**Learning Objective 3**

To be able to use the Apache Spark Scala api

* In the Spark shell, there is a special interpreter-aware SparkContext
* For a complete list of options, run spark-shell --help
* The main abstraction Spark provides is a resilient distributed dataset (RDD)
* RDDs are created by starting with a file in HDFS or local fs, or an existing Scala collection and transforming it.

1 get some realistic sample data

wget<http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/spam.data>

Very Simple task

Take input and transform

in this case String to double

2. val inFile = sc.textFile("./spam.data")

val nums = inFile.map(x => x.split(' ').map(\_.toDouble))

inFile.first()

nums.first()

**Learning Objective 4**

To be able to Run Apache Spark sample code on the cluster

* Spark applications can be written with Scala, Java and Python
* Spark application consists of a driver program that runs the main function and executes various parallel operations on a cluster.
* Spark 1.1.1 uses Scala 2.10. To write applications in Scala, you will need to use a compatible Scala version
* Spark 1.1.1 works with Python 2.6 or higher (but not Python 3). It uses the standard CPython interpreter, so C libraries like NumPy can be used
* Spark 1.1.1 works with Java 6 and higher. If you are using Java 8, Spark supports [lambda expressions](http://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html)

We will look initially at the Java Api because it is the primary

Hadoop related language.

**Anatomy of a Spark Application**

/\*\*

\* Computes an approximation to pi

\* Usage: JavaSparkPi [slices]

\*/

**public final class JavaSparkPi {**

**public static void main(String[] args) throws Exception {**

**SparkConf sparkConf = new SparkConf().setAppName("JavaSparkPi");**

**JavaSparkContext jsc = new JavaSparkContext(sparkConf);**

**int slices = (args.length == 1) ? Integer.parseInt(args[0]) : 2;**

**int n = 100000 \* slices;**

**List<Integer> l = new ArrayList<Integer>(n);**

**for (int i = 0; i < n; i++) {**

**l.add(i);**

**}**

**JavaRDD<Integer> dataSet = jsc.parallelize(l, slices);**

**int count = dataSet.map(new Function<Integer, Integer>() {**

**@Override**

**public Integer call(Integer integer) {**

**double x = Math.random() \* 2 - 1;**

**double y = Math.random() \* 2 - 1;**

**return (x \* x + y \* y < 1) ? 1 : 0;**

**}**

**}).reduce(new Function2<Integer, Integer, Integer>() {**

**@Override**

**public Integer call(Integer integer, Integer integer2) {**

**return integer + integer2;**

**}**

**});**

**System.out.println("Pi is roughly " + 4.0 \* count / n);**

**}**

**}**

Fragment from run-example.sh

"$FWDIR"/bin/spark-submit \

--master $EXAMPLE\_MASTER \

--class $EXAMPLE\_CLASS \

"$SPARK\_EXAMPLES\_JAR" \

"$@"

./bin/spark-submit --class org.apache.spark.examples.SparkPi \

--master yarn-cluster --num-executors 3 --driver-memory 512m \

--executor-memory 512m --executor-cores 1 lib/spark-examples\*.jar 10

export YARN\_CONF\_DIR=/etc/hadoop/conf

http://127.0.0.1:8088/cluster/app/application\_1421653738249\_0001

<http://sandbox.hortonworks.com>: → http://127.0.0.1:

http://127.0.0.1:8088/cluster/app/application\_1421653738249\_0001

Pi is roughly 3.142388