Apache Spark

Session 7 - Advanced Spark Programming

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

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

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 2006 2002	**tBits Global**	Founded tBits Global Built an enterprise grade Document Management System
	D.E.Shaw	Built the big data systems before the term was coined
2002	**IIT Roorkee**	Finished B.Tech.

Shared variables:

When we pass functions such map()
Every node gets a copy of the variable
The change to these variables is no communicated back
After starting of the map(), changes to the variable on driver doesn't impact the worker.

Two Kinds:

1. **Accumulators** to aggregate information
2. **Broadcast variables** to efficiently distribute large values

Operator Types

An op is

a. commutative
b. associative

If

1. (a *op* b) *op* c = a *op* (b *op* c)
2. a *op* b = b *op* a

Operator Types

An op is

a. commutative
b. associative

If

1. (a *op* b) *op* c = a *op* (b *op* c)
2. a *op* b = b *op* a

Answer:

a-2

b-1

Operator Types

Question: sum and max are commutative and associative? (true/false)?

Operator Types

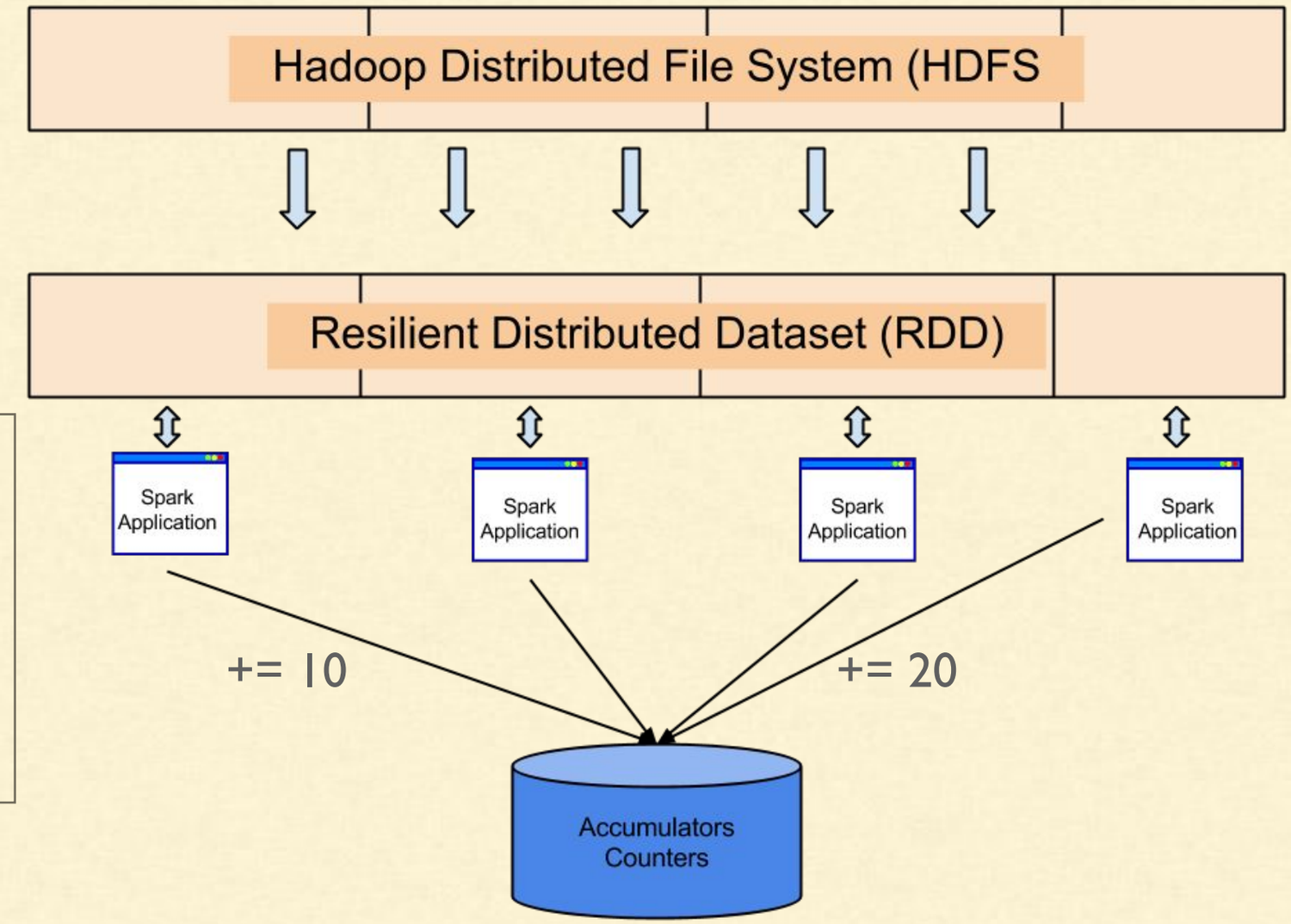
Question: sum and max are commutative and associative? (true/false)?

Answer: true

SHARED MEMORY - Accumulators

are only "added" to through associative operation

assoc.: 2+3+4=2+4+3=9



Accumulators

- Accumulators are variables that are only “added” to through an associative operation
- Can therefore be efficiently supported in parallel.
- They can be used to implement counters (as in MapReduce) or sums.

Accumulator : empty line count in Python

```
file = sc.textFile("hdfs://h1.cloudxlab.com/data/mr/wordcount/input/" )

# Create Accumulator[Int] initialized to 0
   blankLines = sc.accumulator(0)
   def countBL(line):
      global blankLines
      # Make the global variable accessible
      if (line == ""):
           blankLines += 1
      return line.split(" ")

words = file.flatMap(countBL)
words.saveAsTextFile("words")
print "Blank lines: %d" % blankLines.value
```

Accumulators work as follows:

1. We create them in the driver by calling the SparkContext. accumulator(initial Value) method, which produces an accumulator holding an initial value. The return type is an org.apache.spark. Accumulator[T] object, where T is the type of initialValue.
2. Worker code in Spark closures can add to the accumulator with its += method (or add in Java).
3. The driver program can call the value property on the accumulator to access its value (or call value() and setValue() in Java).

Note: tasks on worker nodes cannot access the accumulator's value()

Accumulators and Fault Tolerance

- Spark Re-executes failed or slow tasks.
- Preemptively launches “speculative” copy of slow worker task

The net result is ???

Accumulators and Fault Tolerance

- Spark Re-executes failed or slow tasks.
- Preemptively launches “speculative” copy of slow worker task

The net result is: The same function may run multiple times on the same data.

Accumulators and Fault Tolerance

- Spark Re-executes failed or slow tasks.
- Preemptively launches “speculative” copy of slow worker task

The net result is: The same function may run multiple times on the same data.

Does it mean accumulators will give wrong result?

Accumulators and Fault Tolerance

- Spark Re-executes failed or slow tasks.
- Preemptively launches “speculative” copy of slow worker task

The net result is: The same function may run multiple times on the same data.

Does it mean accumulators will give wrong result?
YES, for accumulators in Transformation.
No, for accumulators in Action

Accumulators and Fault Tolerance

- For accumulators in actions, Each task's accumulator update applied once.
- For reliable absolute value counter, put it inside an action
- In transformations, this guarantee doesn't exist.
- In transformations, use accumulators for debug only.

Custom Accumulators

- Out of the box, Spark supports accumulators of type Double, Long, and Float.
- Spark also includes an API to define custom accumulator types and custom aggregation operations
  - (e.g., finding the maximum of the accumulated values instead of adding them).
- Custom accumulators need to extend AccumulatorParam.

Custom Accumulators

```
from pyspark.accumulators import AccumulatorParam

class VectorAccumulatorParam(AccumulatorParam):
    def zero(self, initialValue):
        return Vector.zeros(initialValue.size)

    def addInPlace(self, v1, v2):
        v1 += v2
        return v1

# Then, create an Accumulator of this type:
vecAccum = sc.accumulator(Vector(...),
VectorAccumulatorParam())
```

Accumulators Question

```
accum = sc.accumulator(0)
def g(x):
  accum.add(x)
  return f(x)
data.map(g)
print accum.value()
```

Question: What is the value of accum at the end?

Accumulators Question

```
accum = sc.accumulator(0)
def g(x):
  accum.add(x)
  return f(x)
data.map(g)
print accum.value()
```

Question: What is the value of accum at the end?
Answer: 0 because of lazy evaluation.

Broadcast Variables : Introduction

```
commonWords = ["a", "an", "the", "of", "at", "is",
"am", "are", "this", "that", '', 'at']

If we need to remove the common words from our
wordcount, what do we need to do?
```

Broadcast Variables : Introduction

```
commonWords = ["a", "an", "the", "of", "at", "is",
"am", "are", "this", "that", '', 'at']

If we need to remove the common words from our
wordcount, what do we need to do?

> We can create a local variable and use it
```

Broadcast Variables : Introduction

```
commonWords = ["a", "an", "the", "of", "at", "is",
"am", "are", "this", "that", '', 'at']

If we need to remove the common words from our
wordcount, what do we need to do?

> We can create a local variable and use it

> Is it inefficient?
```

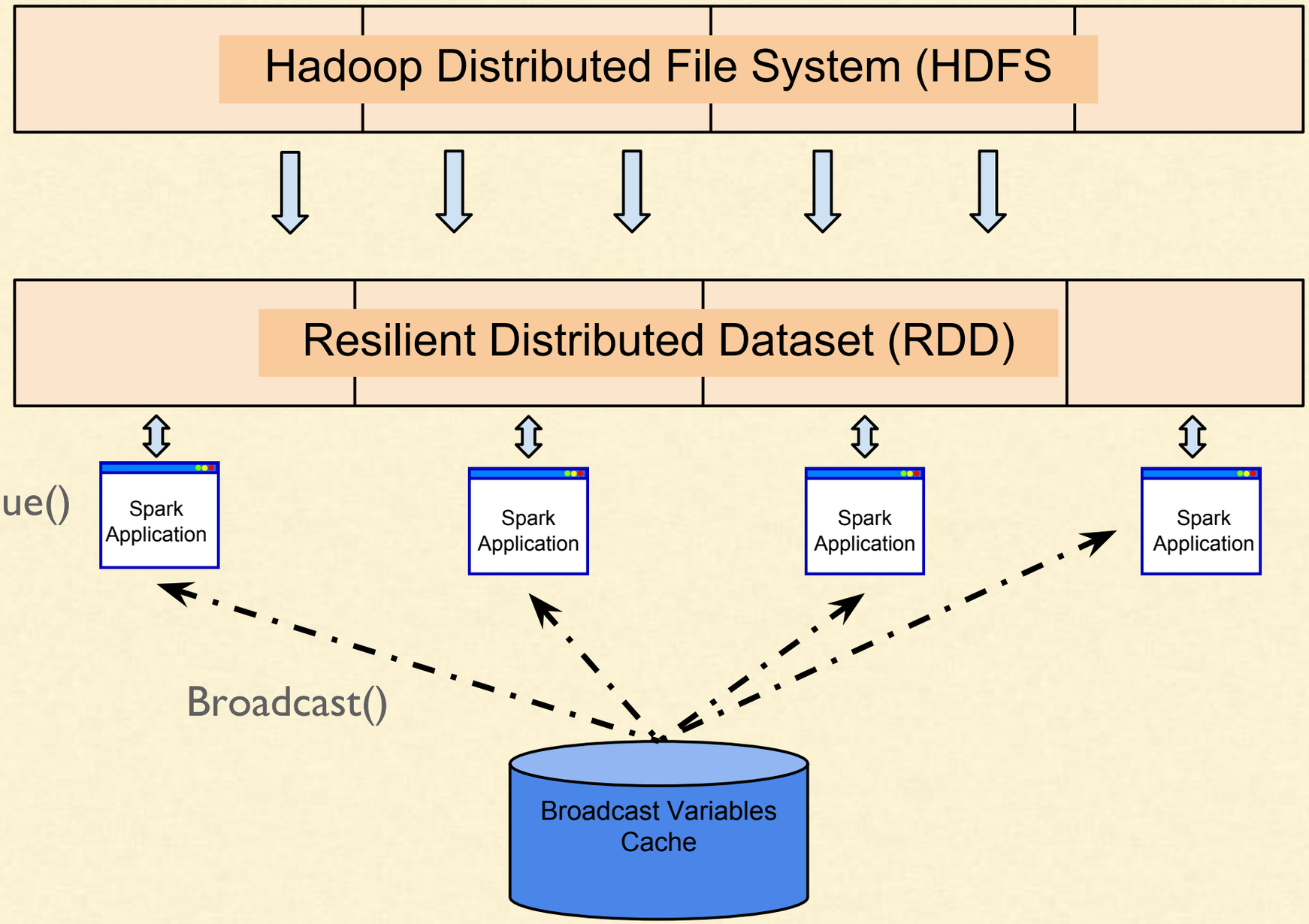
Broadcast Variables : Introduction

1. Yes because spark sends all the reference variables to all workers.
2. But it is inefficient because:
    a. The default task launching mechanism is optimised for small task sizes.
    b. If using multiple times, spark will be sending it again to all nodes
3. So, we use broadcast variable instead.

SHARED MEMORY

Broadcast Variables

broadcast.value()



Broadcast Variables

- Allows the program to efficiently send a large, read-only value To all the worker nodes
- For example:
  - Send a large, read-only lookup table to all the nodes
  - Large feature vector in a machine learning algorithm
- It is like a distributed cache of Hadoop
- Spark distributes broadcast variables efficiently to reduce communication cost.
- Useful when
  - Tasks across multiple stages need the same data
  - Caching the data in deserialized form is important.

Broadcast Variables : Example

Removing Common Words using Broadcast.

```
commonWords = ["a", "an", "the", "of", "at", "is", "am", "are", "this", "that", '', 'at']
file = sc.textFile("hdfs://hadoop1.knowbigdata.com/data/mr/wordcount/input/")
commonWordsBC = sc.broadcast(commonWords)
# Create Accumulator[Int] initialized to 0
def toWords(line):
   words = line.split(" ")
   output = [];
   for word in words:
      word = word.lower();
      cws = commonWordsBC.value;
      if word not in cws:
         output.append(word)
   return output;

uncommonWords = file.flatMap(toWords)
uncommonWords.collect()
```

Working on a Per-Partition Basis

```
def process(signs):
    """Lookup call signs using a connection pool"""
    # Create a connection pool
    http = urllib3.PoolManager()
    # the URL associated with each call sign record
    urls = map(lambda x: "http://73s.com/qsos/%s.json" % x, signs)
    # create the requests (non-blocking)
    requests = map(lambda x: (x, http.request('GET', x)), urls)
    # fetch the results
    result = map(lambda x: (x[0], json.loads(x[1].data)), requests)
    # remove any empty results and return
    return filter(lambda x: x[1] is not None, result)

def fetchCallSigns(input):
    """Fetch call signs"""
    return input.mapPartitions(lambda callSigns: process(callSigns))

contactsContactList = fetchCallSigns(validSigns)
```

Piping to External Programs

1. Sometimes we need to handle the legacy code
2. Or we need to use domain specific language such as R

R distance program

```
#!/usr/bin/env Rscript
library("Imap")
f <- file("stdin")
open(f)
while(length(line <- readLines(f,n=1)) > 0) {
  # process line
  contents <- Map(as.numeric, strsplit(line, ","))
  mydist <- gdist(contents[[1]][1], contents[[1]][2],
                  contents[[1]][3], contents[[1]][4],
                  units="m", a=6378137.0, b=6356752.3142, verbose = FALSE)
  write(mydist, stdout())
}
```

```
$ ./src/R/finddistance.R 37.75889318222431,-122.42683635321838,37.7614213,-122.4240097 349.2602
coffee
NA
ctrl-d
```

using pipe() to call finddistance.R

```
# Compute the distance of each call using an external R program
distScript = "./src/R/finddistance.R"
distScriptName = "finddistance.R"
sc.addFile(distScript)
def hasDistInfo(call):
    """Verify that a call has the fields required to compute the distance"""
    requiredFields = ["mylat", "mylong", "contactlat", "contactlong"]
    return all(map(lambda f: call[f], requiredFields))
def formatCall(call):
    """Format a call so that it can be parsed by our R program"""
    return "{0},{1},{2},{3}".format( call["mylat"], call["mylong"], call["contactlat"], call
["contactlong"])
pipeInputs = contactsContactList.values().flatMap(
    lambda calls: map(formatCall, filter(hasDistInfo, calls)))
distances = pipeInputs.pipe(SparkFiles.get(distScriptName))
print distances.collect()
```

Numeric RDD Operations

Summary statistics available	
count()	Number of elements in the RDD
mean()	Average of the elements
sum()	Total
max()	Maximum value
min()	Minimum value
variance()	Variance of the elements
sampleVariance()	Variance of the elements, computed for a sample
stdev()	Standard deviation
sampleStdev()	Sample standard deviation

Removing outliers in Python

```
# Convert our RDD of strings to numeric data
# so we can compute stats and remove the outliers.
a = sc.parallelize([1,2,3,4,5,5,6,6,6,6,10,14,15,1,2,2,10,50000])
stdev = a.stdev()
mean = a.mean()
def isNotOutLier(x):
    return math.fabs(x - mean) < (3 * stdev)
b = a.filter(isNotOutLier)
```

Apache Spark

Thank you.

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

reachus@knowbigdata.com

Subscribe to our Youtube channel for latest videos - https://www.youtube.com/channel/UCxugRFe5wETYA7nMH6VGyEA