Hortonworks

HDP Developer: Apache Pig and Hive

Hortonworks. We do Hadoop.

Revision 4

Hortonworks

Introducing Apache Spark

Topics Covered

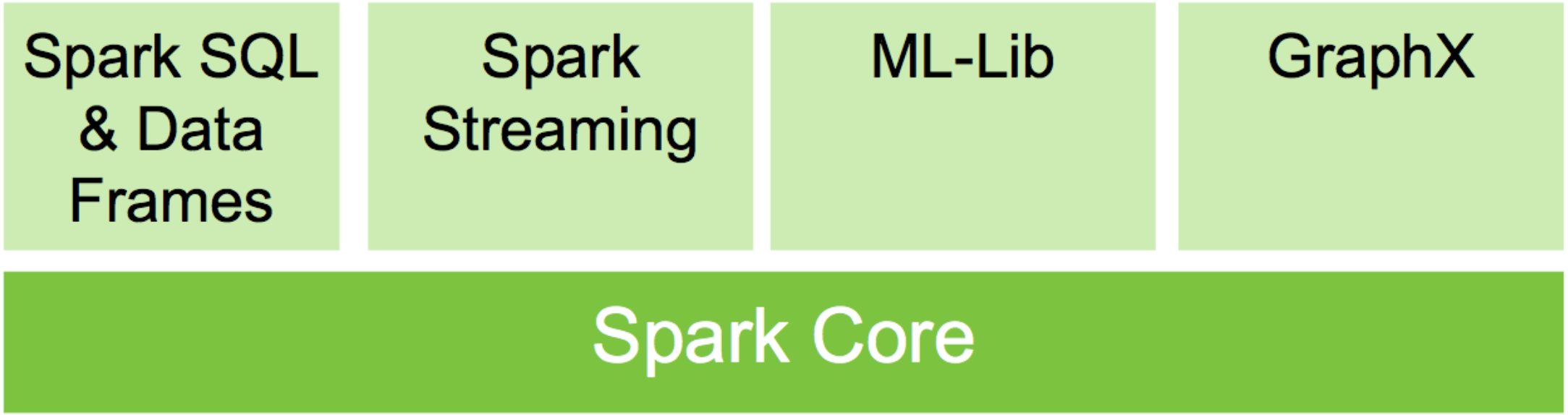
- The origin of Apache Spark
- Rapid rate of growth of the Spark ecosystem
- Spark use cases
- Major differences between Spark and MapReduce

Hortonworks

What is Apache Spark?

- Apache open source project, originally developed at AmpLab at UC-Berkeley
  - 2009: Research project; BDAS (Berkley Data Analysis Stack)
  - Jun 2013: Accepted into Apache Incubator
  - Feb 2014: Became a top-level Apache project
  - Dec 2014: Included in HDP 2.2
- A general data processing engine, focused on in-memory distributed computing use-cases
- APIs in Scala, Python and Java
  - Recently API for R was introduced

The Spark ecosystem



Hortonworks

Why Spark?

- Elegant Developer APIs: Data Frames/SQL, Machine Learning, Graph algorithms and streaming
  - Scala, Python, Java and R
  - Single environment for importing, transforming, and exporting data
- In-memory computation model
  - Effective for iterative computations
- High level API
  - Allows users to focus on the business logic and not internals

Why Spark cont.

- Supports wide variety of workloads
  - Mllib for Data Scientists
  - Spark SQL for Data Analysts
  - Spark Streaming for micro batch use cases
  - Spark Core, SQL, Streaming, Mllib, and GraphX for Data Processing Applications
- Integrated fully with Hadoop and an open source tool
- Faster than MapReduce

Who uses Spark!?

- NASA JPL
  - Deep Space Network
- eBay
  - Analysts are clustering sellers together
- Conviva
  - Video stream health statistics
- Yahoo
  - News story personalization

Hortonworks

Spark vs MapReduce

- Higher level API
- In-memory data storage
  - Up to 100x performance improvement

pyspark

```
text_file = spark.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
             .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

Java MapReduce

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
 }

 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
      throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
    }
 }

 public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
 }

}
```


Hortonworks

Spark vs MapReduce Cont

- Why is Spark faster?
  - Caching data to memory can avoid extra reads from disk
  - Scheduling of tasks from 15-20s to 15-20ms
  - Resources are dedicated the entire life of the application
  - Can link multiple maps and reduces together without having to write intermediate data to HDFS
  - Every reduce doesn't require a map

Spark Growth is Massive

- One of the largest open source projects
  - Last release had over 1000 commits and 230 developers contributing
- On average release a .X version every 3 months
- Currently at spark 1.5.2 (Nov 2015)
  - Mar 2015 – Spark SQL Dataframes Release (v1.3)
  - Dec 2014 – Spark Streaming on Python Released (v1.2)

Spark and HDP

- HDP 2.3.2 – Spark 1.4.1
- HDP 2.2.8 – Spark 1.3.1
- HDP 2.2.4 – Spark 1.2.1

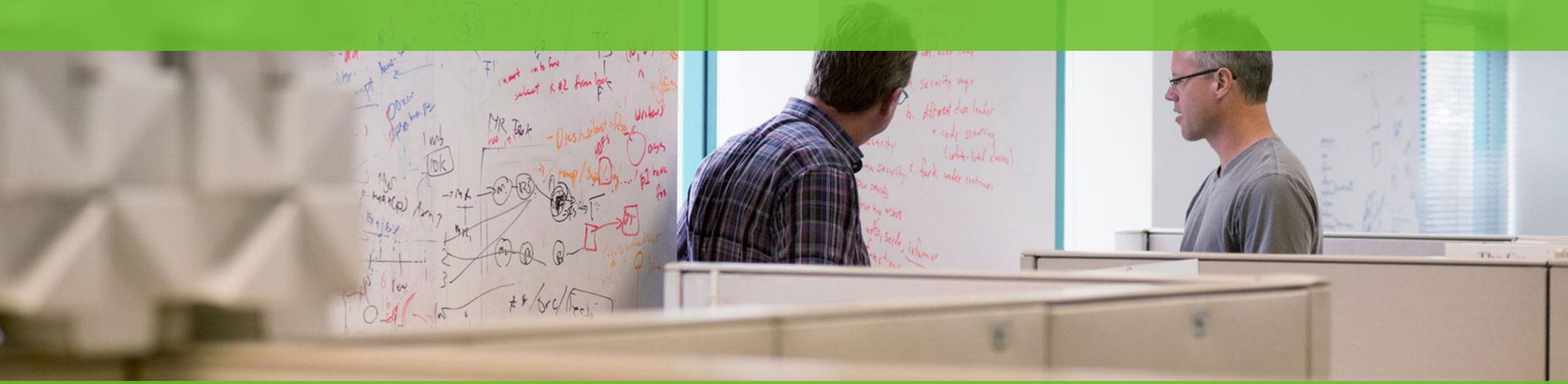
Hortonworks

Lesson Review

1. **What are some of the reasons Spark is faster than MR?**
2. **What distribution of HDP has Spark 1.4.1?**
3. **What are the four libraries that build on Spark Core?**
4. **Name another benefit to using Spark vs MR.**

Hortonworks

Programming with Apache Spark



Hortonworks

Topics Covered

- Starting the spark shell
- Understanding what an RDD is
- Loading data from the HDFS and perform a word count
- The differences between Transformation and Action
- Lazy Evaluation
- *Lab: Getting Started with Apache Spark*

How to start using Apache Spark?

- The Spark Shell provides an interactive way to learn Spark, explore data, and debug applications
- Available for python and scala
  - pyspark
  - spark-shell
- REPL

The SparkContext

- Main entry point for Spark applications
- All Spark applications require one
- The SparkContext has a few responsibilities
  - Represent the connection to a Cluster
  - Used to create RDDs, accumulator and broadcast variables on the cluster
- The REPLs automatically create one for you
  - In Spark 1.3 and on, the shell creates a SQL context too

Working with the Spark Context

Attributes:

- sc.appName: Spark application name
- sc.master: Spark Master (local, yarn-client, etc)
- sc.version: Version of Spark being used

Functions:

- sc.parallelize(): create an RDD from local data
- sc.textFile(): create RDD from a text file in HDFS
- sc.stop(): stop the spark context

The Resilient Distributed Dataset

- An *Immutable* collection of objects (or records) that can be operated on in parallel
  - **Resilient:** can be recreated from parent RDDs - An RDD keeps its lineage information
  - **Distributed:** partitions of data are distributed across nodes in the cluster
  - **Dataset**: a set of data that can be accessed
  - Each RDD is composed of 1 or more partitions - The user can control the number of partitions - More partitions => more parallelism

Create an RDD

- Load data from a file (HDFS, S3, Local, etc)
  - From a single file

```
rdd1 = sc.textFile(“file:/path/to/file.txt”)
rdd2 = sc.textFile(“hdfs://namenode:8020/mydata/data.txt”)
```

  - Also accepts a comma separated list of files, or a wildcard list of files

```
rdd3 = sc.textFile(“mydata/*.txt”)
rdd4 = sc.textFile(“data1.txt,data2.txt”)
```

Create an RDD

- With parallelize() function in driver – useful for learning Spark, distributing local collections of data

```
rdd5 = sc.parallelize([1, 2, 3, 4, 5])
rdd6 = sc.parallelize([“cat”, “dog”, “mouse”])

mydata = (“lets try this”)
rdd7 = sc.parallelize([mydata])
```


Hortonworks

Working with RDDs and Lazy Evaluation

- RDDs have two types of operations
  - Transformations: the RDD is transformed into a new RDD
  - Actions: an action is performed on the RDD and a result is returned to the driver, or data is saved somewhere
- Transformations are lazy: they do not compute until an action is performed

What does “Lazy Execution” mean?

```
file = sc.textFile("hdfs://some-text-file”)
counts = file.flatMap(lambda line: line.split(" ")) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b)

counts.saveAsTextFile("hdfs://wordcount-out”)
```

DAG of transformations is built by Spark on driver side

Action triggers execution of whole DAG

Hortonworks

Spark Uses Functional Programming

- Program built on Functions instead of Objects
- Mutation is forbidden – all variables are final
- Functional purity – if you pass *A* into a function, you're always getting back *B*
- Functions have input and output only – no state or side effects
- Passing functions as input to other functions
- Anonymous Functions – undefined functions passed inline

Actions – count()

- The count() action returns the number of elements in the RDD

```
data = [5, 12, -4 , 7, 20]
rdd = sc.parallelize(data)
rdd.count()
```

The output is: 5

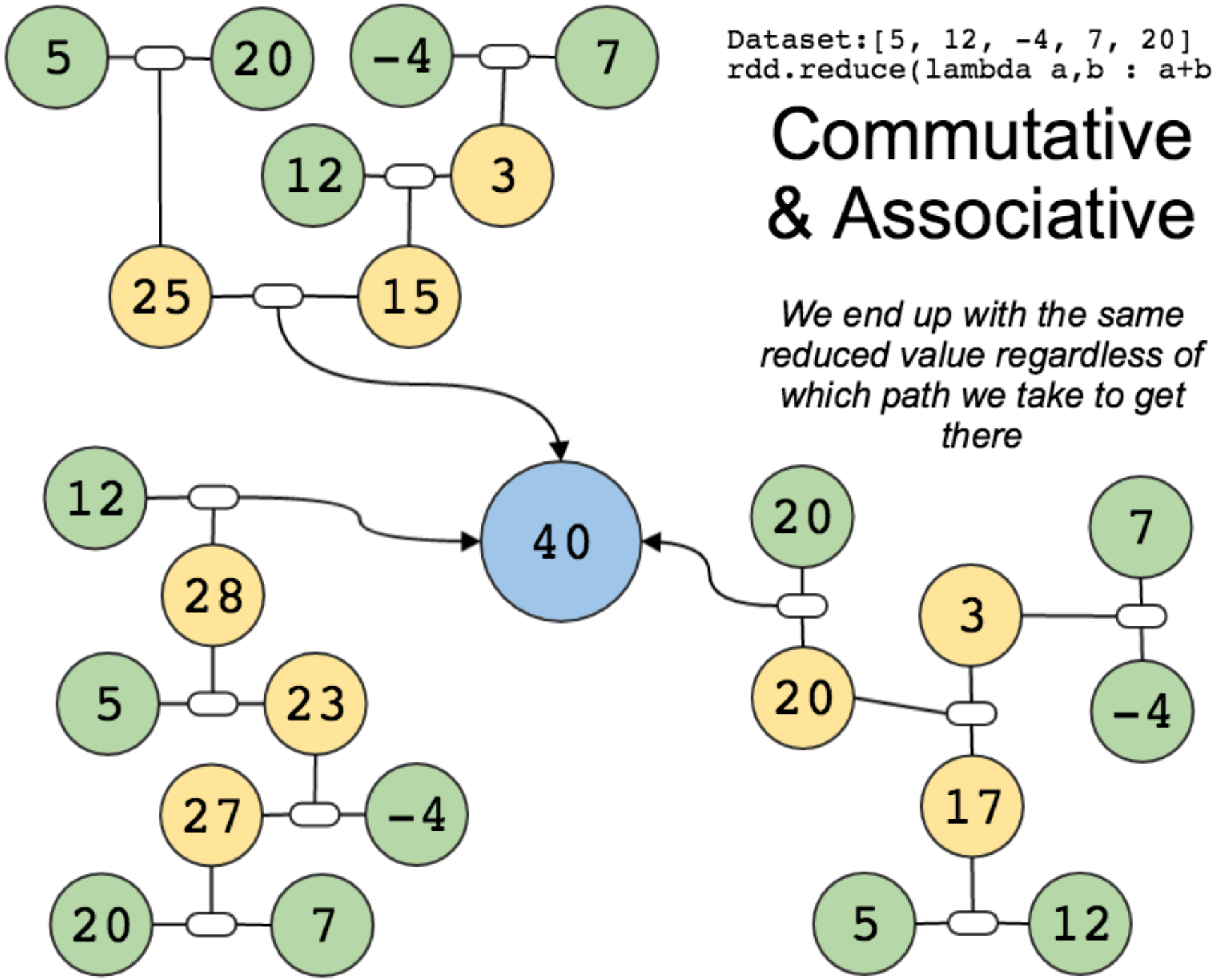
Hortonworks

Actions – reduce()

- The reduce() action has a lot of use cases in Spark
  - Aggregating elements of an RDD using a defined function
  - That function must be commutative and associative
    - a+b = b+a and a+(b+c)=(a+b)+c

```
Dataset:[5, 12, -4 , 7, 20]
rdd.reduce(lambda a, b : a+b)
40

rdd.reduce(lambda a, b: a if (a>b) else b)
20
```




```
Dataset:[5, 12, -4, 7, 20]
rdd.reduce(lambda a,b : a+b)
```

Commutative & Associative

We end up with the same reduced value regardless of which path we take to get there

Other Useful Spark Actions

- first(): return the first element in the RDD
- take(n): return the first n elements of the RDD
- collect(): return all the elements in the RDD to the driver
  - Make sure you only call this on small datasets or risk crashing your driver!
- saveAsTextFile(path): write the RDD to a file

Spark Actions: Examples

```
Dataset:[5, 12, -4 , 7, 20]

rdd.first(): 5

rdd.take(3): [5, 12, -4]

rdd.saveAsTextFile(“myfile”)
```


Hortonworks

Spark Transformations

- Spark Transformations create new RDD's from existing ones
- The transformation is lazy, and processing doesn't occur until an action is called on the RDD, or subsequent RDD
  - Transformation create a recipe, or lineage, for processing
  - The actions trigger data to flow through the transformation and create the result

Transformations: map()

- Map applies a function to each element of the RDD (provides a one input to one output)

```
rdd=sc.parallelize([1, 2, 3, 4, 5])

rdd.map(lambda x: x*2+1).collect()

[3, 5, 7, 9, 11]
```


Hortonworks

Transformations: flatMap()

- Map applies a function to each element of the RDD and returns a collection (provides a one input to many output)

```
rdd=sc.parallelize([1, 2, 3, 4, 5])

rdd.map(lambda x: [x, x*2]).collect()
[(1,2), (2, 4), (3,6), (4,8), (5,10)]

rdd.flatMap(lambda x: [x, x*2]).collect()
[1, 2, 2, 4, 3, 6, 4, 8, 5, 10]
```

Transformation: filter()

- Keep some elements based on a predicate

```
rdd=sc.parallelize([1, 2, 3, 4, 5])

rdd.filter( lambda x:  x%2 == 0).collect()
[2, 4]

rdd.filter( lambda x: x<3).collect()
[1, 2]
```

Key Value Pair Intro (Pair RDDs)

- A Key/Value RDD is an RDD whose elements comprise a pair of values – key and value

- Pair-RDDs are very useful for many applications
  - Allow to group operations by key
  - Examples
    - join()
    - groupByKey()
    - reduceByKey()

Hortonworks

Creating Pair RDDs

- Pair RDDs are often created from regular RDDs by using the map() or flatMap() transformation:

```
wordlist = ‘this is my list and it is a nice list’
rdd1 = sc.parallelize([wordlist])
kv_rdd = rdd1.flatMap(lambda x: x.split(‘ ‘)). \
       .map(lambda x: (x,1))
kv_rdd.collect()
[(this, 1), (is, 1), (my, 1), (list, 1), (and, 1), … (list,1)]
```

Pair RDD Action: reduceByKey()

- reduceByKey() performs a reduce function on all elements of a key/value pair RDD that share a key
  - The function still must be commutative and associative
    - a+b = b+a and a+(b+c)=(a+b)+c

```
kv_rdd.reduceByKey(lambda a,b: a+b).collect()
[('this', 1), ('my', 1), ('and', 1), ('list', 2), ('a', 1), ('it', 1),
 ('is', 2), ('nice', 1)]
```


Hortonworks

Keys & *Values* Can Contain Rich Tuples

```
>>> notSimplePair = sc.parallelize(['I do not like green eggs and ham I do
not like them Sam I am']).flatMap(lambda sent: sent.split(' ')).map(lambda
word: ((word,'bogus'),('notCount',1)))
>>> notSimplePair.sortByKey(ascending=False).take(3)
[(('them', 'bogus'), ('notCount', 1)), (('not', 'bogus'), ('notCount', 1)),
(('not', 'bogus'), ('notCount', 1))]
>>>
>>> notSimplePair.reduceByKey(lambda oneVal,anotherVal:
('noise',oneVal[1]+anotherVal[1])).sortByKey(ascending=False).collect()
[(('them', 'bogus'), ('notCount', 1)), (('not', 'bogus'), ('noise', 2)),
(('like', 'bogus'), ('noise', 2)), (('ham', 'bogus'), ('notCount', 1)),
(('green', 'bogus'), ('notCount', 1)), (('eggs', 'bogus'), ('notCount', 1)),
(('do', 'bogus'), ('noise', 2)), (('and', 'bogus'), ('notCount', 1)), (('am',
'bogus'), ('notCount', 1)), (('Sam', 'bogus'), ('notCount', 1)), (('I',
'bogus'), ('noise', 3))]
```


Hortonworks

Tips for Navigating Within pyspark

- Take advantage of command history with “up arrow” key & add operations one at a time leveraging `take()`

- Use `dir()` to get a list of current variables
  - Like with Pig’s aliases command, there will be additional system-oriented variable names present

- Use `sc.setLogLevel('WARN')` to limit extra “noise”
  - Looses some visibility to helpful `INFO` messages at time

Lesson Review

1. **What are the three ways we can create an RDD?**
2. **What are the two types of operations we can perform on an RDD?**
    1. **Give an example of each**
3. **What is functional programming?**
4. **What is Lazy Execution?**
5. **What does the R stand for in RDD? What does that mean?**

Hortonworks

Conclusion and Key Points

- There are two* types of operations
  - Transformation which returns a new RDD
  - Action which returns a result
- Spark uses functional programming to process data
- Spark is lazy, it only does work when it has too
- RDD's are in your mind
  - They're just a set of directions to transform data, the data is never stored in the RDD

Lab: Getting Started with Apache Spark

TRAINING

Build your career with Hadoop

Real world training designed by the core architects of Hadoop.

Thank you!

training.hortonworks.com