Start the Scala Spark shell:

$SPARK\_HOME/bin/spark-shell

Joining RDDs

1.Next, you are going to create RDDs for the README and the CHANGES file.

val readmeFile = sc.textFile("/tmp/README.md")

val changesFile = sc.textFile("/tmp/CHANGES.txt")

2.How many Spark keywords are in each file?

readmeFile.filter(line => line.contains("Spark")).count()

changesFile.filter(line => line.contains("Spark")).count()

3.Now do a WordCount on each RDD so that the results are (K,V) pairs of (word,count)

val readmeCount = readmeFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(\_ + \_) //flatmap takes one argument and outputs many(one to many) map is one to one.

val changesCount = changesFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(\_ + \_)

4.To see the array

for either of them, just call the collect function on it.

readmeCount.collect()

changesCount.collect()

5.Now let's join these two RDDs together to get a collective set.

The join function combines the two datasets (K,V) and (K,W) together and get (K, (V,W)). Let's

join these two counts together

val joined = readmeCount.join(changesCount)

6.Cache the joined dataset.

joined.cache()

7.Print the value to the console

joined.collect.foreach(println)

8.Let's combine the values together to get the total count

The operations in this command tells Spark to combine the go from (K,V) and (K,W) to (K, V+W). The .\_ notation is a way to access the value on that particular index of the key value pair.

val joinedSum = joined.map(k => (k.\_1, (k.\_2).\_1 + (k.\_2).\_2))

joinedSum.collect()

9.To check if it is correct, print the first five elements from the joined

and the joinedSum RDD

joined.take(5).foreach(println)

joinedSum.take(5).foreach(println)

10.Broadcast variables are useful for when you have a large dataset that you want to use across all

the worker nodes. Instead of having to send out the entire dataset, only the variable is sent out.

In the same shell from the last section, create a broadcast variable. Type in

val broadcastVar = sc.broadcast(Array(1,2,3))

11.To get the value, type in:

broadcastVar.value

12.Accumulators are variables that can only be added through an associative operation. It is used to implement counters and sum efficiently in parallel. Spark natively supports numeric type accumulators and standard mutable collections. Programmers can extend these for new types. Only the driver can read the values of the accumulators. The workers can only invoke it to increment the value.

13.Create the accumulator variable. Type in:

val accum = sc.accumulator(0)

14.Next parallelize an array of four integers and run it through a loop to add each integer value to the accumulator variable. Type in:

sc.parallelize(Array(1,2,3,4)).foreach(x => accum += x)

15.To get the current value of the accumulator variable, type in:

accum.value

You should get a value of 10.

This command can only be invoked on the driver side. The worker nodes can only increment the accumulator

.

16.Key-value pairs

You have already seen a bit about key-value pairs in the Joining RDD section. Here isa brief example of how to create a key-value pair and access its values.

Remember that certain operations such as map and reduce only works on key-value pairs.

17.Create a key-value pair of two characters. Type in:

val pair = ('a', 'b')

18. To access the value of the first index, type in:

pair.\_1

19.To access the value of the second index, type in:

pair.\_2