Using Spark on CDH Quickstart VM

< Contents >

Spark Example on Quickstart VM

This document is to run Spark on CDH.

Apache Spark has been receiving a lot of deserved attention lately. It is very understandable given the huge importance of distributed data processing for many companies and the pursuit for faster, cheaper and easier to use technologies aiming replace or complement the widely adopted Hadoop ecosystem and its MapReduce paradigm.

You need to some preparation to run this example.

1. VMware Download(http://www.vmware.com/kr) and Install
2. Quickstart tutorial Download(http://www.cloudera.com/content/cloudera/en/downloads.html)
3. Open the Quickstart tutorial file(/cloudera-quickstart-vm-5.3.0-0-vmware/cloudera-quickstart-vm-5.3.0-0-vmware.vmx) on VMware.
4. Download the dataset on Github.*

* Download dataset for running spark example.

Open the terminal on virtual machine of QuickStart tutorial. And input command.

```
[cloudera@quickstart ~] git clone https://github.com/SteveKim0513/spark_exam_dataset.git
```

```
[cloudera@quickstart ~]$ git clone https://github.com/SteveKim0513/spark_exam_dataset.git
Initialized empty Git repository in /home/cloudera/spark_exam_dataset/.git/
remote: Counting objects: 25, done.
remote: Compressing objects: 100% (21/21), done.
remote: Total 25 (delta 3), reused 20 (delta 1)
Unpacking objects: 100% (25/25), done.
[cloudera@quickstart ~]$ ls
cloudera-manager  Desktop    eclipse  Pictures          Templates
cm_api.sh         Documents  lib      Public            Videos
datasets          Downloads  Music    spark_exam_dataset  workspace
```

You can see the new folder named spark_exam_dataset.

```
[cloudera@quickstart ~] ls spark_exam_dataset
```

```
[cloudera@quickstart ~]$ ls spark_exam_dataset
lpsa.data   sample_libsvm_data.txt            shakespeare.txt
people.txt
[cloudera@quickstart ~]$
```

Dataset	Example
shakespeare.txt	wordcount
sample_libsvm_data.txt	Logistic Regression
lpsa.data	Linear Regression
people.txt	Running SQL on RDDS

<< Example 1. Estimate Pi >>

To test compute-intensive tasks in Spark, the Pi example calculates pi by “throwing darts” at a circle. The example points in the unit square ((0,0) to (1,1)) and sees how many fall in the unit circle. The fraction should be pi/4, which is used to estimate Pi. This example don’t need to any dataset.

- Open Spark shell

```
[cloudera@quickstart ~] spark-shell --jars /usr/lib/avro/avro-mapred.jar \
--conf spark.serializer=org.apache.spark.serializer.KryoSerializer
```

```
[cloudera@quickstart ~]$ spark-shell --jars /usr/lib/avro/avro-mapred.jar \
> --conf spark.serializer=org.apache.spark.serializer.KryoSerializer
2015-02-05 17:59:13,289 INFO  [main] spark.SecurityManager (Logging.scala:logInfo(59)) - C
hanging view acls to: cloudera
2015-02-05 17:59:13,297 INFO  [main] spark.SecurityManager (Logging.scala:logInfo(59)) - C
hanging modify acls to: cloudera
2015-02-05 17:59:13,299 INFO  [main] spark.SecurityManager (Logging.scala:logInfo(59)) - S
ecurityManager: authentication disabled; ui acls disabled; users with view permissions: Se
t(cloudera); users with modify permissions: Set(cloudera)
2015-02-05 17:59:13,300 INFO  [main] spark.HttpServer (Logging.scala:logInfo(59)) - Starti
ng HTTP Server
2015-02-05 17:59:13,438 INFO  [main] server.Server (Server.java:doStart(272)) - jetty-8.y.
z-SNAPSHOT
2015-02-05 17:59:13,479 INFO  [main] server.AbstractConnector (AbstractConnector.java:doSt
art(338)) - Started SocketConnector@0.0.0.0:59571
2015-02-05 17:59:13,481 INFO  [main] util.Utils (Logging.scala:logInfo(59)) - Successfully
 started service 'HTTP class server' on port 59571.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.2.0
      /_/
```

```
2015-02-05 17:59:47,402 INFO  [main] spark.SparkContext (Logging.scala:logInfo(59)) - Adde
d JAR file:/usr/lib/avro/avro-mapred.jar at http://192.168.206.132:45943/jars/avro-mapred.
jar with timestamp 1423187987401
2015-02-05 17:59:47,634 INFO  [sparkDriver-akka.actor.default-dispatcher-3] executor.Execu
tor (Logging.scala:logInfo(59)) - Using REPL class URI: http://192.168.206.132:59571
2015-02-05 17:59:47,723 INFO  [sparkDriver-akka.actor.default-dispatcher-3] util.AkkaUtils
 (Logging.scala:logInfo(59)) - Connecting to HeartbeatReceiver: akka.tcp://sparkDriver@192
.168.206.132:43415/user/HeartbeatReceiver
2015-02-05 17:59:50,574 INFO  [main] netty.NettyBlockTransferService (Logging.scala:logInf
o(59)) - Server created on 48341
2015-02-05 17:59:50,582 INFO  [main] storage.BlockManagerMaster (Logging.scala:logInfo(59)
) - Trying to register BlockManager
2015-02-05 17:59:50,589 INFO  [sparkDriver-akka.actor.default-dispatcher-13] storage.Block
ManagerMasterActor (Logging.scala:logInfo(59)) - Registering block manager localhost:48341
 with 267.3 MB RAM, BlockManagerId(<driver>, localhost, 48341)
2015-02-05 17:59:50,593 INFO  [main] storage.BlockManagerMaster (Logging.scala:logInfo(59)
) - Registered BlockManager
2015-02-05 17:59:50,880 INFO  [main] repl.SparkILoop (Logging.scala:logInfo(59)) - Created
 spark context..
Spark context available as sc.

scala>
```

If you can't see message "Spark context available as sc.". Then you can't use the "sc"in your coding like this.

```
scala> val data = sc.textFile("spark_exam_dataset/lpsa.data")
<console>:19: error: not found: value sc
       val data = sc.textFile("spark_exam_dataset/lpsa.data")
                  ^
```

In this case, you must exit the spark shell and then check the spark-shell open command.
* sc(spark context) : a connection to the Spark Executive Engine

```
[cloudera@quickstart ~] spark-shell --jars /usr/lib/avro/avro-mapred.jar \
--conf spark.serializer=org.apache.spark.serializer.KryoSerializer
```

- Input the code

```
scala> val count = sc.parallelize(1 to Num_Sample).map{ i =>
val x = Math.random()
val y = Math.random()
if(x*x + y*y < 1) 1 else 0
}.reduce(_ + _)
```

"Num_Sample" is the number of squares which has area an area of 1. So you can input this variable at any number you want. In this case, we input 100.

```
scala> val count = sc.parallelize(1 to 100).map{i =>
     | val x = Math.random()
     | val y = Math.random()
     | if(x*x + y*y<1) 1 else 0
     | }.reduce(_ + _)
2015-02-05 22:18:45,216 INFO  [main] spark.SparkContext (Logging.scala:logInfo(5
9)) - Starting job: reduce at <console>:16
2015-02-05 22:18:45,316 INFO  [sparkDriver-akka.actor.default-dispatcher-3] sche
duler.DAGScheduler (Logging.scala:logInfo(59)) - Got job 0 (reduce at <console>:
16) with 1 output partitions (allowLocal=false)
2015-02-05 22:18:45,317 INFO  [sparkDriver-akka.actor.default-dispatcher-3] sche
duler.DAGScheduler (Logging.scala:logInfo(59)) - Final stage: Stage 0(reduce at
<console>:16)
2015-02-05 22:18:45,318 INFO  [sparkDriver-akka.actor.default-dispatcher-3] sche
duler.DAGScheduler (Logging.scala:logInfo(59)) - Parents of final stage: List()
2015-02-05 22:18:45,326 INFO  [sparkDriver-akka.actor.default-dispatcher-3] sche
duler.DAGScheduler (Logging.scala:logInfo(59)) - Missing parents: List()
2015-02-05 22:18:45,354 INFO  [sparkDriver-akka.actor.default-dispatcher-3] sche
duler.DAGScheduler (Logging.scala:logInfo(59)) - Submitting Stage 0 (MappedRDD[1
] at map at <console>:12), which has no missing parents
2015-02-05 22:18:45,703 INFO  [sparkDriver-akka.actor.default-dispatcher-3] stor
age.MemoryStore (Logging.scala:logInfo(59)) - ensureFreeSpace(1728) called with
curMem=0, maxMem=280248975
2015-02-05 22:18:45,706 INFO  [sparkDriver-akka.actor.default-dispatcher-3] stor
```

scala> println("Pi is roughly" + 4.0 * count / *Num_Sample*)

```
r to /tmp/fetchFileTemp8650823145158895317.tmp
2015-02-05 22:18:47,875 INFO  [Executor task launch worker-0] executor.Executor
(Logging.scala:logInfo(59)) - Adding file:/tmp/spark-29a18af1-1ff2-4347-b6c7-a8c
937719d03/avro-mapred.jar to class loader
2015-02-05 22:18:48,041 INFO  [Executor task launch worker-0] executor.Executor
(Logging.scala:logInfo(59)) - Finished task 0.0 in stage 0.0 (TID 0). 572 bytes
result sent to driver
2015-02-05 22:18:48,089 INFO  [sparkDriver-akka.actor.default-dispatcher-6] sche
duler.DAGScheduler (Logging.scala:logInfo(59)) - Stage 0 (reduce at <console>:16
) finished in 1.538 s
2015-02-05 22:18:48,094 INFO  [task-result-getter-0] scheduler.TaskSetManager (L
ogging.scala:logInfo(59)) - Finished task 0.0 in stage 0.0 (TID 0) in 1067 ms on
 localhost (1/1)
2015-02-05 22:18:48,107 INFO  [task-result-getter-0] scheduler.TaskSchedulerImpl
 (Logging.scala:logInfo(59)) - Removed TaskSet 0.0, whose tasks have all complet
ed, from pool
2015-02-05 22:18:48,132 INFO  [main] scheduler.DAGScheduler (Logging.scala:logIn
fo(59)) - Job 0 finished: reduce at <console>:16, took 2.915489 s
count: Int = 78

scala> println("Pi is roughly" + 4.0*count/100)
Pi is roughly3.12
                Result

scala>
```

The result may change every time because this result is a estimating value.

- Think about it : Why we use Spark?

Problems with MR
- Very low-level : requires a lot of code to do simple things
- Very constrained : everything must be described as “map” and “reduce”. Powerful but sometimes difficult to think in these terms.

⬇

Spark is a general purpose computational framework retaining the advantages(Linear scalability, Fault-tolerance, Data Locality based computations) of MapReduce.
But offers so much more :
- Leverages distributed memory for better performance
- Supports iterative algorithms that are not feasible in MR
- improved developer experience
- Full Directed Graph expressions for data parallel computations
- Comes with libraries for machine learning, graph analysis, etc.

<< Example 2. Wordcount with Spark >>

■ Why do we care about count words?

a. Word count is challenging over massive amounts of data
  - Using a single compute note would be too time-consuming
  - Using distributed nodes requires moving data
  - Number of unique words can easily exceed available memory

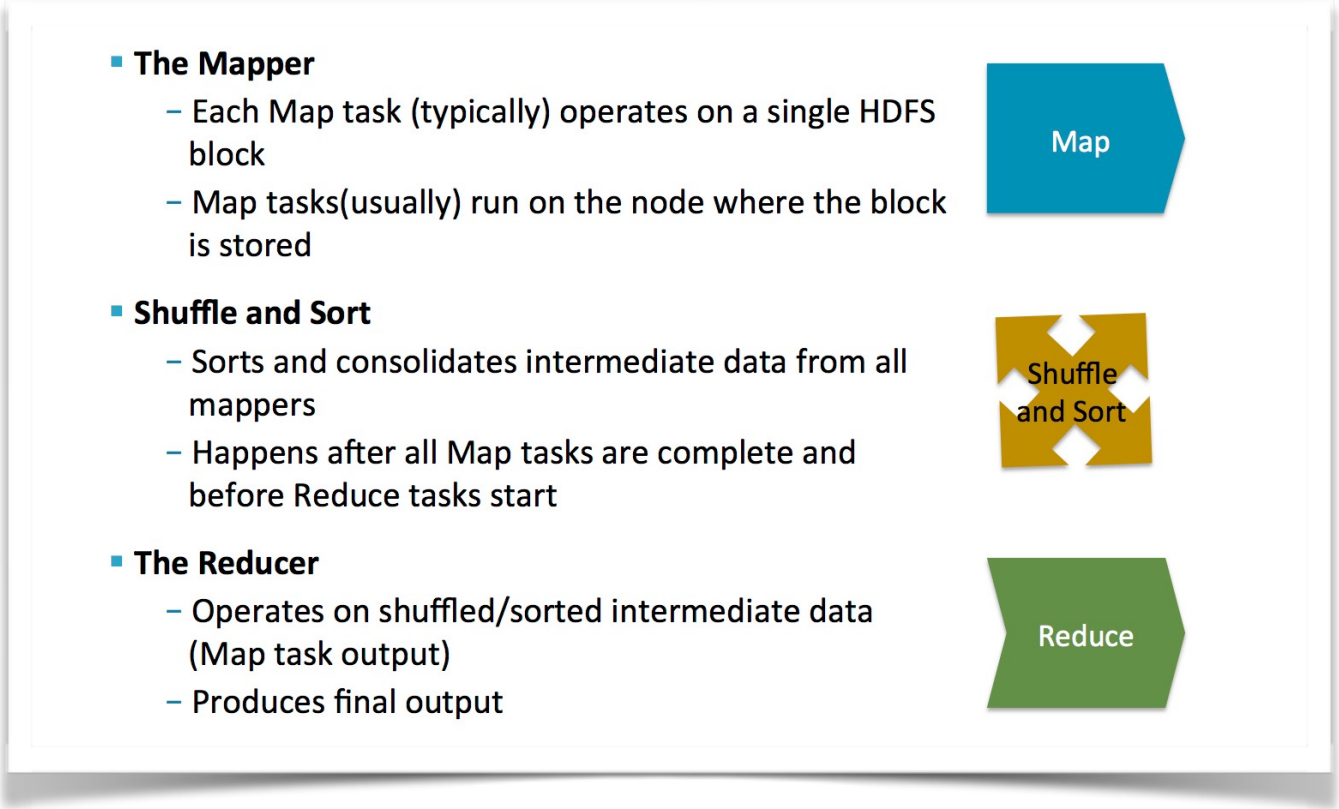
b. Statices are simple aggregate functions
  - Distributive in nature
  - e.g., max, min, sum, count

c. MapReduce breaks complex tasks down into smaller elements which can be executed in parallel

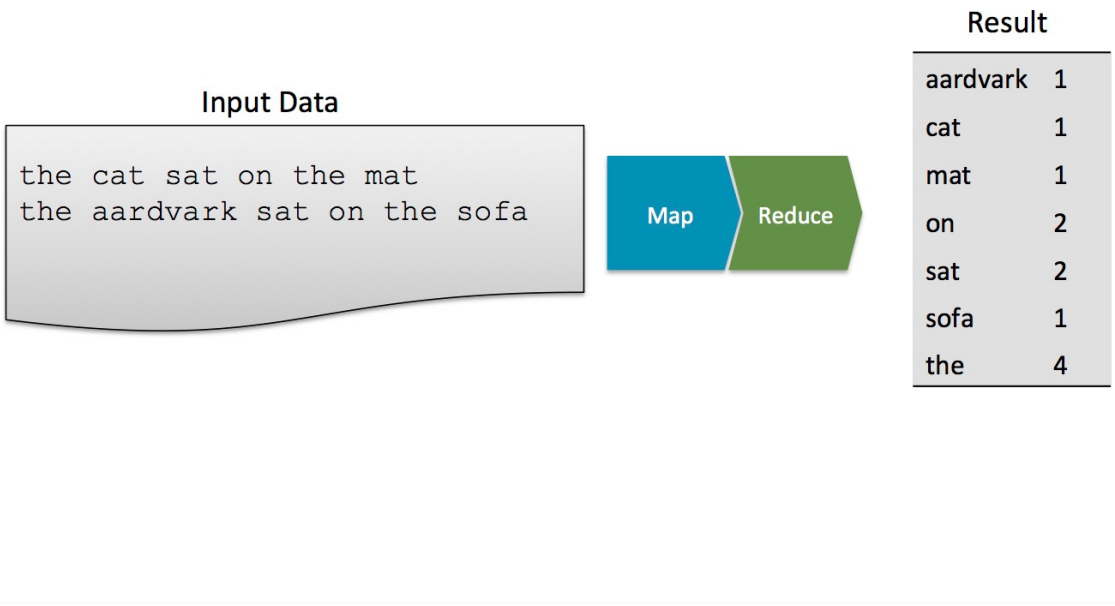
d. Many common tasks are very similar to word count
  - e.g., log file analysis

■ Process of MapReduce[1]

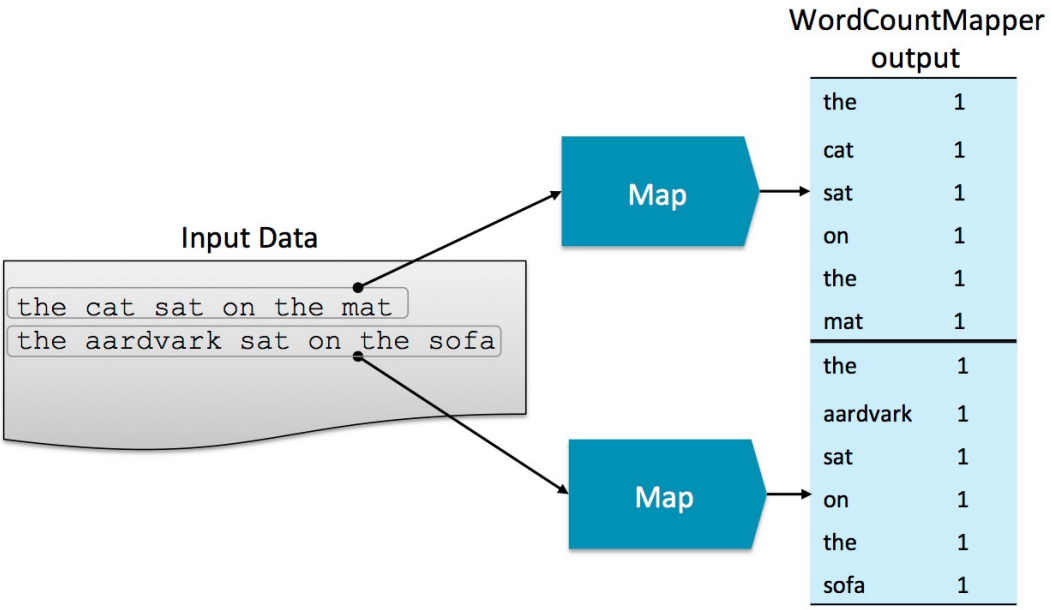
- Components



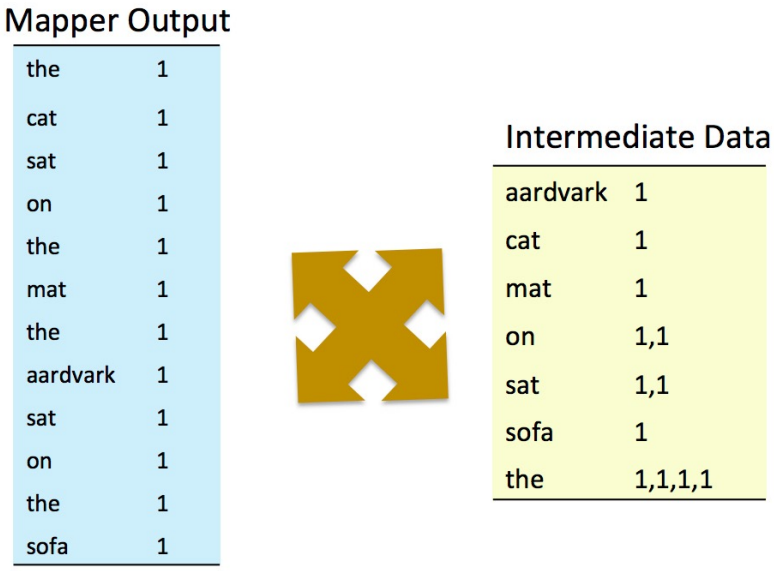
- Overall of wordcount



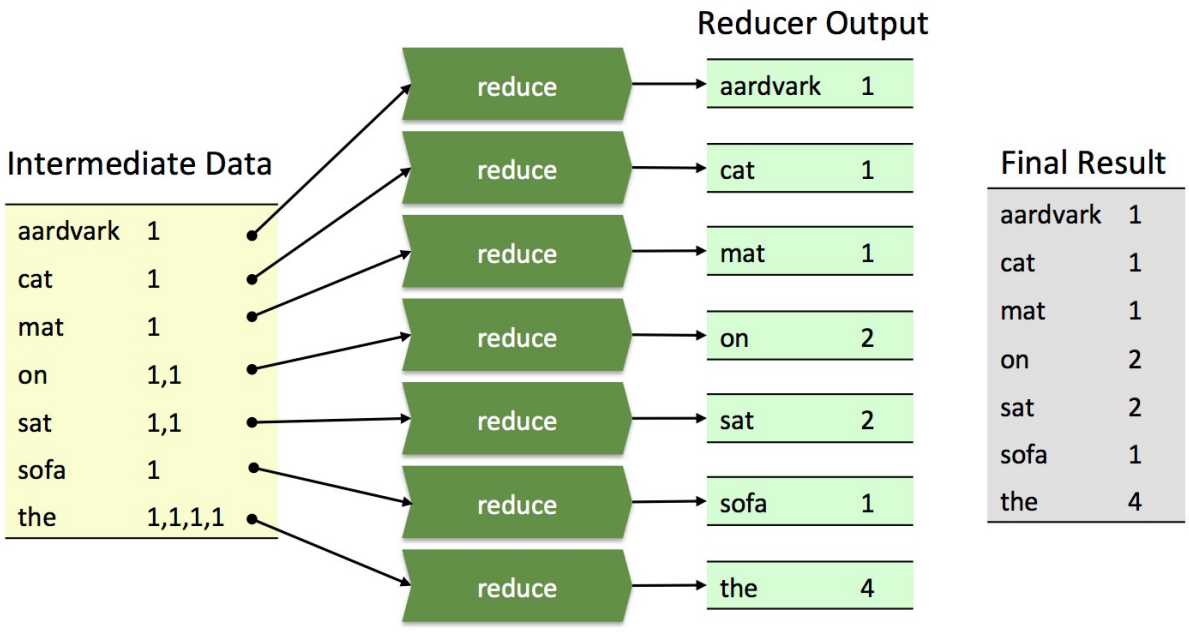
- Map



- Shuffle



- Reduce



- Copy the local dataset to Hadoop file system

```
//make the folder at hadoop file system
[cloudera@quickstart ~] sudo -u hdfs hadoop fs -mkdir /spark_exam

//copy the local dataset to hadoop file system
[cloudera@quickstart ~] hadoop fs -copyFromLocal ./spark_exam_dataset/*.* /spark_exam

//check the files
[cloudera@quickstart ~] hadoop fs -ls /spark_exam
```

```
[cloudera@quickstart ~]$ sudo -u hdfs hadoop fs -mkdir /spark_exam/
[cloudera@quickstart ~]$ hadoop fs -copyFromLocal ./spark_exam_dataset/*.* /spark_exam
[cloudera@quickstart ~]$ hadoop fs -ls /spark_exam
Found 5 items
-rw-r--r--   1 cloudera supergroup      10395 2015-02-08 03:04 /spark_exam/lpsa.data
-rw-r--r--   1 cloudera supergroup        177 2015-02-08 03:04 /spark_exam/people.txt
-rw-r--r--   1 cloudera supergroup     104736 2015-02-08 03:04 /spark_exam/sample_libsvm
 data.txt
-rw-r--r--   1 cloudera supergroup    4538523 2015-02-08 03:04 /spark_exam/shakespeare.t
xt
[cloudera@quickstart ~]$
```

Once you complete this step, you don't need to copy the dataset from local to hadoop file system again. We use these four files to another example.

- Open Spark shell

```
[cloudera@quickstart ~] spark-shell --jars /usr/lib/avro/avro-mapred.jar \
--conf spark.serializer=org.apache.spark.serializer.KryoSerializer
```

```
[cloudera@quickstart ~]$ spark-shell --jars /usr/lib/avro/avro-mapred.jar \
> --conf spark.serializer=org.apache.spark.serializer.KryoSerializer
2015-02-05 17:59:13,289 INFO  [main] spark.SecurityManager (Logging.scala:logInfo(59)) - C
hanging view acls to: cloudera
2015-02-05 17:59:13,297 INFO  [main] spark.SecurityManager (Logging.scala:logInfo(59)) - C
hanging modify acls to: cloudera
2015-02-05 17:59:13,299 INFO  [main] spark.SecurityManager (Logging.scala:logInfo(59)) - S
ecurityManager: authentication disabled; ui acls disabled; users with view permissions: Se
t(cloudera); users with modify permissions: Set(cloudera)
2015-02-05 17:59:13,300 INFO  [main] spark.HttpServer (Logging.scala:logInfo(59)) - Starti
ng HTTP Server
2015-02-05 17:59:13,438 INFO  [main] server.Server (Server.java:doStart(272)) - jetty-8.y.
z-SNAPSHOT
2015-02-05 17:59:13,479 INFO  [main] server.AbstractConnector (AbstractConnector.java:doSt
art(338)) - Started SocketConnector@0.0.0.0:59571
2015-02-05 17:59:13,481 INFO  [main] util.Utils (Logging.scala:logInfo(59)) - Successfully
 started service 'HTTP class server' on port 59571.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.2.0
      /_/
```

- Input the code

```
scala> val file = sc.textFile(“hdfs://quickstart.cloudera/spark_exam/ shaekspeare.txt“)
```

```
scala> val file = sc.textFile("hdfs://quickstart.cloudera/spark_exam/shakespeare.txt")
2015-02-05 18:02:29,276 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(59)) - ens
ureFreeSpace(181427) called with curMem=0, maxMem=280248975
2015-02-05 18:02:29,280 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(59)) - Blo
ck broadcast_0 stored as values in memory (estimated size 177.2 KB, free 267.1 MB)
2015-02-05 18:02:29,844 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(59)) - ens
ureFreeSpace(14118) called with curMem=181427, maxMem=280248975
2015-02-05 18:02:29,845 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(59)) - Blo
ck broadcast_0_piece0 stored as bytes in memory (estimated size 13.8 KB, free 267.1 MB)
2015-02-05 18:02:29,848 INFO  [sparkDriver-akka.actor.default-dispatcher-2] storage.BlockM
anagerInfo (Logging.scala:logInfo(59)) - Added broadcast_0_piece0 in memory on localhost:4
8341 (size: 13.8 KB, free: 267.3 MB)
2015-02-05 18:02:29,849 INFO  [main] storage.BlockManagerMaster (Logging.scala:logInfo(59)
) - Updated info of block broadcast_0_piece0
2015-02-05 18:02:29,855 INFO  [main] spark.SparkContext (Logging.scala:logInfo(59)) - Crea
ted broadcast 0 from textFile at <console>:12
file: org.apache.spark.rdd.RDD[String] = hdfs://quickstart.cloudera/spark_exam/shakespeare
.txt MappedRDD[1] at textFile at <console>:12

scala>
```

```
scala> val counts = file.flatMap(line => line.split(" “)).map( word =>
(word,1)).reduceByKey(_ + _)
```

```
2015-02-05 18:02:29,280 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(59)) - Blo
ck broadcast_0 stored as values in memory (estimated size 177.2 KB, free 267.1 MB)
2015-02-05 18:02:29,844 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(59)) - ens
ureFreeSpace(14118) called with curMem=181427, maxMem=280248975
2015-02-05 18:02:29,845 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(59)) - Blo
ck broadcast_0_piece0 stored as bytes in memory (estimated size 13.8 KB, free 267.1 MB)
2015-02-05 18:02:29,848 INFO  [sparkDriver-akka.actor.default-dispatcher-2] storage.BlockM
anagerInfo (Logging.scala:logInfo(59)) - Added broadcast_0_piece0 in memory on localhost:4
8341 (size: 13.8 KB, free: 267.3 MB)
2015-02-05 18:02:29,849 INFO  [main] storage.BlockManagerMaster (Logging.scala:logInfo(59)
) - Updated info of block broadcast_0_piece0
2015-02-05 18:02:29,855 INFO  [main] spark.SparkContext (Logging.scala:logInfo(59)) - Crea
ted broadcast 0 from textFile at <console>:12
file: org.apache.spark.rdd.RDD[String] = hdfs://quickstart.cloudera/spark_exam/shakespeare
.txt MappedRDD[1] at textFile at <console>:12

scala> val counts = file.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByK
ey(_ + _)
2015-02-05 18:04:53,946 INFO  [main] mapred.FileInputFormat (FileInputFormat.java:listStat
us(247)) - Total input paths to process : 1
counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at <consol
e>:14

scala>
```

scala> counts.saveAsTextFile("hdfs://quickstart.cloudera/spark_exam/wordcount")

```
scala> counts.saveAsTextFile("hdfs://quickstart.cloudera/spark_exam/wordcount")
2015-02-05 18:06:53,592 INFO  [main] Configuration.deprecation (Configuration.java:warnOnc
eIfDeprecated(1022)) - mapred.tip.id is deprecated. Instead, use mapreduce.task.id
2015-02-05 18:06:53,593 INFO  [main] Configuration.deprecation (Configuration.java:warnOnc
eIfDeprecated(1022)) - mapred.task.id is deprecated. Instead, use mapreduce.task.attempt.i
d
2015-02-05 18:06:53,593 INFO  [main] Configuration.deprecation (Configuration.java:warnOnc
eIfDeprecated(1022)) - mapred.task.is.map is deprecated. Instead, use mapreduce.task.ismap
2015-02-05 18:06:53,594 INFO  [main] Configuration.deprecation (Configuration.java:warnOnc
eIfDeprecated(1022)) - mapred.task.partition is deprecated. Instead, use mapreduce.task.pa
rtition
2015-02-05 18:06:53,594 INFO  [main] Configuration.deprecation (Configuration.java:warnOnc
eIfDeprecated(1022)) - mapred.job.id is deprecated. Instead, use mapreduce.job.id
2015-02-05 18:06:53,899 INFO  [main] spark.SparkContext (Logging.scala:logInfo(59)) - Star
ting job: saveAsTextFile at <console>:17
2015-02-05 18:06:53,920 INFO  [sparkDriver-akka.actor.default-dispatcher-2] scheduler.DAGS
cheduler (Logging.scala:logInfo(59)) - Registering RDD 3 (map at <console>:14)
2015-02-05 18:06:53,923 INFO  [sparkDriver-akka.actor.default-dispatcher-2] scheduler.DAGS
cheduler (Logging.scala:logInfo(59)) - Got job 0 (saveAsTextFile at <console>:17) with 1 o
utput partitions (allowLocal=false)
2015-02-05 18:06:53,924 INFO  [sparkDriver-akka.actor.default-dispatcher-2] scheduler.DAGS
cheduler (Logging.scala:logInfo(59)) - Final stage: Stage 1(saveAsTextFile at <console>:17
)
2015-02-05 18:06:53,929 INFO  [sparkDriver-akka.actor.default-dispatcher-2] scheduler.DAGS
```

```
2015-02-05 18:07:00,632 INFO  [Executor task launch worker-0] storage.ShuffleBlockFetcherI
terator (Logging.scala:logInfo(59)) - Getting 1 non-empty blocks out of 1 blocks
2015-02-05 18:07:00,642 INFO  [Executor task launch worker-0] storage.ShuffleBlockFetcherI
terator (Logging.scala:logInfo(59)) - Started 0 remote fetches in 15 ms
2015-02-05 18:07:03,057 INFO  [Executor task launch worker-0] output.FileOutputCommitter (
FileOutputCommitter.java:commitTask(439)) - Saved output of task 'attempt_201502051806_000
1_m_000000_1' to hdfs://quickstart.cloudera/spark_exam/wordcount/_temporary/0/task_2015020
51806_0001_m_000000
2015-02-05 18:07:03,058 INFO  [Executor task launch worker-0] spark.SparkHadoopWriter (Log
ging.scala:logInfo(59)) - attempt_201502051806_0001_m_000000_1: Committed
2015-02-05 18:07:03,073 INFO  [Executor task launch worker-0] executor.Executor (Logging.s
cala:logInfo(59)) - Finished task 0.0 in stage 1.0 (TID 1). 1697 bytes result sent to driv
er
2015-02-05 18:07:03,136 INFO  [sparkDriver-akka.actor.default-dispatcher-5] scheduler.DAGS
cheduler (Logging.scala:logInfo(59)) - Stage 1 (saveAsTextFile at <console>:17) finished i
n 2.877 s
2015-02-05 18:07:03,143 INFO  [task-result-getter-1] scheduler.TaskSetManager (Logging.sca
la:logInfo(59)) - Finished task 0.0 in stage 1.0 (TID 1) in 2877 ms on localhost (1/1)
2015-02-05 18:07:03,144 INFO  [task-result-getter-1] scheduler.TaskSchedulerImpl (Logging.
scala:logInfo(59)) - Removed TaskSet 1.0, whose tasks have all completed, from pool
2015-02-05 18:07:03,157 INFO  [main] scheduler.DAGScheduler (Logging.scala:logInfo(59)) -
Job 0 finished: saveAsTextFile at <console>:17, took 9.257511 s

scala>
```

```
scala> counts.count()
```

```
scala> counts.count()
2015-02-05 18:08:25,416 INFO  [main] spark.SparkContext (Logging.scala:logInfo(59)) - Star
ting job: count at <console>:17
2015-02-05 18:08:25,422 INFO  [sparkDriver-akka.actor.default-dispatcher-13] spark.MapOutp
utTrackerMaster (Logging.scala:logInfo(59)) - Size of output statuses for shuffle 0 is 145
 bytes
2015-02-05 18:08:25,428 INFO  [sparkDriver-akka.actor.default-dispatcher-13] scheduler.DAG
Scheduler (Logging.scala:logInfo(59)) - Got job 1 (count at <console>:17) with 1 output pa
rtitions (allowLocal=false)
2015-02-05 18:08:25,428 INFO  [sparkDriver-akka.actor.default-dispatcher-13] scheduler.DAG
Scheduler (Logging.scala:logInfo(59)) - Final stage: Stage 3(count at <console>:17)
2015-02-05 18:08:25,428 INFO  [sparkDriver-akka.actor.default-dispatcher-13] scheduler.DAG
Scheduler (Logging.scala:logInfo(59)) - Parents of final stage: List(Stage 2)
2015-02-05 18:08:25,431 INFO  [sparkDriver-akka.actor.default-dispatcher-13] scheduler.DAG
Scheduler (Logging.scala:logInfo(59)) - Missing parents: List()
2015-02-05 18:08:25,432 INFO  [sparkDriver-akka.actor.default-dispatcher-13] scheduler.DAG
Scheduler (Logging.scala:logInfo(59)) - Submitting Stage 3 (ShuffledRDD[4] at reduceByKey
at <console>:14), which has no missing parents
2015-02-05 18:08:25,434 INFO  [sparkDriver-akka.actor.default-dispatcher-13] storage.Memor
yStore (Logging.scala:logInfo(59)) - ensureFreeSpace(2072) called with curMem=368591, maxM
em=280248975
```

```
scala> counts.toArray().foreach(println)
```

This means that the result of reduce is to make array format and print it to the screen.

RESULT

```
(Arise,15)
(chinks,1)
(chamberers,1)
(understand,99)
(Placentio,1)
(singing,13)
(bellow,1)
(well-knit,1)
(hostility,3)
(stoup,4)
(Sleep,17)
(memorable,4)
(Alas,168)
(scratching,1)
(rip,2)
(quired,1)
(vine,7)
(heart-sore,2)
(Able,2)
(Presumptuous,3)
(subsidies,1)
(absolutely,2)

scala>
```

- Think about it : How to translate MapReduce to Spark?[2]

The key to getting the most out of Spark is to understand the differences between its RDD API and the original Mapper and Reducer API.

Venerable **MapReduce** has been **Apache Hadoop**'s work-horse computation paradigm since its inception. It is ideal for the kinds of work for which Hadoop was originally designed: large-scale log processing, and batch-oriented ETL (extract-transform-load) operations.

Today, **Apache Spark** is another such alternative, and is said by many to succeed MapReduce as Hadoop's general-purpose computation paradigm. But if MapReduce has been so useful, how can it suddenly be replaced? After all, there is still plenty of ETL-like work to be done on Hadoop, even if the platform now has other real-time capabilities as well.

Thankfully, it's entirely possible to re-implement MapReduce-like computations in Spark. They can be simpler to maintain, and in some cases faster, thanks to Spark's ability to optimize away spilling to disk. For MapReduce, re-implementation on Spark is a homecoming. Spark, after all, mimics **Scala**'s functional programming style and APIs. And the very **idea of MapReduce comes from** the functional programming language **LISP**.

Although Spark's primary abstraction, the **RDD** (Resilient Distributed Dataset), plainly exposes map() and reduce() operations, these are not the direct analog of Hadoop's **Mapper** or **Reducer** APIs. This is often a stumbling block for developers looking to move Mapper and Reducer classes to Spark equivalents.

Viewed in comparison with classic functional language implementations of map() and reduce() in Scala or Spark, the Mapper and Reducer APIs in Hadoop are actually both more flexible and more complex as a result.

Reducer and reduce() versus reduceByKey()

To produce a count of line lengths, it's necessary to sum the counts per length in a Reducer :

```
public class LineLengthReducer
   extends Reducer<IntWritable,IntWritable,IntWritable,IntWritable> {
  @Override
  protected void reduce(IntWritable length, Iterable<IntWritable> counts, Context context)
      throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable count : counts) {
      sum += count.get();
    }
    context.write(length, new IntWritable(sum));
  }
}
```

The equivalent of the Mapper and Reducer above together is a one-liner in Spark :

```
val lengthCounts = lines.map(line => (line.length, 1)).reduceByKey(_ + _)
```

Spark's RDD API has a reduce() method, but it will reduce the entire set of key-value pairs to one single value. This is not what Hadoop MapReduce does. Instead, Reducers reduce all values for a key and emit a key along with the reduced value. reduceByKey() is the closer analog. But, that is not even the most direct equivalent in Spark; see groupByKey() below.
It is worth pointing out here that a Reducer's **reduce()** method receives a stream of many values, and produces 0, 1 or more results. reduceByKey(), in contrast, accepts a function that turns exactly two values into exactly one — here, a simple addition function that maps two numbers to their sum. This associative function can be used to reduce many values to one for the caller. It is a simpler, narrower API for reducing values by key than what a Reducer exposes.

Mapper and map() versus flatMap()

Now, instead consider counting the occurrences of only words beginning with an uppercase character. For each line of text in the input, a Mapper might emit 0, 1 or many key-value pairs :

```
public class CountUppercaseMapper
    extends Mapper<LongWritable,Text,Text,IntWritable> {
  @Override
  protected void map(LongWritable lineNumber, Text line, Context context)
      throws IOException, InterruptedException {
    for (String word : line.toString().split(" ")) {
      if (Character.isUpperCase(word.charAt(0))) {
        context.write(new Text(word), new IntWritable(1));
      }
    }
  }
}
```

The equivalent in Spark is :

```
lines.flatMap(
  _.split(" ").filter(word => Character.isUpperCase(word(0))).map(word => (word,1))
)
```

map() will not suffice here, because map() must produce exactly one output per input, but unlike before, one line needs to yield potentially many outputs. Again, the map() function in Spark is simpler and narrower compared to what the Mapper API supports.
The solution in Spark is to first map each line to an array of output values. The array may be empty, or have many values. Merely map()-ing lines to arrays would produce an RDD of arrays as the result, when the result should be the contents of those arrays. The result needs to be "flattened" afterward, and flatMap() does exactly this. Here, the array of words in the line is filtered and converted into tuples inside the function. In a case like this, it's flatMap() that's required to emulate such a Mapper, not map().

<< Example 3. Logistics Regression >>

Logistic Regression

Logistic regression is widely used to predict a binary response.
It is a linear method as described above in equation:

$$f(w) := \lambda R(w) + \frac{1}{n}\sum_{i=1}^{n} L(w;x_i,y_i)$$

, with the loss function in the formulation given by the logistic loss:

$$L(w;x,y) := \log(1 + \exp(-yw^T x))$$

The logistic regression algorithm outputs a logistic regression model.
Given a new data point, denoted by x, the model makes predictions by applying the logistic function

$$f(z) = \frac{1}{1+e^{-z}}$$

where z=wTx. By default, if f(wTx)>0.5, the outcome is positive, or negative otherwise, though unlike linear SVMs, the raw output of the logistic regression model, f(z), has a probabilistic interpretation (i.e., the probability that x is positive).

Evaluation Metrics

MLlib* supports common evaluation metrics for binary classification (not available in PySpark). This includes precision, recall, F-measure, receiver operating characteristic (ROC), precision-recall curve, and area under the curves (AUC). AUC is commonly used to compare the performance of various models while precision/recall/F-measure can help determine the appropriate threshold to use for prediction purposes.

> * MLlib(Machine Learning Library)
> MLlib is Spark's scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as underlying optimization primitives.

- Intro

The following code snippet illustrates how to load a sample dataset, execute a training algorithm on this training data using a static method in the algorithm object, and make predictions with the resulting model to compute the training error.

- Open Spark shell

```
[cloudera@quickstart ~] spark-shell --jars /usr/lib/avro/avro-mapred.jar \
--conf spark.serializer=org.apache.spark.serializer.KryoSerializer
```

- Input the code

```
// Import the libraries.
scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
scala>  import org.apache.spark.SparkContext
scala> import org.apache.spark.mllib.classification.SVMWithSGD
scala> import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
scala> import org.apache.spark.mllib.regression.LabeledPoint
scala> import org.apache.spark.mllib.linalg.Vectors
scala> import org.apache.spark.mllib.util.MLUtils
```

```
scala> import org.apache.spark.SparkContext
import org.apache.spark.SparkContext

scala> import org.apache.spark.mllib.classification.SVMWithSGD
import org.apache.spark.mllib.classification.SVMWithSGD

scala> import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics
import org.apache.spark.mllib.evaluation.BinaryClassificationMetrics

scala> import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LabeledPoint

scala> import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Vectors

scala> import org.apache.spark.mllib.util.MLUtils
import org.apache.spark.mllib.util.MLUtils
```

```
// Load training data in LIBSVM format.
scala> val data = MLUtils.loadLibSVMFile(sc, "hdfs://quickstart.cloudera/spark_exam/sample_libsvm_data.txt")
```

```
scala> val data = MLUtils.loadLibSVMFile(sc, "hdfs://quickstart.cloudera/spark_e
xam/sample_libsvm_data.txt")
2015-02-07 08:33:42,204 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(
59)) - ensureFreeSpace(181427) called with curMem=0, maxMem=280248975
2015-02-07 08:33:42,208 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(
59)) - Block broadcast_0 stored as values in memory (estimated size 177.2 KB, fr
ee 267.1 MB)
2015-02-07 08:33:43,635 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(
59)) - ensureFreeSpace(14118) called with curMem=181427, maxMem=280248975
2015-02-07 08:33:43,637 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(
59)) - Block broadcast_0_piece0 stored as bytes in memory (estimated size 13.8 K
B, free 267.1 MB)
2015-02-07 08:33:43,639 INFO  [sparkDriver-akka.actor.default-dispatcher-13] sto
rage.BlockManagerInfo (Logging.scala:logInfo(59)) - Added broadcast_0_piece0 in
memory on localhost:55181 (size: 13.8 KB, free: 267.3 MB)
2015-02-07 08:33:43,640 INFO  [main] storage.BlockManagerMaster (Logging.scala:l
ogInfo(59)) - Updated info of block broadcast_0_piece0
2015-02-07 08:33:43,646 INFO  [main] spark.SparkContext (Logging.scala:logInfo(5
9)) - Created broadcast 0 from textFile at MLUtils.scala:74
```

```
// Split data into training (60%) and test (40%). And set the training, test variables.
scala> val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)
scala> val training = splits(0).cache()*
scala> val test = splits(1)
```

* cache() : spark will load the parsed data into memory

```
scala> val splits = data.randomSplit(Array(0.6, 0.4), seed = 11L)
2015-02-07 08:37:54,653 INFO  [sparkDriver-akka.actor.default-dispatcher-4] stor
age.BlockManager (Logging.scala:logInfo(59)) - Removing broadcast 1
2015-02-07 08:37:54,658 INFO  [sparkDriver-akka.actor.default-dispatcher-4] stor
age.BlockManager (Logging.scala:logInfo(59)) - Removing block broadcast_1_piece0
2015-02-07 08:37:54,660 INFO  [sparkDriver-akka.actor.default-dispatcher-4] stor
age.MemoryStore (Logging.scala:logInfo(59)) - Block broadcast_1_piece0 of size 1
977 dropped from memory (free 279879318)
2015-02-07 08:37:54,688 INFO  [sparkDriver-akka.actor.default-dispatcher-5] stor
age.BlockManagerInfo (Logging.scala:logInfo(59)) - Removed broadcast_1_piece0 on
 localhost:55181 in memory (size: 1977.0 B, free: 267.1 MB)
2015-02-07 08:37:54,688 INFO  [sparkDriver-akka.actor.default-dispatcher-4] stor
age.BlockManagerMaster (Logging.scala:logInfo(59)) - Updated info of block broad
cast_1_piece0
```

```
scala> val training = splits(0).cache()
training: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoin
t] = PartitionwiseSampledRDD[7] at randomSplit at <console>:20

scala> val test = splits(1)
test: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoint] =
 PartitionwiseSampledRDD[8] at randomSplit at <console>:20

scala>
```

```
// Run training algorithm to build the model.
scala> val numIterations = 100
scala> val model = SVMWithSGD.train(training, numIternations)*
```

* The SVMWithSGD.train() method by default performs L2 regularization with the regularization parameter set to 1.0. If we want to configure this algorithm, we can customize SVMWithSGD further by creating a new object directly and calling setter methods. All other MLlib algorithms support customization in this way as well. For example, the following code produces an L1 regularized variant of SVMs with regularization parameter set to 0.1, and runs the training algorithm for 200 iterations.

```
import org.apache.spark.mllib.optimization.L1Updater

val svmAlg = new SVMWithSGD()
svmAlg.optimizer.
  setNumIterations(200).
  setRegParam(0.1).
```

LogisticRegressionWithSGD can be used in a similar fashion as SVMWithSGD.

```
2015-02-07 08:40:43,251 INFO  [main] scheduler.DAGScheduler (Logging.scala:logIn
fo(59)) - Job 103 finished: reduce at RDDFunctions.scala:112, took 0.171191 s
2015-02-07 08:40:43,253 INFO  [main] optimization.GradientDescent (Logging.scala
:logInfo(59)) - GradientDescent.runMiniBatchSGD finished. Last 10 stochastic los
ses 3275.1870023499514, 3268.323943712561, 3261.5125697178705, 3254.752014747527
, 3248.04143649104, 3241.380015080995, 3234.76695226901, 3228.2014706401237, 322
1.6828128634725, 3215.2102409771383
2015-02-07 08:40:43,254 INFO  [task-result-getter-3] scheduler.TaskSetManager (L
ogging.scala:logInfo(59)) - Finished task 0.0 in stage 103.0 (TID 103) in 99 ms
on localhost (1/1)
2015-02-07 08:40:43,254 INFO  [task-result-getter-3] scheduler.TaskSchedulerImpl
 (Logging.scala:logInfo(59)) - Removed TaskSet 103.0, whose tasks have all compl
eted, from pool
model: org.apache.spark.mllib.classification.SVMModel = (weights=[0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.
0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,-0.8238181569207137,-3.63362651356
10044,-1.7800356604893992,-1.0297726961508917,1.8579866496626125,3.7797031989598
784,-1.6868567829169763,-0.07355519258220666,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,-0.9267954265358038,-3.574782359495243,-3.2
36428473617091,-5.281262827402429,-11.636431466505092,-14.328551515013842,-12...
scala>
```

// Clear the default threshold.
scala> model.clearThreshold()

```
scala> model.clearThreshold()
res0: model.type = (weights=[0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,-0.8238181569207137,-3.6336265135610044,-1.7800356604893992,-1.02977269
61508917,1.8579866496626125,3.7797031989598784,-1.6868567829169763,-0.0735551925
8220666,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,
-0.9267954265358038,-3.574782359495243,-3.236428473617091,-5.281262827402429,-11
.636431466505092,-14.328551515013842,-12.17330767540053,-12.733387739695708,-...
scala>
```

// Compute raw scores on the test set.
In statistics and data analysis, a raw score is an original datum that has not been transformed.
scala> val scoreAndLabels = test.map { point =>
val score = model.predict(point.features)
(score, point.label)
}

```
scala> val scoreAndLabels = test.map { point =>
     | val score = model.predict(point.features)
     | (score, point.label)
     | }
2015-02-07 08:44:25,255 INFO  [sparkDriver-akka.actor.default-dispatcher-3] stor
age.BlockManager (Logging.scala:logInfo(59)) - Removing broadcast 204
2015-02-07 08:44:25,255 INFO  [sparkDriver-akka.actor.default-dispatcher-3] stor
age.BlockManager (Logging.scala:logInfo(59)) - Removing block broadcast_204_piec
e0
2015-02-07 08:44:25,255 INFO  [sparkDriver-akka.actor.default-dispatcher-3] stor
age.MemoryStore (Logging.scala:logInfo(59)) - Block broadcast_204_piece0 of size
 3501 dropped from memory (free 279640101)
```

```
// Get evaluation metrics.
scala> val metrics = new BinaryClassificationMetrics(scoreAndLabels)
scala> val auROC = metrics.areaUnderROC()
```

```
scala> val metrics = new BinaryClassificationMetrics(scoreAndLabels)
metrics: org.apache.spark.mllib.evaluation.BinaryClassificationMetrics = org.apa
che.spark.mllib.evaluation.BinaryClassificationMetrics@68b40d50
```

```
scala> val auROC = metrics.areaUnderROC()
2015-02-07 08:45:53,578 INFO  [main] spark.SparkContext (Logging.scala:logInfo(5
9)) - Starting job: collect at BinaryClassificationMetrics.scala:110
2015-02-07 08:45:53,590 INFO  [sparkDriver-akka.actor.default-dispatcher-3] sche
duler.DAGScheduler (Logging.scala:logInfo(59)) - Registering RDD 211 (map at <co
nsole>:30)
2015-02-07 08:45:53,591 INFO  [sparkDriver-akka.actor.default-dispatcher-3] sche
duler.DAGScheduler (Logging.scala:logInfo(59)) - Registering RDD 212 (combineByK
ey at BinaryClassificationMetrics.scala:101)
```

```
2015-02-07 08:46:04,264 INFO  [task-result-getter-2] scheduler.TaskSetManager (L
ogging.scala:logInfo(59)) - Finished task 0.0 in stage 112.0 (TID 110) in 95 ms
on localhost (2/3)
2015-02-07 08:46:04,303 INFO  [Executor task launch worker-2] executor.Executor
(Logging.scala:logInfo(59)) - Finished task 2.0 in stage 112.0 (TID 111). 576 by
tes result sent to driver
2015-02-07 08:46:04,307 INFO  [sparkDriver-akka.actor.default-dispatcher-15] sch
eduler.DAGScheduler (Logging.scala:logInfo(59)) - Stage 112 (aggregate at AreaUn
derCurve.scala:45) finished in 0.182 s
2015-02-07 08:46:04,308 INFO  [main] scheduler.DAGScheduler (Logging.scala:logIn
fo(59)) - Job 106 finished: aggregate at AreaUnderCurve.scala:45, took 0.297890
s
auROC: Double = 1.0

scala> 2015-02-07 08:46:04,313 INFO  [task-result-getter-3] scheduler.TaskSetMan
ager (Logging.scala:logInfo(59)) - Finished task 2.0 in stage 112.0 (TID 111) in
 67 ms on localhost (3/3)
2015-02-07 08:46:04,313 INFO  [task-result-getter-3] scheduler.TaskSchedulerImpl
 (Logging.scala:logInfo(59)) - Removed TaskSet 112.0, whose tasks have all compl
eted, from pool
val auR

scala>
```

```
scala> println("Area under ROC = " + auROC)
```

```
scala> println("Area under ROC = " + auROC)
Area under ROC = 1.0

scala>
```

Result

- Dataset

```
0 128:51 129:159 130:253 131:159 132:50 155:48 156:238 157:252 158:252 159:25
2 160:237 182:54 183:227 184:253 185:252 186:239 187:233 188:252 189:57 190:6
 208:10 209:60 210:224 211:252 212:253 213:252 214:202 215:84 216:252 217:253
 218:122 236:163 237:252 238:252 239:252 240:253 241:252 242:252 243:96 244:1
89 245:253 246:167 263:51 264:238 265:253 266:253 267:190 268:114 269:253 270
:228 271:47 272:79 273:255 274:168 290:48 291:238 292:252 293:252 294:179 295
:12 296:75 297:121 298:21 301:253 302:243 303:50 317:38 318:165 319:253 320:2
33 321:208 322:84 329:253 330:252 331:165 344:7 345:178 346:252 347:240 348:7
1 349:19 350:28 357:253 358:252 359:195 372:57 373:252 374:252 375:63 385:253
 386:252 387:195 400:198 401:253 402:190 413:255 414:253 415:196 427:76 428:2
46 429:252 430:112 441:253 442:252 443:148 455:85 456:252 457:230 458:25 467:
7 468:135 469:253 470:186 471:12 483:85 484:252 485:223 494:7 495:131 496:252
 497:225 498:71 511:85 512:252 513:145 521:48 522:165 523:252 524:173 539:86
540:253 541:225 548:114 549:238 550:253 551:162 567:85 568:252 569:249 570:14
6 571:48 572:29 573:85 574:178 575:225 576:253 577:223 578:167 579:56 595:85
596:252 597:252 598:252 599:229 600:215 601:252 602:252 603:252 604:196 605:1
30 623:28 624:199 625:252 626:252 627:253 628:252 629:252 630:233 631:145 652
:25 653:128 654:252 655:253 656:252 657:141 658:37
```

Linear regression

Linear regression belongs to the family of regression algorithms.
The goal of regression is to find relationships and dependencies between variables. It is modeling the relationship between a continuous scalar dependent variable y (also label or target in machine learning terminology) and one or more (a D-dimensional vector) explanatory variables (also independent variables, input variables, features, observed data, observations, attributes, dimensions, data point, ..) denoted X using a linear function.

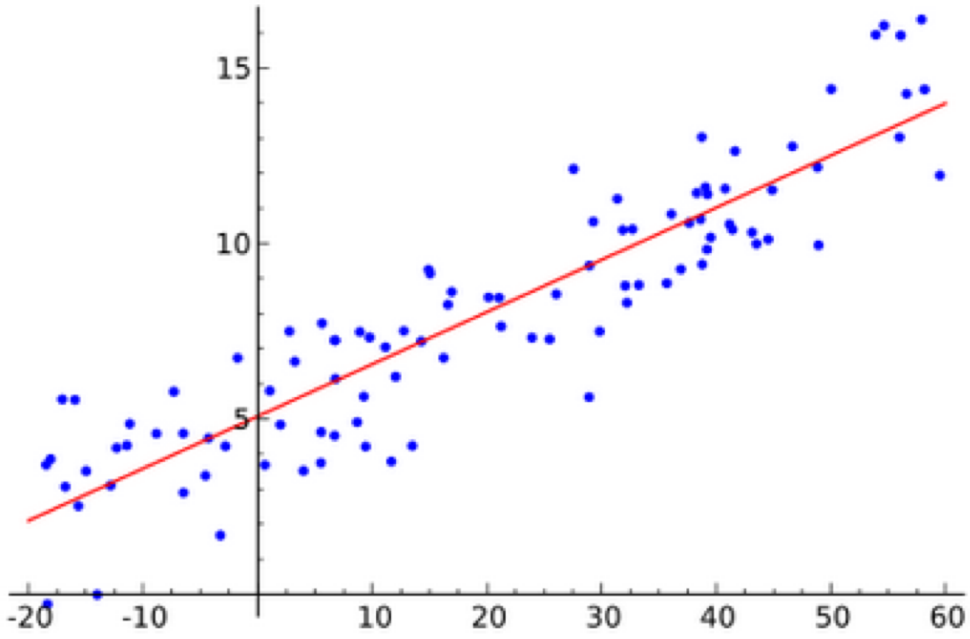
The model for a multiple regression which involves linear combination of input variables takes the form

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \ldots.$$

Linear regression belongs to a category of supervised learning algorithms.
It means we train the model on a set of labeled data (training data) and then use the model to predict labels on unlabeled data (test data).

Many algorithms to calculate linear regression exist, but it is not a focus of this article. Perhaps the most known approach is the least squares method.

Below is an example of simple linear regression with one independent variable (x axis). The model (red line) is calculated using training data (blue points) where each point has a known label (y axis) to fit the points as accurately as possible by minimizing the value of a chosen loss function. We can then use the model to predict unknown labels (we only know x value and want to predict y value).



Linear least square, Lasso, and ridge regression

Linear least squares is the most common formulation for regression problems.
It is a linear method as described above in equation

,

$$f(w) := \lambda R(w) + \frac{1}{n}\sum_{i=1}^{n} L(w;x_i,y_i),$$

with the loss function in the formulation given by the squared loss:

$$L(w;x,y) := \frac{1}{2}(w^T x - y)^2$$

Various related regression methods are derived by using different types of regularization:
ordinary least squares or linear least squares uses no regularization; ridge regression uses L2 regularization; and Lasso uses L1 regularization. For all of these models, the average loss or training error, $\frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2$ is known as the mean squared error.

- Intro

The following example demonstrate how to load training data, parse it as an RDD of LabeledPoint. The example then uses LinearRegressionWithSGD to build a simple linear model to predict label values. We compute the mean squared error at the end to evaluate goodness of fit.

- Open Spark shell

```
[cloudera@quickstart ~] spark-shell --jars /usr/lib/avro/avro-mapred.jar \
--conf spark.serializer=org.apache.spark.serializer.KryoSerializer
```

- Input the code

```
// Import the libraries.
scala> import org.apache.spark.SparkContext
scala> import org.apache.spark.mllib.regression.LinearRegressionWithSGD
scala> import org.apache.spark.mllib.regression.LabeledPoint
scala> import org.apache.spark.mllib.linalg.Vectors
```

```
scala> import org.apache.spark.SparkContext
import org.apache.spark.SparkContext

scala> import org.apache.spark.mllib.regression.LinearRegressionWithSGD
import org.apache.spark.mllib.regression.LinearRegressionWithSGD

scala> import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.regression.LabeledPoint

scala> import org.apache.spark.mllib.linalg.Vectors
import org.apache.spark.mllib.linalg.Vectors
```

```
// Load and parse the data
scala> val data = sc.textFile("hdfs://quickstart.cloudera/spark_exam/lpsa.data")
scala> val parsedData = data.map { line =>
scala> val parts = line.split(',')
LabeledPoint(parts(0).toDouble, Vectors.dense(parts(1).split(' ').map(x => x.toDouble).toArray))
```

```
scala> val data = sc.textFile("hdfs://quickstart.cloudera/spark_exam/lpsa.data")

2015-02-08 01:05:34,813 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(
59)) - ensureFreeSpace(181475) called with curMem=195545, maxMem=280248975
2015-02-08 01:05:34,814 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(
59)) - Block broadcast_1 stored as values in memory (estimated size 177.2 KB, fr
ee 266.9 MB)
2015-02-08 01:05:34,918 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(
59)) - ensureFreeSpace(14118) called with curMem=377020, maxMem=280248975
2015-02-08 01:05:34,919 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(
59)) - Block broadcast_1_piece0 stored as bytes in memory (estimated size 13.8 K
B, free 266.9 MB)
2015-02-08 01:05:34,920 INFO  [sparkDriver-akka.actor.default-dispatcher-4] stor
age.BlockManagerInfo (Logging.scala:logInfo(59)) - Added broadcast_1_piece0 in m
emory on localhost:59200 (size: 13.8 KB, free: 267.2 MB)
2015-02-08 01:05:34,921 INFO  [main] storage.BlockManagerMaster (Logging.scala:l
ogInfo(59)) - Updated info of block broadcast_1_piece0
2015-02-08 01:05:34,922 INFO  [main] spark.SparkContext (Logging.scala:logInfo(5
9)) - Created broadcast 1 from textFile at <console>:16
data: org.apache.spark.rdd.RDD[String] = hdfs://quickstart.cloudera/spark_exam/l
psa.data MappedRDD[3] at textFile at <console>:16
```

```
scala> val parsedData = data.map { line =>
     | val parts = line.split(',')
     | LabeledPoint(parts(0).toDouble, Vectors.dense(parts(1).split(' ').map(x =
>
     | x.toDouble).toArray))
     | }
parsedData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPo
int] = MappedRDD[4] at map at <console>:18
```

```
// Building the model
scala> val numIterations = 20
scala> val model = LinearRegressionWithSGD.train(parsedData, numIterations)*
```

* class LinearRegressionWithSGD
  : extends GeneralizedLinearAlgorithm[LinearRegressionModel] with Serializable.

```
model: org.apache.spark.mllib.regression.LinearRegressionModel = (weights=[-68.6
7074199762735,-30.426817230514445,-36.39140640283543,-5.8700508585256514,-59.414
866895701664,-68.84295783757955,-56.98851847143271,-67.42652465402384], intercep
t=0.0)

scala> 2015-02-08 01:08:25,108 INFO  [task-result-getter-1] scheduler.TaskSetMan
ager (Logging.scala:logInfo(59)) - Finished task 0.0 in stage 21.0 (TID 21) in 1
56 ms on localhost (1/1)
2015-02-08 01:08:25,109 INFO  [task-result-getter-1] scheduler.TaskSchedulerImpl
 (Logging.scala:logInfo(59)) - Removed TaskSet 21.0, whose tasks have all comple
ted, from pool
```

```
// Evaluate model on training examples and compute training error
scala> val valuesAndPreds = parsedData.map { point =>
scala> val prediction = model.predict(point.features)
(point.label, prediction)
}
```

```
scala> val valuesAndPreds = parsedData.map { point =>
     | val prediction = model.predict(point.features)
     | (point.label, prediction)
     | }
valuesAndPreds: org.apache.spark.rdd.RDD[(Double, Double)] = MappedRDD[46] at ma
p at <console>:24
```

```
scala> val MSE = valuesAndPreds.map{ case(v, p) => math.pow((v - p), 2)}.reduce(_ + _)/
valuesAndPreds.count

scala> println("training Mean Squared Error = " + MSE)*
```

* Mean Squared Error
In statistics, the mean squared error (MSE) of an estimator measures the average of the squares of the "errors", that is, the difference between the estimator and what is estimated.

Similarly you can use RidgeRegressionWithSGD and LassoWithSGD and compare training Mean Squared Errors.

```
scala> println("training Mean Squared Error = " + MSE)
training Mean Squared Error = 81871.4647274916
```

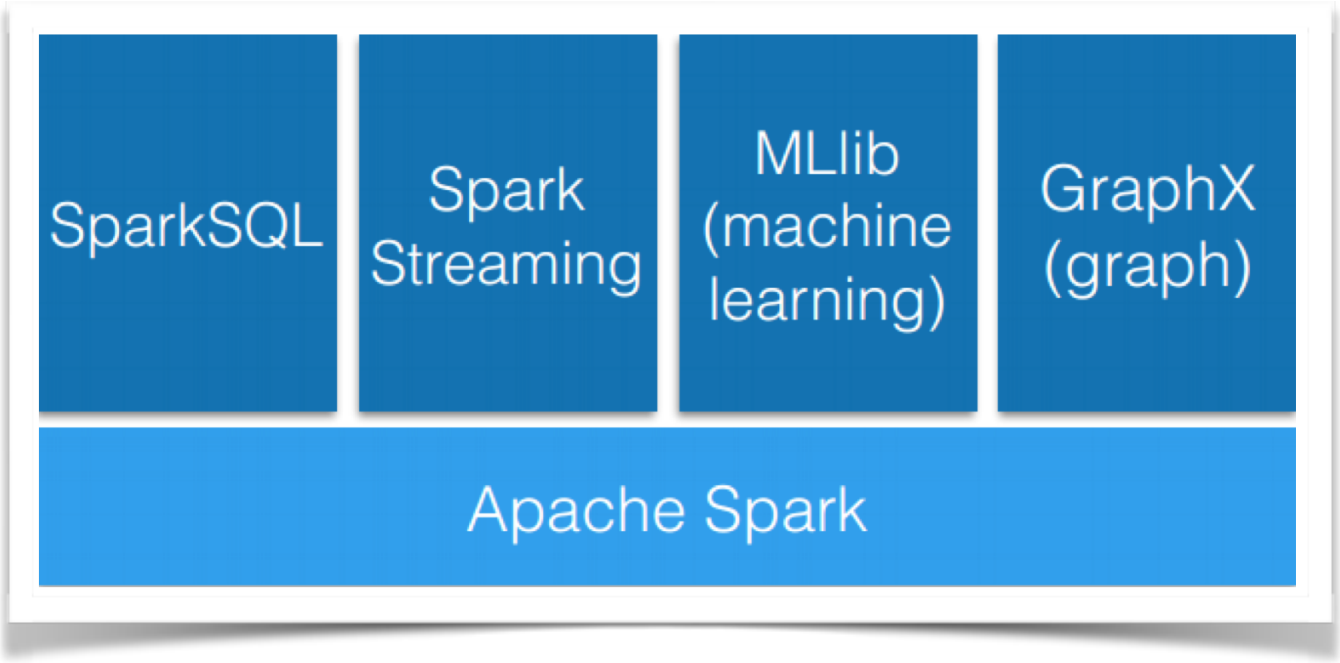
Result

- Dataset

```
-0.4307829,-1.63735562648104 -2.00621178480549 -1.86242597251066 -1.024705801670
82 -0.522940888712441 -0.863171185425945 -1.04215728919298 -0.864466507337306
-0.1625189,-1.98898046126935 -0.722008756122123 -0.787896192088153 -1.0247058016
7082 -0.522940888712441 -0.863171185425945 -1.04215728919298 -0.864466507337306
-0.1625189,-1.57881887548545 -2.1887840293994 1.36116336875686 -1.02470580167082
 -0.522940888712441 -0.863171185425945 0.342627053981254 -0.155348103855541
-0.1625189,-2.16691708463163 -0.807993896938655 -0.787896192088153 -1.0247058016
7082 -0.522940888712441 -0.863171185425945 -1.04215728919298 -0.864466507337306
0.3715636,-0.507874475300631 -0.458834049396776 -0.250631301876899 -1.0247058016
7082 -0.522940888712441 -0.863171185425945 -1.04215728919298 -0.864466507337306
0.7654678,-2.03612849966376 -0.933954647105133 -1.86242597251066 -1.024705801670
82 -0.522940888712441 -0.863171185425945 -1.04215728919298 -0.864466507337306
0.8544153,-0.557312518810673 -0.208756571683607 -0.787896192088153 0.99014685253
7193 -0.522940888712441 -0.863171185425945 -1.04215728919298 -0.864466507337306
1.2669476,-0.929360463147704 -0.0578991819441687 0.152317365781542 -1.0247058016
7082 -0.522940888712441 -0.863171185425945 -1.04215728919298 -0.864466507337306
1.2669476,-2.28833047634983 -0.0706369432557794 -0.116315079324086 0.80409888772
376 -0.522940888712441 -0.863171185425945 -1.04215728919298 -0.864466507337306
1.2669476,0.223498042876113 -1.41471935455355 -0.116315079324086 -1.024705801670
82 -0.522940888712441 -0.29928234305568 0.342627053981254 0.199211097885341
1.3480731,0.107785900236813 -1.47221551299731 0.420949810887169 -1.0247058016708
2 -0.522940888712441 -0.863171185425945 0.342627053981254 -0.687186906466865
```

- Think about it : What is the MLlib in Spark?

MLlib is Spark's Machine Learning Library.
That is, MLlib is a Spark implementation of some common machine learning (ML) functionality, as well associated tests and data generators. MLlib currently supports four common types of machine learning problem settings, namely, binary classification, regression, clustering and collaborative filtering, as well as an underlying gradient descent optimization primitive.
MLlib is also part of Apache Spark Echosystem.



Active Development

Initial Release
• Developed by AMPLab (11 contributors)
• Shipped with Spark v0.8 (Sep 2013)

Current Version
• 48 contributors from various organizations
• Shipped with Spark v1.0 (May 2014)

Algorithms in v1.0
• classification: logistic regression, linear support vector machines (SVM), naive Bayes, decision trees
• regression: linear regression, regression trees
• collaborative filtering: alternating least squares (ALS)
• clustering: k-means
• optimization: stochastic gradient descent (SGD), limitedmemory BFGS (L-BFGS)
• dimensionality reduction: singular value decomposition (SVD), principal component analysis (PCA)

<< Example 5. Running SQL on RDDs >>

One type of table that is supported by Spark SQL is an RDD of Scala case classes. The case class defines the schema of the table. The names of the arguments to the case class are read using reflection and become the names of the columns. Case classes can also be nested or contain complex types such as Sequences or Arrays. This RDD can be implicitly converted to a SchemaRDD and then be registered as a table. Tables can be used in subsequent SQL statements.

- Open Spark shell

```
[cloudera@quickstart ~] spark-shell --jars /usr/lib/avro/avro-mapred.jar \
--conf spark.serializer=org.apache.spark.serializer.KryoSerializer
```

- Input the code

```
// Importing the SQL context gives access to all the public SQL functions and implicit conversions.
scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
scala> import sqlContext._
```

```
scala> val sqlContext = new org.apache.spark.sql.SQLContext(sc)
sqlContext: org.apache.spark.sql.SQLContext = org.apache.spark.sql.SQLContext@36
bd59d4

scala> import sqlContext._
import sqlContext._

scala>
```

```
// Define the schema using a case class.
// Note: Case classes in Scala 2.10 can support only up to 22 fields. To work around this limit,
// you can use custom classes that implement the Product interface.
scala> case class Person(name: String, age: Int)
```

```
scala> case class Person(name: String, age: Int)
defined class Person

scala>
```

```
// Create an RDD of Person objects and register it as a table.
scala> val people = sc.textFile("examples/src/main/resources/people.txt").map(_.split(",")).map(p
=> Person(p(0), p(1).trim.toInt))
scala > people.registerAsTable("people")
```

```
scala> val people = sc.textFile("hdfs://quickstart.cloudera/spark_exam/people.tx
t").map( .split(",")).map(p => Person(p(0), p(1).trim.toInt))
2015-02-07 06:44:29,790 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(
59)) - ensureFreeSpace(181427) called with curMem=0, maxMem=280248975
2015-02-07 06:44:29,793 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(
59)) - Block broadcast_0 stored as values in memory (estimated size 177.2 KB, fr
ee 267.1 MB)
2015-02-07 06:44:30,490 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(
59)) - ensureFreeSpace(14118) called with curMem=181427, maxMem=280248975
2015-02-07 06:44:30,491 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(
59)) - Block broadcast_0_piece0 stored as bytes in memory (estimated size 13.8 K
B, free 267.1 MB)
2015-02-07 06:44:30,494 INFO  [sparkDriver-akka.actor.default-dispatcher-4] stor
age.BlockManagerInfo (Logging.scala:logInfo(59)) - Added broadcast_0_piece0 in m
emory on localhost:55344 (size: 13.8 KB, free: 267.3 MB)
2015-02-07 06:44:30,495 INFO  [main] storage.BlockManagerMaster (Logging.scala:l
ogInfo(59)) - Updated info of block broadcast_0_piece0
2015-02-07 06:44:30,507 INFO  [main] spark.SparkContext (Logging.scala:logInfo(5
9)) - Created broadcast 0 from textFile at <console>:19
people: org.apache.spark.rdd.RDD[Person] = MappedRDD[3] at map at <console>:19

scala>
```

```
scala> people.registerAsTable("people")
warning: there were 1 deprecation warning(s); re-run with -deprecation for detai
ls

scala>
```

```
// SQL statements can be run by using the sql methods provided by sqlContext.
scala> val teenagers = sql("SELECT name FROM people WHERE age >= 13 AND age <= 19")
```

```
scala> val teenagers = sql("SELECT name FROM people WHERE age >= 13 AND age <= 1
9")
teenagers: org.apache.spark.sql.SchemaRDD =
SchemaRDD[6] at RDD at SchemaRDD.scala:108
== Query Plan ==
== Physical Plan ==
Project [name#0]
 Filter ((age#1 >= 13) && (age#1 <= 19))
  PhysicalRDD [name#0,age#1], MapPartitionsRDD[4] at mapPartitions at ExistingRD
D.scala:36

scala>
```

```
// The results of SQL queries are SchemaRDDs and support all the normal RDD operations.
// The columns of a row in the result can be accessed by ordinal.
scala> teenagers.map(t => "Name: " + t(0)).collect().foreach(println)
```

```
scala> teenagers.map(t => "Name: " +t(0)).collect().foreach(println)
2015-02-07 06:47:29,183 INFO  [main] mapred.FileInputFormat (FileInputFormat.jav
a:listStatus(247)) - Total input paths to process : 1
2015-02-07 06:47:29,378 INFO  [main] spark.SparkContext (Logging.scala:logInfo(5
9)) - Starting job: collect at <console>:20
2015-02-07 06:47:29,407 INFO  [sparkDriver-akka.actor.default-dispatcher-2] sche
duler.DAGScheduler (Logging.scala:logInfo(59)) - Got job 0 (collect at <console>
:20) with 1 output partitions (allowLocal=false)
2015-02-07 06:47:29,415 INFO  [sparkDriver-akka.actor.default-dispatcher-2] sche
duler.DAGScheduler (Logging.scala:logInfo(59)) - Final stage: Stage 0(collect at
 <console>:20)
2015-02-07 06:47:29,416 INFO  [sparkDriver-akka.actor.default-dispatcher-2] sche
duler.DAGScheduler (Logging.scala:logInfo(59)) - Parents of final stage: List()
2015-02-07 06:47:29,425 INFO  [sparkDriver-akka.actor.default-dispatcher-2] sche
duler.DAGScheduler (Logging.scala:logInfo(59)) - Missing parents: List()
2015-02-07 06:47:29,437 INFO  [sparkDriver-akka.actor.default-dispatcher-2] sche
duler.DAGScheduler (Logging.scala:logInfo(59)) - Submitting Stage 0 (MappedRDD[7
] at map at <console>:20), which has no missing parents
2015-02-07 06:47:29,450 INFO  [sparkDriver-akka.actor.default-dispatcher-2] stor
age.MemoryStore (Logging.scala:logInfo(59)) - ensureFreeSpace(6456) called with
curMem=195545, maxMem=280248975
2015-02-07 06:47:29,451 INFO  [sparkDriver-akka.actor.default-dispatcher-2] stor
age.MemoryStore (Logging.scala:logInfo(59)) - Block broadcast_1 stored as values
 in memory (estimated size 6.3 KB, free 267.1 MB)
```

```
2015-02-07 06:47:30,479 INFO  [Executor task launch worker-0] executor.Executor
(Logging.scala:logInfo(59)) - Finished task 0.0 in stage 0.0 (TID 0). 1750 bytes
 result sent to driver
2015-02-07 06:47:30,512 INFO  [sparkDriver-akka.actor.default-dispatcher-3] sche
duler.DAGScheduler (Logging.scala:logInfo(59)) - Stage 0 (collect at <console>:2
0) finished in 0.982 s
2015-02-07 06:47:30,513 INFO  [task-result-getter-0] scheduler.TaskSetManager (L
ogging.scala:logInfo(59)) - Finished task 0.0 in stage 0.0 (TID 0) in 950 ms on
localhost (1/1)
2015-02-07 06:47:30,526 INFO  [task-result-getter-0] scheduler.TaskSchedulerImpl
 (Logging.scala:logInfo(59)) - Removed TaskSet 0.0, whose tasks have all complet
ed, from pool
2015-02-07 06:47:30,541 INFO  [main] scheduler.DAGScheduler (Logging.scala:logIn
fo(59)) - Job 0 finished: collect at <console>:20, took 1.162845 s
Name: Apache
Name: Sally
Name: Sam
Name: Seeun
Name: Dev

scala>
```

- Dataset

```
Apache,19
Spark,25
Jiho,20
Jungeun,32
Hansol,90
Hyeyeon,24
Mike,39
Jim,64
Steve,42
Lion,22
Sally,15
Sam,17
Seeun,13
coffee,24
Moca,33
Milk,12
Ham,46
Eunkyong,31
Dev,18
Ikhwan,68
```

- Think about it : Meaning of Spark.

Building a unified platform for big data analytics has long been the vision of Apache Spark, allowing a single program to perform ETL, MapReduce, and complex analytics. An important aspect of unification that our users have consistently requested is the ability to more easily import data stored in external sources, such as Apache Hive. Today, we are excited to announce Spark SQL, a new component recently merged into the Spark repository.
Spark SQL brings native support for SQL to Spark and streamlines the process of querying data stored both in RDDs (Spark's distributed datasets) and in external sources. Spark SQL conveniently blurs the lines between RDDs and relational tables. Unifying these powerful abstractions makes it easy for developers to intermix SQL commands querying external data with complex analytics, all within in a single application

Since the results of Spark SQL are also stored in RDDs, interfacing with other Spark libraries is trivial. Furthermore, Spark SQL allows developers to close the loop, by making it easy to manipulate and join the output of these algorithms, producing the desired final result.
To summarize, the unified Spark platform gives developers the power to choose the right tool for the right job, without having to juggle multiple systems.

References

[1] **training.cloudera.com/essentials.pdf** “Cloudera Essentials for Apache Hadoop”
[2] **http://blog.cloudera.com/blog/2014/09/how-to-translate-from-mapreduce-to-apache-spark/**