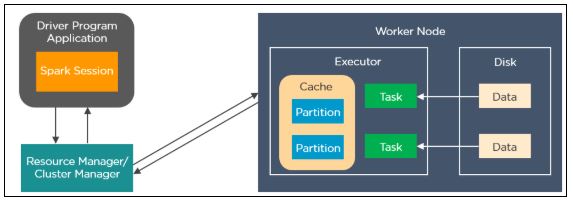
**Spark:**

**A fast general-purpose computing engine for processing large datasets and is a replacement for Hadoop’s MapReduce.**

1. Spark contains a driver and a worker or executor where the driver launches various ops like Analyzing and assigning the tasks to the driver node and it runs in local or cluster mode. And the worker node takes care of the actual computation.



2. We can run the spark application in cluster mode or local mode.

In cluster mode – The job runs in the cluster and Spark context creates a Spark application and upon action, the program submits the job to the cluster manager. The cluster manager takes care of resource allocation and starts the execution of the worker node. The jobs need to run in cluster mode to avoid local computer dependencies and can run independently.

In Client mode – The job runs in the machine and it is called a single-node cluster with no worker node, so the driver takes care of both the master and worker nodes. It utilizes as much executors’ memory as possible. The jobs can run in Client mode when testing or debugging issues.

And another mode is local mode where no cluster is needed.

3. Spark is a computing engine like Python, Scala, and R, and we have a cluster manager and storage system as external as yarn or mesos and hdfs or s3.

Spark contains structured and unstructured APIs like datasets/Dataframe and RDD/accumulators and broadcast variables.

4. Features of spark are lazy evaluation, real-time processing, speed, and in-memory computing

And it also involves job submission, resource allocation, task scheduling, task execution, and result collection.

5. Operations: Transformations and actions

Overview:

The spark job is called after an action in local or cluster mode. The driver creates the directed acyclic graph and divides the job into tasks. Many tasks can take place in the same stage creating one or more RDD. The cluster manager takes care of resource allocation and finally executor takes care of the operation. And spark automatically takes care of any shuffle operation. Finally, the output is collected by the driver from each executor and stored.

6. DAG - stands for directed acyclic graph with finite vertices and edges. It takes place in a sequential manner with unidirectional operations.

7. Features of spark

In-memory computation

Distributed processing using parallelize.

Can be used with many cluster managers (Spark, Yarn, Mesos e.t.c)

Fault-tolerant

Immutable

Lazy evaluation

Cache & persistence

8. RDD: Resilient distributed dataset means it can auto recover from any failure which means its fault tolerance. In any loss of data or failure, the spark has the ability to recover the lost data.

Distributed collection of data and immutable data

Cache: Results can be saved in memory only

Persistence: Results can be saved in local storage which helps with time and cost

9. Repartition and Coalesce:

Using repartition, we can increase or decrease the number of partitions but with coalesce we can only decrease the partitions.

Repartition ops create a new partition and perform full shuffle but coalesce uses the existing partition and reduces shuffling.

Repartition internally calls coalesce which results in the redistribution of data.

Coalese: It is used to reduce the number of partitions and avoid full shuffle and used hash partitioning.

Repartition: It is used to either increase or decrease the number of partitions and it is a full shuffle

10. Narrow and wide transformation:

Narrow doesn’t require data to be shuffled (map, flatmap, filter) whereas wide requires data to be shuffled (reduceByKey, groupByKey).

11. Difference between RDD, DataFrame, and Dataset

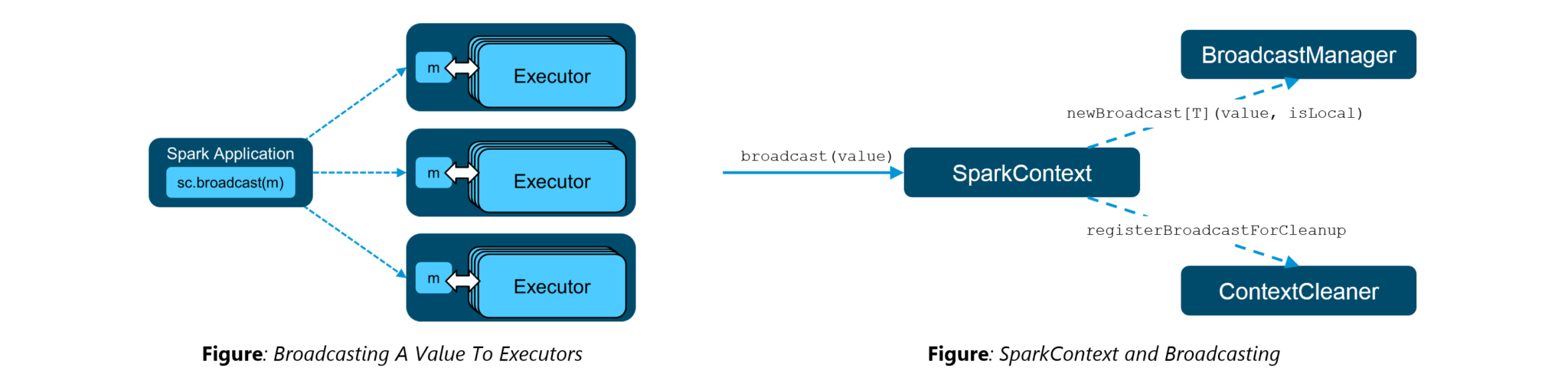
RDD represents distributed data providing low-level API and is dynamically typed and can automatically recover from node failure.

The data frame is similar to a table, and it is strongly typed which means it is dependent on the schema and provides high-level API.

The dataset is an extension of the data frame.

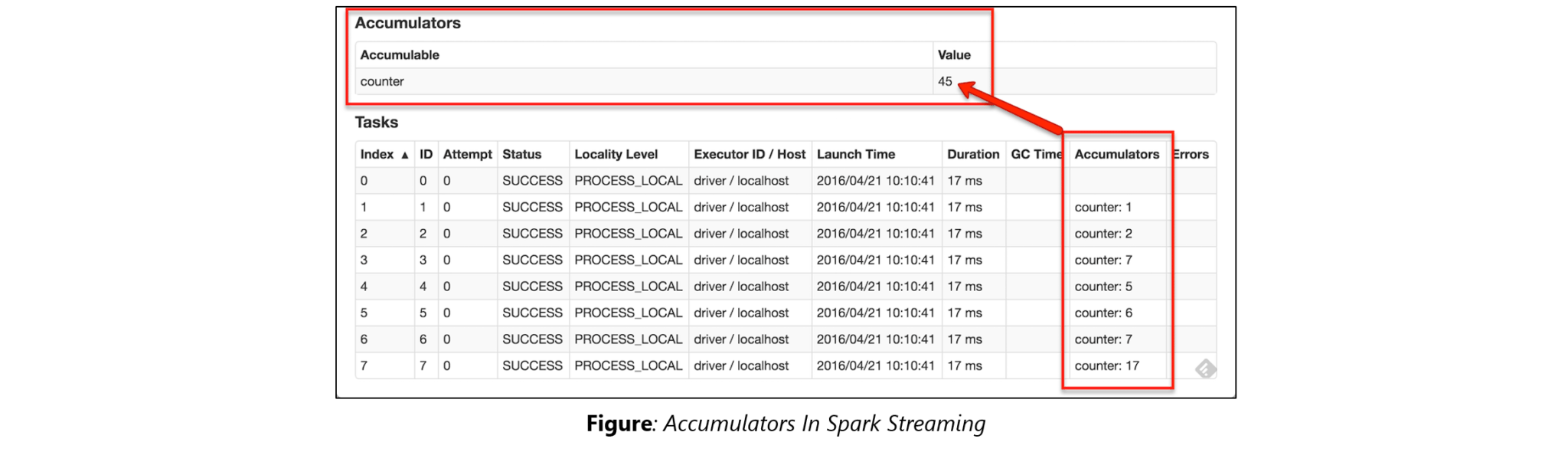
12. Broadcast variables:

Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. Broadcast variables are read only variables, present in-memory cache on every machine. When working with Spark, usage of broadcast variables eliminates the necessity to ship copies of a variable for every task, so data can be processed faster. Broadcast variables help in storing a lookup table inside the memory which enhances the retrieval efficiency when compared to an RDD *lookup()*

****

13. Accumulators:

Accumulators are variables that are only added through an associative and commutative operation. They are used to implement counters or sums. Tracking accumulators in the UI can be useful for understanding the progress of running stages. Spark natively supports numeric accumulators. We can create named or unnamed accumulators.

****

1. Spark session:

Introduced in spark 2.0 and is the entry point for underlying functionality (sparkcontext, sql, hive, streaming)

Prior – spark context is entry point

import org.apache.spark.sql.SparkSession

object SparkSessionTest extends App {

val spark = SparkSession.builder() - class

.master("local[1]") – methods for running cluster

.appName("SparkByExamples.com") – just a name

.getOrCreate(); - return spark object

.enableHiveSupport()

println(spark)

println("Spark Version : "+spark.version)

}

Spark sql - we need createOrReplaceTempView

// Spark SQL

df.createOrReplaceTempView("sample\_table")

val df2 = spark.sql("SELECT \_1,\_2 FROM sample\_table")

df2.show()

2. rdd – represents dataset distributed across multiple nodes in the cluster.

Immutable and fault tolerance – which mean resilient

Creation – using parallelize or text file or another rdd

1. scala> val rdd = sc.parallelize(Array(1,2,3,4,5,6,7,8,9,10))0

Create empty rdd:

val rdd = spark.sparkContext.emptyRDD // creates EmptyRDD[0]

val rddString = spark.sparkContext.emptyRDD[String] // creates EmptyRDD[1]

Text file: Reads single or multiple line and return an RDD and wholetextfile: same but first element is a tuple and 2nd is content of file

val rdd = spark.sparkContext.textFile("C:/tmp/files/\*")

rdd.foreach(f=>{

println(f)

})

Reading file and split with ,

val rdd5 = spark.sparkContext.textFile("C:/tmp/files/\*")

val rdd6 = rdd5.map(f=>{

f.split(",")

})

rdd6.foreach(f => {

println("Col1:"+f(0)+",Col2:"+f(1)) })

3. Actions in RDD

Performs operations and creates another RDD

Eg: collect, count, countApprox, countApproxDistinct, countByvalue, first, top, take,

4. Repartition: incr or decr the num of partitions and coalesce: decrease the num of partitions

Both has shuffle the data across many nodes

Repartition: full shuffle

Coalesce: data movement is less

5. shuffle partition: re-distributing data

Aggregate ops - Groupby, groupbykey reduceby, join

Expensive because it involved disk i/o, serialization and deserialization and data movement

Manual set partition: spark.conf.set("spark.sql.shuffle.partitions",100)

6. Cache: stores in memory and persist stores in user defined storage level

Optimization technique to store intermediate output of rdd, dataframe and dataset for subsequent operation

7. Broadcast variable: read only shared variables are cached and avaible in all nodes of cluster to reduce communication cost and time

8. Accumulators: Spark Accumulators are shared variables which are only “added” through an associative and commutative operation and are used to perform counters (Similar to Map-reduce counters) or sum operations

9. Dataframe: Similar to database it is distributed collection of data in rows and columns

Using implicits we convert rdd to df or using createDataFrame method

Below is df with schema,

//From RDD (USING createDataFrame and Adding schema using StructType)

val schema = StructType(columns

.map(fieldName => StructField(fieldName, StringType, nullable = true)))

//convert RDD[T] to RDD[Row]

val rowRDD = rdd.map(attributes => Row(attributes.\_1, attributes.\_2))

val dfFromRDD3 = spark.createDataFrame(rowRDD,schema)

10. createDataFrame, toDF or read.csv/json

Transformations:

RDD lineage - one rdd creates another rdd and so on…

All rdd are lazy evaluation - no transformation is called until you call action.

Types: 1. Narrow: no data movement and live on single partition – map, mapPartition, flatMap, filter

2. Wide: live one many partitions and has data movement since shuffle take place – groupByKey, reduceByKey, aggregateByKey, join, repartition

Transaformations:

Map. Flatmap,filter,sortbykey,

val structureData = Seq(

Row("James","","Smith","36636","NewYork",3100),

Row("Michael","Rose","","40288","California",4300),

Row("Robert","","Williams","42114","Florida",1400),

Row("Maria","Anne","Jones","39192","Florida",5500),

Row("Jen","Mary","Brown","34561","NewYork",3000)

)

val structureSchema = new StructType()

.add("firstname",StringType)

.add("middlename",StringType)

.add("lastname",StringType)

.add("id",StringType)

.add("location",StringType)

.add("salary",IntegerType)

val df2 = spark.createDataFrame(

spark.sparkContext.parallelize(structureData),structureSchema)

df2.printSchema()

df2.show(false)

fold – aggregate each element in each partitions and then all partitions

aggregate

reduce

reduceByKey

Spark sql:

1. Where and filter based on given condition on a df

2. withColumn: add new column, change value of existing column, derive a new column from existing column, convert datatype

3. convert df into temp view for sql queries

4. df.distinct and df.dropduplicates

5. groupby

6. map: applies function to each row in a dataframe mappartition: function on each partition

Both return a dataset and not a df

7. for each and foreach partition

8. pivot and unpivot – row to column and reverse

9. collect: retrieve all items from the dataset and collectaslist

Collect is action whereas select is transformation

10. Operations in spark:

RDD transformations – Transformations are lazy operations, instead of updating an RDD, these operations return another RDD.

RDD actions – operations that trigger computation and return RDD value