**Apache Spark**

Session 4 - Concepts

www.KnowBigData.com

WELCOME - KNOWBIGDATA

❏ Expert Instructors

❏ CloudLabs

❏ Lifetime access to LMS

❏ Presentations

❏ Class Recording

❏ Assignments + Quizzes

❏ Project Work

❏ Real Life Project

❏ Course Completion Certificate

❏ 24x7 support

❏ KnowBigData - Alumni

❏ Jobs

❏ Stay Abreast (Updated Content,

Complimentary Sessions)

❏ Stay Connected

www.KnowBigData.com

COURSE CONTENT

I Introduction to Big Data with Apache Spark II Downloading Spark and Getting Started III Programming with RDDs IV Working with Key/Value Pairs V Loading and Saving Your Data VI Advanced Spark Programming VII Running on a Cluster VIII Tuning and Debugging Spark IX Spark SQL, SparkR X Spark Streaming XI Machine Learning with MLlib, GraphX

www.KnowBigData.com

About Instructor?

**2014 KnowBigData Founded 2014**

2012

**Amazon**

Built High Throughput Systems for Amazon.com site using in-house NoSql.

2012 InMobi Built Recommender that churns 200 TB 2011

**tBits Global**

Founded tBits Global Built an enterprise grade Document Management System

2006

**D.E.Shaw**

Built the big data systems before the term was coined 2002 2002 IIT Roorkee Finished B.Tech.

www.KnowBigData.com

Starting Spark With Python Interactive Shell

$ cd spark-1.5.0-bin-hadoop2.4 $ bin/pyspark

It is basically the python interactive shell with one extra variable “sc”. Check dir(sc) or help(sc)

www.KnowBigData.com

Starting Spark With Python Job Submit

$ cd cd spark-1.5.0-bin-hadoop2.4 $ bin/spark-submit ../myprog.py

www.KnowBigData.com

Submitting a java job

1. Download the project from LMS 2. Create fresh workspace inside eclipse 3. import the project 4. Fix lib path of spark jar

a. Go to Project properties b. Under libraries: remove missing file and add external jar 5. Export jar 6. scp ~/Downloads/wc\* spark1@hadoop3.knowbigdata.com: 7. cd spark 8. bin/spark-submit --class sparkex.WordCount ../wc-p1.jar

www.KnowBigData.com

Getting started with Scala Interactive Shell

$ cd cd spark-1.5.0-bin-hadoop2.4 $ bin/spark-shell

www.KnowBigData.com

RDDs - Resilient Distributed Datasets

*Dataset: Collection of data elements. e.g. Array, Tables, Data frame (R), collections of mongodb*

**What is RDD?**

*Distributed: Parts Multiple machines*

*Resilient: Recovers on Failure*

www.KnowBigData.com

SPARK - CONCEPTS - RESILIENT DISTRIBUTED DATASET

**A collection of elements partitioned across cluster**

Machine 1 Machine 2 Machine 3 Machine 4

www.KnowBigData.com

SPARK - CONCEPTS - RESILIENT DISTRIBUTED DATASET

**A collection of elements partitioned across cluster**

• An immutable distributed collection of objects.

• Split in partitions which may be on multiple nodes

• Can contain any data type:

○ Python,

○ Java,

○ Scala objects

○ including user defined classes

www.KnowBigData.com

SPARK - CONCEPTS - RESILIENT DISTRIBUTED DATASET

• RDD Can be persisted in memory

• RDD Auto recover from node failures

• Can have any data type but has a special dataset type for key-value

• Supports two type of operations: transformation and action

• Each Element of RDD across cluster is run through map function

www.KnowBigData.com

Creating RDD

**Method 1: By Directly Loading a file from remote**

>>lines = sc.textFile(

'hdfs://hadoop1.knowbigdata.com/data/mr/wordcount/input/big.txt'

**Method 2: By distributing existing object**

>> arr = range(1, 1000000) >> numbers = sc.parallelize(arr)

www.KnowBigData.com

)

Churning RDD

Two Kinds Operations

Transformation Action

www.KnowBigData.com

**• Transformations are operations on RDDs**

**• return a new RDD**

**• such as map() and filter()**

RDD - Operations : Transformation

www.KnowBigData.com

Transformation Example

➢ arr = range(1, 1000000) ➢ nums = sc.parallelize(arr) ➢ def multipleByTwo(x):

return x\*2;

***➢ dbls = nums.map (multipleByTwo); ➢ dbls.take(5) ➢ [2, 4, 6, 8, 10]***

www.KnowBigData.com

Transformation Example (2)

➢ words = sc.parallelize(['hi', 'how', 'are', 'you']) ➢ def toTuple(word):

return (word, 1); ➢ tuples = words.map(toTuple)

*[('hi', 1), ('how', 1), ('are', 1), ('you', 1)]*

hi how are you

toTuple() toTuple() toTuple() toTuple()

(hi,1) (how,1) (are,1) (you,1)

www.KnowBigData.com

RDD - Operations : Actions

**• Brings back the data to driver**

**• Causes the full execution of transformations**

**• Involves both spark driver as well as the nodes**

www.KnowBigData.com

Action Example

➢ arr = range(1, 1000000) ➢ nums = sc.parallelize(arr) ➢ def multipleByTwo(x):

return x\*2;

➢ dbls = nums.map (multipleByTwo); ➢ dbls.take(4) ➢ [2, 4, 6, 8]

www.KnowBigData.com

Transformations - filter()

➢ arr = range(1, 1000000) ➢ nums = sc.parallelize(arr) ➢ def isEven(x):

return x%2 == 0;

nums

**1 2 3 4 5 6 7**

**isEven() isEven() isEven()**

**isEven()**

**isEven() isEven() isEven()**

evens

**2 4 6**

***➢ evens = nums.filter(isEven); ➢ evens.take(3) ➢ [2, 4, 6]***

www.KnowBigData.com

**linesRDD**

**wordsRDD**

Transformations:: flatMap()

➢ linesRDD = sc.parallelize( ["this is a dog", "named jerry"]) ➢ def toWords(line):

return lines.split() ➢ wordsRDD = linesRDD.flatMap(lines) ➢ wordsRDD.collect() ➢ ['this', 'is', 'a', 'dog', 'named', 'jerry']

this is a dog named jerry

toWords() toWords()

this is a dog named jerry

www.KnowBigData.com

Transformations:: Union

➢ a = sc.parallelize([1,2,3]); ➢ b = sc.parallelize(['A','B','C']); ➢ c=a.union(b) ➢ c.collect();

*[1, 2, 3, 'A', 'B', 'C']*

[1, 2, 3] ['A','B','C'])

**Union**

[1, 2, 3, 'A','B','C']]

www.KnowBigData.com

Transformations:: union()

RDD lineage graph created during log analysis

www.KnowBigData.com

Actions: collect()

Brings all the elements back to you. Data must fit into memory. Mostly it is impractical.

➢ a = sc.parallelize([1,2,3, 4, 5 , 6, 7]); ➢ a

ParallelCollectionRDD[3] at parallelize at PythonRDD.scala:391 ➢ localarray = a.collect(); ➢ localarray

***[1, 2, 3, 4, 5, 6, 7]***

**1 2 3 4 5 6 7**

www.KnowBigData.com

Actions: take()

Bring only few elements to the driver. This is more practical than collect()

***➢ a = sc.parallelize([1,2,3, 4, 5 , 6, 7]); ➢ localarray = a.take(4); ➢ localarray [1, 2, 3, 4]***

**1 2 3 4 5 6 7**

www.KnowBigData.com

Actions: count()

➢ a = sc.parallelize([1,2,3, 4, 5 , 6, 7]); ➢ mycount = a.count(); ➢ mycount

***7***

**1, 2, 3 4,5 6,7**

3

2

2

3+ 2 + 2 = 7

www.KnowBigData.com

Actions: Lazy Evaluation

1. Every time we call an action, entire RDD must be computed from scratch 2. Everytime d gets executed, a,b,c would be run

a. lines = sc.textFile("myfile"); b. fewlines = lines.filter(...) c. uppercaselines = fewlines.map(...) d. uppercaselines.count() 3. When we call a transformation, it is not evaluated immediately. 4. It helps Spark optimize the performance 5. Similar to Pig, LinQ etc. 6. Instead of thinking RDD as dataset, think of it as the instruction on how to

compute data

def Map1(x):

return x.strip(); def Map2(x):

return upper(x);

www.KnowBigData.com

Actions: Lazy Evaluation - Optimization

def Map1(x):

return x.strip();

def Map2(x):

return upper(x);

lines = sc.textFile(...) lines1 = lines.map(Map1); lines2 = lines1.map(Map1);

dump lines2

def Map(x):

val = x.strip(); return upper(val);

lines = sc.textFile(...) lines2 = lines.map(Map); dump lines2

www.KnowBigData.com

Function Passing - Python

**#Method1: Inline Functions #Good for smaller functions**

*word = rdd.filter(lambda s: "error" in s)*

**#Method2: Defining a function or using a global function**

*def containsError(s): return "error" in s*

*word = rdd.filter(containsError)*

www.KnowBigData.com

Function Passing - Python (pitfalls)

**#Avoid passing instance functions. it sends entire object to workers**

**# Instead only pass the local variables**

www.KnowBigData.com

Function Passing - Scala

www.KnowBigData.com

Function Passing - Java

Passing Java Function with 1. Anonymous inner class 2. Named class 3. Lambda expression in Java 8

www.KnowBigData.com

Function Passing - Java

1. Java function passing with anonymous inner class

www.KnowBigData.com

Function Passing - Java

2. Java function passing with named class

www.KnowBigData.com

Function Passing - Java

3. Java function passing with lambda expression in Java 8

www.KnowBigData.com

Common Transformations and Actions (continued..)

**sample(withReplacement, fraction, [seed])**

Sample an RDD, with or without replacement.

seq = sc.parallelize(range(1,100)) seq.sample(False, 0.1).collect();

*[8, 19, 34, 37, 43, 51, 70, 83]*

seq.sample(True, 0.1).collect();

*[14, 26, 40, 47, 55, 67, 69, 69]*

Please note that the result will be different on every run.

www.KnowBigData.com

Common Transformations (continued..)

**mapPartitions(f, preservesPartitioning=False)**

Return a new RDD by applying a function to each partition of this RDD.

rdd = sc.parallelize([1, 2, 3, 4], 2) def f(iterator): yield sum(iterator) rdd.mapPartitions(f).collect()

*[3, 7]*

www.KnowBigData.com

Common Transformations (continued..)

**sortBy(keyfunc, ascending=True, numPartitions=None)**

Sorts this RDD by the given keyfunc

*⋙ tmp = [('a', 1), ('b', 2), ('1', 3), ('d', 4), ('2', 5)]*

*⋙ sc.parallelize(tmp).sortBy(lambda x: x[0]).collect()*

*[('1', 3), ('2', 5), ('a', 1), ('b', 2), ('d', 4)]*

*⋙ sc.parallelize(tmp).sortBy(lambda x: x[1]).collect()*

*[('a', 1), ('b', 2), ('1', 3), ('d', 4), ('2', 5)]*

www.KnowBigData.com

Common Transformations (continued..)

Pseudo set operations

Though RDD is not really sets but still the set operations try to provide you utility set functions

www.KnowBigData.com

Set operations (Pseudo)

**distinct()**

+ Give the set property to your rdd + Expensive as shuffling is required

**union()**

+ Simply appends one rdd to another + Is not same as mathematical function + It may have duplicates

**intersection()**

+ Finds common values in RDDs + Also removes duplicates + Requires shuffling

**subtract()**

+ Returns values in first RDD and not second + Requires Shuffling like intersection()

www.KnowBigData.com

Set operations (Pseudo)

**cartesian()**

+ Returns all possible pairs of (a,b) + a is in source RDD and b is in other RDD

www.KnowBigData.com

Questions - Set Operations

What will be the result of the following?

*a = sc.parallelize(['a','a','b','c']) b = a.distinct() b.collect()*

www.KnowBigData.com

Questions - Set Operations

What will be the result of the following?

*a = sc.parallelize(['a','a','b','c']) b = a.distinct() b.collect()*

***['a', 'c', 'b']***

www.KnowBigData.com

Questions - Set Operations

What will be the result of the following?

*a = sc.parallelize(['a','a','b','c']) b = a.distinct() a1 = sc.parallelize(['a','d']); c = a1.intersection(b);*

***c.collect();***

www.KnowBigData.com

Questions - Set Operations

What will be the result of the following?

*a = sc.parallelize(['a','a','b','c']) b = a.distinct() a1 = sc.parallelize(['a','d']); c = a1.intersection(b);*

***c.collect();***

***['a']***

www.KnowBigData.com

Questions - Set Operations

What will be the result of the following?

*a = sc.parallelize(['a','a','b','c']) b = a.distinct() a1 = sc.parallelize(['a','d']); d = b.subtract(a1) d.collect();*

www.KnowBigData.com

Questions - Set Operations

What will be the result of the following?

*a = sc.parallelize(['a','a','b','c']) b = a.distinct() a1 = sc.parallelize(['a','d']); d = b.subtract(a1) d.collect();*

***['c', 'b']***

www.KnowBigData.com

Questions - Set Operations

What will be the result of the following?

*a = sc.parallelize(['a','a','b','c']) b = a.distinct() a1 = sc.parallelize(['a','d']); u = a.union(a1) u.collect();*

www.KnowBigData.com

Questions - Set Operations

What will be the result of the following?

*a = sc.parallelize(['a','a','b','c']) b = a.distinct() a1 = sc.parallelize(['a','d']); u = a.union(a1) u.collect();*

***['a', 'a', 'b', 'c', 'a', 'd']***

www.KnowBigData.com

Questions - Set Operations

What will be the result of the following?

*a = sc.parallelize(['a','a','b','c']) b = a.distinct() a1 = sc.parallelize(['a','d']); v = b.union(a1) v.collect();*

www.KnowBigData.com

Questions - Set Operations

What will be the result of the following?

*a = sc.parallelize(['a','a','b','c']) b = a.distinct() a1 = sc.parallelize(['a','d']); v = b.union(a1) v.collect();*

***['a', 'c', 'b', 'a', 'd']***

www.KnowBigData.com

Questions - Set Operations

What will be the result of the following?

*noun = sc.parallelize(['boy', 'girl']) adj = sc.parallelize(['good', 'bad']) result = adj.cartesian(noun) result.collect()*

www.KnowBigData.com

Questions - Set Operations

What will be the result of the following?

*noun = sc.parallelize(['boy', 'girl']) adj = sc.parallelize(['good', 'bad']) result = adj.cartesian(noun) result.collect()*

***[('good', 'boy'), ('good', 'girl'), ('bad', 'boy'), ('bad', 'girl')]***

www.KnowBigData.com

More Actions - Reduce()

reduce(func)

Aggregate elements of dataset using a function:

• Takes 2 arguments and returns only one

•

Commutative and associative for parallelism

• Return type of function has to be same as argument

>>> seq = sc.parallelize(range(1,100)) >>> def sum(x, y): ... return x+y; >>> total = seq.reduce(sum); >>> total 4950

www.KnowBigData.com

REDUCE SUM FUNCTION (Walk through)

//Single Node lines = ["san giri g", "san giri", "giri", "bhagwat kumar", "mr. shashank sharma", "anto"] lineLengths = [11, 9, 4, 14, 20, 4] sum = ???

//Node1 lines = ["san giri g", "san giri", "giri"] lineLengths = [11, 9, 4]

totalLength = [20, 4] totalLength = 24 //sum or min or max or sqrt(a\*a + b\*b)

//Node2 lines = ["bhagwat kumar"] lineLengths = [14] totalLength = 14

//Node3 lines = ["mr. shashank sharma", "anto"] lineLengths = [20, 4] totalLength = 24

//Driver Node lineLengths = [24, 14, 24] lineLength = [38, 24] lineLength = [62]

www.KnowBigData.com

More Actions - fold()

fold(initial value, func)

Aggregate the elements of each partition and then the results for all the partitions using a given associative and commutative function and a neutral "zero value".

seq = sc.parallelize(range(1,100)) def conca(x, y):

if type(x) is list:

y.extend(x) else:

y.append(x); return y; arr = [] v = seq.fold(arr, conca)

www.KnowBigData.com

More Actions - aggregate()

aggregate(initial value, seqOp, combOp)

1. First, all values of each partitions are merged to

Initial value using SeqOp() 2. Second, all partitions result is combined together

using combOp

**1, 2, 3 4,5 6,7**

SeqOp()

SeqOp() SeqOp()

CombOp()

**Output**

www.KnowBigData.com

More Actions - aggregate()

aggregate(initial value, seqOp, combOp)

1. First, all values of each partitions are merged to

Initial value using SeqOp() 2. Second, all partitions result is combined together

using combOp

seq = sc.parallelize(range(1,10)) def seqOp(x, y):

return str(x) + ":" + str(y); def comOp(x, y):

return x + ";" + y; v = seq.aggregate("+", seqOp, comOp)

*'+;+:1:2:3:4;+:5:6:7:8:9'*

www.KnowBigData.com

More Actions: countByValue()

Number of times each element occurs in the RDD.

**1 2 3 3 5 5 5**

rdd = sc.parallelize([1, 2, 3, 3, 5, 5, 5]) dict = rdd.countByValue() dict

**{1: 1, 2: 1, 3: 2, 5: 3}**

www.KnowBigData.com

More Actions: top(n)

Sorts and gets the maximum n values.

**4 5 8 1 2 3 10 9**

a=sc.parallelize([4,5,8,1,2, 3, 10, 9]) a.top(10)

**[10, 9, 8, 5, 4, 3, 2, 1]**

www.KnowBigData.com

More Actions: takordered()

Get the N elements from a RDD ordered in ascending order or as specified by the optional key function.

*>>> sc.parallelize([10, 1, 2, 9, 3, 4, 5, 6, 7]).takeOrdered(6) [1, 2, 3, 4, 5, 6]*

*>>> sc.parallelize([10, 1, 2, 9, 3, 4, 5, 6, 7]).takeOrdered(6, key=lambda x: -x) [10, 9, 7, 6, 5, 4]*

*>>>sc.parallelize([10, 1, 2, 9, 3, 4, 5, 6, 7]).takeOrdered(6, key=lambda x: str(x)) [1, 10, 2, 3, 4, 5]*

www.KnowBigData.com

More Actions: takeSample()

Return a fixed-size sampled subset of this RDD (currently requires numpy).

*>>> rdd = sc.parallelize(range(0, 10)) >>> len(rdd.takeSample(True, 20, 1)) 20 >>> len(rdd.takeSample(False, 5, 2)) 5 >>> len(rdd.takeSample(False, 15, 3)) 10*

www.KnowBigData.com

More Actions: foreach()

Applies a function to all elements of this RDD.

*>>> def f(x): print x >>> sc.parallelize([1, 2, 3, 4, 5]).foreach(f)*

www.KnowBigData.com

More Actions: foreachPartition(f)

Applies a function to each partition of this RDD.

*>>> def f(itr): for x in itr:*

*print "+++" + str(x) yield None >>> sc.parallelize([1, 2, 3, 4, 5]).foreachPartition(f)*

+++1 +++2 +++3 +++4 +++5

www.KnowBigData.com

Persistence (caching)

*1. RDDs are lazily evaluated 2. RDD and all of its dependencies are recomputed on an*

*action 3. We may wish to use the same RDD multiple times. 4. To avoid re-computing, we can persist the RDD*

www.KnowBigData.com

Persistence (caching)

*1. If a node that has data persisted on it fails, Spark will*

*recompute the lost partitions of the data when needed. 2. We can also replicate our data on multiple nodes if we*

*want to be able to handle node failure without slowdown.*

www.KnowBigData.com

Persistence (caching) - Example

*⋙ nums = sc.parallelize(range(1, 100000), 5) ⋙ def sumparts(itr): ⋙ yield sum(itr) ⋙ ⋙ partitions = nums.mapPartitions(sumparts) ⋙ def incrByOne(x): ⋙ return x+1;*

*⋙ nums = sc.parallelize(range(1, 100000), 5) ⋙ def sumparts(itr): ⋙ yield sum(itr) ⋙ ⋙ partitions = nums.mapPartitions(sumparts) ⋙ def incrByOne(x): ⋙ return x+1;*

www.KnowBigData.com

Persistence (caching) - Example

persist(storageLevel= pyspark.StorageLevel(useDisk, useMemory, useOffHeap, deserialized, replication=1))

*⋙ nums = sc.parallelize(range(1, 100000), 5) ⋙ def sumparts(itr): ⋙ yield sum(itr) ⋙ ⋙ partitions = nums.mapPartitions(sumparts) ⋙ def incrByOne(x): ⋙ return x+1; ⋙ ⋙ partitions1 = partitions.map(incrByOne); ⋙ partitions1.persist() ⋙ partitions.is\_cached*

*False ⋙ partitions1.is\_cached*

True

www.KnowBigData.com

Persistence (caching) - Details

www.KnowBigData.com

Persistence (caching) - Details

www.KnowBigData.com

Persistence (caching) - Usage

1. Create an object of StorageLevel

*a. sl=StorageLevel(useDisk=True, useMemory=True, useOffHeap=False,*

*deserialized=True, replication=2) OR b. sl=StorageLevel(True, True, False, True, replication=1)*

2. Then pass it while persisting:

*a = sc.parallelize(range(1, 1000)) a.persist(storageLevel=sl);*

3. In default persist(), the following storage level is used StorageLevel(False, True, False, False, 1))

For more detail, please check the Python API

www.KnowBigData.com

Key - Value Pair RDDs

1. Special operations on RDDs containing key/value pairs 2. These RDDs are called pair RDDs. 3. Pair RDDs are a useful building block in many programs 4. For example, pair RDDs have a reduceByKey()

1. A Pair in Python is defined as (x, y) 2. Also known as tuple 3. A tuple is a sequence of immutable Python objects. 4. You can convert a list into tuple

www.KnowBigData.com

Creating Key-Value Pair RDDs

**By Directly Loading as Key-Value Pairs**

We will discuss it later

**By using a map function:**

⋙ lines = sc.parallelize("this is a bird".split()) ⋙ def toWords(w): return (w, 1); ⋙ lines.map(toWords).collect()

www.KnowBigData.com

Transformations on Pair RDDs

reduceByKey(func)

Combine values with the same key.

rdd = sc.parallelize([(1, 2), (3, 4), (3, 6)]); def sm(x, y):

return x+y;

rdd1 = rdd.reduceByKey(sm) rdd1.collect() [(1, 2), (3, 10)]

www.KnowBigData.com

**Word Count example**

#STEP 1: Load the data lines = sc.textFile('hdfs://hadoop1.knowbigdata.com/data/mr/wordcount/input/big.txt')

#STEP 2: Prepare Words def lineToWords(line): return line.split(" ");

words = lines.flatMap(lineToWords)

#STEP 3: convert to key-value pairs def toKV(word):

return (word, 1);

wordsKV = words.map(toKV);

www.KnowBigData.com

**Word Count example**

#Step 4: Reduce def sm(v1, v2):

return v1+v2;

wc = wordsKV.reduceByKey(sm) wc.take(10);

#Step 5: Save the data wc.saveAsTextFile('hdfs://hadoop1.knowbigdata.com/user/spark1/wordcount-output')

www.KnowBigData.com

+1 419 665 3276 (US) +91 803 959 1464 (IN)

Subscribe to our Youtube channel for latest videos -

https://www. youtube.com/channel/UCxugRFe5wETYA7nMH6VGyEA

**Apache Spark**

Thank you.

reachus@knowbigdata.com

www.KnowBigData.com