-source project on Apache, developed around 2009 at AMPlab in UC Berkeley and released in February 2013.
* It has gained popularity due to its ease of use and speed.
* It has a number of advantages over MapReduce such as being more flexible, being abe to handle different types of data formats such as AWS S3, Cassandra, HDFS etc.
* Spark is an alternative to MapReduce.
* MapReduce requires data to be stored in Hadoop while Spark doesn’t; spark is up to 100 times faster.
* Spark achieves this speed by holding data in memory after every operation and only spilling to the hard drive when memory (RAM) is filled whereas, MapReduce writes to Hard disk after every Map and Reduce operation.
* The core of Spark is Resilient Distributed Dataset (RDD) with four main features: fault tolerance, ability to use may data sources, parallel operation (partitioned), and distributed collection of data.
* RDDs are immutable, cacheable and lazily evaluated.
* There are 2 types of RDD operations: transformation and action.
* The four most basic RDD actions are first (returns the first element of the RDD), collect (returns all the elements of the RDD as an array), count (returns the number of elements in the RDD), take (returns an array of the first n elements of the RDD).
* The basic RDD transformations are: filter (similar to python’s built in filter, it applies a function to each element and returns those that are true), map (similar to pands’s ‘.apply’, applies a function transforms every element but preserves the number of elements ) and flatmap (transforms the elements into a 0-N elements and therefore changes the number of elements).
* RDDs typically hold data in tuples of key, value pairs which offers better partitioning and leads to functionality based on reduction.
* The reduction methods are: reduce (aggregates RDD elements using a function that returns a single element) and reduce by key (aggregates pair RDD elements with a function that returns a pair RDD). These are similar to groupby operation
* the spark ecosystem is expanding really quickly and already includes: MLlib, GraphX, Spark data frames, spark SQL, spark streaming etc.

# Pyspark set up

* create an aws account
* login
* create an ec2 instance
* generate a key pair
* log in the ssh on putty
* download anaconda for this virtual instance
* install it
* make sure to set the python to the anaconda pyton
* configure the Jupyter by generating config, make certifications
* install spark and its dependencies, java and scala.
* Install pip to the anaconda and with pip, install py4
* Download and install spark Hadoop

# Lambda expression review

* Lambda expressions are a briefer version of functions.
* Takes in input and returns a transformation.
* Syntax is lambda a: b . a is input, b is transformation to be returned example is lambda num: a\*\*2, lambda a: a%2, lambda a: a[: :-1], lambda x, y: x + y etc.

# Introduction to spark and python

* From pyspark import SparkContext
* Instantiate spark context for example sc = SparkContext()
* Sparkcontext represents a connection to a spark cluster and can be used to create RDD and broadcast to it
* You can only have one spark context in the way this course is running it.
* Create a text file with magic function %%writefile filename.txt and then in the same cell, write te contents of the file.
* You can create an RDD object with a = sc.textfile(filename). Spark context’s text file method creates an RDD from a text file. A is object name
* You can count number of lines with a.count() were a is object name (an action)
* You can grab the first line with a.first(). (an action)
* You can transform by filtering and return a new RDD with b = a.filter(lambda line: ‘second’ in line)
* Note that it is very fast because it doesn’t actually perform the action until asked to do so. Just keeps the instructions.
* You can count the number of lines that met the condition with b.count() or call an action e.g., b.collect()

# RDD Transformations and Actions

* RDD is a Resilient Distributed Data set; a transformation produces an RDD, an action produces a local object you can then use in different ways; a spark job is a sequence of transformation on data with a final action.
* You can create an RDD from a text file or from an array with sc.textFile (file path) or sc.parallelize (array) were sc is an instance of SparkContext
* Common RDD transformations include: filter (lambda x: x%2) which returns only elements that are true; map (lambda x: x \*2) which multiplies each element by 2 or simply maps every element by the given lambda expression or function such as ‘.split’; flatMap (lambda x: x.split()) which maps and flattens the result; sample (withReplacement = True/False, 0.3) which gets a random sample of the RDD with or without replacement and at the specified sample size which is 0.25 in this example; union(rdd) which appends an RDD to another; distinct() that removes duplicates; sortBy (lambda x:x ascending = True or False) for sorting in ascending or descending order etc.
* Common RDD actions include collect() which converts RDD to in memory list; take(n) which grabs the first n elements of the RDD; top() which functions like take but is used after sorting; takeSample (withReplacement = True/False, n) where n is of elements in the sample; sum(), mean() and stdev() which sums, returns mean and standard deviation respectively.
* For use on text RDDs, map Vs Flat map:
* Import SparkContext from pyspark
* Instantiate it e.g., sc = SparkContext()
* Create or import file and set as an object e.g., text\_rdd = sc.textFile(a) were a is file name or path
* You can do any desired transformation on the RDD and call ‘.collect’ to perform the action.
* Flat map in the above scenario will have returned everything as a single list instead of for each line
* For use on key value pairs RDD:
* Set the text file to an object
* Clean the data
* Get rid of undesired parts
* Perform an action