732A54 - Big Data Analytics

Spark Lesson

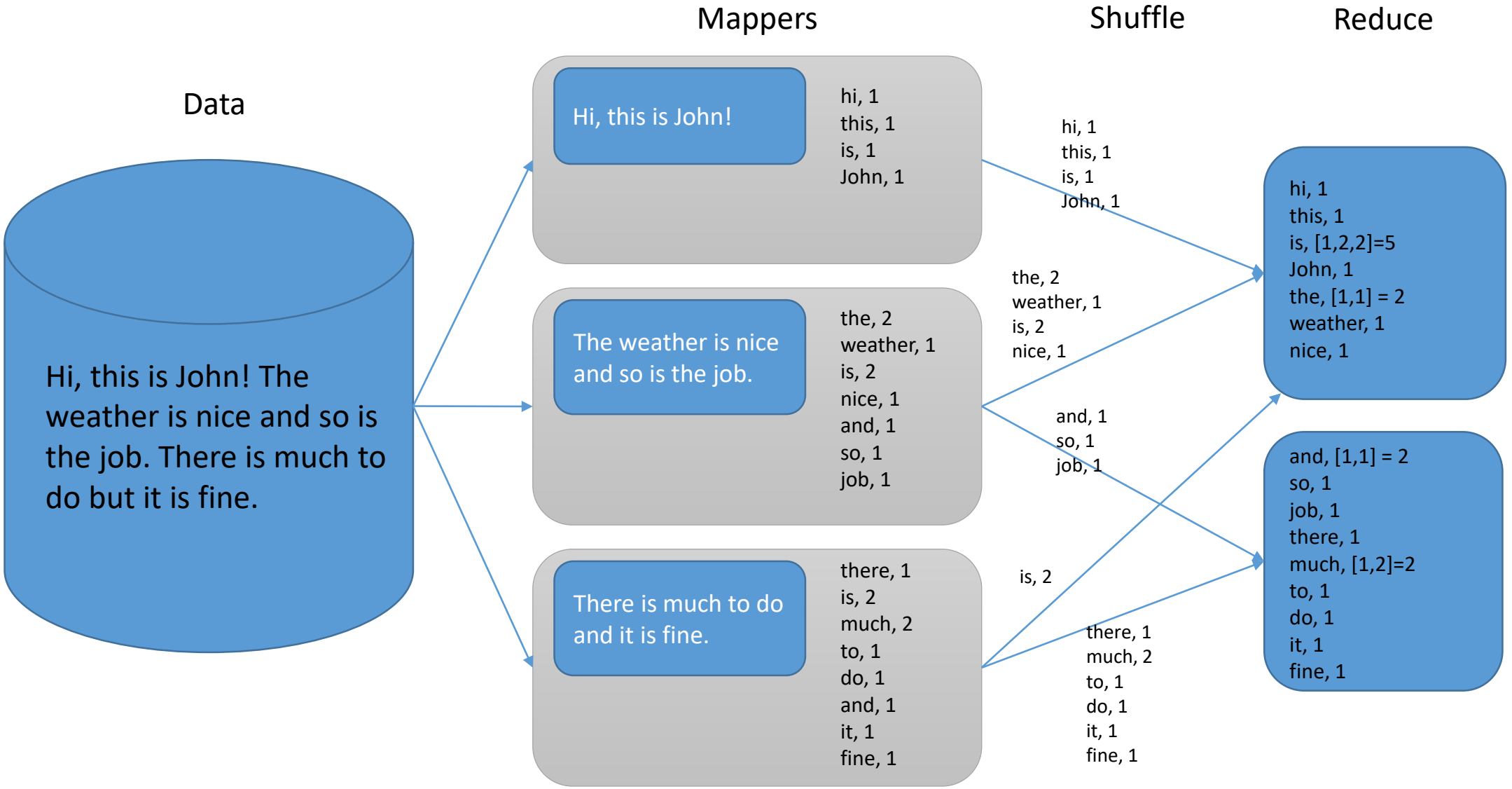
Word Count - problem

- Find the number of occurrences of each word in the DavidCopperfield.txt file.

Word Count – conceptual solution

- We divide the text into a number of more manageable pieces – partitions
- From each partition, we extract words from the text
- Next, in each worker count the occurrences of each word in that partition and send your intermediate results to node(s) which sum up all intermediate results for each word

Word Count – conceptual solution



Word Count - algorithm

- Pre-step 1: the text file first needs to be distributed. The simplest approach is to do this using hdfs commands:

  hdfs dfs –copyFromLocal DavidCopperfield.txt data/

  This command copies the DavidCopperfield.txt file to hdfs data folder

  Good to know:

  hdfs dfs –rm file.txt - remove the file

  hdfs dfs –ls - check the content of the folder

  hdfs dfs –rm –r folder - remove the folder and its content

  hdfs dfs –copyToLocal results/ . - copy the results/ folder to the current folder(.)
- Step 1: read the file from hdfs, Spark produces RDD
- Step 2: write the code which extracts words from the text
- Step 3: produce key-values pairs for each word - (word, 1)
- Step 4: write the reducer code which sums up 1s for each key/word
- Step 5: output the results

Lambda functions

- General form:

  `lambda arguments: expression`

- Examples

  `lambda a: 2*a` – doubles the argument `a`

  `lambda a, b: a+b` – produces the sum of arguments `a` and `b`

  `lambda a: (a%2 == 0)` – can be used to filter data, `a` is accepted only if `a` is divisible by 2

  `lambda a, b: a if (a%2 == 0) else b` – another filtering example, returns `a` if divisible by 2, otherwise `b`

Tuples

- A tuple is a sequence of immutable Python objects e.g.
  - `('a', 1, 3) (1, 2, (3, 4)) (3, 'a', ('c', [1]))`
- Empty tuple - `()`, tuple with one value - `(1,)`
- Accessing elements done with [index], for example for
  - `x = (3, 'a', ('c', [1]))`

    `x[0] = 3`

    `x[2] = x[-1] = ('c', [1])`

    `x[2][1] = [1]`
- Some functions: len, max, min

Word count - PySpark Code

```
from pyspark import SparkContext

sc = SparkContext(appName="Wordcount")

myfile = sc.textFile("/user/zladr41/data/DavidCopperfield.txt")
words = myfile.flatMap(lambda line: line.split(" "))
counts = words.map(lambda word: (word, 1))
counts = counts.reduceByKey(lambda v1,v2: v1 + v2)
counts.saveAsTextFile("/user/zladr41/data/results/")
```

Import SparkContext
Get the reference to it
Give a name to the application

Get the file on hdfs
Absolute path needed!

Map to extract words

Map to produce key-value pairs (word, 1)

Reduction, works on pairs of key-value pairs
It will sum up values of key-value pairs with the same key

Save the results to hdfs

Word count - variations

- Modify the wordcount program by only considering words with at least 4 characters.
- Sort the output

Word count – PySpark Code (2)

```
from pyspark import SparkContext


sc = SparkContext(appName="Wordcount")


myfile = sc.textFile("/user/zladr41/data/DavidCopperfield.txt")
words = myfile.flatMap(lambda line: line.split(" "))
words = words.filter(lambda w: len(w)==4)
counts = words.map(lambda word: (word, 1))
counts = counts.reduceByKey(lambda v1,v2: v1 + v2)
counts = counts.sortBy(ascending=False, keyfunc=lambda a: a[1])
counts.saveAsTextFile("/user/zladr41/data/results/").repartition(1)
```

Some built-ins

- Count the total number of words

```
from pyspark import SparkContext
sc = SparkContext(appName="Wordcount")
myfile = sc.textFile("/user/zladr41/data/DavidCopperfield.txt")
numberOfWords = myfile.flatMap(lambda line: line.split(" ")).count()
print numberOfWords
```

- Count the total number of unique words

```
from pyspark import SparkContext
sc = SparkContext(appName="Wordcount")
myfile = sc.textFile("/user/zladr41/data/DavidCopperfield.txt")
numberOfWords = myfile.flatMap(lambda line: line.split(" ")).distinct().count()
print numberOfWords
```

Finding max/min temperature

- What are the lowest and highest temperatures measured each year for the period 1950-2014. Provide the lists sorted in the descending order w.r.t. temperature. In this exercise you will use the *temperature-readings.csv* file.

Finding max/min temperatures – conceptual solution

- Divide the input into more manageable parts
- In each part, select only the necessary information, i.e. year and temperature
- Filter out the data that is not in the interval 1950-2014
- Find the maximum for each year in each partition
- Send the intermediate results to node(s) which then find the maximum from the intermediate results

- Minimum done analogously

Finding max/min temperatures - algorithm

- Pre-Step 1: distribute the *temperature-readings.csv* file using hdfs commands
- Step 1: read the file, Spark produces an RDD
- Step 2: select only relevant values in each row (year and temperature), produce key-value pairs
- Step 3: filter out the years that are not relevant
- Step 4: write the reducer code which finds the maximum value for each year
- Step 5: output the results

Finding max/min – PySpark Code

```
from pyspark import SparkContext
sc = SparkContext(appName="maxMin")
lines = sc.textFile("/user/zladr41/data/temperature-readings.csv").cache()
lines = lines.map(lambda a: a.split(";"))
lines = lines.filter(lambda x: int(x[1][0:4]) >= 1950 and int(x[1][0:4])<=2014)

temperatures = lines.map(lambda x: (x[1][0:4], float(x[3])))


maxTemperatures = temperatures.reduceByKey(max)
maxTemperaturesSorted =
        maxTemperatures.sortBy(ascending=False,keyfunc=lambda k: k[1])
maxTemperaturesSorted.saveAsTextFile("data/resultsMax")
```

Import SparkContext
Get the reference to it
Give a name to the application

Get the file on hdfs
Absolute path needed!

Map to split the row to get values

Remove rows where year not in interval, x[1][0:4] is the year. **NOTE datatypes**

Map to create key-value pairs (year, temperature)

Reduction, works on pairs of key-value pairs. Uses built-in function max, essentially, lambda a,b: max(a,b)

Sort data, the key is the temperature value

Save the results to hdfs

Finding max/min temperatures – PySpark Code (2)

```
from pyspark import SparkContext
sc = SparkContext(appName="maxMin")
lines = sc.textFile("/user/zladr41/data/temperature-readings.csv").cache()
lines = lines.map(lambda a: a.split(";"))
lines = lines.filter(lambda x: int(x[1][0:4]) >= 1950 and int(x[1][0:4])<=2014)

temperatures = lines.map(lambda x: (x[1][0:4], float(x[3])))

maxTemperatures = temperatures.groupByKey()
maxTemperatures = maxTemperatures.map(lambda a: (a[0], max(a[1])))

maxTemperaturesSorted = maxTemperatures.sortBy(ascending=False,keyfunc=lambda k: k[1])
maxTemperaturesSorted.saveAsTextFile("data/resultsMax")
```

Returns all values for this key, (K, [V_1, V_2, …, V_n])

Map which selects only the maximum element from the list of all elements for some key

Finding max/min temperatures - variation

- Extend the program to include the station number (**not the station name**) where the maximum/minimum temperature was measured.