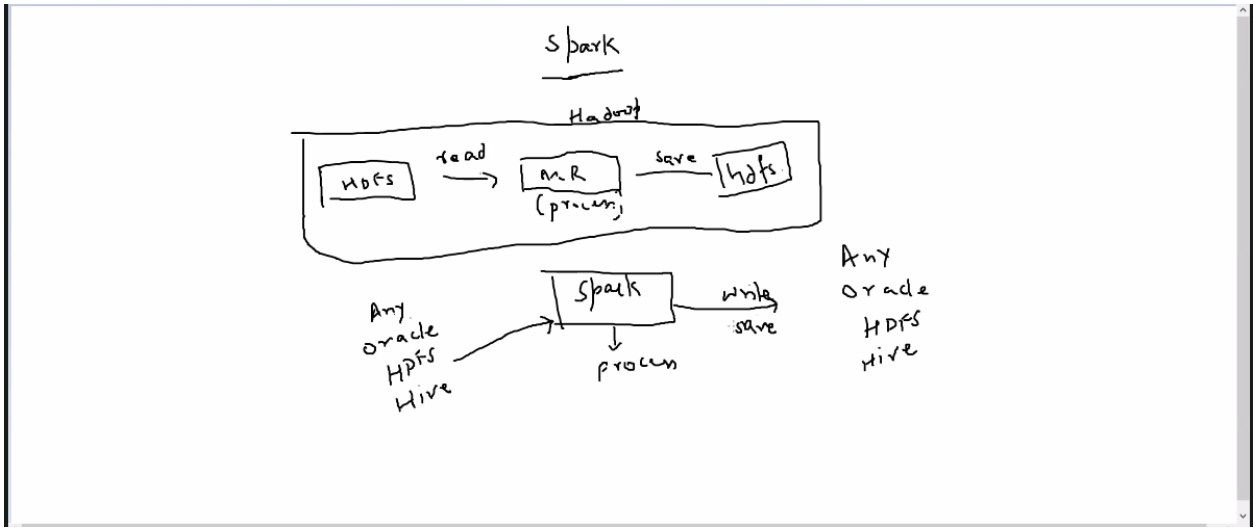
→ Spark

What is Spark ?

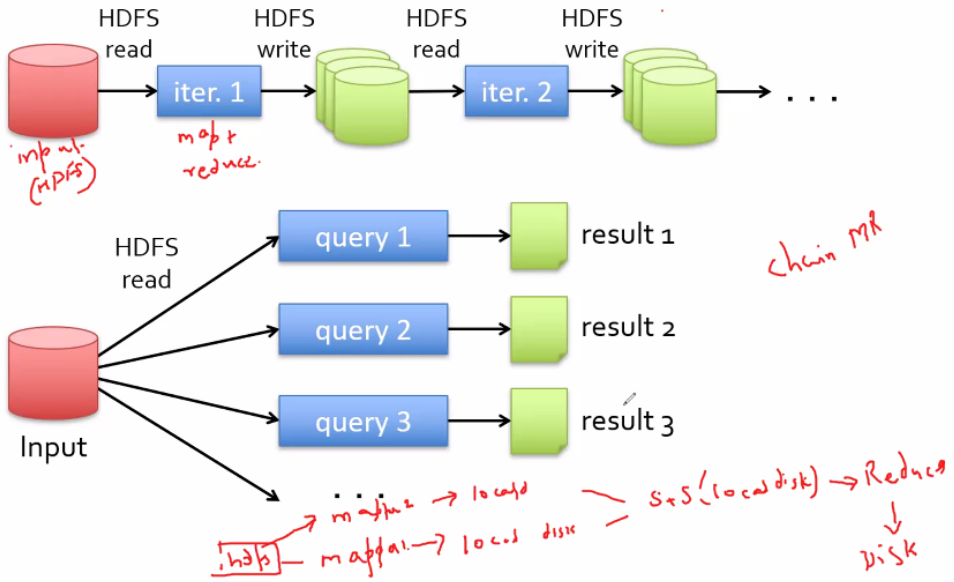
Spark is …

- Not a modified version of Hadoop.
- A low latency cluster computing system.
- Separate, fast, MapReduce-like engine
  - In-memory data storage for very fast iterative queries
  - General execution graphs and powerful optimizations
  - 40x to 100x faster than Hadoop*
  - 100 x faster than MapReduce for Iterative algorithms.
- Compatible with Hadoop's storage APIs
  - Can read/write to any Hadoop-supported system, including HDFS, HBase, SequenceFiles, etc

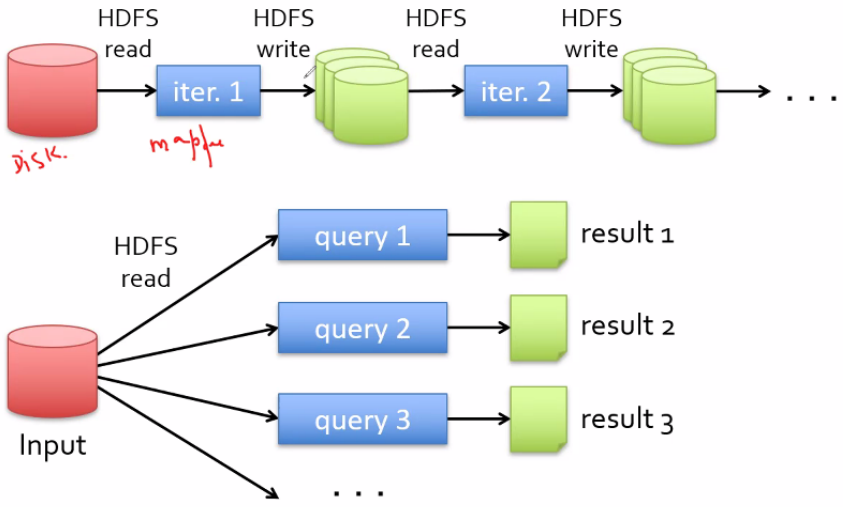
MetaScale

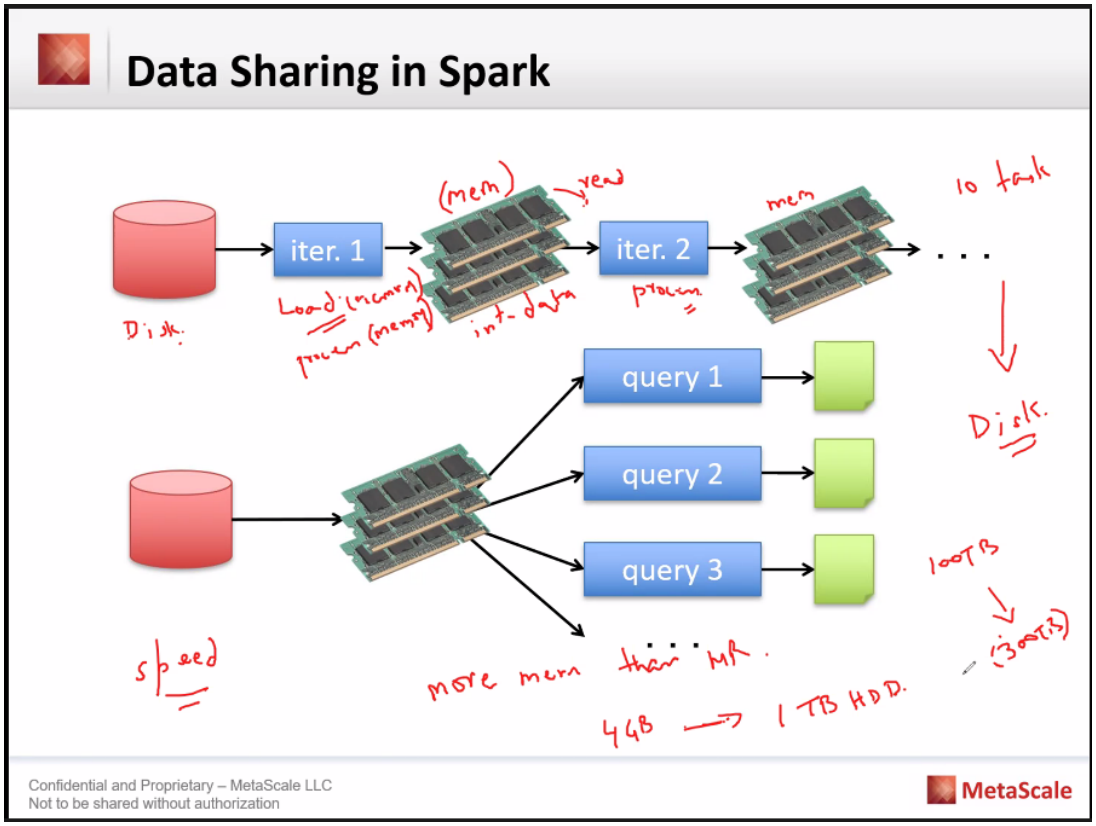


Data Sharing in MapReduce

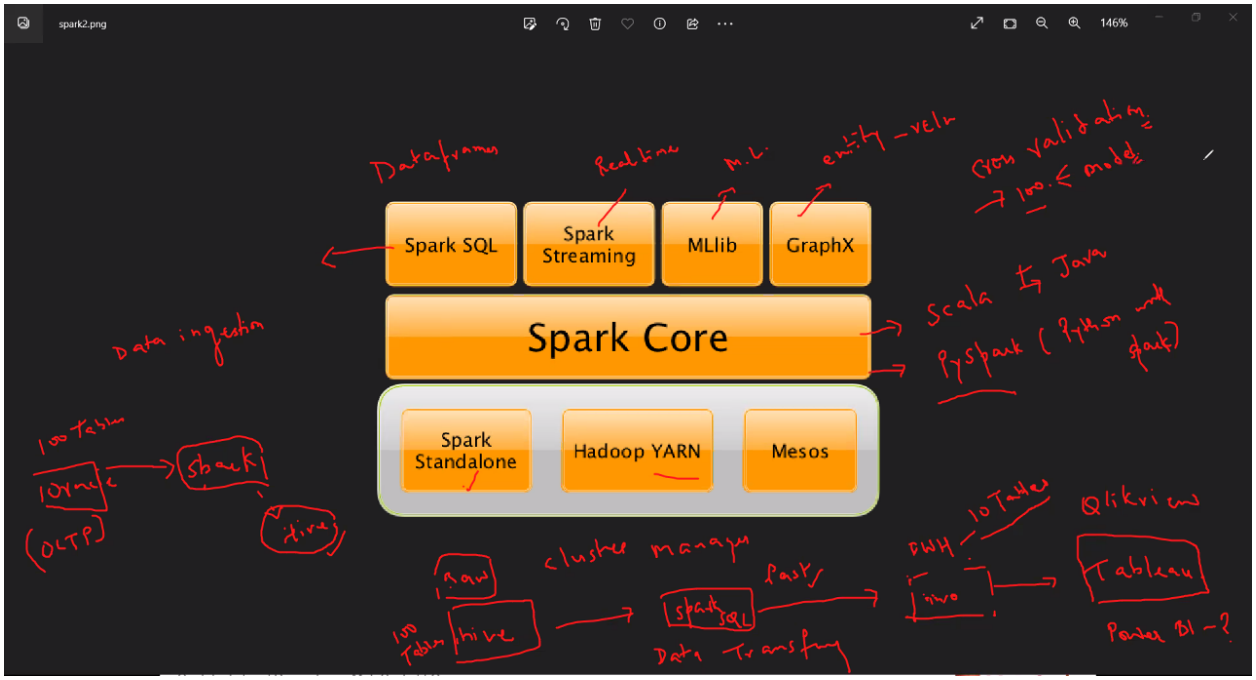


Data Sharing in MapReduce





a. No file system
b. Totally new thing, not a modified version of hadoop
c. Purely for processing
d. Can read/write data from/to any source like HDFS, Hive, Oracle, etc.
e. Costs /Needs more memory

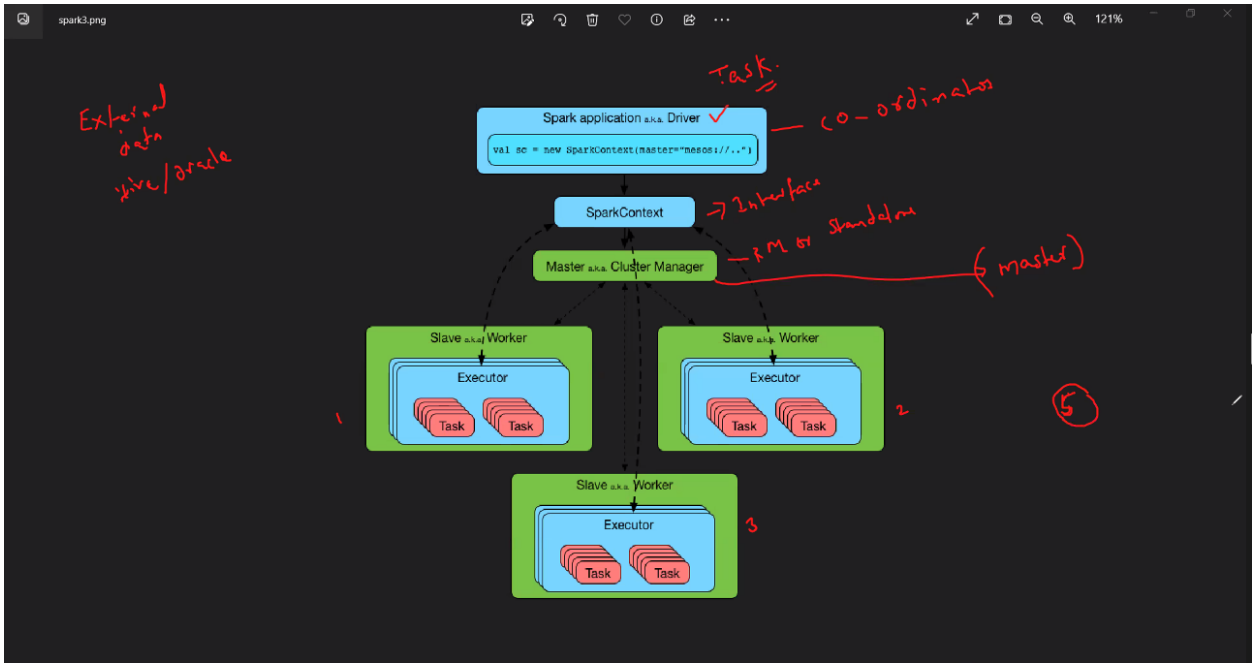


a. spark standalone or Mesos: For spark, we use spark standalone or Mesos
b. Mesos: standalone cluster/resource manager
c. Hadoop YARN: For Hadoop & spark, we use Hadoop YARN
d. spark core: For Scala/Java with Spark, we use spark core
e. PySpark: For python with spark, we use PySpark
f. SparkSQL: for tabular or structured schema to create dataframes
g. Spark streaming: for real time processing
h. MLib: for ML
i. GraphX: for entirety relationships

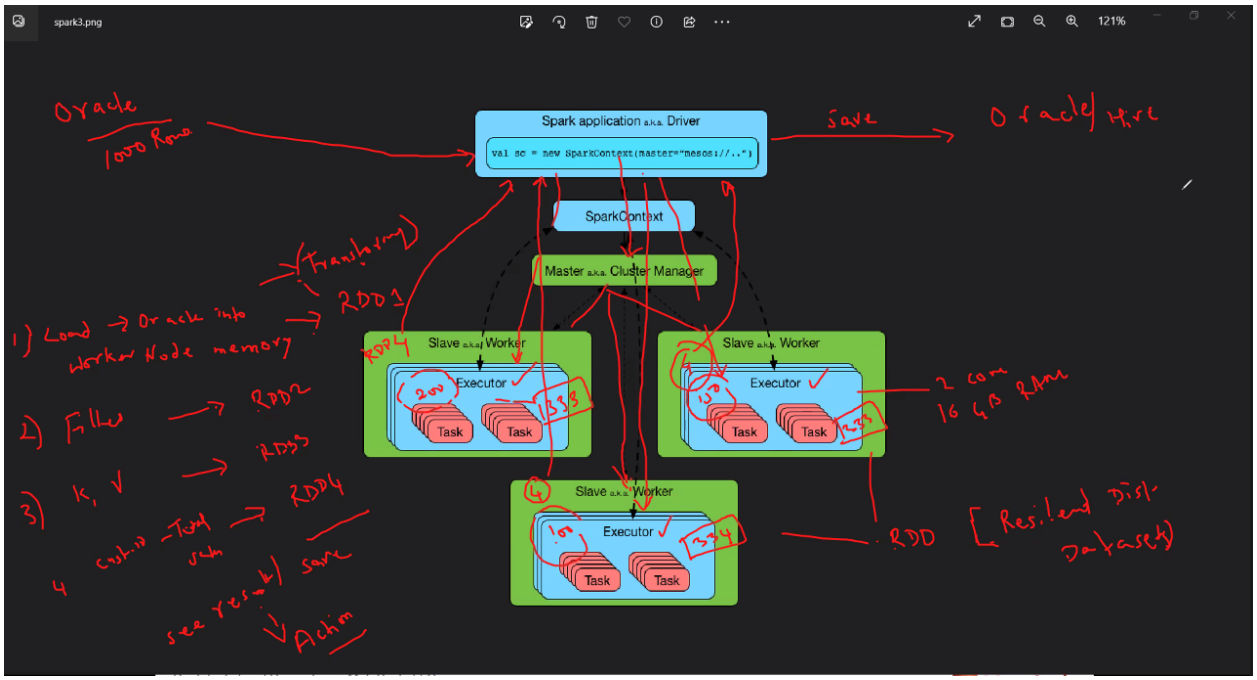
a. Data ingestion: reading data (OLTP data) from source table like Oracle to warehouse like Hive table
b. Data Processing/transforming: reading from warehouse data (OLAP data) like Hive table and then processing using spark and write back to hive and then display on dashboard like tableau

a. Cross-validation: to create models out of validated different models

→ spark Flow



a. Spark app / Driver : coordinates the task
b. Spark context: is a service/interface that helps in connecting spark driver to worker nodes, need to create spark context at start
c. Master / cluster manager: acts as resource manager or masternode, standalone
d. Worker nodes: act upon tasks assigned to them by Master



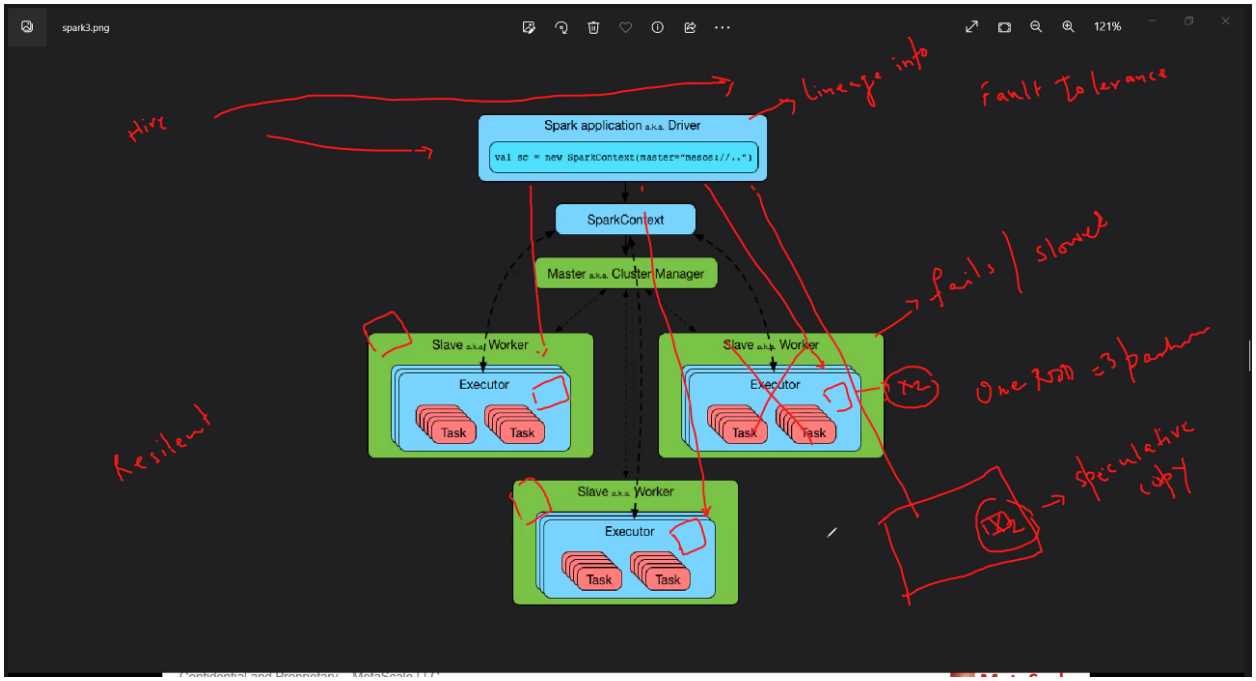
a. Executor: is a container of resources, launched on each worker node by cluster manager, on request of spark app/driver
b. After executor is launched, data is distributed evenly by driver, not in a physical division, but in logical division
c. RDD: Resilient Distributed dataset, is created on worker node memory, to run filter job, data is distributed as RDD for each job
d. Driver node sends filter tasks to each worker node
e. New RDD is launched for new task /job that driver node sends to worker node
f. All these commands are called transformation command, where you transform one RDD to another RDD
g. At the end you, run action command to transform the last RDD to see result / save it,
h. And after saving/printing it, then only memory is freed

i. Actions in PySpark RDDs:
    i. `.collect()`, `.count()`, `.first()`, `.take(n)`, `.reduce()`, `.saveAsTextFile()`
j. Transformations in PySpark RDDs:
    i. `.map()`, `.filter()`, `.union()`, `.flatMap()`
k. Actions in Pair RDDs :
    i. `.countByKey()`
l. Transformations in Pair RDDs :
    i. `.reduceByKey()`, `.sortByKey()`, `.groupByKey()`
m. Lazy evaluation: till you run action command, none of the previous command are run, all previous RDDs are created and released only once you run action command

Spark Programming Model

- Key idea: *resilient distributed datasets (RDDs)*
  - Distributed collections of objects that can be cached in memory across cluster nodes
  - Manipulated through various parallel operators
  - Automatically rebuilt on failure
- Interface
  - Clean language-integrated API in Scala
  - Can be used *interactively* from Scala console

→ Fault tolerance in Spark



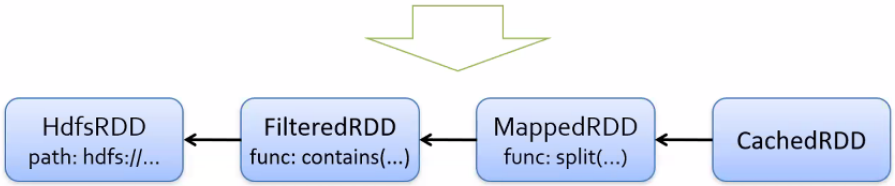
a. Driver maintains lineage information to check if all worker nodes are online
b. If one node fails/ becomes slower, the same data is loaded to a new node for processing
c. Speculative copy: data loaded onto new worker node when one worker node goes down/ becomes slow, whereas data is exactly same as data from the worker node that we lost
d. Out of *slow node* and *speculative copy*, whichever complete first, that data will be taken by spark app/driver

RDD Fault Tolerance

RDDs maintain lineage information that can be used to reconstruct lost partitions

Ex:

```
cachedMsgs = textFile(...).filter(_.contains("error"))
                          .map(_.split('\t')(2))
                          .cache()
```



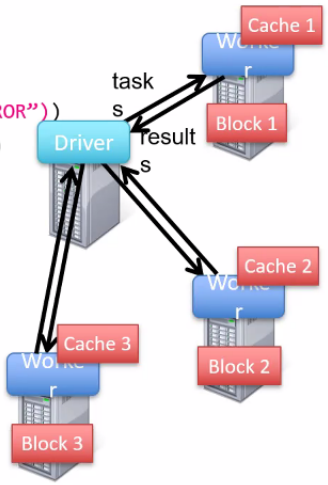
→

Example: Mining Console Logs

- Load error messages from a log into memory, then interactively search for patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split('\t')[2])
messages.cache()

messages.filter(lambda s: "foo" in s).count()
messages.filter(lambda s: "bar" in s).count()
```



→ find the category of transaction with highest net value

start PySpark

```
[bigdatalab456422@ip-10-1-1-204 ~]$ pyspark ;
```

```
[bigdatalab456422@ip-10-1-1-204 ~]$ pyspark ;
Python 3.7.6 (default, Jan  8 2020, 19:59:22)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/06/01 11:12:17 WARN cluster.YarnSchedulerBackend$YarnSchedulerEndpoint: Attempted to request executors before the AM has registered!
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.4.0-cdh6.2.1
      /_/

Using Python version 3.7.6 (default, Jan  8 2020 19:59:22)
SparkSession available as 'spark'.
>>>
```

→ sc stands for spark context

```
>>> txnRDD =
sc.textFile("hdfs://nameservice1/user/bigdatalab456422/training/txns1
.txt",1)
```

```
>>> txnRDD = sc.textFile("hdfs://nameservice1/user/bigdatalab456422/training/txns1.txt",1)
```

```
>>> txnRDD.count()
```

```
>>> txnRDD.count()
[Stage 0:>                                                          (0 + 0) / 1]23/06/01 11:22:44 WARN cluster.YarnScheduler: Initial job has not accepted any resources
; check your cluster UI to ensure that workers are registered and have sufficient resources
50000
>>>
```

```
## IGN
>>> txnRDD1 = txnRDD.map(lambda a : a.encode("ascii","ignore"))
```

```
>>> txnRDD1 = txnRDD.map(lambda a : a.encode("ascii","ignore"))
>>>
```

```
## IGN
>>> txnRDD1.count()
```

```
>>> txnRDD1.count()
50000
>>>
```

```
>>> txnKVRDD = txnRDD.map(lambda row : (row.split(',')[5],
float(row.split(',')[3])))
```

```
>>> txnKVRDD = txnRDD.map(lambda row : (row.split(',')[5], float(row.split(',')[3])))
>>>
```

→ Here take() is action command

```
>>> for line in txnKVRDD.take(5):
...     print(line)
...
```

```
>>> for line in txnKVRDD.take(5):
...     print(line)
...
[Stage 5:>                                                          (0 + 0) / 1]
('Cardio Machine Accessories', 40.33)
('Weightlifting Gloves', 198.44)
('Weightlifting Machine Accessories', 5.58)
('Gymnastics Rings', 198.19)
('Field Hockey', 98.81)
>>>
```

```
>>> spendbyProd = txnKVRDD.reduceByKey(lambda a,b : a+b)
```

```
>>> spendbyProd = txnKVRDD.reduceByKey(lambda a,b : a+b)
>>>
```

→ Here, collect() is also action command

```
>>> for line in spendbyProd.collect():
...     print(line)
...
```

```
>>> for line in spendbyProd.collect():
...     print(line)
...
('Cardio Machine Accessories', 46485.540000000045)
('Weightlifting Gloves', 38438.72)
('Weightlifting Machine Accessories', 41571.10999999999)
('Gymnastics Rings', 39871.54000000002)
('Field Hockey', 40813.67999999997)
('Camping & Backpacking & Hiking', 39993.520000000004)
('Jigsaw Puzzles', 41183.07999999999)
('Sandboxes', 42535.79999999998)
('Snowmobiling', 40357.51000000005)
('Bungee Jumping', 38975.590000000004)
('Archery', 37088.659999999945)
('Swing Sets', 47204.13999999999)
('Bowling', 40052.85999999999)
('Vaulting Horses', 41052.8)
('Fencing', 40604.29000000001)
('Free Weight Bars', 41915.61999999996)
('Basketball', 44954.68000000004)
('Gymnastics Bars', 41031.35000000001)
('Billiards', 38614.520000000004)
('Lacrosse', 42732.60999999997)
('Boating', 43049.069999999934)
('Fishing', 37144.60999999998)
('Free Weights', 41966.6)
('Weightlifting Machines', 42701.39999999998)
('Wetsuits', 38980.340000000004)
('Ping Pong', 39973.01999999999)
('Outdoor Playsets', 39532.59000000003)
('Bobsledding', 43157.45999999999)
('Luge', 40649.73)
>>>
```

```
>>> spendbyProd.count()
```

```
>>> spendbyProd.count()
125
>>>
```

```
>>> sortbyval = spendbyProd.sortBy(lambda a : -a[1])
```

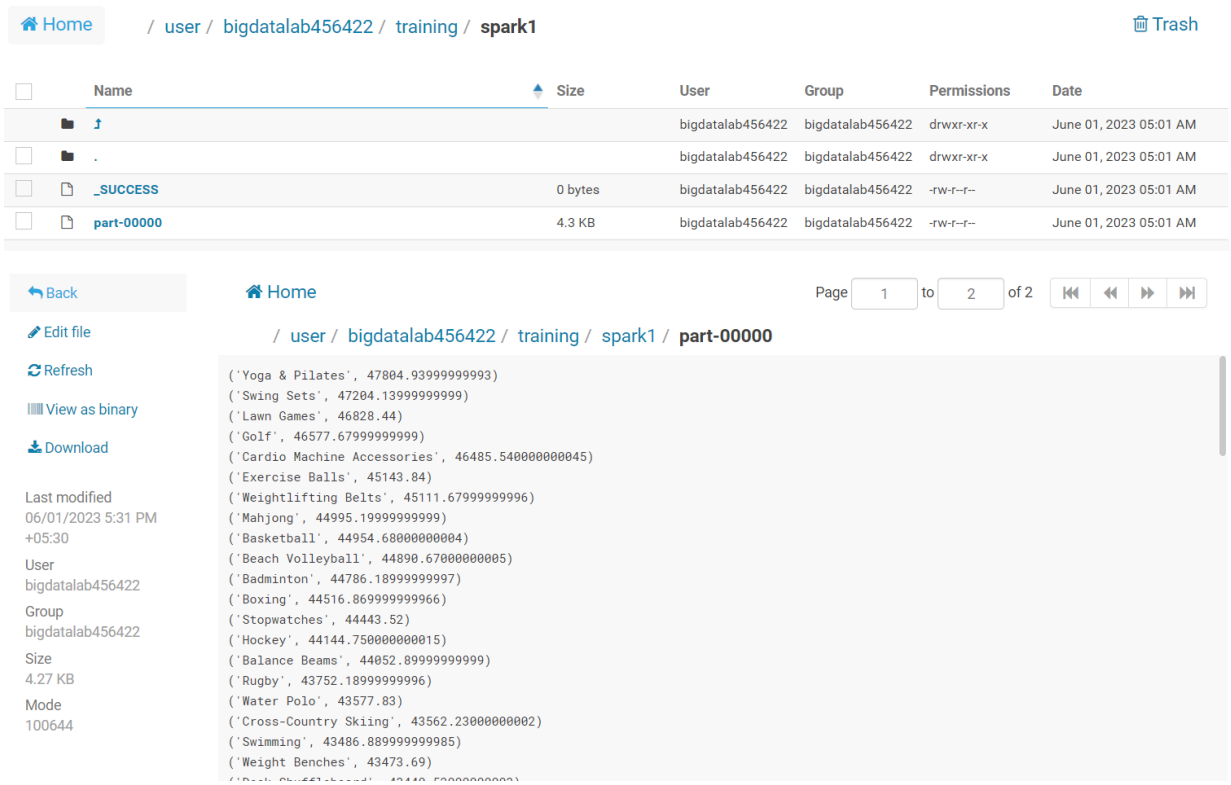
```
>>> sortbyval = spendbyProd.sortBy(lambda a : -a[1])
>>>
```

```
>>> for line in sortbyval.collect():
...     print(line)
...
```

```
>>> for line in sortbyval.collect():
...     print(line)
...
('Yoga & Pilates', 47804.93999999993)
('Swing Sets', 47204.13999999999)
('Lawn Games', 46828.44)
('Golf', 46577.67999999999)
('Cardio Machine Accessories', 46485.540000000045)
('Exercise Balls', 45143.84)
('Weightlifting Belts', 45111.67999999996)
('Mahjong', 44995.19999999999)
('Basketball', 44954.68000000004)
('Beach Volleyball', 44890.67000000005)
('Badminton', 44786.18999999997)
('Boxing', 44516.869999999966)
('Stopwatches', 44443.52)
('Hockey', 44144.750000000015)
('Balance Beams', 44052.89999999999)
('Rugby', 43752.18999999996)
('Water Polo', 43577.83)
('Cross-Country Skiing', 43562.23000000002)
('Swimming', 43486.889999999985)
('Weight Benches', 43473.69)
('Tetherball', 35611.92999999999)
('Surfing', 35069.079999999965)
('Disc Golf', 33944.02000000002)
('Mechanical Puzzles', 20381.669999999984)
('Air Suits', 19371.570000000007)
>>>
```

```
>>> sortbyval.saveAsTextFile("hdfs://nameservice1/user/bigdatalab456422/training/spark1")
```

```
>>> sortbyval.saveAsTextFile("hdfs://nameservice1/user/bigdatalab456422/training/spark1")
>>>
```




```
>>> txnRDD5 = txnRDD4.map(lambda a: a[0]+", "+str(a[1]))
```

```
>>> txnRDD5 = txnRDD4.map(lambda a: a[0]+", "+str(a[1]))
```

```
>>> for a in txnRDD5.take(5):
...     print(a)
```

```
>>> for a in txnRDD5.take(5):
...     print(a)
...
Yoga & Pilates, 47804.93999999993
Swing Sets, 47204.13999999999
Lawn Games, 46828.44
Golf, 46577.67999999999
Cardio Machine Accessories, 46485.540000000045
>>>
```

```
>>> sortbyval2 = sortbyval.map(lambda a : a[0] + "," + str(round(a[1],2)))
```

```
>>> sortbyval2 = sortbyval.map(lambda a : a[0] + "," + str(round(a[1],2)))
>>>
```

```
>>> sortbyval2.saveAsTextFile("hdfs://nameservice1/user/bigdatalab456422/training/spark2")
```

```
>>> sortbyval2.saveAsTextFile("hdfs://nameservice1/user/bigdatalab456422/training/spark2")
>>>
```

Home / user / bigdatalab456422 / training / **spark2** Trash

Name	Size	User	Group	Permissions	Date
↑		bigdatalab456422	bigdatalab456422	drwxr-xr-x	June 01, 2023 05:06 AM
.		bigdatalab456422	bigdatalab456422	drwxr-xr-x	June 01, 2023 05:07 AM
_SUCCESS	0 bytes	bigdatalab456422	bigdatalab456422	-rw-r--r--	June 01, 2023 05:07 AM
part-00000	2.7 KB	bigdatalab456422	bigdatalab456422	-rw-r--r--	June 01, 2023 05:07 AM

Back
Edit file
Refresh
View as binary
Download

Last modified
06/01/2023 5:37 PM
+05:30
User
bigdatalab456422
Group
bigdatalab456422
Size
2.69 KB
Mode
100644

Home Page 1 to 1 of 1

/ user / bigdatalab456422 / training / spark2 / **part-00000**

```
Yoga & Pilates,47804.94
Swing Sets,47204.14
Lawn Games,46828.44
Golf,46577.68
Cardio Machine Accessories,46485.54
Exercise Balls,45143.84
Weightlifting Belts,45111.68
Mahjong,44995.2
Basketball,44954.68
Beach Volleyball,44890.67
Badminton,44786.19
Boxing,44516.87
Stopwatches,44443.52
Hockey,44144.75
Balance Beams,44052.9
Rugby,43752.19
Water Polo,43577.83
Cross-Country Skiing,43562.23
Swimming,43486.89
Weight Benches,43473.69
Deck Shuffleboard,43440.52
```

→ reduceByKey() & sortBy()

```
('Boating', 43049.069999999934)
('Fishing', 37144.60999999998)
('Free Weights', 41966.6)
('Weightlifting Machines', 42701.39999999998)
('Wetsuits', 38980.340000000004)
('Ping Pong', 39973.01999999999)
('Outdoor Playsets', 39532.59000000003)
('Bobsledding', 43157.45999999999)
('Luge', 40649.73)
>>> spendbyProd.count()
125
>>> sortbyval = spendbyProd.sortBy(lambda a : -a[1])
>>> for line in sortbyval.collect():
...     print(line)
...
('Yoga & Pilates', 47804.93999999993)
('Swing Sets', 47204.13999999999)
('Lawn Games', 46828.44)
('Golf', 46577.67999999999)
('Cardio Machine Accessories', 46485.540000000045)
('Exercise Balls', 45143.84)
('Weightlifting Belts', 45111.67999999996)
('Mahjong', 44995.19999999999)
('Basketball', 44954.68000000004)
('Beach Volleyball', 44890.67000000005)
('Badminton', 44786.18999999997)
('Boxing', 44516.869999999966)
('Stopwatches', 44443.52)
('Hockey', 44144.750000000015)
('Balance Beams', 44052.89999999999)
('Rugby', 43752.18999999996)
('Water Polo', 43577.83)
('Cross-Country Skiing', 43562.23000000002)
('Swimming', 43486.889999999985)
('Weight Benches', 43473.69)
('Deck Shuffleboard', 43440.52000000003)
('Table Shuffleboard', 43405.149999999994)
('Abdominal Equipment', 43304.109999999986)
('Darts', 43243.41999999999)
('Gymnastics Mats', 43224.54999999999)
('Bobsledding', 43157.45999999999)
('Foosball', 43055.959999999997)
```