Chapter 1 Spark Introduction Review Done

==========

http://vishnuviswanath.com/spark_rdd.html

1. What is Apache Spark?
2. A brief History of Apache Spark
3. Spark shines on where?
4. Processing terminology
5. Spark Before and After
6. A Unified Stack
7. Apache Spark Components
7.1. Spark Core
7.2. Spark SQL
7.3. Spark Streaming
7.4. Spark MLlib
7.5. Spark GraphX
7.6. SparkR s
8. Cluster Managers
9. Storage Layers for Spark
10. Is Spark depending on Hadoop?
11. What kind of files spark support?

Chapter 2 Spark core concepts

==========

1. What is Drive program
2. What is SparkContext in Apache Spark?
3. How to Create SparkContext object?
4. Stopping SparkContext
5. Spark Scala Word Count Example
6. Functions of SparkContext in Apache Spark

Chapter 3 RDD

==========

1. What is RDD?
2. Why do we need RDD in Spark?
3. RDD vs DSM (Distributed Shared Memory)
4. Features of RDD in Spark
5. Spark RDD Operations
5.1. Transformations
5.1.1. Narrow Transformations
5.1.2. Wide Transformations
5.2. Actions
6. Limitation of Spark RDD

Chapter 4 RDD Persistence and Caching Mechanism in Apache Spark

==========

1. What is RDD Persistence and Caching in Spark?
2. Need of Persistence in Apache Spark
3. Benefits of RDD Persistence in Spark
4. How to Un-persist RDD in Spark?

Chapter 5 Spark RDD Operations-Transformation & Action with Example

==========

To Do List

2. Apache Spark RDD Operations
3. RDD Transformation
3.1. map(func)
3.2. flatMap()
3.3. filter(func)

==========

==========

1. What is Lazy Evaluation in Apache Spark?
2. Advantages of Lazy Evaluation in Spark Transformation
a. Increases Manageability
b. Saves Computation and increases Speed
c. Reduces Complexities

Chapter 8 Directed Acyclic Graph DAG in Apache Spark

==========

1. What is DAG in Apache Spark?
2. Need of Directed Acyclic Graph in Spark
3. How DAG works in Spark?
4. How is Fault tolerance achieved through DAG?
5. Working of DAG Optimizer in Spark
6. Advantages of DAG in Spark

Chapter 8 How Apache Spark Works – Run-time Spark Architecture

==========

1. Internals of How Apache Spark works?
2. Terminologies of Spark
2.1. Apache SparkContext
2.2. Apache Spark Shell
2.3. Spark Application
2.4. Task
2.5. Job
2.6. Stage
3. Components of Spark Run-time Architecture of Spark
3.1. Apache Spark Driver
3.2. Apache Spark Cluster Manager
3.3. Apache Spark Executors

4. How to launch a Program in Spark?
5. How to Run Apache Spark Application on a cluster

Chapter 9 Spark Performance Tuning-Learn to Tune Apache Spark Job

==========

1. What is Performance Tuning in Apache Spark?
2. Data Serialization in Spark
3. Memory Tuning in Spark

a. Spark Data Structure Tuning

b. Spark Garbage Collection Tuning

4. Memory Management in Spark
5. Determining Memory Consumption in Spark
6. Spark Garbage Collection Tuning
7. Other consideration for Spark Performance Tuning

a. Level of Parallelism

b. Memory Usage of Reduce Task in Spark

c. Broadcasting Large Variables

d. Data Locality in Apache Spark

==========

Introduction to Apache Spark

1. Apache Spark
2. History of Apache Spark
3. Spark shines on where?
4. Processing terminology
5. Spark - Before and After
6. A Unified Stack
7. Apache Spark Components

1. Apache Spark

- ✓ Apache Spark is a cluster computing platform designed to be fast and general purpose.
- ✓ It is written in Scala programming language.
- ✓ It provides high-level API. For example, Java, Scala, Python and R.
- ✓ Easily integrated with Hadoop and process existing HDFS data.

2. History of Spark

- ✓ Spark started in the year of 2009 as a research project.
- ✓ It was open sourced in 2010.
- ✓ In 2013 spark was donated to Apache Software Foundation where it became top-level Apache project in 2014.

3. Spark shines on where?

- ✓ MapReduce is a bit slow while processing large datasets because it requires more read and write operations.
- ✓ Spark offers speed while processing large datasets because it's not requires more read and write operations (it can able to do RAM level processing)

4. Processing terminology

	Batch	Stream	Interactive	Graph
Definition	To processing historical data.	As soon as data generated need to process. (Real time data)	Takes input from the users to run quickly.	It uses graphical representation.
Best use case	If working with very large data	If you want the results in real time mode	frequent user interaction	social network analysis

5. Spark - Before and After

Before

- ✓ Hadoop MapReduce can only perform batch processing.
- ✓ Apache Storm / S4 can only perform stream processing.
- ✓ Apache Impala / Apache Tez can only perform interactive processing.
- ✓ Neo4j / Apache Giraph can only perform to graph processing.

After

- ✓ Apache Spark can perform,
  - Batch processing.
  - Stream processing.
  - Interactive processing

- Graph processing

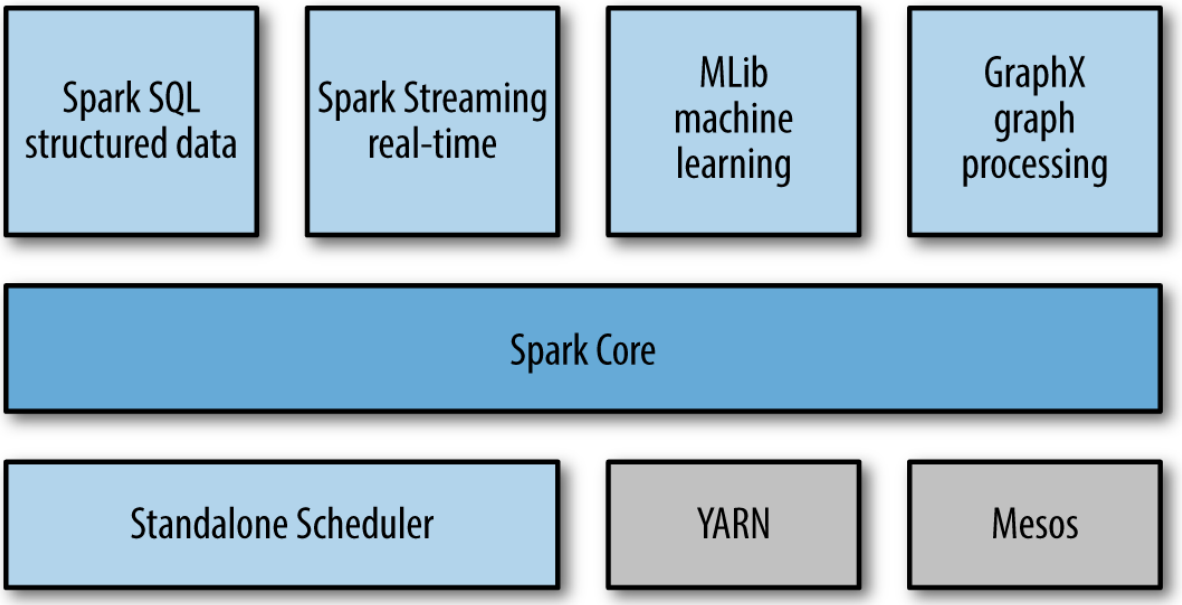
6. A Unified stack

- ✓ Any organization prefers to run only one software which having all stack of processing, instead of running 5–10 independent software systems.
- ✓ These costs include deployment, maintenance, testing, support etc.
- ✓ Spark is a unified stack integrated with different processing components.

Make a note

- ✓ So, we can say as, Spark is a single point of contact for all types of processing.

7. Apache Spark Components



7.1. Spark Core

- ✓ Spark core module provides built in features and functionalities.
- ✓ It delivers speed by providing in-memory computation capability.
- ✓ It is the foundation of parallel and distributed processing of huge dataset.
- ✓ key features,
  - It is in-charge of essential I/O functionalities.
  - Task scheduling
  - Memory management
  - Fault recovery
  - Interacting with storage systems etc.
- ✓ Spark Core provides different types of API to define RDD (Resilient Distributed Datasets)
- ✓ RDD is a spark's main programming abstraction.
- ✓ RDD represent a collection of items distributed across many nodes that can be processed in parallel.

7.2. Spark SQL

- ✓ Spark SQL is used to process structured data by using SQL and HQL.
- ✓ It supports many sources of data
  - Hive tables
  - Parquet
  - JSON

7.3. Spark Streaming

- ✓ Spark Streaming module enables processing of live streams of data.
- ✓ In spark, live streams are converted into micro-batches which are executed on top of spark core.

7.4. MLlib

- ✓ Spark comes with a library containing common machine learning (ML) functionality, called MLlib.
- ✓ MLlib provides multiple types of machine learning algorithms, including
  - classification
  - regression
  - clustering
  - collaborative filtering
  - model evaluation and data import.

7.5. GraphX

- ✓ GraphX is a library for manipulating graphs and performing graph-parallel computations.(e.g., a social network's friend graph)
- ✓ Like Spark Streaming and Spark SQL, GraphX extends the Spark RDD API.

7.6. SparkR

- ✓ It is R package that gives lightweight frontend to use Apache Spark from R.
- ✓ It allows data scientists to analyse large datasets and interactively run jobs on them from the R shell.
- ✓ The main idea behind SparkR was to explore different techniques to integrate the usability of R with the scalability of Spark.

8. Cluster Managers

- ✓ Spark can run over a variety of cluster managers, including Hadoop YARN, Apache Mesos.
- ✓ Simple cluster manager included in Spark itself called the Standalone Scheduler.

9. Storage Layers for Spark

- ✓ HDFS
- ✓ Local File System
- ✓ Amazon S3
- ✓ Cassandra
- ✓ Hive
- ✓ HBase etc.

10. Is Spark depending on Hadoop?

- ✓ Spark does not require Hadoop.
- ✓ By default, spark search and store files in HDFS.

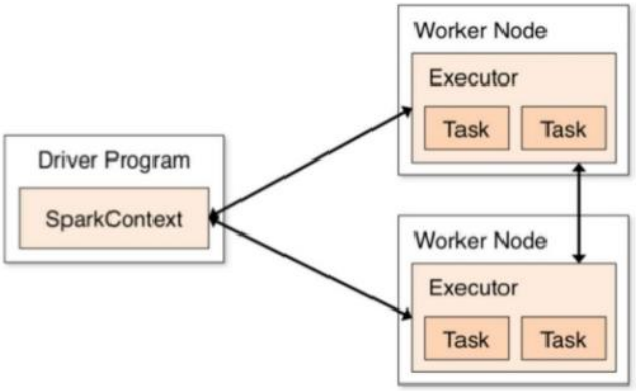
11. What kind of files spark support?

- Text files
- SequenceFiles
- Avro
- Parquet
- Any other Hadoop InputFormat.

2. Spark core concepts

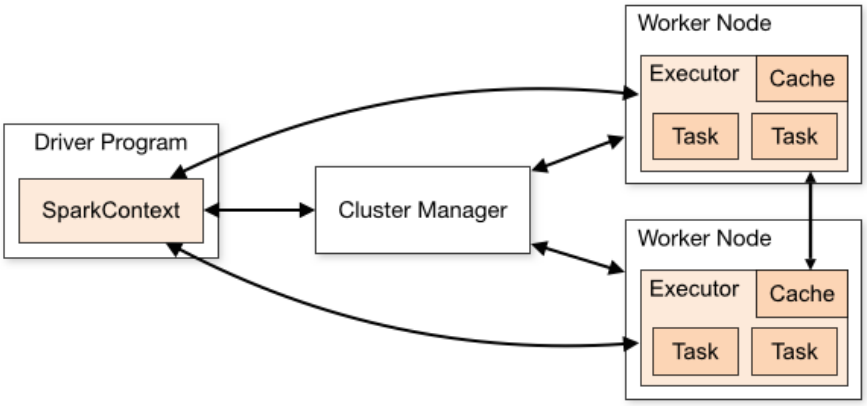
1. Driver program

- ✓ Every Spark application contains driver program that launches various parallel operations on a cluster.
- ✓ It contains your application's main function and defines distributed datasets on the cluster, then applies operations on them.
- ✓ To run operations, driver programs manage number of nodes called executors.
- ✓ Driver programs communicate to cluster by using SparkContext.



2. SparkContext?

- ✓ It is the entry point of Spark functionality.
- ✓ Without SparkContext no computation can be started.
- ✓ The most important step of any Spark driver application is to create SparkContext object.
- ✓ SparkContext uses resource manager to connect the cluster.
- ✓ The resource manager can be one of these three,
  - Spark Standalone,
  - YARN,
  - Apache Mesos.



3. Creating SparkContext object

- ✓ SparkConf and SparkContext are available in org. apache. spark package.
- ✓ SparkConf object is required because to create SparkContext object.
- ✓ We need to set two parameters for SparkConf object.

- ✓ val sparkConf = new SparkConf (). setMaster("local"). setAppName ("My App")

    - local is value, which tells Spark how to connect to a cluster.
        - If you are giving local, then it runs on standalone mode.

    - My App is a application name.
        - This will identify your application on the cluster manager's UI if you connect to a cluster.

- ✓ SparkContext constructor will take parameter as SparkConf object.
- ✓ val sparkContext = new SparkContext(sparkConf)

- ✓ With SparkContext object we can call,
    - We can create RDDs
    - broadcast variable
    - accumulator
    - to run jobs.
- ✓ All these things can be carried out until SparkContext is stopped.

6. How many SparkContext objects are available for application?

- ✓ Only one SparkContext object is available for whole over Spark application.
- ✓ We should not create more than one SparkContext objects.
- ✓ We should stop existing SparkContext object before creating new sparkcontext object.

7. Stopping SparkContext

- ✓ We can stop SparkContext object by using stop () method.

Syntax

```
sparkContext.stop()
```

- ✓ It will display the following message:
- ✓ INFO SparkContext: Successfully stopped SparkContext

4. Deployment environments – Run Modes

1. local
2. clustered
    - 2.1 Spark Standalone
    - 2.2 Spark on Apache Mesos
    - 2.3 Spark on Hadoop YARN

- ✓ Spark deployment environment are two types namely local and clustered.
- ✓ local
    - local mode is non-distributed single-JVM deployment mode.
    - All the execution components – driver, executor, LocalSchedulerBackend, and master are present in same single JVM.
- ✓ cluster
    - While in clustered mode, the Spark runs in distributive mode.

Modes

https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-deployment-environments.html

https://data-flair.training/blogs/apache-spark-cluster-managers-tutorial/

5. When we suppose to use local and cluster modes

- ✓ For testing, debugging, the local mode is suitable because it requires no earlier setup to launch spark application.
- ✓ While in clustered mode, the Spark runs in distributive mode.

Program purpose : Word count

Program name : WordCount.scala

output :

hi,2

how, 3,

are, 2

```
package com. nireekshan. spark
import org. apache. spark. SparkContext
import org. apache. spark. SparkConf
object WordCount
{
        def main (args: Array[String])
        {
        val sparkConf = new SparkConf (). setAppName ("WordCount")
        val sparkContext = new SparkContext(sparkConf)

        val inputRDD = sparkContext. textFile ("file:///home
        /input/wordcount.txt")
        val wordsRDD = inputRDD.flatMap(x => x. split (" "))
        val keyValueRDD = wordsRDD.map (x => (x, 1))
        val wordCountRDD = keyValueRDD.reduceByKey((x, y) => x + y)
        wordCountRDD.count()
        wordCount. saveAsTextFile("WCoutput")
        sparkContext.stop
        }
}
```

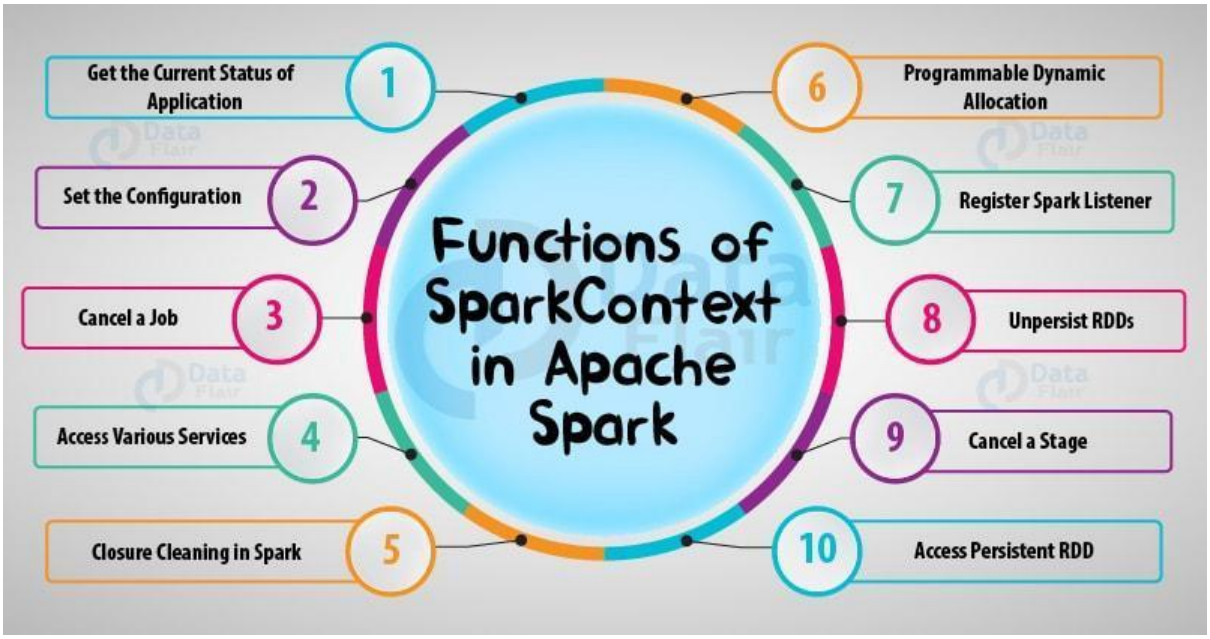
We can run this application in two ways:

1. By using intelliJ

2. By using spark shell.

Compile	:	Executing by using Spark shell
Run	:	Executing by using Spark shell

5. Functions of SparkContext in Apache Spark



1. To get the status of Spark Application
2. To set the configuration
3. To Access various services
4. To Cancel a job
5. To Cancel a stage
6. For Closure cleaning in Spark
7. To Register Spark listener
8. Programmable Dynamic allocation
9. To access persistent RDD
10. To un-persist RDDs

Spark RDD – Introduction, Features & Operations of RDD

Why RDD

In Spark all work is expressed as,

- ✓ Either creating new RDDs.
- ✓ Transforming existing RDDs.
- ✓ Calling operations on RDDs to compute a result.

1. What is RDD?

- ✓ RDD stands for "Resilient Distributed Dataset".
- ✓ It is the fundamental data structure of Apache Spark.

- ✓ Definition 1:
  - A group of items distributed in cluster.

- ✓ Definition 2:
  - A collection of partitions in cluster.

- ✓ Definition 3:
  - RDD is an immutable collection of objects distributed in many nodes that can be processed in parallel.

- ✓ Definition 4:
  - RDDs are immutable, that means cannot be modified once created.

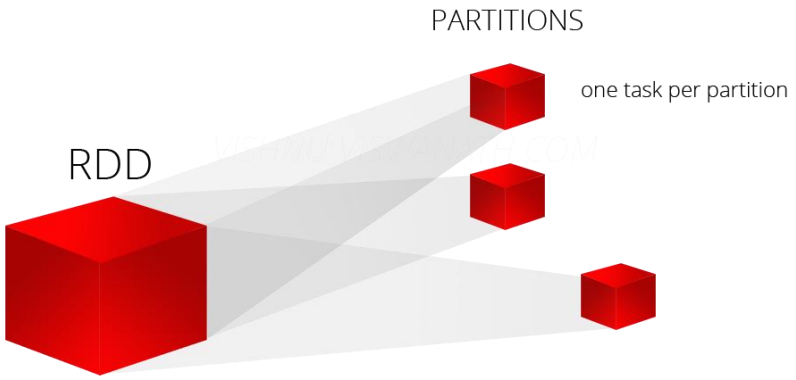
RDD full form:

Resilient	Distributed	Dataset
Fault-tolerant with the help of RDD lineage graph(DAG), able to re-compute missing or damaged partitions due to node failures.	Data distributed on multiple nodes.	Load the data set externally which can be either JSON file, CSV file, text file or database.

RDD computation

- ✓ Every dataset in RDD is logically partitioned across many nodes, so they can be computed parallelly on different nodes of the cluster.

Partitions in RDD



- ✓ RDDs are divided into smaller chunks called Partitions.
- ✓ RDD is logically partitioned in a cluster
- ✓ When you execute some action, a task is launched per partition.
- ✓ So, it means, the more number of partitions brings more parallelism.
- ✓ Spark achieves parallelism by using partitions.
- ✓ Spark automatically decides the number of partitions to RDD.
- ✓ We can specify the number of partitions to RDD while creating.

Creating RDD

- ✓ There are two ways to create RDDs in Spark such as,
  - Parallelized Collections.
  - From External Datasets.

Caching

- ✓ Spark RDD can be cached and partitioned.
- ✓ If we are using RDD many times, then it's good to cache the RDD.
- ✓ The reason behind for partitioning is to bring the parallelism.
- ✓ One task per one partition

Persistent

- ✓ Programmers can use persist method to store RDD for future operations.
- ✓ By default, spark uses in-memory to store RDD, but it can spill them to disk if not enough memory in RAM.

Coarse-grained

- ✓ Transform the whole dataset but not an individual element on the dataset.

Fine-grained

- ✓ Transform individual element on the dataset.

1. Read

- ✓ The read operation in RDD is either coarse grained or fine grained.

2. Write

- ✓ The write operation in RDD is coarse grained.

3. Fault-Recovery Mechanism

- ✓ The lost data can be easily recovered by using lineage graph.
- ✓ Each transformation creates new RDD.
- ✓ RDD is immutable so it's easy to recover.

4. If RAM is inefficient to store RDD then where it stores?

- ✓ If there is no space to store RDD in RAM, then RDD will be stored in disk.

5. RDD Features

5.1. In-memory Computation

- ✓ RDD stores intermediate results in RAM to compute instead of disk.

5.2. Lazy Evaluations

- ✓ All transformations are lazy, means they do not compute results immediately.
- ✓ Transformations compute results, only when action got triggered.

5.3. Fault Tolerance

- ✓ Spark RDDs are automatically rebuilt on lost data by using lineage, each RDD remembers how it was created from other datasets to recreate itself.

5.4. Immutability

- ✓ Once the RDD created then it is constant means cannot modify.
- ✓ Immutability means, if we are trying to apply operations on existing RDD then with those changes new RDD will be created.

5.5. Partition

- ✓ RDD is logically partitioned in cluster.
- ✓ Partitioning is the fundamental unit of parallelism in Spark.

5.6. Persistence

- ✓ Based on requirement programmers can store the RDD in-memory or disk for reusing purpose, this process is called persistence.

5.7. Coarse-grained Operations

- ✓ It applies to all elements in datasets through maps or filter or group by operation.

5.8. Location-Stickiness

- ✓ The DAGScheduler launches the task nearest to the partitions, it speeds up the computation.

6. Spark RDD Operations

- ✓ RDD supports two types of operations:

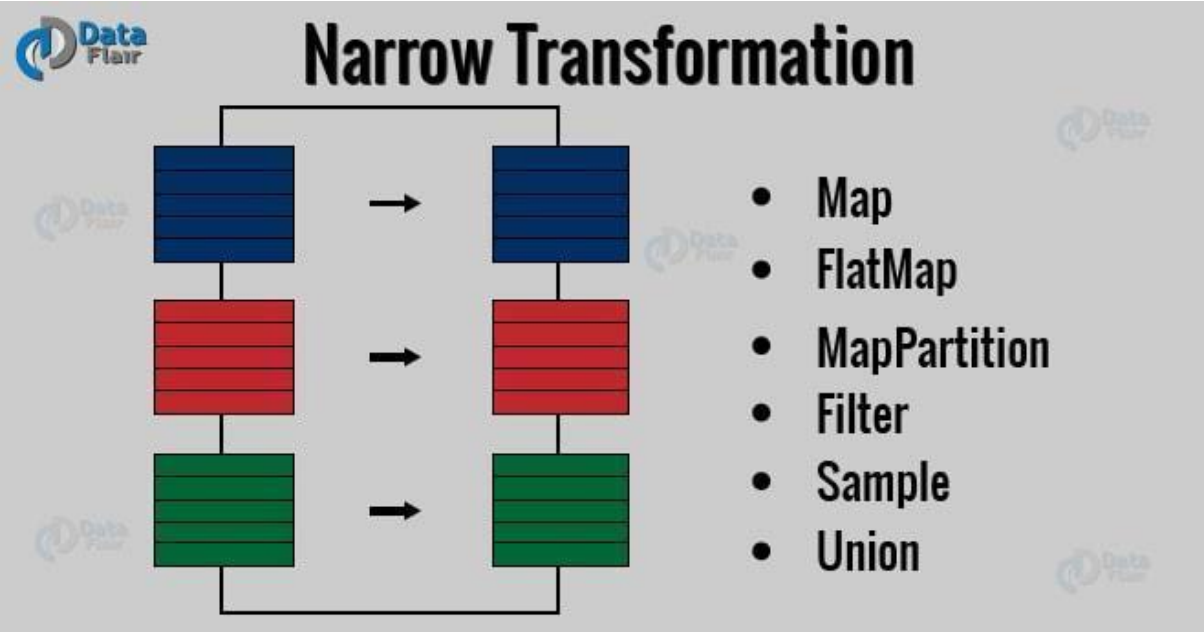
- Transformations.
- Actions.

6.1. Transformations

- ✓ Transformations are functions that take an RDD as the input and creates one or more new RDDs as the output.
- ✓ Transformations are executed when an action got triggered.
- ✓ Transformations are lazy operations.
- ✓ Transformations do not change the input RDD because RDDs are immutable.
- ✓ There are two kinds of transformations:
  - Narrow transformations.
  - Wide transformations.

6.1.1. Narrow Transformations

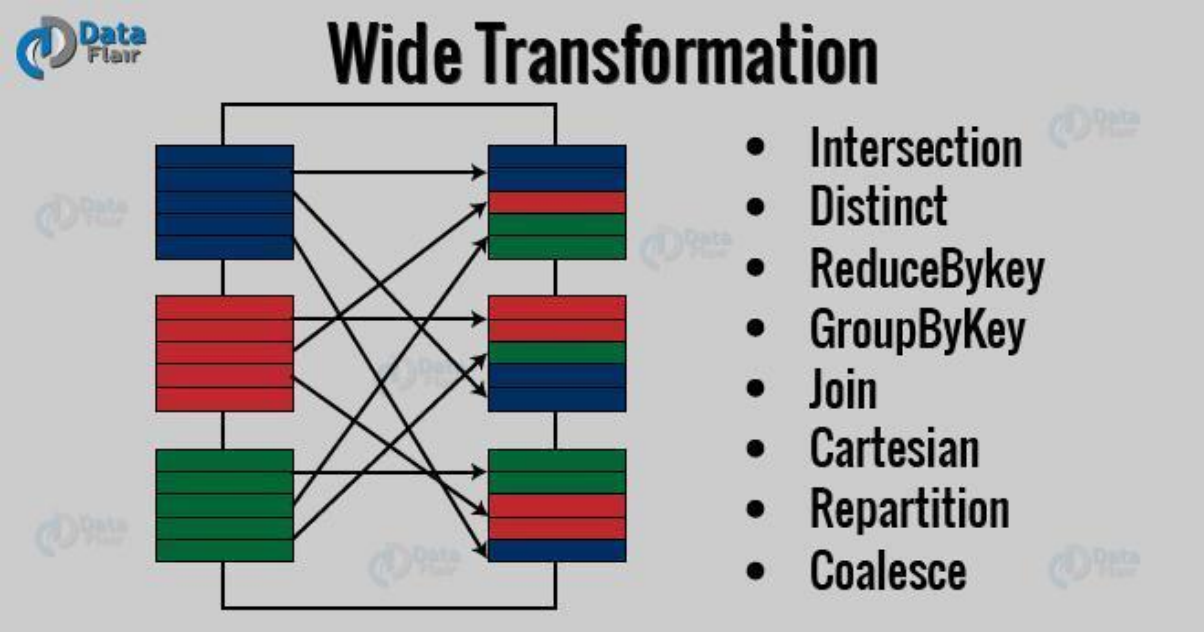
- ✓ It is the result of map, flatMap, MapPartition and filter methods.
- ✓ An output RDD partition may come from single parent partition.
- ✓ In Narrow transformations, data is from a single partition only.
- ✓ Spark groups narrow transformations as a stage known as pipelining.



6.1.2. Wide Transformations

- ✓ It is the result of groupByKey () and reduceByKey () functions.
- ✓ The output of RDD partition may come from many parent partitions.

- ✓ Wide transformations are also known as shuffle transformations because they may or may not depend on a shuffle.



Make a note

- ✓ Lineage graph is a dependency graph of all parallel RDDs of RDD.

6.2. Actions

- ✓ An action is a result of RDD computations.
- ✓ When action got triggered then,
  - It loads data into original RDD by using lineage graph and
  - Executes all intermediate transformations and return final results to Driver program or write it out to file system.
- ✓ An Action is one of the ways to send result from executors to the driver.
- ✓ Few examples of actions
  - first (),
  - take (),
  - reduce (),
  - collect (),
  - count () etc.

Review done till here - 1202201811pm

7. Limitation of RDD

7.1. No inbuilt optimization engine

- ✓ RDDs are unfit when working with structured data.

- ✓ RDDs cannot take advantages of catalyst optimizer and Tungsten execution engine.
- ✓ Developers need to optimize each RDD based on its attributes.

7.2. RDDs are unable to handle structured data

- ✓ RDDs are unable to infer schema for data.
- ✓ While working with RDD, developers need to write the schema explicitly.

7.3. Performance limitation

- ✓ RDDs involve the overhead of Garbage Collection
- ✓ Java serialization is required which is expensive when data grows.

RDD Examples: To-Do List

Review done till here - 1202201812pm

Creating RDDs

1. How to Create RDDs in Apache Spark?

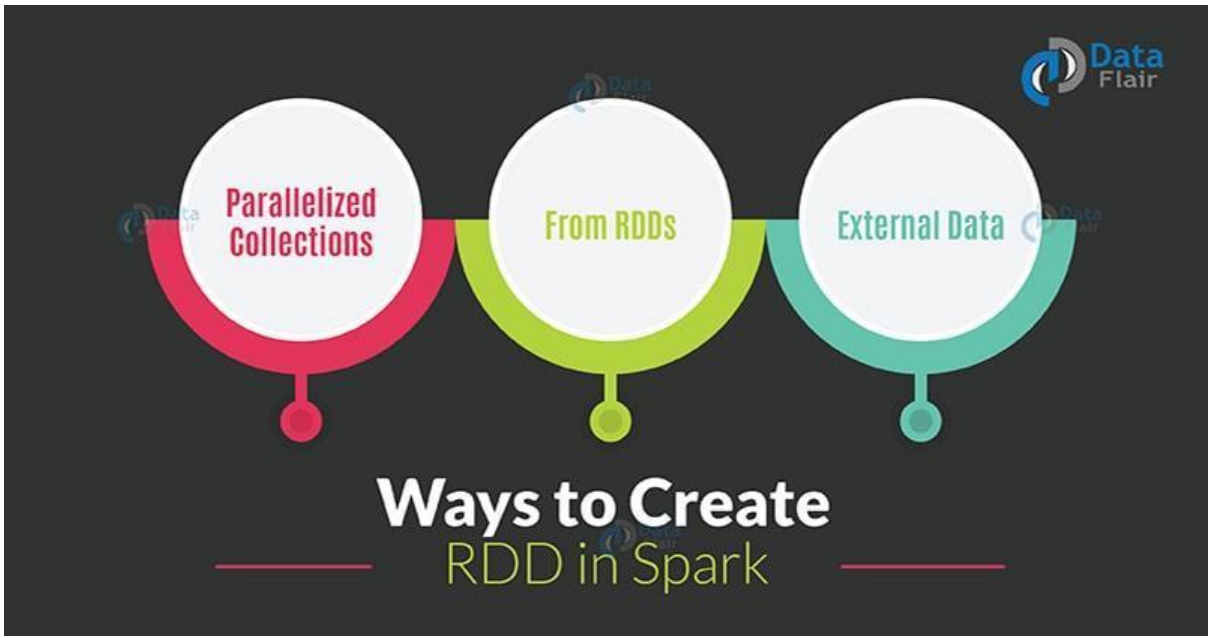
1.1. Parallelized collection (parallelizing)

1.2. External Datasets (Referencing a dataset)

i) csv (String path)

ii) json (String path)

iii) textFile (String path)



Review done till here - 1202201812pm

Spark Shell

- ✓ Spark shell is an interactive environment, by using that we can create and execute the logic quickly.

1. How to Create RDDs in Apache Spark?

There are two ways to create an RDD in Spark.

- ✓ By parallelize () method which is available in SparkContext.
- ✓ From any external storage systems (HDFS, HBase, any database).

2.1. parallelize () method

- ✓ In the initial stage when we are learning Spark,
  - RDDs are generally created by parallelize () method.
- ✓ This method quickly creates RDD and performs operations.
- ✓ This method is rarely used outside testing and prototyping because this method requires entire dataset on one machine.

Consider the following example of sortByKey ().

In this, the data to be sorted is taken through parallelized collection:

```
val seqCol = Seq (("maths",52), ("english",75), ("science",82))

val data = sc. parallelize (seqCol)

val sorted = data. sortByKey ()

sorted. foreach(println)
```

Points to be noted:

- ✓ As we know RDD is a group of partitions, spark will run one task for each partition of cluster.
- ✓ We require two to four partitions for each CPU in cluster.
- ✓ Spark sets number of partition based on our cluster.

Can we set number of partitions to RDD?

- ✓ We can manually set the number of partitions to RDD.
- ✓ This is achieved by passing number of partition as second parameter to parallelize method.
  - e.g. sc. parallelize (seqCol, 10), here we have manually given number of partition as 10.
- ✓ We can increase or decrease the partitions by using coalesce and repartition methods (we will learn this in upcoming chapter)

```
val a = Array ("jan", "feb", "mar", "april", "may", "jun")

val rdd1 = sc.parallelize(a, 3)

val result = rdd1.coalesce(2)

result.foreach(println)
```

2.2. Creating RDD by using External Datasets (Referencing a dataset)

- ✓ We can create RDD by using external datasets as well, means HDFS, LFS, Cassandra, HBase, and any other database.
- ✓ In this scenario the data is loaded from the external dataset.
- ✓ If the data exists in HDFS and LFS, in this case to create RDD we need to call textFile method of SparkContext.
  - val data = sc.textFile("file:///home/nireekshan/input") --- LFS
  - val data = sc.textFile("/hdfc/path") ---- HDFS
- ✓ This method takes URL of the file and read it as a collection of line.

Point to be noted:

- ✓ By default, spark reads below format files,
  - Json
  - Text

To-Do --- Need to write the examples

Review done till here : 140220184pm

RDD Persistence and Caching Mechanism in Apache Spark

1. What is RDD Persistence and Caching in Spark?
2. Need of Persistence in Apache Spark

3. Benefits of RDD Persistence in Spark

4. How to Un-persist RDD in Spark?

1. What is RDD Persistence and Caching in Spark?

- ✓ Spark RDD persistence is an optimization technique, it saves the result of RDD evaluation.
- ✓ Using this we can save the intermediate result, so that we can use it for future operations if it is required.
- ✓ It reduces the computation overhead.
- ✓ We can make persisted RDD through cache () and persist () methods.

cache ()

- ✓ When we use the cache () method we can store all the RDD in-memory.
- ✓ We can persist the RDD in memory and use it efficiently across parallel operations.

persist ()

- ✓ When we use the persist () method we can store all the RDD in-memory.
- ✓ persist () method having different storage levels.

Diff b/w cache () and persist ()

- ✓ The default storage level for cache () is MEMORY_ONLY.
- ✓ Persist () method having different storage levels.

How cache and persist works?

- ✓ When we persist RDD,
  - Each node stores partitions of RDD
  - That RDD computes in In-memory.
  - This process makes it reusable for future use.
- ✓ This process speeds up the further computation ten times.
- ✓ When the RDD is computed for the first time, it is kept in memory on the node.
- ✓ The cache memory of the Spark is fault tolerant.
- ✓ Whenever any partition of RDD is lost, it can be recovered by transformation operation that originally created it.

2. Need of Persistence in Apache Spark

- ✓ While we are doing computations if we are using same RDDs multiple times then it’s good to cache or persist based on requirement.

- ✓ If we didn't cache then this task may consume time and memory, especially for iterative algorithms that look at data multiple times.
- ✓ To solve this problem of repeated computation the technique of persistence came into the picture.

3. Benefits of RDD Persistence in Spark

- Time efficient
- Cost efficient
- Improve the execution time.

5. Storage levels of Persisted RDDs

- ✓ persist () method having different storage levels to store persisted RDD,
  - MEMORY_ONLY
  - MEMORY_AND_DISK
  - MEMORY_ONLY_SER
  - MEMORY_AND_DISK_SER
  - DISK_ONLY

6. How to Un-persist RDD in Spark?

- ✓ Spark monitors the cache of each node.
- ✓ Automatically drop out the old data partition in the LRU (least recently used) fashion algorithm.
- ✓ It spills out that data from the cache.
- ✓ We can also remove the cache manually using RDD.unpersist() method.

To Do List

2. Apache Spark RDD Operations

3. RDD Transformation

    3.1. map(func) done

    3.2. flatMap() done

    3.3. filter(func) done

    3.4. mapPartitions(func) not done

    3.5. mapPartitionWithIndex() not done

    3.6. union(dataset) done

    3.7. intersection(other-dataset) done

    3.8. distinct() done

    3.9. groupByKey() done

    3.10. reduceByKey(func, [numTasks]) done

    3.11. sortByKey() done

    3.12. join() not done

    3.13. coalesce() not done

4. RDD Action

    4.1. count() done

    4.2. collect()done

    4.3. take(n) done

    4.4. top() done

    4.5. countByValue() done

    4.6. reduce() done

    4.7. fold() not done

    4.8. aggregate() not done

    4.9. foreach() done

Map vs FlatMap

To-Do List write all the tables from Spark Learning

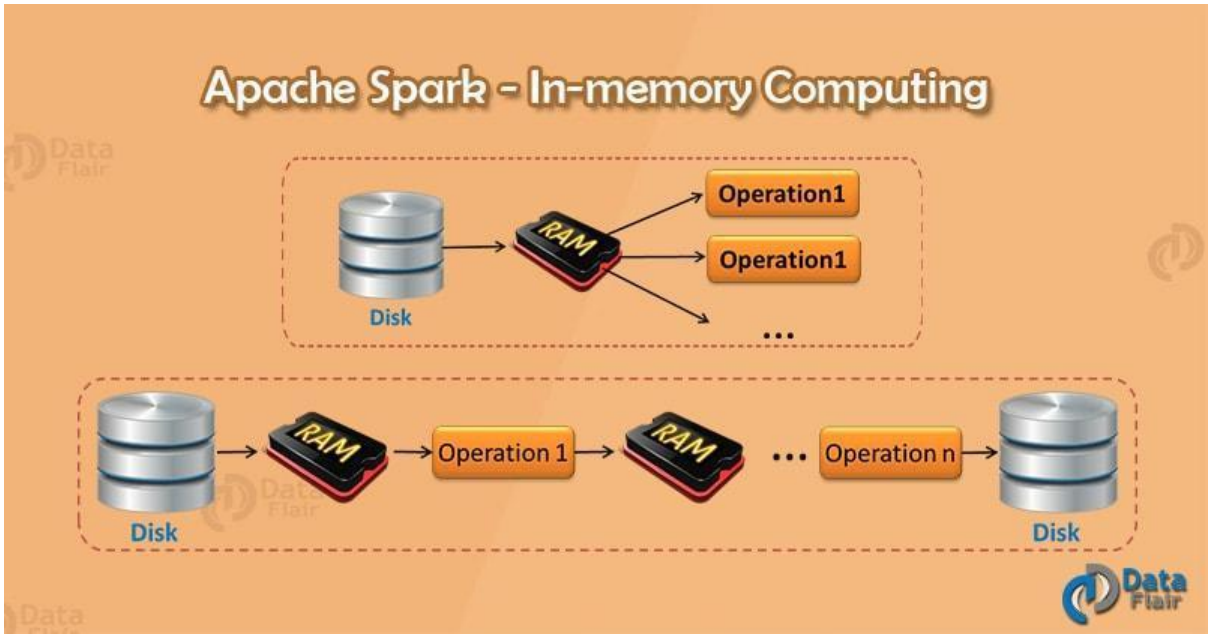
Table 3-2. Basic RDD transformations on an RDD containing {1, 2, 3, 3}

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

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)}`

1. What is In-memory Computing?
2. Introduction to Spark In-memory Computing
3. Storage levels of RDD persist () in Spark
4. Advantages of In-memory Processing



1. What is In-memory Computing?

- ✓ In in-memory computation, the data is kept in RAM (Random access memory) to improve the performance instead of some disk.
- ✓ Advantages of in-memory computation are,
  - RAM storage
  - Parallel distributed processing.

2. Introduction to Spark In-memory Computing

- ✓ Storing the data in-memory improves the performance.
- ✓ The main abstraction of Spark is its RDDs.
- ✓ RDDs are cached using the cache () or persist () method.
- ✓ The in-memory capability of Spark is good for machine learning and micro-batch processing.
- ✓ It provides faster execution for iterative jobs.
- ✓ The difference between cache () and persist () is that using cache () the default storage level is MEMORY_ONLY while using persist () we can use various storage levels.

4. Storage levels of RDD persist () in Spark

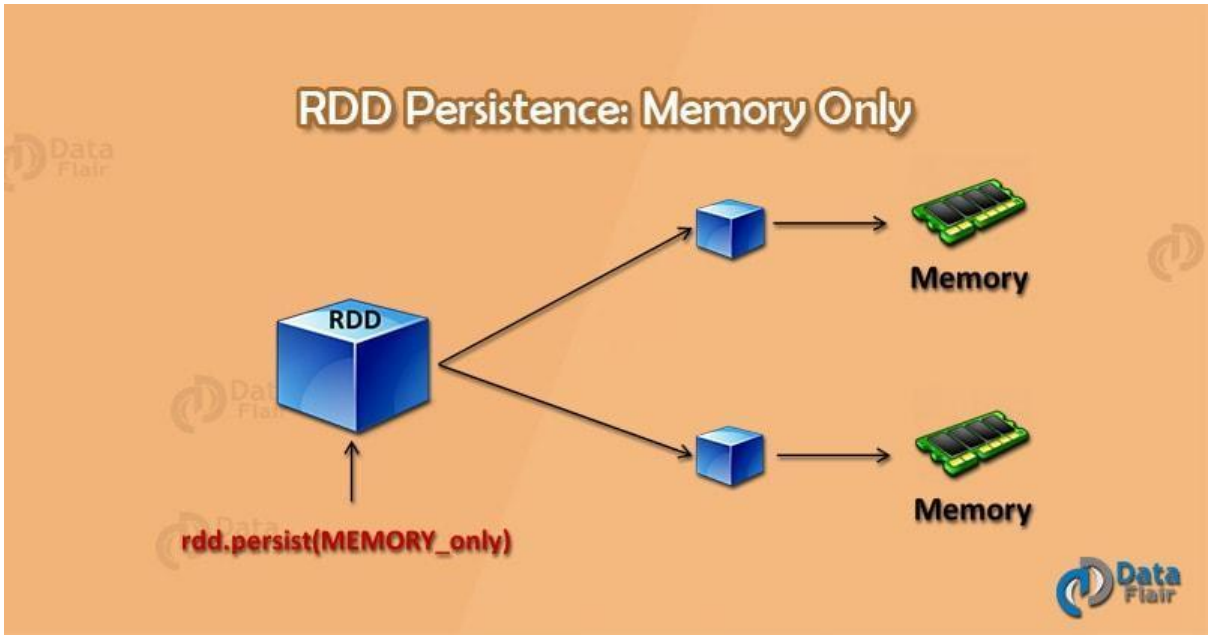
The various storage level of persist () method in Apache Spark RDD are:

- ✓ MEMORY_ONLY

- ✓ MEMORY_AND_DISK
- ✓ MEMORY_ONLY_SER
- ✓ MEMORY_AND_DISK_SER
- ✓ DISK_ONLY
- ✓ MEMORY_ONLY_2 and MEMORY_AND_DISK_2

4.1. MEMORY_ONLY

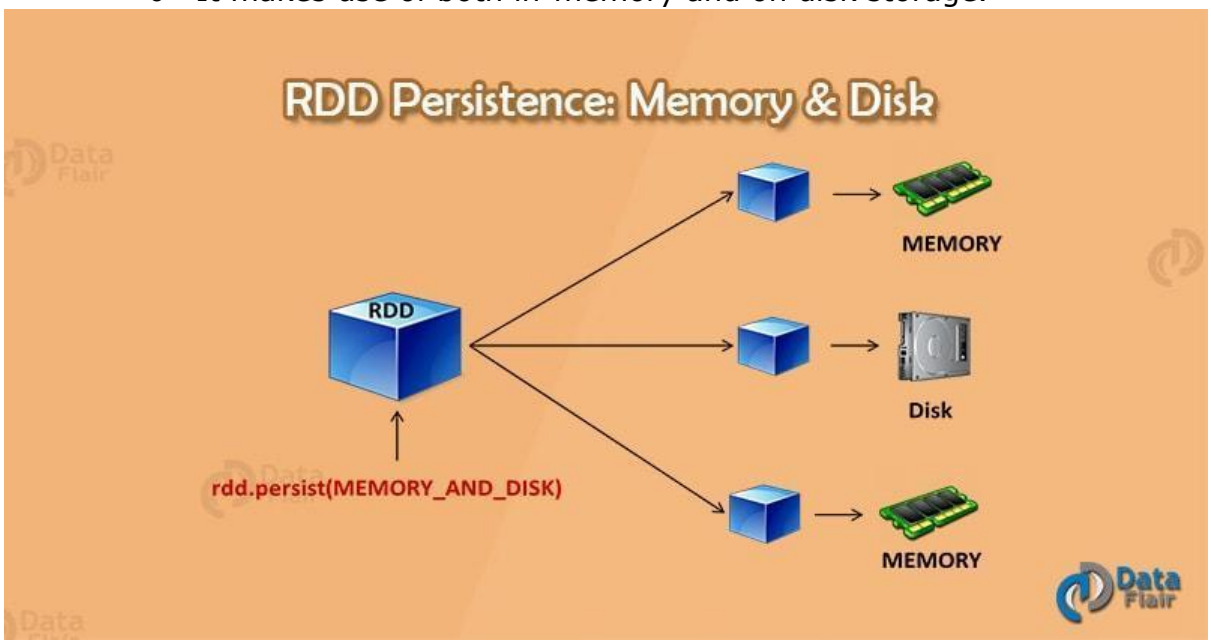
- ✓ In this level, RDD is stored as deserialized Java object in the JVM.
- ✓ If RDD does not fit in memory, then the remaining will be recomputed each time they are needed.
- ✓ In this level,
  - The space used for storage is very high.
  - The CPU computation time is low.
  - The data is stored in-memory.
  - It does not make use of the disk.



4.2. MEMORY_AND_DISK

- ✓ In this level, RDD is stored as deserialized Java object in the JVM.
- ✓ If the full RDD does not fit in memory then the remaining partition is stored on disk, instead of recomputing it every time when it is needed.

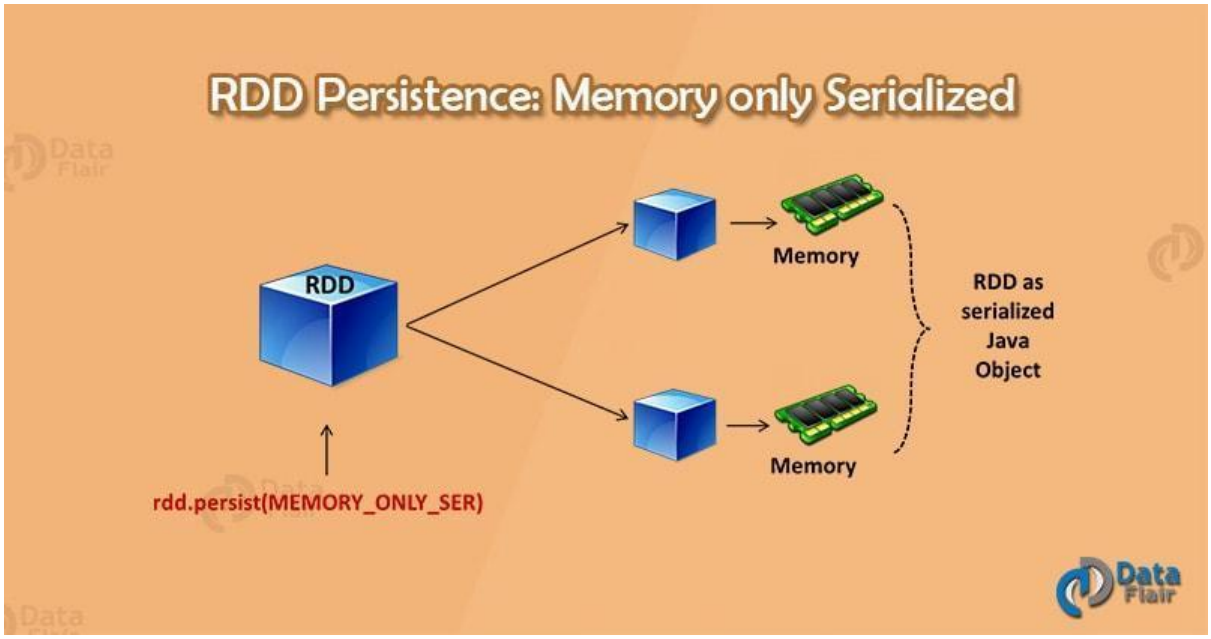
- ✓ In this level,
  - The space used for storage is high
  - The CPU computation time is medium
  - It makes use of both in-memory and on disk storage.



4.3. MEMORY_ONLY_SER

- ✓ RDD is stored as serialized Java object (one-byte array per partition).
- ✓ It is like MEMORY_ONLY but more space efficient as compared to deserialized objects, especially when it uses fast serializer.

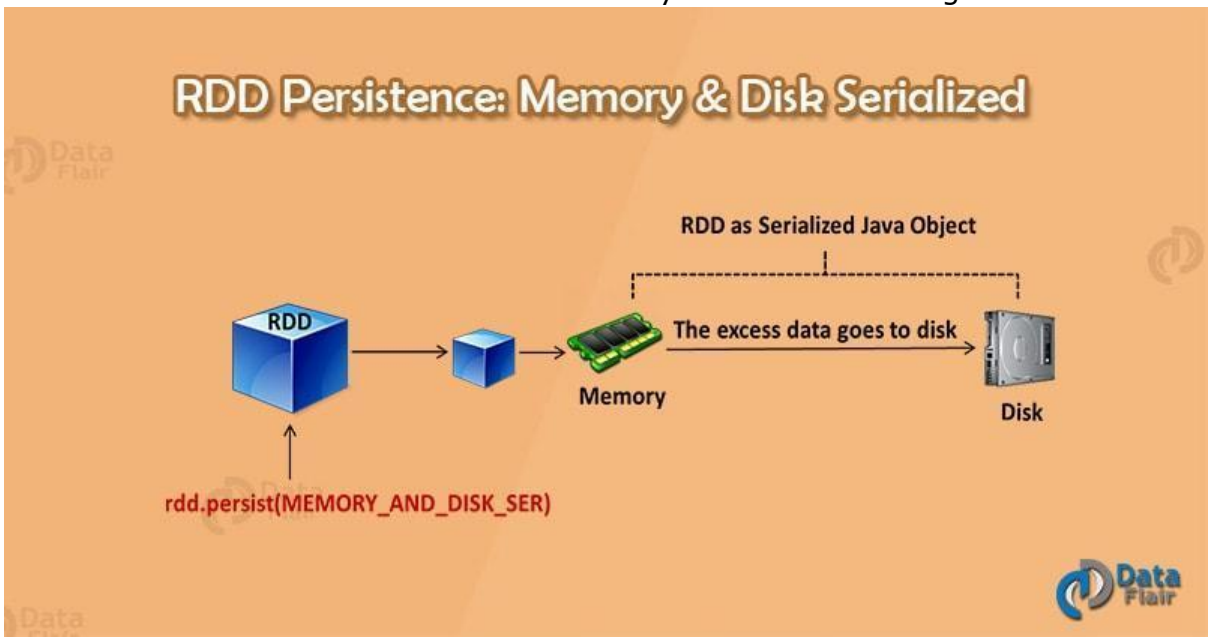
- ✓ It increases the overhead on CPU.
- ✓ In this level,
  - The storage space is low
  - The CPU computation time is high
  - The data is stored in-memory.
- ✓ It does not make use of the disk.



4.4. MEMORY_AND_DISK_SER

- ✓ This level stores RDD as serialized JAVA object.

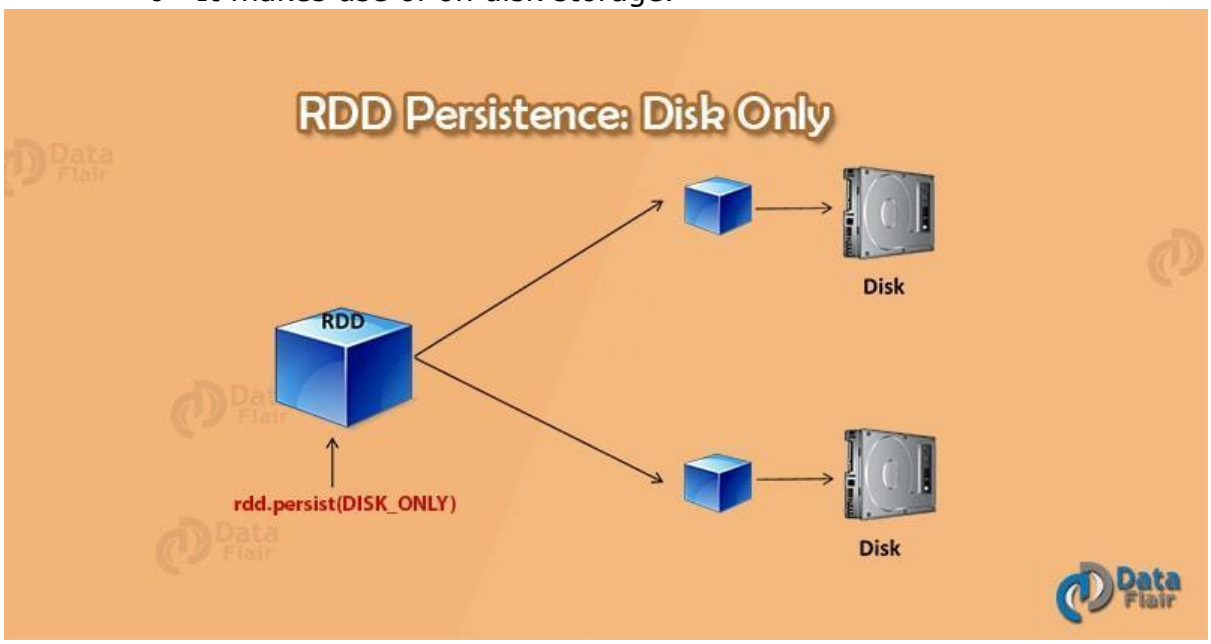
- ✓ It is same as MEMORY_ONLY_SER, If the full RDD does not fit in the memory then it stores the remaining partition on the disk, instead of recomputing it every time when we need.
- ✓ In this storage level,
  - The space used for storage is low
  - The CPU computation time is high
  - It makes use of both in-memory and on disk storage.



4.5. DISK_ONLY

- ✓ This storage level stores the RDD partitions only on disk.

- ✓ In this storage level,
  - The space used for storage is low
  - The CPU computation time is high
  - It makes use of on disk storage.



4.6. MEMORY_ONLY_2 and MEMORY_AND_DISK_2

- ✓ It is like MEMORY_ONLY and MEMORY_AND_DISK.
- ✓ The only difference is that each partition gets replicate on two nodes in the cluster.

5. Advantages of In-memory Processing

- ✓ The data becomes highly accessible.
- ✓ The computation speed of the system increases.
- ✓ It is economic, as the cost of RAM has fallen over a period.

Summary of the story

Level	Space used	CPU time	In memory	On disk	Comments
MEMORY_ONLY	High	Low	Y	N	
MEMORY_ONLY_SER	Low	High	Y	N	
MEMORY_AND_DISK	High	Medium	Some	Some	Spills to disk if there is too much data to fit in memory.
MEMORY_AND_DISK_SER	Low	High	Some	Some	Spills to disk if there is too much data to fit in memory. Stores serialized representation in memory.
DISK_ONLY	Low	High	N	Y	

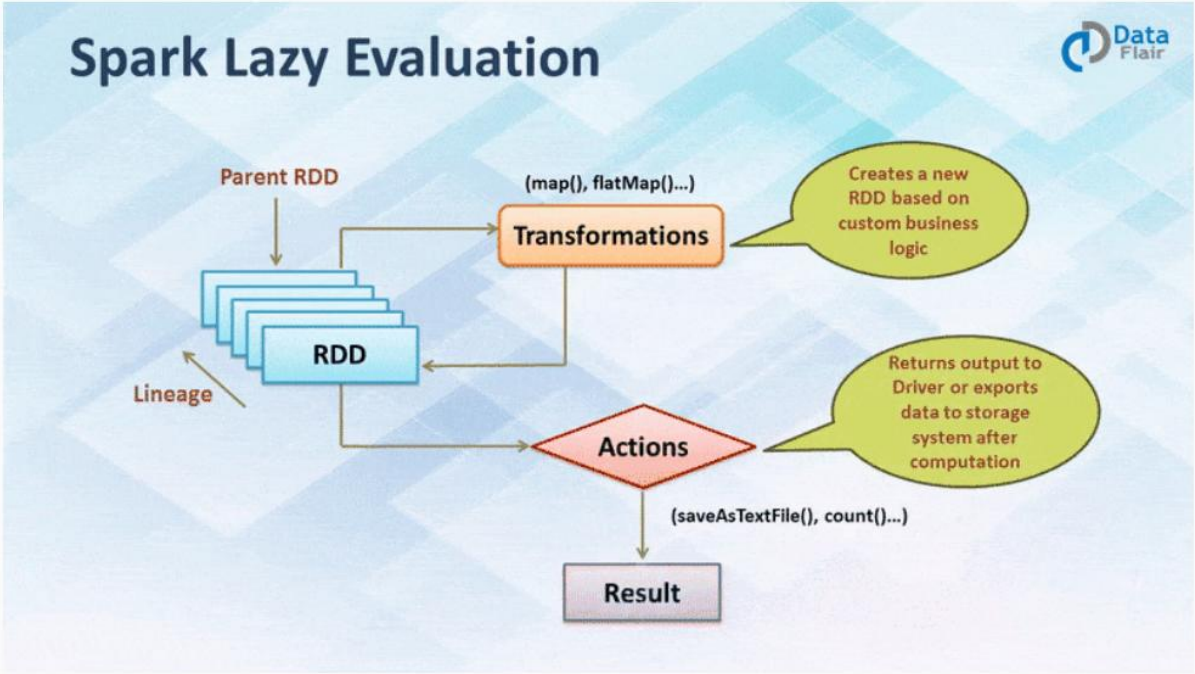
Review done till here : 140220187pm

Lazy Evaluation in Apache Spark – A Quick guide 1

1. What is Lazy Evaluation in Apache Spark?
2. Advantages of Lazy Evaluation in Spark Transformation
    a. Increases Manageability
    b. Saves Computation and increases Speed
    c. Reduces Complexities

1. What is Lazy Evaluation in Apache Spark?

- ✓ Lazy evaluation means that the execution will not start until an action is triggered.



2. Advantages of Lazy Evaluation in Spark Transformation

a. Increases manageability

- ✓ users can organize their program into smaller operations.
- ✓ It reduces the number of passes on data by grouping operations.

b. Saves computation and increases Speed

- ✓ Lazy evaluation plays a key role in saving calculation overhead.
- ✓ Since only necessary values get compute.
- ✓ It saves the round-trip operations between driver and cluster, thus speeds up the process.

c. Reduces Complexities

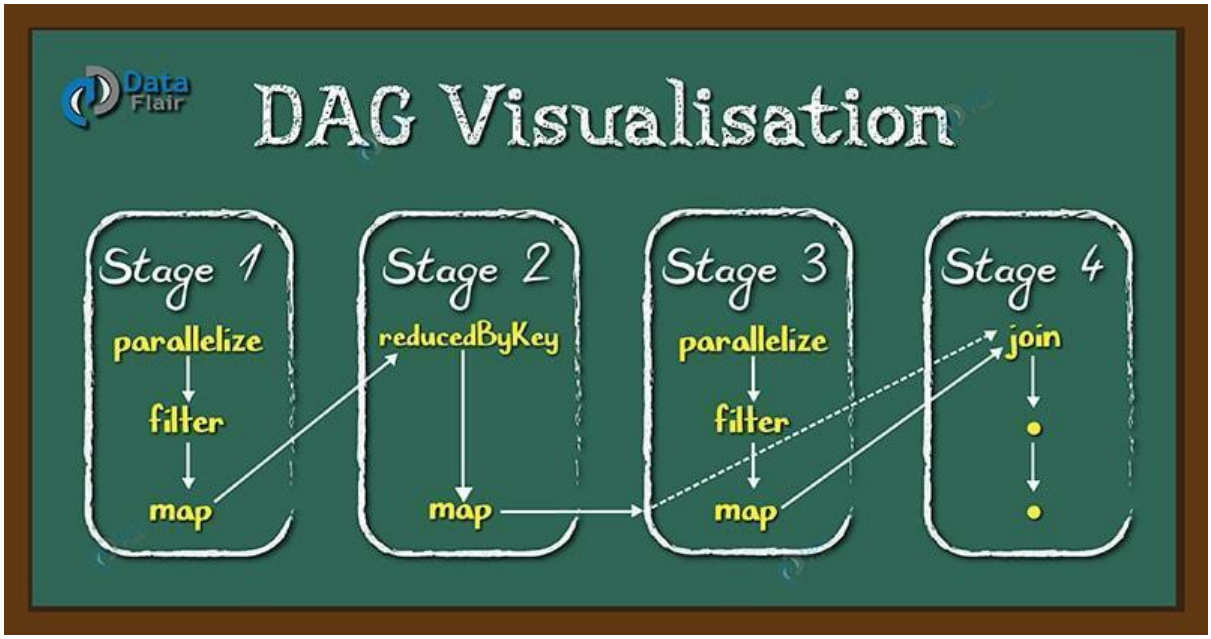
- ✓ The two main complexities of any operation are
  - Time

  - space complexity.
- ✓ Using lazy evaluation, we can overcome both.
- ✓ Since we do not execute every operation, Hence, the time gets saved.
- ✓ The action is triggered only when the data is required, it reduces overhead.

Review done till here : 140220187.30pm

Directed Acyclic Graph DAG in Apache Spark

1. What is DAG in Apache Spark?
2. Need of Directed Acyclic Graph in Spark
3. How DAG works in Spark?
4. How is Fault tolerance achieved through DAG?
5. Working of DAG Optimizer in Spark
6. Advantages of DAG in Spark



1. What is DAG in Apache Spark?

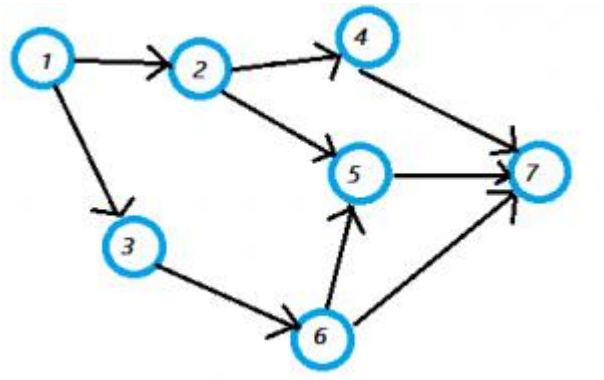
- ✓ The full form of DAG is Directed Acyclic Graph.
- ✓ DAG is a set of Vertices and Edges,
  - Vertices represent the RDDs.
  - Edges represent the Operation to be applied on RDD.
- ✓ Every edge is directed from earlier to later in the sequence.
- ✓ On calling of Action, the created DAG is submitted to DAG Scheduler.
- ✓ The Scheduler splits the Spark RDD into stages based on various transformation applied.
- ✓ DAG allows the user to dive into the stage and expand on any stage.
- ✓ Each stage is comprised of tasks, based on the partitions of the RDD, which will perform computation in parallel.

Why DAG required?

- ✓ DAG works better than MapReduce model.

- ✓ The picture of DAG becomes clear for more complex jobs.

Make a note: A directed acyclic graph is an acyclic graph that has a direction.



The parts of the above graph are:

Integer = the set for the Vertices.

Vertices set = {1,2,3,4,5,6,7}.

Edge set = {(1,2), (1,3), (2,4), (2,5), (3,6), (4,7), (5,7), (6,7)}.

A directed acyclic graph has a topological ordering.

2. Need of Directed Acyclic Graph in Spark

2. 1 Limitations of MapReduce

- ✓ The limitations of MapReduce became a key point to introduce DAG in Spark.
- ✓ While processing MapReduce,
  - Reading the data from HDFS,
  - Applying Map and Reduce operations.
  - Results written back to HDFS

- ✓ Each operation in MapReduce is independent of other.
- ✓ HADOOP has no idea of which Map reduce operation would come in next.
- ✓ Sometimes for some iteration, it is irrelevant to read and write back the immediate result between two map-reduce jobs.
- ✓ In such case, the memory in stable storage (HDFS) or disk memory gets wasted.
- ✓ In multiple-step, till the completion of the previous job all the jobs are blocked from the beginning.
- ✓ As a result, complex computation can require a long time with small data volume.

How Spark achieved?

- ✓ In Spark, a DAG (Directed Acyclic Graph) of consecutive computation stages is formed.
- ✓ In this way, the execution plan is optimized,
  - minimize shuffling data around.
- ✓ In contrast, it is done manually in MapReduce by tuning each MapReduce step.

Spark Application

- ✓ The Spark application is a self-contained computation that runs user-supplied code to compute a result.
- ✓ A Spark application can have processes running on its behalf even when it's not running a job.

Task

- ✓ A task is a unit of work that is sent to the executor.
- ✓ Each stage has some task, one task per partition.
- ✓ The Same task is done over different partitions of RDD.

Job

- ✓ The job is parallel computation consisting of multiple tasks that get spawned in response to actions in Apache Spark.

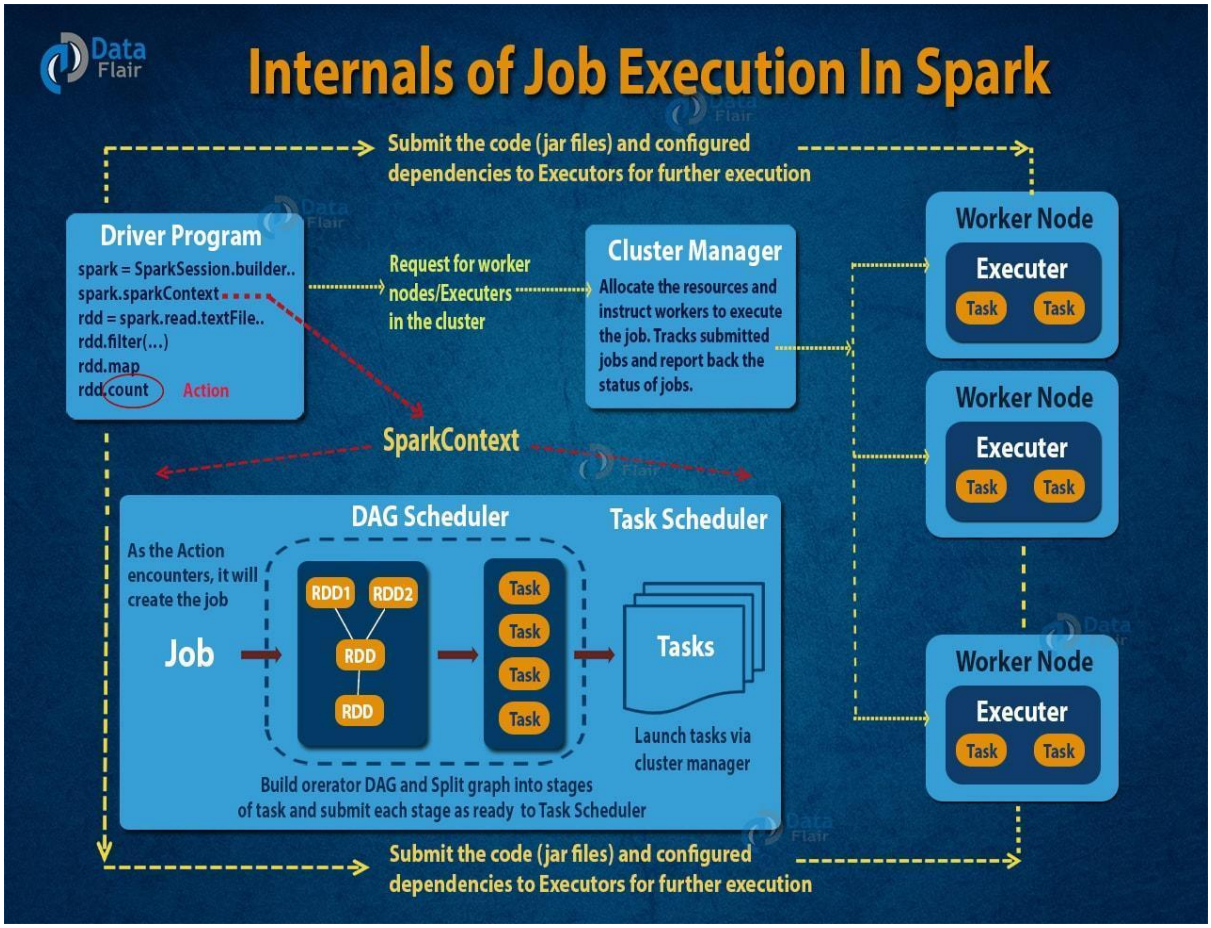
Stage

- ✓ Each job gets divided into smaller sets of tasks called stages that depend on each other.
- ✓ Stages are classified as computational boundaries.
- ✓ All computation cannot be done in single stage.
- ✓ It is achieved over many stages.

4. How DAG works in Spark?

- ✓ The interpreter is the first layer in DAG.
- ✓ Spark uses Scala interpreter to interprets the code.
- ✓ Spark creates an operator graph for your written code.
- ✓ When an Action is called on Spark RDD at a high level then Spark submits the operator graph to the DAG Scheduler.
- ✓ Operators are divided into stages of the task in the DAG Scheduler.

- ✓ A stage contains task based on the partition of the input data.
- ✓ The DAG scheduler pipelines operators together.
- ✓ For example, map operators are scheduled in a single stage.
- ✓ The stages are passed on to the Task Scheduler.
- ✓ It launches task through cluster manager.
- ✓ The dependencies of stages are unknown to the task scheduler.
- ✓ The workers execute the task on the slave.



- ✓ At higher level, two types of RDD transformations can be applied:
  - Narrow transformation (e.g. map (), filter () etc.)
  - Wide transformation (e.g. reduceByKey ()).
- ✓ Narrow transformation does not require the shuffling of data across a partition, the narrow transformations will be grouped into single stage.
- ✓ While in wide transformation data shuffling is required.
- ✓ Hence, Wide transformation results in stage boundaries.
- ✓ Each RDD maintains a pointer to one or more parent along with metadata about what type of relationship it has with the parent.
- ✓ For example, if we call val b = a.map () on an RDD, the RDD b keeps a reference to its parent RDD a, that's an RDD lineage.

5. How Fault tolerance achieved through DAG?

- ✓ RDD is split into the partition and each node is operating on a partition at any point in time.
- ✓ Here, the series of functions are executing on a partition of the RDD.
- ✓ These operations are composed together and Spark execution engine view these as DAG (Directed Acyclic Graph).
- ✓ When any node crashes in the middle of any operation say O3 which depends on operation O2, which in turn O1.
- ✓ The cluster manager finds out the node is dead and assign another node to continue processing.
- ✓ This node will operate on the partition of the RDD and the series of operation that it must execute (O1->O2->O3).
- ✓ Now there will be no data loss.

6. Working of DAG Optimizer in Spark

- ✓ The DAG is optimized by rearranging and combining operators wherever possible.
- ✓ For, example if we submit a spark job which has a map () operation followed by a filter operation.
- ✓ The DAG Optimizer will rearrange the order of these operators since filtering will reduce the number of records to undergo map operation.

7. Advantages of DAG in Spark

- ✓ The lost RDD can be recovered using the DAG.
- ✓ Map Reduce has just two queries the map and reduce but in DAG we have multiple levels.
- ✓ So, to execute SQL query, DAG is more flexible.
- ✓ DAG helps to achieve fault tolerance, means the lost data can be recovered.
- ✓ It can do a better global optimization than a system like Hadoop MapReduce.

Review done till here : 21022018 2:50pm

How Apache Spark Works – Run-time Spark Architecture

2. Internals of How Apache Spark works?

- ✓ Spark uses master/slave architecture i.e. one central coordinator and many distributed workers.
- ✓ Here, the central coordinator is called the driver.
- ✓ The driver runs in its own Java process.
- ✓ Driver communicate with large number of distributed workers called executors.
- ✓ Each executor is a separate java process.
- ✓ A Spark Application is a combination of driver and its own executors.
- ✓ With the help of cluster manager, a Spark Application is launched on a set of machines.
- ✓ Standalone Cluster Manager is the default built in cluster manager of Spark.
- ✓ Apart from its built-in cluster manager, Spark also works with some open source cluster manager like Hadoop Yarn, Apache Mesos etc.

4. Components of Spark Run-time Architecture of Spark

4.1. Apache Spark Driver

- ✓ The main () method of the program runs in the driver.

- ✓ The driver is the process that runs the user code that creates RDDs, and performs transformation and action, and creates SparkContext.
- ✓ When the Spark Shell is launched, this signifies that we have created a driver program.
- ✓ On the termination of the driver, the application is finished.
- ✓ The driver program splits the Spark application into the task and schedules them to run on the executor.
- ✓ The task scheduler resides in the driver and distributes task among workers.
- ✓ The two main key roles of drivers are:
  - Converting user program into the task.
  - Scheduling task on the executor.
- ✓ The structure of Spark program at a higher level is:
  - RDDs are created from some input data, derive new RDD from existing using various transformations, and then after it performs an action to compute data.
  - In Spark program, the DAG (directed acyclic graph) of operations are created implicitly.
  - And when the driver runs, it converts that Spark DAG into a physical execution plan.

4.2. Apache Spark Cluster Manager

- ✓ Spark relies on cluster manager to launch executors and in some cases, even the drivers are launched through it.
- ✓ It is a pluggable component in Spark.
- ✓ On the cluster manager, jobs and action within a spark application are scheduled by Spark Scheduler in a FIFO fashion.
- ✓ Alternatively, the scheduling can also be done in Round Robin fashion.
- ✓ The resources used by a Spark application can be dynamically adjusted based on the workload.
- ✓ Thus, the application can free unused resources and request them again when there is a demand.
- ✓ This is available on all coarse-grained cluster managers, i.e. standalone mode, YARN mode, and Mesos coarse-grained mode.

4.3. Apache Spark Executors

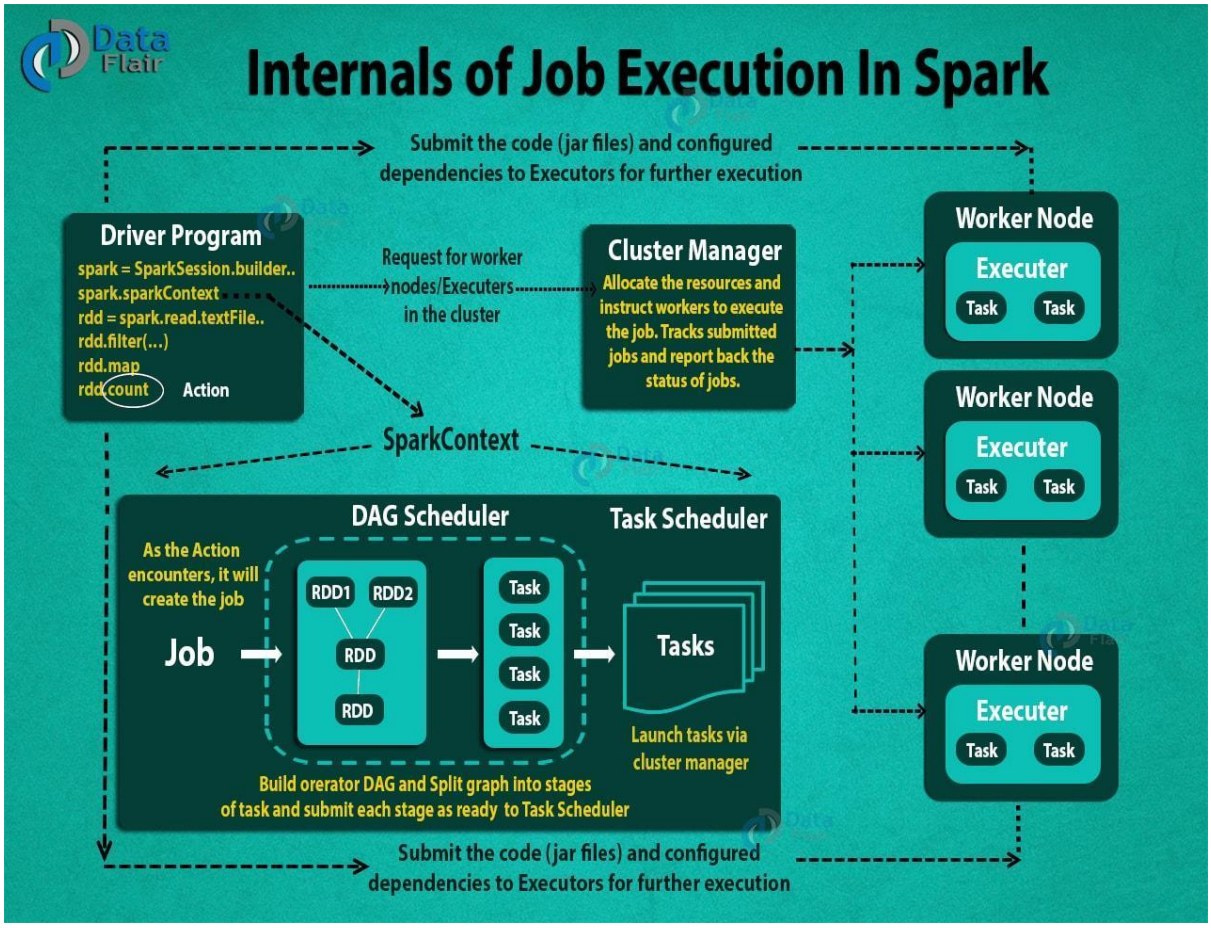
- ✓ The individual task in the given Spark job runs in the Spark executors.
- ✓ Executors are launched once in the beginning of Spark Application and then they run for the entire lifetime of an application.
- ✓ Even if the Spark executor fails, the Spark application can continue with ease.
- ✓ There are two main roles of the executors:
  - Runs the task that makes up the application and returns the result to the driver.
  - Provide in-memory storage for RDDs that are cached by the user.

5. How to launch a Program in Spark?

- ✓ Despite using any cluster manager, Spark comes with the facility of a single script that can be used to submit a program, called as spark-submit.
- ✓ It launches the application on the cluster.

- ✓ There are various options through which spark-submit can connect to different cluster manager and control how many resources our application gets.
- ✓ For some cluster managers, spark-submit can run the driver within the cluster (e.g., on a YARN worker node), while for others, it can run only on your local machine.

6. How to Run Apache Spark Application on a cluster



- ✓ Using spark-submit, the user submits the application.
- ✓ In spark-submit, the main () method specified by the user is invoked.
- ✓ It also launches the driver program.
- ✓ The driver program asks for the resources to the cluster manager that is required to launch executors.
- ✓ The cluster manager launches executors on behalf of the driver program.
- ✓ The driver process runs with the help of user application.
- ✓ Based on the actions and transformation on RDDs, the driver sends work to executors in the form of tasks.
- ✓ The executors process the task and the result is sent back to the driver through cluster manager.

Spark Performance Tuning-Learn to Tune Apache Spark Job

1. What is Performance Tuning in Apache Spark?

2. Data Serialization in Spark
3. Memory Tuning in Spark
    a. Spark Data Structure Tuning
    b. Spark Garbage Collection Tuning
4. Memory Management in Spark
5. Determining Memory Consumption in Spark
6. Spark Garbage Collection Tuning
7. Other consideration for Spark Performance Tuning
    a. Level of Parallelism
    b. Memory Usage of Reduce Task in Spark
    c. Broadcasting Large Variables
    d. Data Locality in Apache Spark

2. What is Performance Tuning in Apache Spark?

- ✓ The process of adjusting settings to record for memory, cores, and instances used by the system is called as a tuning.
- ✓ This process guarantees that the Spark has optimal performance and prevents resource bottlenecks.

- ✓ Effective changes are made to each property and settings, to ensure the correct usage of resources based on system specific setup.
- ✓ Apache Spark has in-memory computation nature.
- ✓ As a result, resources in the cluster (CPU, memory etc.) may get bottlenecked.
- ✓ Sometimes to decrease memory usage RDDs are stored in serialized form.
- ✓ Data serialization plays important role in good network performance and can also help in reducing memory usage, and memory tuning.

If used properly, tuning can:

- ✓ Ensure proper use of all resources in an effective manner.
- ✓ Eliminates those jobs that run long.
- ✓ Improves the performance time of the system.
- ✓ Guarantees that jobs are on correct execution engine.

3. Data Serialization in Spark

- ✓ It is the process of converting the in-memory object to another format that can be used to store in a file or send over the network.
- ✓ It plays a mail role in the performance of any distributed application.
- ✓ The computation gets slower due to formats that are slow to serialize or consume large number of files.

Apache Spark gives two serialization libraries:

- ✓ Java serialization
- ✓ Kryo serialization

Java serialization

- ✓ Objects are serialized in Spark using an ObjectOutputStream framework, and can run with any class that implements java.io. Serializable.
- ✓ The performance of serialization can be controlled by extending java.io. Externalizable.
- ✓ It is flexible but slow and leads to large serialized formats for many classes.

Kryo serialization

- ✓ To serialize objects, Spark can use the Kryo library (Version 2).
- ✓ Although it is more compact than Java serialization, it does not support all Serializable types.
- ✓ For better performance, we need to register the classes in advance.
- ✓ We can switch to Kryo by initializing our job with SparkConf and calling-conf.set (“spark.serializer ”, “org.apache.spark.serializer.KyroSerializer”)

- ✓ We use the registerKryoClasses method, to register our own class with Kryo.
- ✓ In case our objects are large we need to increase spark. kryoserializer. buffer config.

- ✓ The value should be large so that it can hold the largest object we want to serialize.

4. Memory Tuning in Spark

- ✓ Consider the following three things in tuning memory usage:
  - Amount of memory used by objects (the entire dataset should fit in-memory)
  - The cost of accessing those objects
  - Overhead of garbage collection.
- ✓ The Java objects can be accessed but consume 2-5x more space than the raw data inside their field.

The reasons for such behaviour are:

- ✓ Every distinct Java object has an "object header".
- ✓ The size of this header is 16 bytes.
- ✓ Sometimes the object has little data in it, thus in such cases, it can be bigger than the data.
- ✓ There are about 40 bytes of overhead over the raw string data in Java String.
- ✓ It stores each character as two bytes because of String's internal usage of UTF-16 encoding.
- ✓ If there are 10 characters String, it can easily consume 60 bytes.

- ✓ Common collection classes like HashMap and LinkedList make use of linked data structure, there we have "wrapper" object for every entry.
- ✓ This object has both header and pointer (8 bytes each) to the next object in the list.

- ✓ Collections of primitive types often store them as "boxed objects".
- ✓ For example, java. lang. Integer.

a. Spark Data Structure Tuning

- ✓ By avoiding the Java features that add overhead we can reduce the memory consumption. There are several ways to achieve this:
- ✓ Avoid the nested structure with lots of small objects and pointers.
- ✓ Instead of using strings for keys, use numeric IDs or enumerated objects.
- ✓ If the RAM size is less than 32 GB, set JVM flag to – xx:+UseCompressedOops to make a pointer to four bytes instead of eight.

b. Spark Garbage Collection Tuning

- ✓ JVM garbage collection is problematic with large churn RDD stored by the program.
- ✓ To make room for new objects, Java removes the older one; it traces all the old objects and finds the unused one.
- ✓ But the key point is that cost of garbage collection in Spark is proportional to many Java objects.
- ✓ Thus, it is better to use a data structure in Spark with lesser objects.
- ✓ One more way to achieve this is to persist objects in serialized form.

- ✓ As a result, there will be only one object per RDD partition.

5. Memory Management in Spark

- ✓ We consider Spark memory management under two categories:
  - Execution
  - Storage.

- ✓ The memory which is for computing in shuffles, Joins, aggregation is Execution memory.
- ✓ While the one for caching and propagating internal data in the cluster is storage memory.
- ✓ Both execution and storage share a unified region M.
- ✓ When the execution memory is not in use, the storage can use all the memory.
- ✓ The same case lies true for Storage memory.
- ✓ Execution can drive out the storage if necessary.
- ✓ This is done only until storage memory usage falls under certain threshold R.

- ✓ We can get several properties by this design.
- ✓ First, the application can use entire space for execution if it does not use caching.
- ✓ While the applications that use caching can reserve a small storage (R), where data blocks are immune to evict.

- ✓ Even though we have two relevant configurations, the users need not adjust them.
- ✓ Because default values are relevant to most workloads:

- ✓ memory. fraction describes the size of M as a fraction of the (JVM heap space-300MB)(default 0.6).

- ✓ The remaining 40% is stored in user data structure, internal metadata in Spark and safeguarding against OOM error in case of Sparse and large records.
- ✓ memory. storageFraction shows the size of R as the fraction of M (default 0.5).

6. Determining Memory Consumption in Spark

- ✓ If we want to know the size of Spark memory consumption a dataset will require to create an RDD, put that RDD into the cache and look at "Storage" page in Web UI.
- ✓ This page will let us know the amount of memory RDD is occupying.

- ✓ If we want to know the memory consumption of a specific object, use SizeEstimator'S estimate method.

7. Spark Garbage Collection Tuning

- ✓ In garbage collection, tuning in Apache Spark, the first step is to gather statistics on how frequently garbage collection occurs.
- ✓ It also gathers the amount of time spent in garbage collection.
- ✓ Thus, can be achieved by adding -verbose: gc -XX: +PrintGCDetails -XX: +PrintGCTimeStamps to Java option.
- ✓ The next time when Spark job run, a message will display in workers log whenever garbage collection occurs.
- ✓ These logs will be in worker node, not on driver's program.

- ✓ Java heap space divides into two regions Young and Old.
- ✓ The young generation holds short-lived objects while Old generation holds objects with longer life.
- ✓ The garbage collection tuning aims at, long-lived RDDs in the old generation.
- ✓ It also aims at the size of a young generation which is enough to store short-lived objects.
- ✓ With this, we can avoid full garbage collection to gather temporary object created during task execution.
- ✓ Some steps that may help to achieve this are:

- ✓ If full garbage collection is invoked several times before a task is complete this ensures that there is not enough memory to execute the task.
- ✓ In garbage collection statistics, if OldGen is near to full we can reduce the amount of memory used for caching. This can be achieved by lowering spark.memory.fraction. the better choice is to cache fewer objects than to slow down task execution. Or we can decrease the size of young generation i.e., lowering –Xmn.
- ✓ The effect of Apache Spark garbage collection tuning depends on our application and amount of memory used.

8. Other consideration for Spark Performance Tuning

a. Level of Parallelism

- ✓ To use the full cluster the level of parallelism of each program should be high enough.
- ✓ According to the size of the file, Spark sets the number of "Map" task to run on each file.
- ✓ The level of parallelism can be passed as a second argument.
- ✓ We can set the config property spark.default.parallelism to change the default.

b. Memory Usage of Reduce Task in Spark

- ✓ Although RDDs fit in our memory many times we come across a problem of OutOfMemoryError.
- ✓ This is because the working set of our task say groupByKey is too large.
- ✓ We can fix this by increasing the level of parallelism so that each task's input set is small.
- ✓ We can increase the number of cores in our cluster because Spark reuses one executor JVM across many tasks and has low task launching cost.

c. Broadcasting Large Variables

- ✓ The size of each serialized task reduces by using broadcast functionality in SparkContext.
- ✓ If a task uses a large object from driver program inside of them, turn it into the broadcast variable.
- ✓ Generally, it considers the tasks that are about 20 Kb for optimization.

d. Data Locality in Apache Spark

- ✓ Data locality plays an important role in the performance of Spark Jobs.
- ✓ The case in which the data and code that operates on that data are together, the computation is faster.
- ✓ But if the two are separate, then either the code should be moved to data or vice versa.
- ✓ It is faster to move serialized code from place to place then the chunk of data because the size of the code is smaller than the data.

- ✓ Based on data current location there are various levels of locality. The order from closest to farthest is:

- ✓ The best possible locality is that the PROCESS_LOCAL resides in same JVM as the running code.

- ✓ NODE_LOCAL resides on the same node in this. It is because the data travel between processes is quite slower than PROCESS_LOCAL.

- ✓ There is no locality preference in NO_PREF data is accessible from anywhere.

- ✓ RACK_LOCAL data is on the same rack of the server. Since the data is on the same rack but on the different server, so it sends the data in the network, through a single switch.

- ✓ ANY data resides somewhere else in the network and not in the same rack.

Need to update

1.Spark shell

2. Web UI

https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-webui.html

Explain about all tabs

- Jobs
- Stages
- Storage
- Environment
- Executors
- SQL