SPARK & RDD CHEAT SHEET

Spark & RDD Basics

Apache Spark

It is an open source, Hadoop compatible fast and expressive cluster computing platform

RDD

The core concept in Apache Spark is **RDD** (Resilient Distributed Datasheet) , which is an immutable distributed collection of data which is partitioned across machines in a cluster.

Transformation: It is an operation on an **RDD** such as filter (), map () or union () that yields another **RDD**.

Action: It is an operation that triggers a computation such as count (), first (), take(n) or collect ().

Partition: It is a logical division of data stored on a node in a cluster

Spark Context

It holds a connection with spark cluster management

Driver: The process of running the main () function of an application and creating the SparkContext is managed by driver

Worker: Any node which can run program on the cluster is called worker

Components of Spark

Executors: It consists of multiple tasks; basically it is a JVM process sitting on all nodes. Executors receive the tasks, deserialize it and run it as a task. Executors utilize cache so that so that the tasks can run faster.

Tasks: Jars along with the code is a task

Node: It comprises of a multiple executors

RDD: It is a big data structure which is used to represent data that cannot be stored on a single machine. Hence, the data is distributed, partitioned and split across the computers.

Input: Every **RDD** is made up of some input such as a text file, Hadoop file etc.

Output: An output of functions in Spark can produce **RDD**, it is functional as a function one after other receives an input RDD and outputs an output **RDD**.

Shared Variables on Spark

Broadcast variables: It is a read only variable which will be copied to the worker only once. It is similar to the Distributor cache in MapReduce. We can set, destroy and unpersist these values. It is used to save the copy of data across all the nodes

Example syntax:

```
broadcastVariable = sparkContext.broadcast(500)
broadcastVariable.value
```

Accumulators: The worker can only add using an associative operation, it is usually used in parallel sums and only a driver can read an accumulator value. It is same as counter in MapReduce. Basically, accumulators are variables that can be incremented in a distributed tasks and used for aggregating information

Example syntax:

```
exampleAccumulator = sparkContext.accumulator(1)
exampleAccumulator.add(5)
```

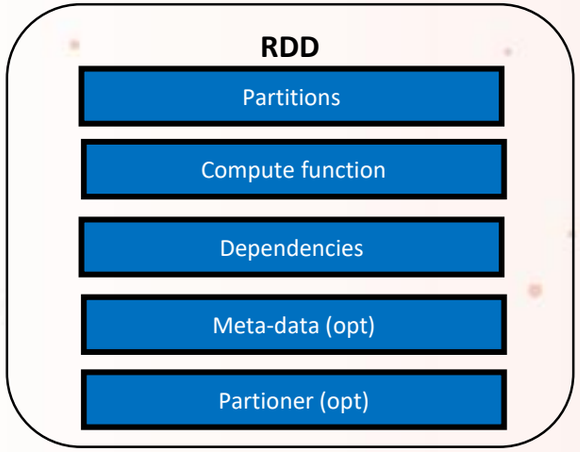
Unified Libraries in Spark

Spark SQL: It is a Spark module which allows working with structured data. The data querying is supported by **SQL** or **HQL**

Spark Streaming: It is used to build scalable application which provides fault tolerant streaming. It also processes in real time using web server logs, Facebook logs etc. in real time.

Mlib(Machine Learning): It is a scalable machine learning library and provides various algorithms for classification, regression, clustering etc.

Graph X: It is an **API** for graphs. This module can efficiently find the shortest path for static graphs.

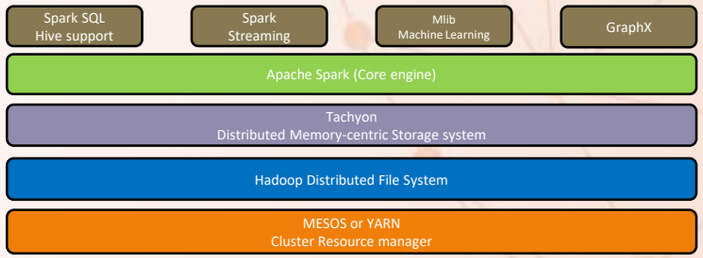


Function Transformations	Description
map(function)	Returns a new **RDD** by applying function on element
filter(function)	Returns a new dataset formed by selecting those elements of the source on which function returns true
filterByRange(lower, upper)	Returns an **RDD** with elements in the specified range
flatMap(function)	It is similar to the map function but the function returns a sequence instead of a value
reduceByKey(function,[num Tasks])	It is used to aggregate values of a key using a function.
groupByKey([num Tasks])	To convert(K,V) to (K, <iterable V>)
distinct([num Tasks])	This is used to eliminated duplicates from **RDD**
mapPartitions(function)	Runs separately on each partition of **RDD**
mapPartitionsWithIndex(function)	Provides function with an integer value representing the index of the partition
sample(withReplacement, fraction, seed)	Samples a fraction of data using a given random number generating seeds
union()	This returns a new **RDD** containing all elements and arguments from source **RDD**
intersection()	Returns a new **RDD** that contains an intersection of elements in the datasets
Cartesian()	Returns the Cartesian product of all pair of elements
subtract()	New **RDD** created by removing the elements from the source **RDD** in common with arguments
join(RDD,[numTasks])	It joins two elements of the dataset with common arguments. When invoked on (A,B) and (A,C) it creates a new **RDD** (A,(B,C))
cogroup(RDD,[numTasks])	It converts (A,B) to (A, <iterable B>)

Cluster Manager

It is used to allocate resources to each application in a driver program. There are 3 types of cluster managers which are supported by **Apache Spark**

- Standalone
- Mesos
- Yarn



Action Functions	Description
count()	Get the number of data elements in the **RDD**
collect()	Get all the data elements in the **RDD** as an array
reduce(function)	It is used to aggregate data elements into the **RDD** by taking two arguments and returning one
take(n)	It is used to fetch the first n elements of the **RDD**
foreach(function)	It is used to execute function for each data element in the **RDD**
first()	It retrieves the first data element of the **RDD**
saveastextfile(path)	It is used to write the content of **RDD** to a text file or set of text files to the local system
takeordered(n, [ordering])	It will return the first n elements of **RDD** using either the natural order or a custom comparator

RDD Persistence Method Functions	Description
MEMORY_ONLY (default level)	It stores the **RDD** in an available cluster memory as deserialized Java object
MEMORY_AND_DISK	This will store **RDD** as a deserialized Java object. If the **RDD** does not fit in the cluster memory it stores the partitions on the disk and reads them
MEMORY_ONLY_SER	This stores **RDD** as a serialized Java object, this is more CPU intensive
MEMORY_ONLY_DISK_SER	This option is same as above but stores in a disk when the memory is not sufficient
DISC_ONLY	This option stores **RDD** only on the disk
MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc.	This is same as the other levels except that the partitions are replicated on 2 slave nodes

Persistence Method Function	Description
cache()	It is used to avoid unnecessary recomputation . This is same as persist(MEMORY_ONLY)
persist([Storage Level])	Persisting the **RDD** with the default storage level
unpersist()	Marking the **RDD** as non persistent and removing the block from memory and disk
checkpoint()	It saves a file inside the checkpoint directory and all the reference of its parent **RDD** will be removed

IntelliPaat

FURTHERMORE:

Spark Certification Training Course