BIG DATA

In memory analytics with Spark : Introduction

K V Subramaniam
Computer Science and Engineering

Overview of lecture – Spark Introduction

- Why Spark – the motivation?
- Moving to in memory compute
- Distribute data in memory
- Handling fault tolerance
- Programming model – Operations in Spark
- Handling key-value based operations
- Putting it all together : Word count in Spark

Motivation for Spark

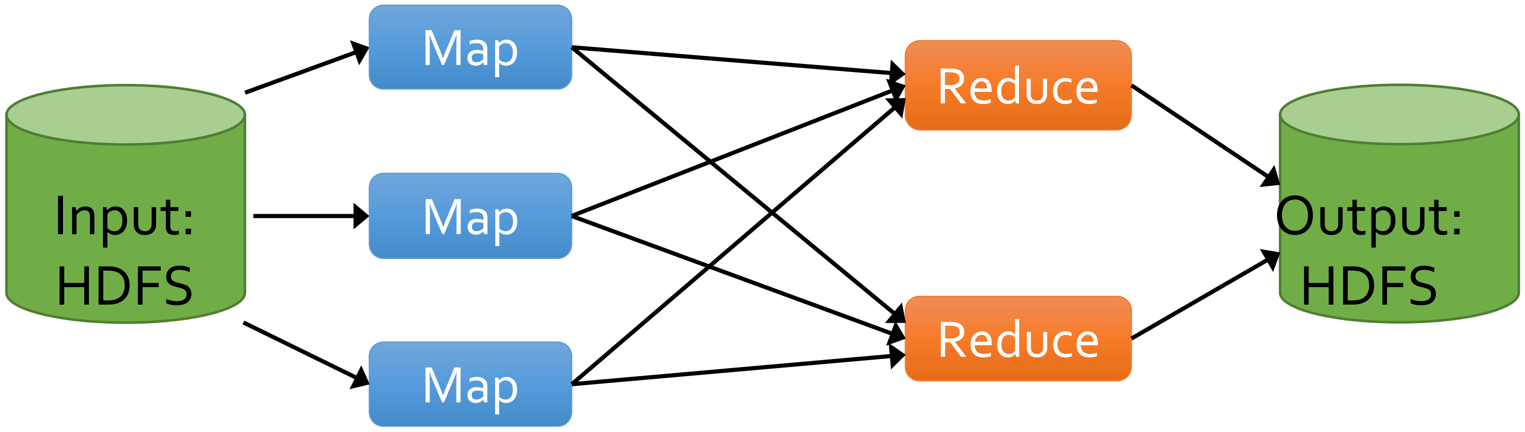
Spark: Motivation

Most current cluster programming models are based on *acyclic data flow* from stable storage to stable storage

Hadoop → reads data from persistent storage in *Map* step

Writes data back to persistent store (HDFS) in reduce

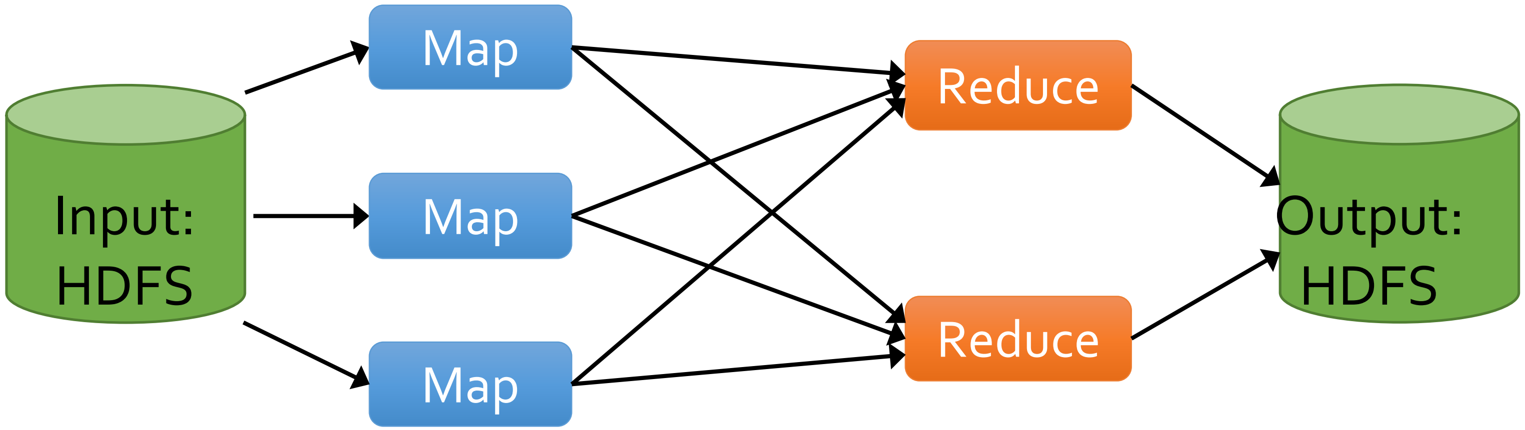
Advantage – dynamically decide machines/handle failures



Issues?

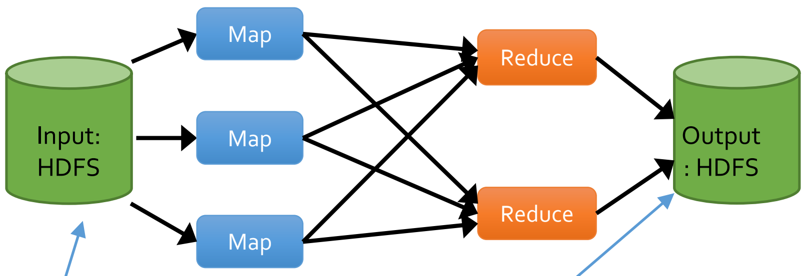
Consider the page rank exercise we did in the last unit?

What are the issues that we see with this?



Challenges

- Acyclic data flow
  - consider operating on a *data working set*
  - **Working set:** same set of data reused
  - in page rank, we keep computing importance vector and reusing in next iteration.
  - Hadoop – inefficient in such cases.
- Example of use
  - Where we need to *iterate*
    - Graph processing
    - Machine Learning
  - Where we need to do *interactive analysis*
    - Python, R



- On every iteration, storing/reloading of data from persistent storage is time consuming.

In Memory Computation

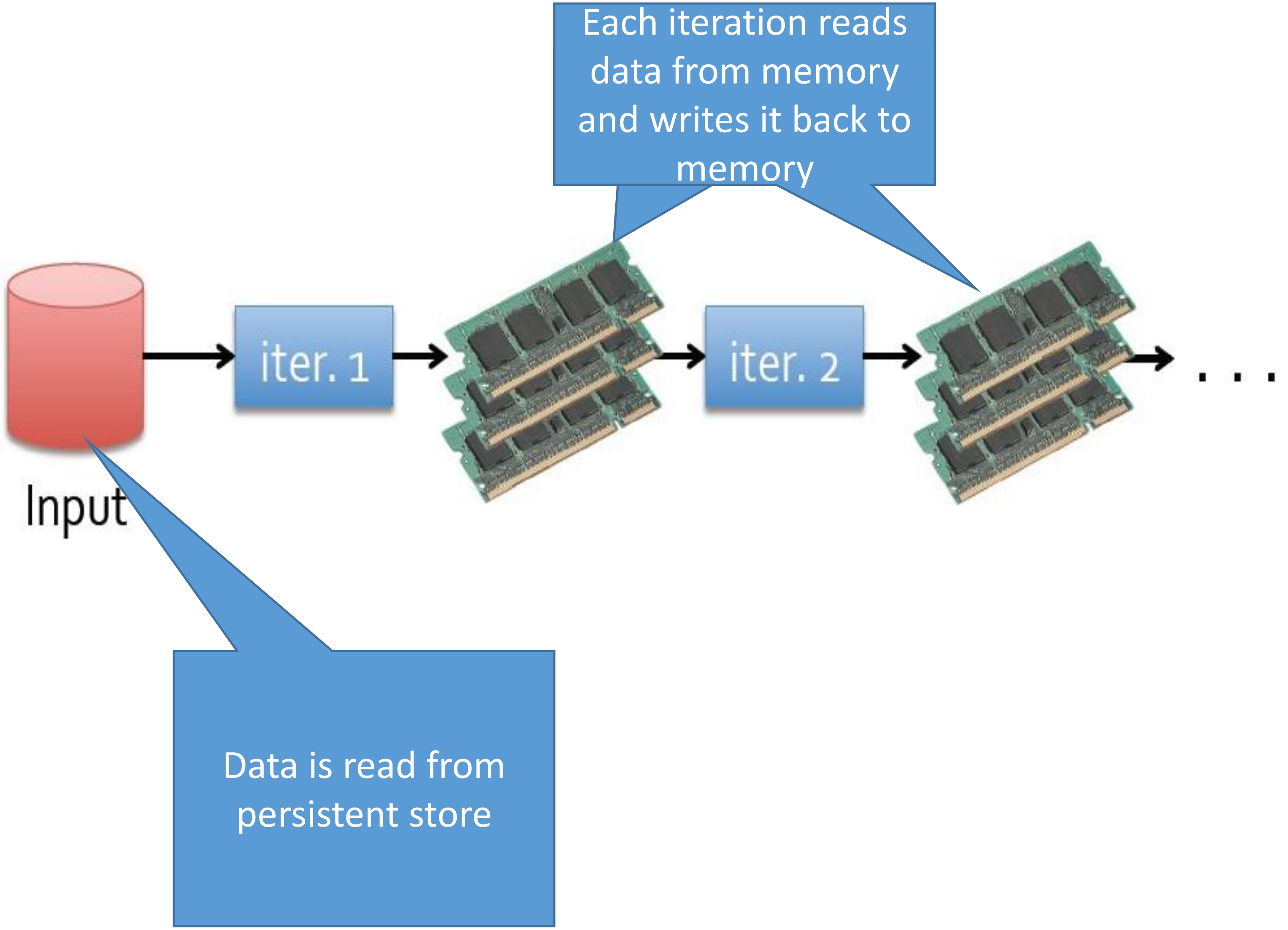
Recap: Word count in scala

```
val lines = scala.io.Source.fromFile("textfile.txt").getLines
val words = lines.flatMap(line => line.split(" ")).toIterable
val counts = words.groupBy(identity).map(words =>
    words._1 -> words._2.size)
val top10 = counts.toArray.sortBy(_._2).reverse.take(10)
println(top10.mkString("\n"))
```

Each operation creates
A data value that can be kept in
Memory and reused.

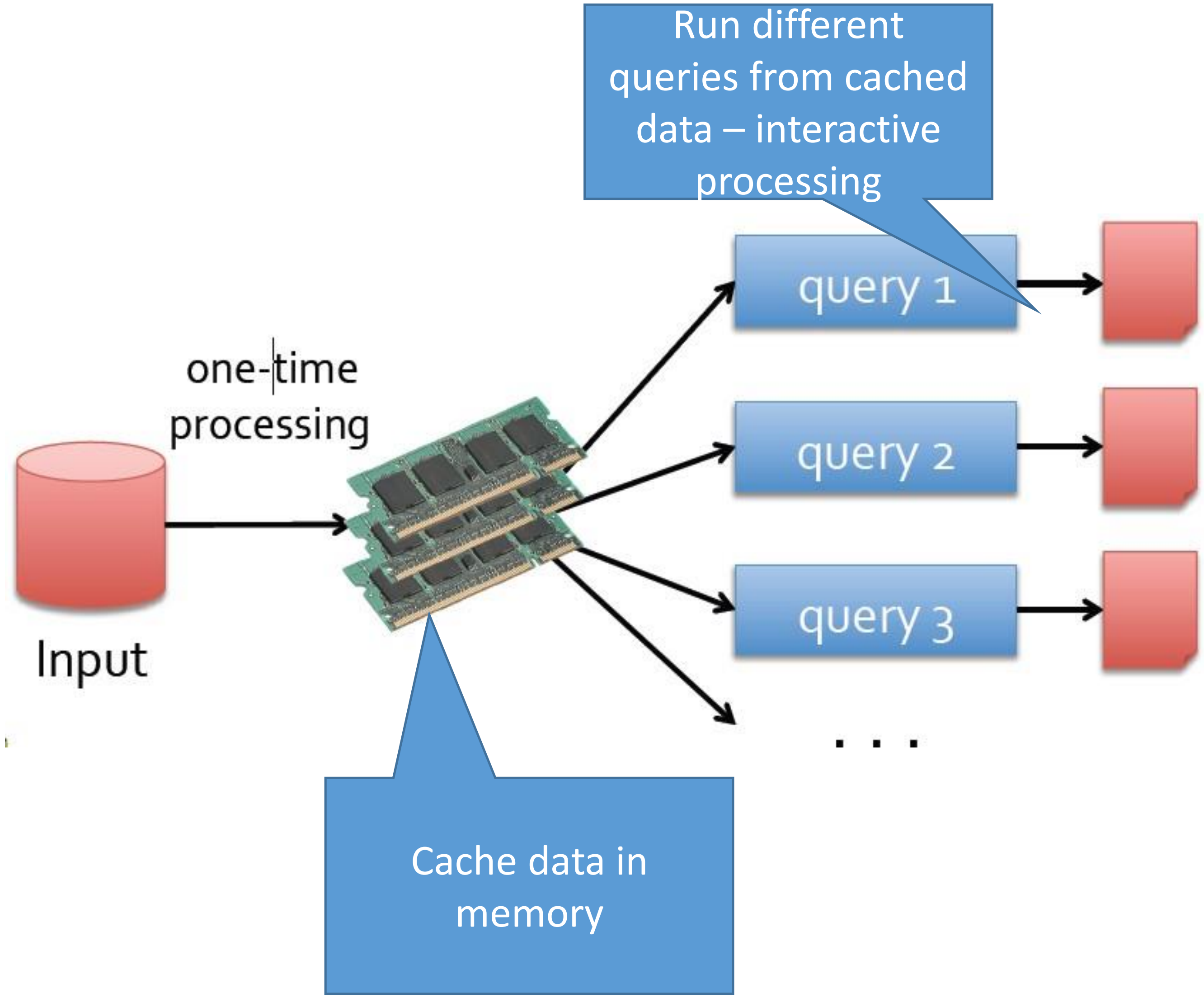
BIG DATA

Doing in memory processing – Iterative processing



BIG DATA

Doing in memory processing – interactive processing



Challenges of in memory processing

- How do we distribute the data among the DRAM of the cluster?

- What happens if this memory is not sufficient?

- How do we handle failures because memory is volatile?

Distributed Dataset

Example: Log Processing

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(startsWithERROR())
messages = errors.map(split("\t"),2)
cachedMsgs = messages.cache()

cachedMsgs.filter(containsfoo()).count()
cachedMsgs.filter(containsbar()).count()
. . .
```

Courtesy: Zaharia et al , "Spark: Fast, Interactive, Language-Integrated Cluster Computing", www.spark-project.org

Example: Log Processing

Loads a file to an in memory struct called an RDD (think of it as a collection of strings)

Filter function to retain only those lines with an error. Creates another RDD

Applies a function to each element(string) in RDD and produces a new RDD

Keep it in memory as it will be reused

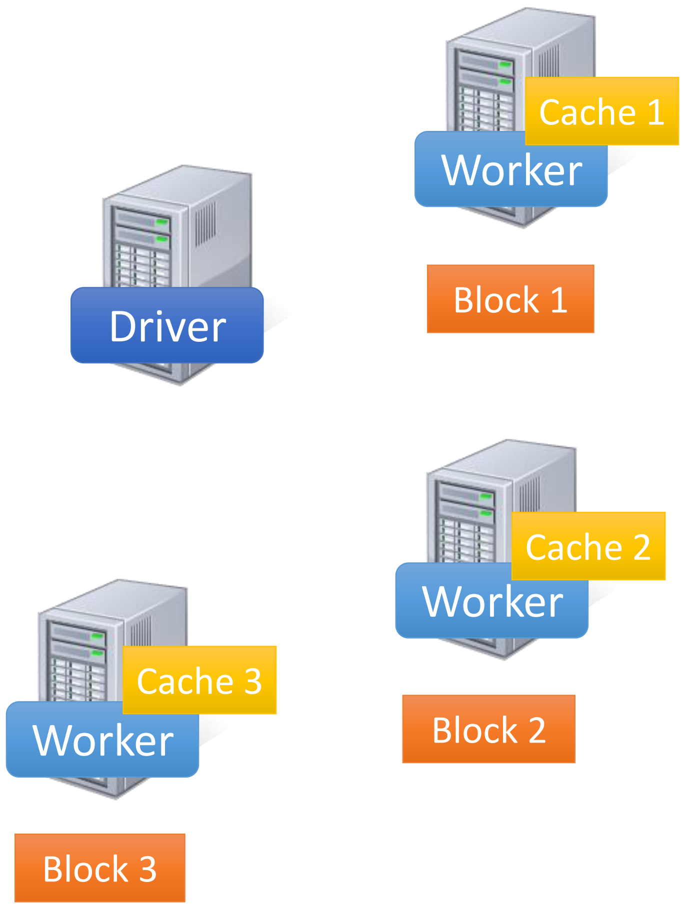
Counts #objects in the RDD

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(startsWithERROR())
messages = errors.map(split("\t"),2)
cachedMsgs = messages.cache()

cachedMsgs.filter(containsfoo()).count()
cachedMsgs.filter(containsbar()).count()
. . .
```

Courtesy: Zaharia et al , "Spark: Fast, Interactive, Language-Integrated Cluster Computing", www.spark-project.org

Java and Scala: Spot the differences

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(startsWithERROR())
messages = errors.map(split("\t"),2)
cachedMsgs = messages.cache()

cachedMsgs.filter(containsfoo()).count()
cachedMsgs.filter(containsbar()).count()
```



Adding fault tolerance – The RDD

Consider the following code:

```
                Step1          Step2
messages = textFile(...).filter(startsWithERROR())
                        .map(split("\t")(2))
                         Step3
```

Step1: Read in the file to an in memory RDD

Step2: remove all lines that don't contain the term ERROR

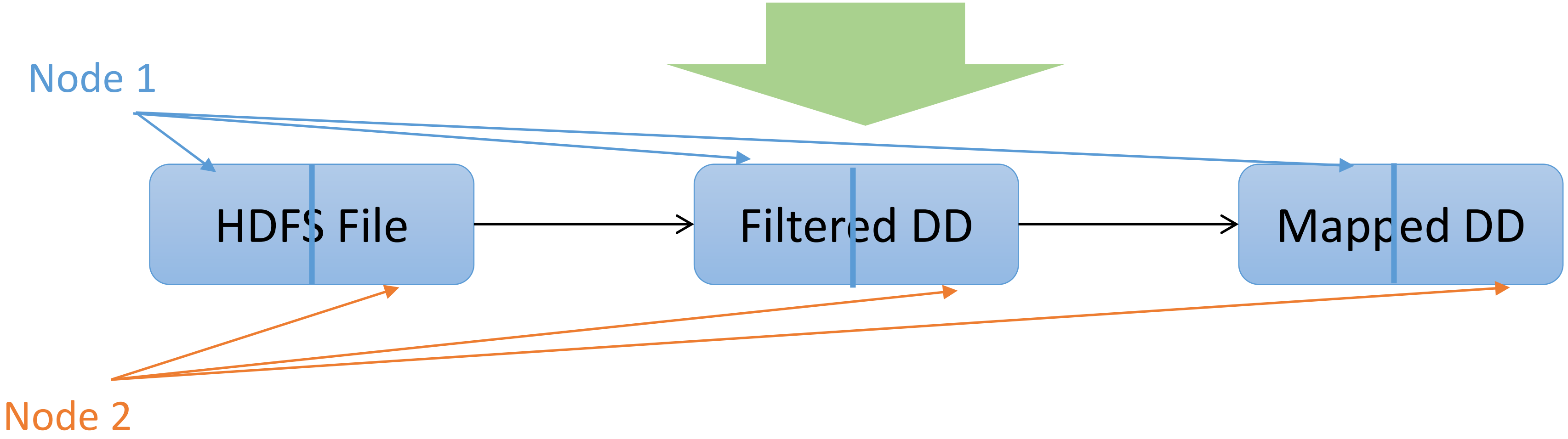
Step3: split the line

map(*func*)	Return a new distributed dataset formed by passing each element of the source through a function *func*.
filter(*func*)	Return a new dataset formed by selecting those elements of the source on which *func* returns true.

Handling fault tolerance

Ex:

```
messages = textFile(...).filter(startsWithERROR())
                        .map(split("\t")(2))
```



What is an RDD?

- When we add lineage information to the concept of a Distributed Dataset
  - We add ability to recreate it in case of failure
  - So, this data is now *resilient* to failures.
  - Hence called an ***RDD:*** *Resilient Distributed Dataset*

How is lineage information stored

- Lineage information is stored by keeping track of
    - Operations that are performed on
        - An RDD
        - That results in another RDD
    - What types of operations are supported?

RDD Operations : Transformations and Actions

Types of Operations

- Operations are of two types
  - ***Transformations***
  - ***Actions***

Transformation

- Are operations that create a *new dataset* from an existing dataset

- For example:
  - *map()* is a transformation

  - Each line on input RDD is passed through the *map()* function
  - result of *map()* function applied on each value is stored in the output RDD.

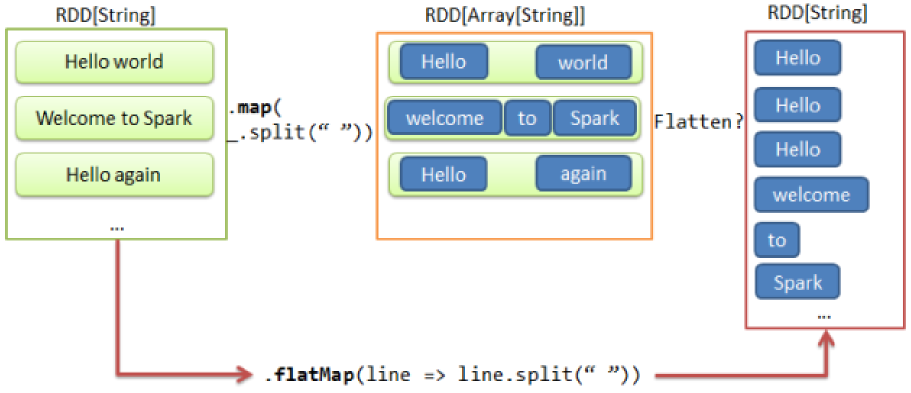
- Note it is similar to the Map of map-reduce, but is more generic.

BIG DATA

Transformations

Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`map()`	Apply a function to each element in the RDD and return an RDD of the result.	`rdd.map(x => x + 1)`	`{2, 3, 4, 4}`
`flatMap()`	Apply a function to each element in the RDD and return an RDD of the contents of the iterators returned. Often used to extract words.	`rdd.flatMap(x => x.to(3))`	`{1, 2, 3, 2, 3, 3, 3}`
`filter()`	Return an RDD consisting of only elements that pass the condition passed to `filter()`.	`rdd.filter(x => x != 1)`	`{2, 3, 3}`
`distinct()`	Remove duplicates.	`rdd.distinct()`	`{1, 2, 3}`
`sample(withReplacement, fraction, [seed])`	Sample an RDD, with or without replacement.	`rdd.sample(false, 0.5)`	Nondeterministic



Transformations

Table 3-3. Two-RDD transformations on RDDs containing {1, 2, 3} and {3, 4, 5}

Function name	Purpose	Example	Result
`union()`	Produce an RDD containing elements from both RDDs.	`rdd.union(other)`	`{1, 2, 3, 3, 4, 5}`
`intersection()`	RDD containing only elements found in both RDDs.	`rdd.intersection(other)`	`{3}`
`subtract()`	Remove the contents of one RDD (e.g., remove training data).	`rdd.subtract(other)`	`{1, 2}`
`cartesian()`	Cartesian product with the other RDD.	`rdd.cartesian(other)`	`{(1, 3), (1, 4), … (3,5)}`

Actions

- Are operations that return a value

- For example:
  - Reduce() is an action
  - Aggregates all elements of a RDD to produce a result.

BIG DATA
Actions

PES UNIVERSITY ONLINE

Table 3-4. Basic actions on an RDD containing {1, 2, 3, 3}

Function name	Purpose	Example	Result
`collect()`	Return all elements from the RDD.	`rdd.collect()`	`{1, 2, 3, 3}`
`count()`	Number of elements in the RDD.	`rdd.count()`	4
`countByValue()`	Number of times each element occurs in the RDD.	`rdd.countByValue()`	`{(1, 1), (2, 1), (3, 2)}`
`take(num)`	Return `num` elements from the RDD.	`rdd.take(2)`	`{1, 2}`
`top(num)`	Return the top `num` elements the RDD.	`rdd.top(2)`	`{3, 3}`
`takeOrdered(num)(ordering)`	Return `num` elements based on provided ordering.	`rdd.takeOrdered(2)(myOrdering)`	`{3, 3}`
`takeSample(withReplacement, num, [seed])`	Return `num` elements at random.	`rdd.takeSample(false, 1)`	Nondeterministic
`reduce(func)`	Combine the elements of the RDD together in parallel (e.g., `sum`).	`rdd.reduce((x, y) => x + y)`	9
`fold(zero)(func)`	Same as `reduce()` but with the provided zero value.	`rdd.fold(0)((x, y) => x + y)`	9

RDD Operations : Working with key-value pairs

Key Value Pairs

- Consider our earlier operation of map/reduce using Spark
- Worked on datasets with only single values
- Let’s consider how to represent *<key, value>* pairs
- Spark provides
  - Separate RDDs called pair RDDs for this
  - Separate operations to function on Pair RDDs

BIG DATA

Pair RDD Transformations

Table 4-1. Transformations on one pair RDD (example: {(1, 2), (3, 4), (3, 6)})

Function name	Purpose	Example	Result
`reduceByKey(func)`	Combine values with the same key.	`rdd.reduceByKey((x, y) => x + y)`	`{(1, 2), (3, 10)}`
`groupByKey()`	Group values with the same key.	`rdd.groupByKey()`	`{(1, [2]), (3, [4, 6])}`
`combineByKey(createCombiner, mergeValue, mergeCombiners, partitioner)`	Combine values with the same key using a different result type.	See Examples 4-12 through 4-14.	

Pair RDD Transformations

`mapValues(func)`	Apply a function to each value of a pair RDD without changing the key.	`rdd.mapValues(x => x+1)`	`{(1, 3), (3, 5), (3, 7)}`
`flatMapValues(func)`	Apply a function that returns an iterator to each value of a pair RDD, and for each element returned, produce a key/value entry with the old key. Often used for tokenization.	`rdd.flatMapValues(x => (x to 5)`	`{(1, 2), (1, 3), (1, 4), (1, 5), (3, 4), (3, 5)}`
`keys()`	Return an RDD of just the keys.	`rdd.keys()`	`{1, 3, 3}`
`values()`	Return an RDD of just the values.	`rdd.values()`	`{2, 4, 6}`

Pair RDD Actions

countByKey(k, V) → returns a HashMap of (k, Int) key value pairs
with count of each key

Word Count in Spark

Word Count in Spark

Create a spark context: tell Spark to create a new job

```
val sc = new SparkContext(new SparkConf().setAppName("Spark Count"))

val tokenized = sc.textFile(args(0)).flatMap(_.split(" "))

val wordCounts = tokenized.map((_, 1)).reduceByKey(_ + _)
```

Read in text file
Split it into words

Each of these is an RDD

Map each word to 1

Reduce by key. Can also use countbykey

https://docs.cloudera.com/documentation/enterprise/5-13-x/topics/spark_develop_run.html

Additional References

- What is Apache Spark? Matei Zaharia
  - https://www.youtube.com/watch?v=p8FGC49N-zM
- RDD, DataFrames and Datasets
  - https://www.youtube.com/watch?v=pZQsDloGB4w

THANK YOU

K V Subramaniam, Usha Devi
Dept. of Computer Science and Engineering

subramaniamkv@pes.edu
ushadevibg@pes.edu