


# 20152850 이재호 Spark실습

## 1. EMR 마스터 노트 접속 후 Scala 기반 스파크 인터프리터 실행

시작 상태

### Your AWS Account Status

 **Active**  
full access ( hiljh96@kookmin.ac.kr )

 **\$49.55**  
remaining credits (estimated)

 **2:60**  
session time

[Account Details](#)

[AWS Console](#)

Please use AWS Educate Account responsibly. Remember to shut down your instances when not in use.

```
[hadoop@ip-172-31-82-98 ~]$ spark
-bash: spark: command not found
[hadoop@ip-172-31-82-98 ~]$ spark-shell
20/11/16 01:44:26 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java
asses where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Spark context Web UI available at http://ip-172-31-82-98.ec2.internal:4040
Spark context available as 'sc' (master = local[*], app id = local-1605491093078).
Spark session available as 'spark'.
Welcome to

  ____      _
 / ___|  _ \| | | |
 \___ \  | | | | | |
  ___) | |_| | | | |
 |____|_||_| |_| |_|

version 2.4.6-amzn-0

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_265)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

아래 실습들의 `sc.textFile` 부분은 `sc.textFile("/dataset/movielens/movies.csv")` 로 대체 됩니다.

## 2. movies.csv 영화 중 Action이 포함된 영화의 개수

```
val movies =
sc.textFile("/Filestore/shared_uploads/hiljh96@kookmin.ac.kr/movies.csv")
val movieInfo = movies.map(movie => movie.split(','))
val genres = movieInfo.map(mi => mi(2).split('|'))
genres.take(6)
val actions = genres.filter(mg => mg.contains("Action"))
val counts = actions.count()
```

```
scala> val movies = sc.textFile("/dataset/movielens/movies.csv")
movies: org.apache.spark.rdd.RDD[String] = /dataset/movielens/movies.csv MapPartitionsRDD[3] at textFile at <console>:24

scala> movies.take(2)
res1: Array[String] = Array(movieId,title,genres, 1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy)

scala> val movieInfo = movies.map(movie => movie.split(','))
movieInfo: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[4] at map at <console>:25

scala> val genres = movieInfo.map(mi => mi(2).split('|'))
genres: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[5] at map at <console>:25

scala> genres.take(6)
res2: Array[Array[String]] = Array(Array(genres), Array(Adventure, Animation, Children, Comedy, Fantasy), Array(Adventure, Children, Fantasy), Array(Comedy, Romance), Array(Comedy, Drama, Romance), Array(Comedy))

scala> val actions = genres.filter(mg => mg.contains("Action"))
actions: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[6] at filter at <console>:25

scala> val counts = actions.count()
counts: Long = 3520

scala> |
```

### 3. movies.csv에 있는 영화 이름 중 unique한 값 만 추출 RDD에서 임의의 제목 10개를 선정하여 출력

```
val movies =
sc.textFile("/Filestore/shared_uploads/hiljh96@kookmin.ac.kr/movies.csv")
val movieInfo = movies.map(movie => movie.split(','))
val mi = movieInfo.take(3)
val movie_names = movieInfo.map(mi => mi(1))

val unique_names = movie_names.distinct()
unique_names.takeSample(false, 10)
```

```
scala> val movie_names = movieInfo.map(mi=>mi(1))
movie_names: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[7] at map at <console>:25

scala> val unique_names = movie_names.distinct()
unique_names: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[10] at distinct at <console>:25

scala> unique_names.takeSample(false, 10)
res3: Array[String] = Array(Dear God (1996), Wizard of Oz The (1925), Victory Through Air Power (1943), Three Crowns of the Sailor (Les trois couronnes du matelot) (1983), Melissa P. (2005), Young Man with a Horn (1950), Return of Dracula The (1958), Man Bites Dog (C'est arrivé près de chez vous) (1992), Cutthroat Island (1995), Gayby (2012))
```

#### filter(function)

- 주어진 함수의 결과가 true 를 리턴하는 원소만 반환

```
scala> val movieGenres = movieInfo.map(mi => (mi(1), mi(2).split('|')))
scala> val actionMovies = movieGenres.filter(m => m._2.contains("Action"))
// m._1 : Tuple 의 원소 접근. 인덱스 1부터 시작: ._1, ._2, ...
scala> actionMovies.take(1)
```

```
1 val movieGenres = movieInfo.map(mi => (mi(1), mi(2).split('|')))
```

## tags.csv 파일에서 가장 많이 태그된 값과 해당 태그가 사용된 횟수를 함께 출력

```
val tags =
sc.textFile("/Filestore/shared_uploads/hiljh96@kookmin.ac.kr/tags.csv")
val tag = tags.take(4)
val tagInfo = tags.map(t => t.split(','))
val ti = tagInfo.take(3)
val t = tagInfo.map(t => t(2))
val tOne = t.map(t => (t,1))
val tCounter = tOne.reduceByKey((x,y) => (x+y))
tCounter.takeOrdered(1)(Ordering[Long].reverse.on(x=>x._2))
```

```
scala> val tags = sc.textFile("/dataset/movielens/tags.csv")
tags: org.apache.spark.rdd.RDD[String] = /dataset/movielens/tags.csv MapPartitionsRDD[13] at textFile at <console>:24

scala> tags.take(3)
res5: Array[String] = Array(userId,movieId,tag,timestamp, 18,4141,Mark Waters,1240597180, 65,208,dark hero,1368150078)

scala> val tagInfo = tags.map(t => t.split(','))
tagInfo: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[14] at map at <console>:25

scala> val ti = tagInfo.take(3)
ti: Array[Array[String]] = Array(Array(userId, movieId, tag, timestamp), Array(18, 4141, Mark Waters, 1240597180), Array(65, 208, dark hero, 1368150078))

scala> val t = tagInfo.map(t => t(2))
t: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[15] at map at <console>:25

scala> val tOne = t.map(t=>(t,1))
tOne: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[16] at map at <console>:25

scala> val tCounter = tOne.reduceByKey((x,y) => (x+y))
tCounter: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[17] at reduceByKey at <console>:25

scala> tCounter.take(2)
res6: Array[(String, Int)] = Array((Josh Brolin,37), (break into lab,1))

scala> tCounter.takeOrdered(1)(Ordering[Int].reverse.on(x=>x._2))
<console>:26: error: not found: value Ordering
      tCounter.takeOrdered(1)(Ordering[Int].reverse.on(x=>x._2))
                              ^
scala> tCounter.takeOrdered(1)(Ordering[Int].reverse.on(x=>x._2))
res8: Array[(String, Int)] = Array((sci-fi,3384))

scala> |
```

## 5. ratings.csv 에 표현된 전체 영화의 평점을 계산하는 프로그램을 작성하세요.

```
val ratings =
sc.textFile("/Filestore/shared_uploads/hiljh96@kookmin.ac.kr/ratings.csv")
val header = ratings.first()
val nhratings = ratings.filter(row => row != header)
nhratings.take(3)
val ratingInfo = nhratings.map(r => r.split(','))
ratingInfo.take(3)
val number = ratingInfo.map(n => n(2).toDouble)
number.take(3)
val avgN = number.reduce((x,y) => (x+y)*0.5)
```

```

Welcome to

  ____  _
 / ___|| | | |
| |___| |_| |
 \___ \|  _/
      |_|_|_|

version 2.4.6-amzn-0

Using Scala version 2.11.12 (OpenJDK 64-Bit Server VM, Java 1.8.0_265)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val ratings = sc.textFile("/dataset/movielens/ratting.csv")
ratings: org.apache.spark.rdd.RDD[String] = /dataset/movielens/ratting.csv MapPartitionsRDD[1] at textFile at <
console>:24

scala> val ratings = sc.textFile("/dataset/movielens/ratings.csv")
ratings: org.apache.spark.rdd.RDD[String] = /dataset/movielens/ratings.csv MapPartitionsRDD[3] at textFile at <
console>:24

scala> val header = ratings.first()
header: String = userId,movieId,rating,timestamp

scala> val nhratings = ratings.filter(row => row != header)
nhratings: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[4] at filter at <console>:27

scala> val ratingInfo = nhratings.map(r => r.split(','))
ratingInfo: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[5] at map at <console>:25

scala> val number = ratingInfo.map(n => n(2).toDouble)
number: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[6] at map at <console>:25

scala> val avgN = number.reduce((x,y) => (x+y)*0.5)
avgN: Double = 3.8993230761532542

```

## 6. ratings.csv 에 표현된 영화별 평점을 계산하여 movieid 와 평균 평점을 계산 평점 상위 100개 영화 movie\_id, 평균 평점

```

val ratingss =
sc.textFile("/Filestore/shared_uploads/hiljh96@kookmin.ac.kr/ratings.csv")
val header = ratingss.first()
val ratings = ratingss.filter(row => row != header)
ratings.take(3)
val ratingInfo = ratings.map(r => r.split(','))
ratingInfo.take(3)
val rating = ratingInfo.map(ri => (ri(1).toString,ri(2).toDouble))
rating.take(3)
// r.take(4)
val id = ratingInfo.map(ri => ri(1).toInt)
val score = ratingInfo.map(ri => ri(2).toDouble)
val merged = id.zip(score)
def parseLine(ratings: String)={
    val fields = ratings.split(",")
    val id = fields(1).toInt
    val score = fields(2).toDouble
    (id, score) // will be appended as element
}
val rdd = ratings.map(parseLine)
rdd.take(10).foreach(println)
val value_expand = rdd.mapValues(x=> (x,1))
value_expand.take(10).foreach(println)
val key_cum = value_expand.reduceByKey((x,y)=> (x._1 + y._1, x._2 + y._2))
key_cum.take(10).foreach(println)
val averageByScore = key_cum.mapValues(x => x._1/x._2)
averageByScore.take(10).foreach(println)
val results = averageByScore.takeOrdered(100)
(Ordering[Double].reverse.on(x=>x._2))

```

```
scala> val ratingss = sc.textFile("/dataset/movielens/ratings.csv")
ratingss: org.apache.spark.rdd.RDD[String] = /dataset/movielens/ratings.csv MapPartitionsRDD[1] at textFile at <console>:24

scala> val header = ratingss.first()
header: String = userId,movieId,rating,timestamp

scala> val ratings = ratingss.filter(row => row!=header)
ratings: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[2] at filter at <console>:27

scala> val ratingInfo = ratings.map(r => r.split(','))
ratingInfo: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[3] at map at <console>:25

scala> val id = ratingInfo.map(ri => ri(1).toInt)
id: org.apache.spark.rdd.RDD[Int] = MapPartitionsRDD[4] at map at <console>:25

scala> val score = ratingInfo.map(ri => ri(2).toDouble)
score: org.apache.spark.rdd.RDD[Double] = MapPartitionsRDD[5] at map at <console>:25

scala> val merged = id.zip(score)
merged: org.apache.spark.rdd.RDD[(Int, Double)] = ZippedPartitionsRDD2[6] at zip at <console>:27

scala> val value_expanded = merged.mapValues(x => (x,1))
value_expanded: org.apache.spark.rdd.RDD[(Int, (Double, Int))] = MapPartitionsRDD[7] at mapValues at <console>:25

scala> value_expanded.take(10).foreach(println_
| )
<console>:24: error: not found: value value_expanded
value_expanded.take(10).foreach(println_
^
<console>:24: error: not found: value println_
value_expanded.take(10).foreach(println_
^

scala> value_expanded.take(10).foreach(println)
<console>:24: error: not found: value value_expanded
value_expanded.take(10).foreach(println)
^
```

```
scala> value_expanded.take(10).foreach(println)
(2,(3.5,1))
(29,(3.5,1))
(32,(3.5,1))
(47,(3.5,1))
(50,(3.5,1))
(112,(3.5,1))
(151,(4.0,1))
(223,(4.0,1))
(253,(4.0,1))
(260,(4.0,1))

scala> val key_cum = value_expanded.reduceByKey((x,y) => (x._1 + y._1, x._2 + y._2))
<console>:25: error: value reduceByKey is not a member of org.apache.spark.rdd.RDD[(Int, (Double, Int))]
val key_cum = value_expanded.reduceByKey((x,y) => (x._1 + y._1, x._2 + y._2))
^

scala> val key_cum = value_expanded.reduceByKey((x,y) => (x._1 + y._1, x._2 + y._2))
key_cum: org.apache.spark.rdd.RDD[(Int, (Double, Int))] = ShuffledRDD[8] at reduceByKey at <console>:25

scala> val averageByScore = key_cum.mapValues(x => x._1/x._2)
averageByScore: org.apache.spark.rdd.RDD[(Int, Double)] = MapPartitionsRDD[9] at mapValues at <console>:25

scala> val result = averageByScore.takeOrdered(100)(Ordering[Double].reverse.on(x=>x._2))
result: Array[(Int, Double)] = Array((101292,5.0), (130996,5.0), (113860,5.0), (105846,5.0), (32230,5.0), (106076,5.0), (119430,5.0), (79866,5.0), (54326,5.0), (115994,5.0), (129034,5.0), (129516,5.0), (129526,5.0), (112790,5.0), (27914,5.0), (105529,5.0), (111546,5.0), (116106,5.0), (117418,5.0), (105526,5.0), (128506,5.0), (26718,5.0), (104317,5.0), (128738,5.0), (113790,5.0), (117314,5.0), (102882,5.0), (86055,5.0), (129478,5.0), (112990,5.0), (99450,5.0), (113947,5.0), (95517,5.0), (111797,5.0), (81117,5.0), (95977,5.0), (109253,5.0), (129305,5.0), (107434,5.0), (98761,5.0), (117606,5.0), (128093,5.0), (120134,5.0), (121029,5.0), (106113,5.0), (103753,5.0), (126945,5.0), (100830,5.0), (114214,5.0), (129530,5.0), (94431,5.0), (125599,5.0), (109571,5.0), (95979,5.0), (114254,5.0), (93...)
```

## 7. ratings.csv 파일로 부터 영화별 평점을 계산 한 후 평점이 기록된 횟수가 10 이상인 상위 10개 영화를 아이디, 영화 제목

```
val ratingss =
sc.textFile("/FileStore/shared_uploads/hiljh96@kookmin.ac.kr/ratings.csv")
```

```

val header = ratings.first()
val ratings = ratings.filter(row => row != header)
ratings.take(3)
val ratingInfo = ratings.map(r => r.split(','))
ratingInfo.take(3)
val rating = ratingInfo.map(ri => (ri(1).toString, ri(2).toDouble))
rating.take(3)
// r.take(4)
val id = ratingInfo.map(ri => ri(1).toInt)
val score = ratingInfo.map(ri => ri(2).toDouble)
val merged = id.zip(score)
def parseLine(ratings: String)={
    val fields = ratings.split(",")
    val id = fields(1).toInt
    val score = fields(2).toDouble
    (id, score) // will be appended as element
}
val rdd = ratings.map(parseLine)
rdd.take(10).foreach(println)
val value_expand = rdd.mapValues(x=> (x,1))
value_expand.take(10).foreach(println)
val key_cum = value_expand.reduceByKey((x,y)=> (x._1 + y._1, x._2 + y._2))
key_cum.take(10).foreach(println)
val averageByScore = key_cum.mapValues(x => x._1/x._2)
averageByScore.take(10).foreach(println)
val results = averageByScore.takeOrdered(100)
(Ordering[Double].reverse.on(x=>x._2))
val selected = key_cum.filter(x => x._2._2 > 100)
val averageAbove = selected.mapValues(x => x._1/x._2)
val movies =
sc.textFile("/FileStore/shared_uploads/hiljh96@kookmin.ac.kr/movies.csv")
val header = movies.first()
val nhmovie = movies.filter(row => row != header)
nhmovie.take(3)
val movieInfo = nhmovie.map(movie => movie.split(','))
val movie = movieInfo.map(mi => (mi(0).toInt, mi(1).toString))
val dist = movie.distinct()
val joined = averageAbove.join(dist)
joined.takeOrdered(10)(Ordering[Double].reverse.on(x=>x._2._1))

```

```
scala> val movies = sc.textFile("/dataset/movielens/movies.csv")
movies: org.apache.spark.rdd.RDD[String] = /dataset/movielens/movies.csv MapPartitionsRDD[12] at textFile at <console>:24

scala> val header = movies.first()
header: String = movieId,title,genres

scala> val nhmovie = movies.filter(row => row != header)
nhmovie: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[13] at filter at <console>:27

scala> val movieInfo = nhmovie.map(movie => movie.split(','))
movieInfo: org.apache.spark.rdd.RDD[Array[String]] = MapPartitionsRDD[14] at map at <console>:25

scala> val movie = movieInfo.map(mi => (mi(0).toInt, mi(1).toString))
movie: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[15] at map at <console>:25

scala> val dist = movie.distinct()
dist: org.apache.spark.rdd.RDD[(Int, String)] = MapPartitionsRDD[18] at distinct at <console>:25

scala> val joined = averageAbove.join(dist)
<console>:25: error: not found: value averageAbove
    val joined = averageAbove.join(dist)
                  ^




scala> val selected = key_cum.filter(x => x._2._2 > 100)
selected: org.apache.spark.rdd.RDD[(Int, (Double, Int))] = MapPartitionsRDD[19] at filter at <console>:25

scala> val averageAbove = selected.mapValues(x => x._1/x._2)
averageAbove: org.apache.spark.rdd.RDD[(Int, Double)] = MapPartitionsRDD[20] at mapValues at <console>:25

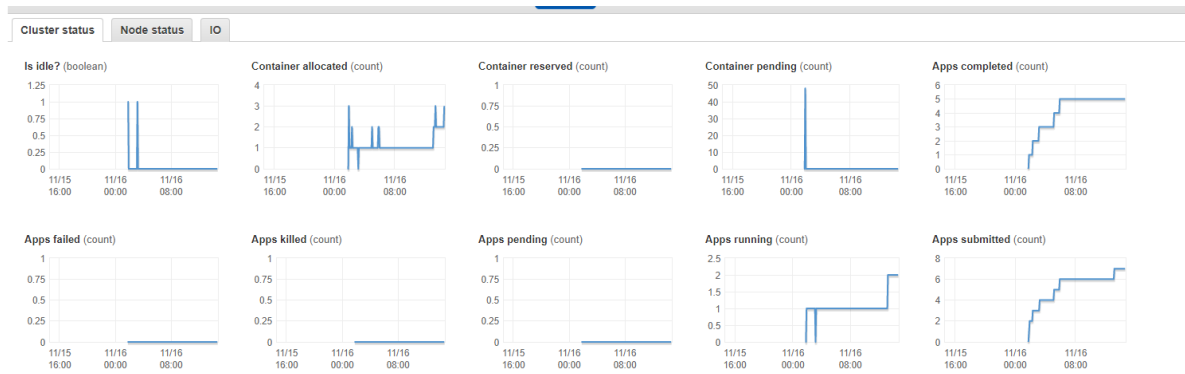
scala> val joined = averageAbove.join(dist)
joined: org.apache.spark.rdd.RDD[(Int, (Double, String))] = MapPartitionsRDD[23] at join at <console>:27

scala> joined.takeOrdered(10)(Ordering[Double].reverse.on(x=>x._2._1))
res5: Array[(Int, (Double, String))] = Array((318,(4.446990499637029,Shawshank Redemption The (1994))), (858,(4.364732196832306,Godfather The (1972))), (50,(4.334372207803259,Usual Suspects The (1995))), (527,(4.310175010988133,Schindler's List (1993))), (1221,(4.275640557704942,Godfather: Part II The (1974))), (2019,(4.2741796572216,Seven Samurai (Shichinin no samurai) (1954))), (904,(4.271333600779414,Rear Window (1954))), (7502,(4.263182346109176,Band of Brothers (2001))), (912,(4.258326830670664,Casablanca (1942))), (922,(4.256934865900383,Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)))
```

## 8. aws 크레딧 분석

	<b>Active</b> full access ( hiljh96@kookmin.ac.kr )
	<b>\$49.55</b> remaining credits (estimated)
	<b>2:60</b> session time

갱신이 돼지 않아서 cluster에서 monitoring으로 대체합니다



container pending는 아마 movie.csv와 tag.csv를 올릴 때 증가했다. 그 이후 ratings.csv를 올릴때 크게 증가 했다.

appsRunning은 spark 환경에서 명령어를 계속 추가해서 apps running이 증가 했다.

Apps submitted 또한 계속해서 증가하는 것을 확인 가능하다.

Apps completed를 보면 spark 명령을 실행하면서 계속 증가한 것이 확인가능하다.

아마 가장 많은 가격이 나간 것은 ratings.csv를 작업하면서 resource를 많이 사용해서 6,7,8번 과정이 많은 리소스를 사용했을 것 같다.