Good questions and explanation

<http://data-flair.training/forums/topic/list-the-advantage-of-parquet-file-in-apache-spark>

<http://data-flair.training/forums/topic/explain-transformation-and-action-in-rdd-in-apache-spark>

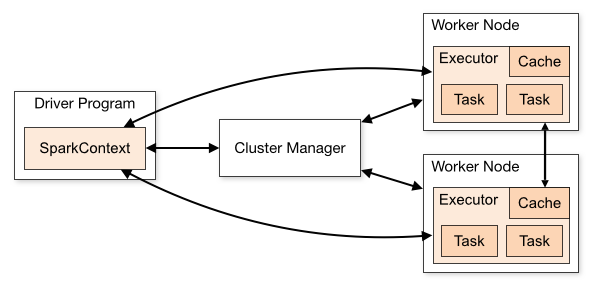
Spark Driver Program:

--------------------------------------

Driver program The process running the main() function of the application and creating the SparkContext.

Roll of Driver in Spark

* Driver program is responsible for launching various parallel operations on the cluster.
* **Driver program contains application's *main()* function.**
* **It is the process which is running the user code which in turn create the SparkContext object,**[**create RDDs**](http://data-flair.training/blogs/how-to-create-rdds-in-apache-spark/)**and performs**[**transformation and action operation on RDD**](http://data-flair.training/blogs/rdd-transformations-actions-apis-apache-spark/)**.**
* Driver program access [**Apache Spark**](http://data-flair.training/blogs/apache-spark-introduction-spark-comprehensive-tutorial/)through a **[SparkContext](http://data-flair.training/blogs/sparkcontext-in-apache-spark-tutorial/)** object which represents a connection to computing cluster (From Spark 2.0 onwards we can access SparkContext object through SparkSession).
* **Driver program is responsible for converting user program into the unit of physical execution called task.**
* It also defines distributed datasets on the cluster and we can apply different operations on Dataset (transformation and action).
* **Spark program creates a logical plan called**[**Directed Acyclic graph**](http://data-flair.training/blogs/directed-acyclic-graph-dag-in-apache-spark/)**which is converted to physical execution plan by the driver when driver program runs.**



What are the benefits of using parquet file-format in Apache Spark?

Parquet is a columnar format supported by many data processing systems. The benefits of having a columnar storage are -

1- Columnar storage limits IO operations.

2- Columnar storage can fetch specific columns that you need to access.

3-Columnar storage consumes less space.

4- Columnar storage gives better-summarized data and follows type-specific encoding.

Parquet is an open source file format for [**Hadoop**](http://data-flair.training/blogs/hadoop-introduction-tutorial-quick-guide/). Parquet stores nested data structures in a flat columnar format compared to a traditional approach where data is stored in row-oriented approach, parquet is more efficient in terms of storage and performance.

There are several advantages to columnar formats:

1)Organizing by column allows for better compression, as data is more homogeneous. The space savings are very noticeable at the scale of a Hadoop cluster.  
2)I/O will be reduced as we can efficiently scan only a subset of the columns while reading the data. Better compression also reduces the bandwidth required to read the input.  
3)As we store data of the same type in each column, we can use encoding better suited to the modern processors’ pipeline by making instruction branching more predictable.

**YARN**

---------------------

- Yet Another Resourec Negotiator (MapReduce-2)

- in MapReduce 1, Scalability a bottle neck when cluster size grows to 400+

- 2010 yahoo began the next generation MapReduce

- which can run different distributed processing frame work in parallel on the same cluster

-Main idea is to split the JOBTRACKER responsibilitys:

- Resource Manager - (Job Scheduling)

- Application Master - (Task Monitoring)

- older program written in MapReduce 1 work well with MapReuce2

- with MR2 only the way of execution of MR program changed

-so the program written in older api still works on MR2

Advantages-

---------------

- Increased Scalability: as the JobTracker task was split in to 2 scalibity increased dramatically

- More than one Yarn could co-exist on the same cluster.

along with MR we can have another distributed data processing framework (spark) on the same cluster

- better memory utilization with the concepts of containers

it is same as slots in classic MR - which are fixed in nature; where as containers are more flexible

in MR1 for single Task tracker would have fixed slots for map task and reduce task;

where as in containers it can run map/reduce or any other task and flexible in nature this results in better in memory utilization

Entities in yarn

---------------------

1. Client - responsible for submitting the job and interact with map and reduce and HDFS framework

2. Yarn Resource Manager - which is responsible for allocating the computing resources that are required by the job

job responsibilities can be classified in to 2

1. Scheduler - which responsible for scheduling of job; which does not perform monitoring/ tracking of job

2. Application Manager - which monitors the application status

3. Yarn Node Manager - it is present on all the slave node; responsible for launch and managing the containers

4. MR Application Master - it is responsible for execution of the job that is associated with;

- it is the one which coorbinates the task running and monitors the progress and aggregates it and sends the report to its client

- it is sponed(launched) under the Node Manager under the instructions of Resource Manager;

- it is sponed for ever job and terminates after the job completion

5.Yarn Child - it manages the execution of map and reduce task; responsible to send updates and progress to the application master

6. Distributed File System - which contains all necessary input and the place where the output files are returned to

Difference b/w MR1 and MR2

-------------------------------------------------

60 mins – Video 2 Edureka

MR1 – default block size 64 MB

MR2 – Default block size is 128 MB

Fedaration and Highavailability

HightAvailability – Standby NameNode

Fedaration – NN1 - /HR/..

NN2 - /Finance/..

MR1

As a result Name node has to store less meta info (meta data) in memory(RAM)

* Only one Name node –

If Name node fail, cluster will be down

* We had secondary Name node – which will take backup of Name node

-it is not hot\_standby – it just takes the backup from NameNode on hourly bases

-manual intervention was need to restore the system.

* Only one Name Node – entire meta data was in RAM for too big cluster it was not well suitable

MR2

As a result Name Node has to store less meta data in memory(RAM) –

* In MR2 we had Highavailability

-----------------------

Active Name node and Standby Name node

-if active name node fails Standby will act as active Name node ; with out manual intervention

And admin have to make one more Standby Name node

* Fedaration (each application data pushed to specific folder under NameNode; separate folder under)

------------------

/HR/.. NameNode 1

/Fin/… NameNode 2

/Sales/… NameNode 3

In MR1 JobTracker was over burdened and single point of failure

In MR1 we have only JobTracker and TaskTracker

JobTracker:

Was responsible for

-Allocating resources

-Job Scheduling

-Monitoring of task of all the job

In MR2

Resource Manager

Node Manager + Application Master

video 3: 130min's

--------------------

Hadoop1

------------

JobTracker => master => overburdened

Complete cluster level resources management

Job scheduling

Monitoring of task of all the job

TaskTracker => slave --> allocating resources

Hadoop 2: MR2 - Yarn => Yet another resource negotiator

================================================

Resource Manager => master

Complete cluster level resources management

Job Scheduling

Node Manager =>slave => Allocating the resources(resources required for a tast is called a container)

+

Application Master=>will be created one per job/application(emp sal or wordcount program)

monitor tasks for each job

Video 3: 130 MR program

video 3: 130min's

--------------------

Hadoop1

------------

JobTracker => master => overburdened

Complete cluster level resources management

Job scheduling

Monitoring of task of all the job

TaskTracker => slave --> allocating resources

Hadoop 2: MR2 - Yarn => Yet another resource negotiator

================================================

Resource Manager => master

Complete cluster level resources management

Job Scheduling

Node Manager =>slave => Allocating the resources(resources required for a tast is called a container)

+

Application Master=>will be created one per job/application(emp sal or wordcount program)

monitor tasks for each job

different modes in which hadoop operates:

----------------------------------------

1. stand alon mode

2. psudo distributed mode

3. fully distributed mode

1.Psudo distributed mode

--------------------

core-site.xml - NN - where namenode has to run

yarn-site.xml - RM - where Resource Manager has to run

slave -- DN & NM - where DN and NM has to run

2. we configure custom parameters for our cluster

--------------------

hdfs-site.xml - repl factor to one, block size(all file related conf is done here)

yarn-site.xml - process specific setttings(processing related conf is done here)

3. We set java path

-------------------

hadoop-env.sh --> java path(environment variables)

Video-2 - 130m's

hadoop-env.sh - Environment variables that are used in the scripts to run Hadoop(java home path)

core-site.xml - Common for Yarn and HDFS ;Configuration settings for hadoop core such as I/O settings that HDFS and MapReduce

hdfs-site.xml - config settings for hdfs daemons, the namenode, the secondary namenode and the data node

mapred-site.xml - config settings for MapReduce application

yarn-site.xml - config settings for Resource manager and NodeManager

master - (if we need secondary name node - which is not used in MR2 if required we can configure )a list of machines (one per line)that each run a secondary namenode

slaves - a list of machines (one per line) that each run a Datanode and a node manager

sudo jps -- to know all the process are up and running

to list a program under jar file

-----------------------------------

hadoop jar <jar-file-name>

--it will list all the program under <jar-file-name>

to run a program from a jar file

------------------------------------

hadoop jar <jar-file-name> <program-name> input-path output-path

to start and stop service

--------------------------

sudo service hadoop-master stop

sudo service hadoop-master start

hadoop dfsadmin -safemode leave

sudo jps

distcp - to copy data from cluster1 to cluster2

-------------------------------------------------

hadoop distcp <cluseter1> <cluster2>

traditional way for bigdata:=> Limitations

--------------------------------------------

1. Spliting the data and managing; manually we need to split and store the data; maintain where it is stored

2. Running the program in parallel with fault tolerance

3. aggregation / consolidation of data

MR Program:video 3: 30mis

------------

1. Map class and implement map method

--Business logic

2. Reduce class and implement Reduce method

--aggregate logic/consolidation logic

Driver Class -> to configure the application i.e - input/output path

Step 1:

--------

map class

reduce class

+

Driver class-> configuring the application/job

step2:

-----------

create jar and run jar

hadoop jar <jar-file> <program-name> inpupath outpath

Map1 output

--------------

welcome,1

to,1

Map2 output

--------------

edu,1

welcome,1

you,1

Reducers Input

------------------

welcome,[1,1]

to,[1]

you,[1]

Spark application building in to jars and shipping it uat or production

---------------------------------------------------------------

1. Once the application is ready
2. We use to build a jar file and ship it to production cluster using command

scp wc\_2.10-1.0.jar madhanrajuj@gw01.itversity.com:~ --(~ --/home/madhanrajuj)

pw: ….

1. We use to run the application using command

Spark-submit \

--class wc \

<jar-filename> <input-args-1> <input-args-2>

Ex:

spark-submit \

--class wc \

wc\_2.10-1.0.jar file:///home/madhanrajuj2/wordCount.txt /user/madhanrajuj/wcop

we use to parameterize the input path and output path

using args(0) and args(1)

ex:

sc.textFile(args(0))

for database connectivity to run under UAT; Production

we use to create a “application.properties” file which use to have following configuration

dev.host = nn01.itversity.com

dev.port = 3306

dev.db = hr

dev.user = hr\_ro

dev.pw = itversity

test.host = nn02.itversity.com

test.port = 3306

test.db = retail\_db

test.user = retail\_dba

test.pw = itversity

while running the application we use to specify where it has to run ie “dev” or “test”

so it is use run accordingly under dev/test environment

1. We use to develop a application using intellij
2. We use to build a jar file using sbt

-go to project dir and say “sbt package” it use to create a jar file

-jar use to store under /project/target/scala10.8/<jarfile >

Ex: /home/cloudera/wc/target/scala-2.10/ wc\_2.10-1.0.jar

1. We use to ship jar file to cluster using command “scp”

scp wc\_2.10-1.0.jar madhanrajuj2@gw01.itversity.com:~ --(~ --/home/madhanrajuj)

pw: ….

1. We use to execute an application using command

Spark-submit \

--class wc \

<jar-file-name> <input-args1> <input-args2>

To externalize the program execution we need to use “typesafe config”

Like : DB cradentials

go to spark config

-------------------

we can get spark env properties

spark executor - how many executer will be used

spark executor cores - how many cores will be given to each executor ( no of cpu threds running in parallel )

spark executor memory - how much memory is given to each executor

spark driver memory - jvm memory

go to user guide and select Overview -> running on yarn

--driver-memory 4g

--executor-memory 2g \

--executor-cores 1 \

we need to understand capacity of our cluster -to tune the spark jobs at run time

-------------------------------------------

--

spark-submit \

--class wc \

--master yarn \

--conf spark.ui.port = 22222 \

--num-executors 6 \

--executor-cores 2 \

--executor-memeory 2g \

wc\_2.10-1.0.jar /public/randomtextwriter/part-m-00000 /user/madhanrajuj/wcop prod

**Hive:**

**Hive** **partitions:**

**Hive** organizes tables into **partitions**. It is a way of dividing a table into related parts based on the values of **partitioned** columns such as date, city, and department. Using **partition**, it is easy to query a portion of the data. ... A query searches the whole table for the required information.

**Bucketing** is another technique for decomposing data sets into more manageable parts. For example, suppose a table using date as the top-level partition and employee\_id as the second-level partition leads to too many small partitions. Instead, if we bucket the employee table and use employee\_id as the bucketing column, the value of this column will be hashed by a user-defined number into buckets. Records with the same employee\_id will always be stored in the same bucket.

So, **bucketing** works well when the field has high cardinality and data is evenly distributed among buckets. **Partitioning** works best when the cardinality of the partitioning field is not too high.

Also, *you can partition on multiple fields*, with an order (year/month/day is a good example), while *you can bucket on only one field*.

In tables, the number of rows (or tuples) is called the **cardinality**.

* Basically, a bucket is a file in Hive whereas partition is a directory.

Suppose, you have a table with five columns, name, server\_date, some\_col3, some\_col4 and some\_col5. Suppose, you have partitioned the table on *server\_date* and bucketed on *name* column in 10 buckets, your file structure will look something like below.

1. server\_date=xyz
   * 00000\_0
   * 00001\_0
   * 00002\_0
   * ........
   * 00010\_0

Here *server\_date=xyz* is the partition and *000* files are the buckets in each partition. Buckets are calculated based on some hash functions, so rows with *name=Sandy* will always go in same bucket.

**8) Explain the difference between partitioning and bucketing.**

* **Partitioning and Bucketing of tables is done to improve the query performance. Partitioning helps execute queries faster,** only if the partitioning scheme has some common range filtering i.e. either by timestamp ranges, by location, etc. Bucketing does not work by default.
* Partitioning helps eliminate data when used in WHERE clause. Bucketing helps organize data inside the partition into multiple files so that same set of data will always be written in the same bucket. Bucketing helps in joining various columns.
* In partitioning technique, a partition is created for every unique value of the column and there could be a situation where several tiny partitions may have to be created. However, with bucketing, one can limit it to a specific number and the data can then be decomposed in those buckets.
* **Basically, a bucket is a file in Hive whereas partition is a directory.**

RDD, DataFrame and Dataset

RDD:

Resilient distributed dataset (RDD), which is a collection of elements partitioned across the nodes of the cluster that can be operated on in parallel.

DataFrame:

**A DataFrame is a distributed collection of data organized into named columns.** It is conceptually equivalent to a table in a relational database or a R/Python Dataframe. Along with Dataframe, Spark also introduced catalyst optimizer, which leverages advanced programming features to build an extensible query optimizer.

Dataframe Limitations:-

* **Compile-time type safety:** As discussed, Dataframe API does not support compile time safety which limits you from manipulating data when the structure is not know. The following example works during compile time. However, you will get a Runtime exception when executing this code.

Example:

case class Person(name : String , age : Int)

val dataframe = sqlContect.read.json("people.json")

dataframe.filter("salary > 10000").show

=> throws Exception : cannot resolve 'salary' given input age , name

This is challenging specially when you are working with several transformation and aggregation steps.

Because DataFrame is weakly typed and developers aren't getting the benefits of the type system. For example, lets say you want to read something from SQL and run some aggregation on it:

val people = sqlContext.read.parquet("...")

val department = sqlContext.read.parquet("...")

people.filter("age > 30")

.join(department, people("deptId") === department("id"))

.groupBy(department("name"), "gender")

.agg(avg(people("salary")), max(people("age")))

**When you say people("deptId"), you're not getting back an Int, or a Long, you're getting back a Column object which you need to operate on**. In languages with a reach type systems such as Scala, you end up loosing all the type safety which increases the number of run-time errors for things that could be discovered at compile time.

On the contrary, DataSet[T] is typed. when you do:

val people: People = val people = sqlContext.read.parquet("...").as[People]

You're actually getting back a People object, where deptId is an actual integral type and not a column type, thus taking advantage of the type system.

As of Spark 2.0, the DataFrame and DataSet APIs will be unified, where DataFrame will be a type alias for DataSet[Row].

**DataSet Api:**

Dataset, is a collection of **strongly-typed** JVM objects, dictated by a case class you define in Scala or a class in Java.

Dataset API is an extension to DataFrames that provides a type-safe, object-oriented programming interface. It is a strongly-typed, immutable collection of objects that are mapped to a relational schema.

## Dataset Features:-

* **Provides best of both RDD and Dataframe:** RDD(functional programming, type safe), DataFrame (relational model, Query optimazation , Tungsten execution, sorting and shuffling)
* **Encoders:** With the use of Encoders, it is easy to convert any JVM object into a Dataset, allowing users to work with both structured and unstructured data unlike Dataframe.
* **Programming Languages supported:** Datasets API is currently only available in Scala and Java. Python and R are currently not supported in version 1.6. Python support is slated for version 2.0.
* **Type Safety:** Datasets API provides compile time safety which was not available in Dataframes. In the example below, we can see how Dataset can operate on domain objects with compile lambda functions.

Example:

case class Person(name : String , age : Int)

val personRDD = sc.makeRDD(Seq(Person("A",10),Person("B",20)))

val personDF = sqlContext.createDataFrame(personRDD)

val ds= personDF.as[Person]

ds.filter(p => p.age > 25)

ds.filter(p => p.salary > 25)

// error : value salary is not a member of person

ds.rdd // returns RDD[Person]

* **Interoperable:** Datasets allows you to easily convert your existing RDDs and Dataframes into datasets without boilerplate code.

Datasets API Limitation:-

* **Requires type casting to String:** Querying the data from datasets currently requires us to specify the fields in the class as a string. Once we have queried the data, we are forced to cast column to the required data type. On the other hand, if we use map operation on Datasets, it will not use Catalyst optimizer.

Example:

ds.select(col("name").as[String], $"age".as[Int]).collect()

Datasets API Limitation:-

* **Requires type casting to String:** Querying the data from datasets currently requires us to specify the fields in the class as a string. Once we have queried the data, we are forced to cast column to the required data type. On the other hand, if we use map operation on Datasets, it will not use Catalyst optimizer.