Vaibhav Shekhar Dave

HW 3 CS6240 Parallel Data Processing – Spring 19.

https://github.ccs.neu.edu/vaibhavdave5/parallelDataProcessing/tree/master/SocialTriagleSpark/

Combining in Spark:

1) RDD_R

```
//Reading csv file
   val textFile = sc.textFile(inputPath)

//Using reduceByKey with a function to perform summation
   val counts = textFile.map(line => line.split(",")(0))
    .map(word => (word, 1))
    .reduceByKey(_ + _)

   counts.saveAsTextFile(outputPath)
```

Corresponding toDebugString()

```
(40) ShuffledRDD[4] at reduceByKey at WordCount.scala:28 []
 +-(40) MapPartitionsRDD[3] at map at WordCount.scala:27 []
    |   MapPartitionsRDD[2] at map at WordCount.scala:26 []
    |   input/edges.csv MapPartitionsRDD[1] at textFile at WordCount.scala:25 []
    |   input/edges.csv HadoopRDD[0] at textFile at WordCount.scala:25 []
```

DOES NOT PERFORM AGGREGATION BEFORE SHUFFLING

2) **RDD_G**

```
// Reading csv file
   val textFile = sc.textFile(inputPath)

//Using groupByKey with a function to perform summation
   val counts = textFile.map(line => line.split(",")(0))
     .map(word => (word, 1))
     .groupByKey()
     .mapValues(id => id.sum)

   counts.saveAsTextFile(outputPath)

 }
```

Corresponding toDebugString()

```
(40) MapPartitionsRDD[5] at mapValues at WordCount.scala:43 []
 |   ShuffledRDD[4] at groupByKey at WordCount.scala:42 []
 +-(40) MapPartitionsRDD[3] at map at WordCount.scala:41 []
    |   MapPartitionsRDD[2] at map at WordCount.scala:40 []
    |   input/edges.csv MapPartitionsRDD[1] at textFile at WordCount.scala:38 []
    |   input/edges.csv HadoopRDD[0] at textFile at WordCount.scala:38 []
```

DOES NOT PERFORM AGGREGATION BEFORE SHUFFLING

3) **RDD_F**

```
// Reading csv file
   val textFile = sc.textFile(inputPath)

//Using foldByKey with a function to perform summation
   val counts = textFile.map(line => line.split(",")(0))
    .map(word => (word, 1))
    .foldByKey(0)(_ + _)
   counts.saveAsTextFile(outputPath)

 }
```

Corresponding toDebugString()

```
(40) ShuffledRDD[4] at foldByKey at WordCount.scala:58 []
 +-(40) MapPartitionsRDD[3] at map at WordCount.scala:57 []
   |   MapPartitionsRDD[2] at map at WordCount.scala:56 []
   |   input/edges.csv MapPartitionsRDD[1] at textFile at WordCount.scala:54 []
   |   input/edges.csv HadoopRDD[0] at textFile at WordCount.scala:54 []
```

DOES PERFORM AGGREGATION BEFORE SHUFFLING

4) **RDD_A**

```
// Reading csv file
   val textFile = sc.textFile(inputPath)

//Using aggregateByKey with a function to perform summation
   val counts = textFile.map(line => line.split(",")(0))
    .map(word => (word, 1))
    .aggregateByKey(0)(_ + _, _ + _)

            counts.saveAsTextFile(outputPath)

   // Printing the RDD lineage graph
   println(counts.toDebugString);

 }
```

Corresponding toDebugString()

```
(40) ShuffledRDD[4] at aggregateByKey at WordCount.scala:73 []
 +-(40) MapPartitionsRDD[3] at map at WordCount.scala:72 []
    |   MapPartitionsRDD[2] at map at WordCount.scala:71 []
    |   input/edges.csv MapPartitionsRDD[1] at textFile at WordCount.scala:69 []
    |   input/edges.csv HadoopRDD[0] at textFile at WordCount.scala:69 []
```

DOES PERFORM AGGREGATION BEFORE SHUFFLING

5) **DSET**

```
  val word = spark.read.csv(inputPath).groupBy("_c1").count()
  println(word.explain(extended = true))
  word.coalesce(1).write.csv(outputPath)

 }
```

Coressponding Explain

```
Start
2019-02-21 21:26:04 INFO  FileSourceStrategy:54 - Pruning directories with:
2019-02-21 21:26:04 INFO  FileSourceStrategy:54 - Post-Scan Filters:
2019-02-21 21:26:04 INFO  FileSourceStrategy:54 - Output Data Schema: struct<_c1: string>
2019-02-21 21:26:04 INFO  FileSourceScanExec:54 - Pushed Filters:
== Parsed Logical Plan ==
Aggregate [_c1#11], [_c1#11, count(1) AS count#17L]
+- AnalysisBarrier
     +- Relation[_c0#10,_c1#11] csv

== Analyzed Logical Plan ==
_c1: string, count: bigint
Aggregate [_c1#11], [_c1#11, count(1) AS count#17L]
+- Relation[_c0#10,_c1#11] csv

== Optimized Logical Plan ==
Aggregate [_c1#11], [_c1#11, count(1) AS count#17L]
+- Project [_c1#11]
  +- Relation[_c0#10,_c1#11] csv

== Physical Plan ==
*(2) HashAggregate(keys=[_c1#11], functions=[count(1)], output=[_c1#11, count#17L])
+- Exchange hashpartitioning(_c1#11, 200)
  +- *(1) HashAggregate(keys=[_c1#11], functions=[partial_count(1)], output=[_c1#11, count#22L])
     +- *(1) FileScan csv [_c1#11] Batched: false, Format: CSV, Location:
InMemoryFileIndex[file:/home/vaibhav/Desktop/lspdpNew/parallelDataProcessing/SocialTriagleSpark/S...,
PartitionFilters: [], PushedFilters: [], ReadSchema: struct<_c1:string>
()
End
```

DOES PERFORM AGGREGATION BEFORE SHUFFLING

Join Implementation

1) R-RS Join

Psuedo code:

```
1)    Set max filter
val maxFilter = 15000
val textFile = sc.textFile("s3://mr-input/edges.csv")

2)  Filter using the maxfilter
val filteredEdges = textFile.map(line => line.split(","))
                            .filter(edge => edge(0).toInt < maxFilter &&  edge(1).toInt < maxFilter)
                    .map(edge => (edge(0), edge(1)))

3) To find pairs a→ b and b→c , we need to find pairs that have the common node 'b'.
// So join the edges dataset on a flipped version of the same dataset, to get Path2.
val edgesOnce = filteredEdges.map(edge => (edge._2, edge._1)) //flip all edges
val edgesTwice = filteredEdges.map(edge => (edge._1, edge._2)) //don't do anything
val edgesThrice = filteredEdges.map(edge => ((edge._1,edge._2) , 1))

4)Calculating path2
val path2 = edgesOnce.join(edgesTwice).map(pair => pair._2)

5)Reverse the endpoints of path2 edges to exactly match with the keys
//of the third edge dataset.
val revPath2 = path2.map(x => ((x._2 , x._1) , 1))

6)Divide by 3 to eliminate redundant counting of same triangles
//with different order of edges.
val matches = revPath2.join(edgesThrice).count()
val triangleCount =  matches/3
println("Number of triangles = "+ triangleCount)
```

Output:
https://github.ccs.neu.edu/vaibhavdave5/parallelDataProcessing/blob/master/SocialTriagleSpark/Spark-Demo/answer.txt

2) **R-Rep Join**

1) Read CSV file
2) Isolate edges and filter using max filter
3) Make RDD from the edges named edges
4) Make a map (userID , List[followers]) from the RDD named broadcastedMap
5) BroadCast the map
6) Path2 = edges.flatMap ((id, follower) =>

```
                broadcastedMap(follower).foreach((follower2) =>
                emit(id, follower2)
        )
```

7) FullTriangle = path2.flatMap ((id, follower) =>

```
                broadcastedMap(follower).foreach((follower2) =>
                if(follower2 == id)
                emit(if, follower2)
        )
```

8) val count = fullTriangle.count()/3 == Answer

OutPut:

https://github.ccs.neu.edu/vaibhavdave5/parallelDataProcessing/blob/master/SocialTriagleSpark/Spark-Demo/answer.txt

3) D-Rep Join

1) Read CSV file
2) Isolate edges and filter using max filter
3) Make DataSet from the edges named edges
4) Make a map (userID , List[followers]) from the RDD named broadcastedMap
5) BroadCast the map
6) Path2 = edges.flatMap ((id, follower) =>

```
        broadcastedMap(follower).foreach((follower2) =>
        emit(id, follower2)
    )
```

7) FullTriangle = path2.flatMap ((id, follower) =>

```
        broadcastedMap(follower).foreach((follower2) =>
        if(follower2 == id)
        emit(if, follower2)
    )
```

8) val count = fullTriangle.count()/3 == Answer

Output:

https://github.ccs.neu.edu/vaibhavdave5/parallelDataProcessing/blob/master/SocialTriagleSpark/Spark-Demo/answer.txt

4) R-Rep Join

`

1) Read the csv

2)Filter the edges

3) Define the following dataFrames

```
val left =  filtered.toDF("a","b")
val right = filtered.toDF("c","d")
val thirdEdge = filtered.toDF("p","q")
```

4) Left join for Path2

```
val path2 = left.join(right, $"b" === $"c").drop("b").drop("c")
```

5) Join for fullTriangle

```
val fullTriangle = path2.join(thirdEdge,$"d" === $"p" &&  $"a" === $"q")
```

6) Remove repeated counts

```
val triangleCount = fullTriangle.count()/3  == answer
```

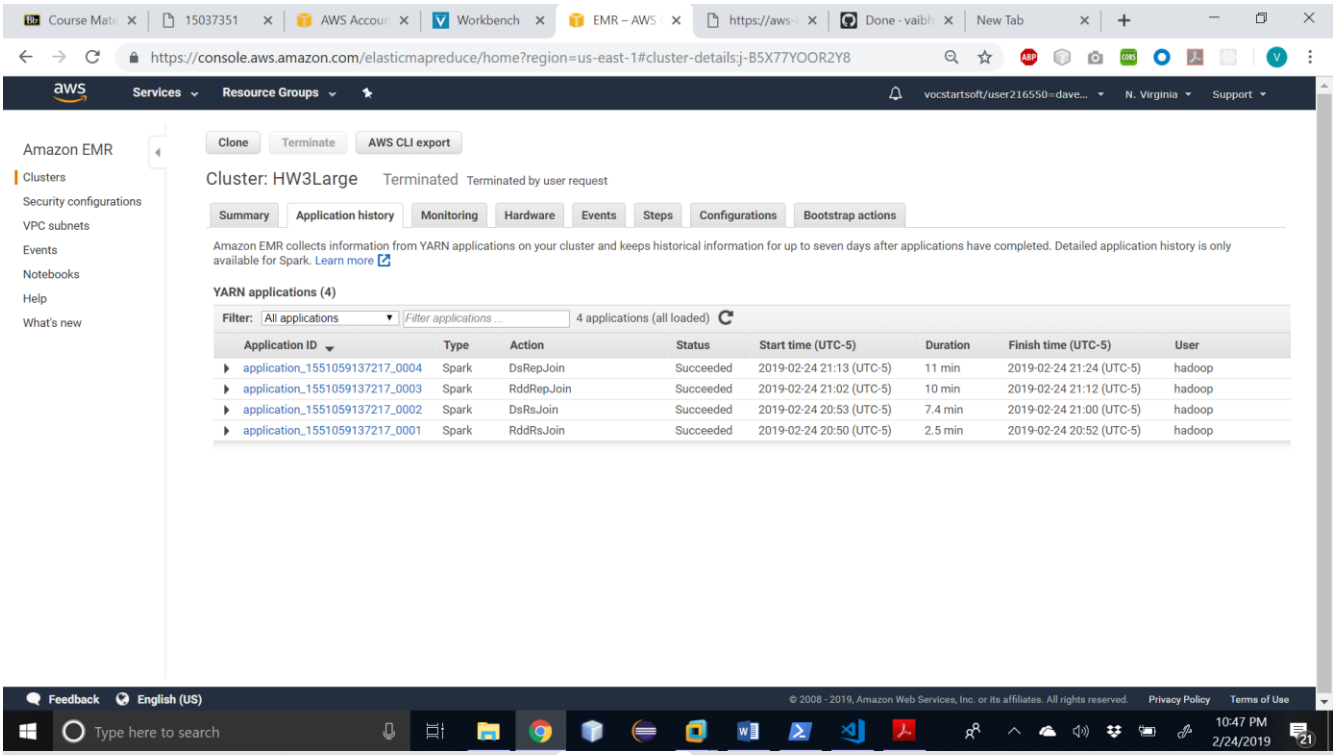
Output:

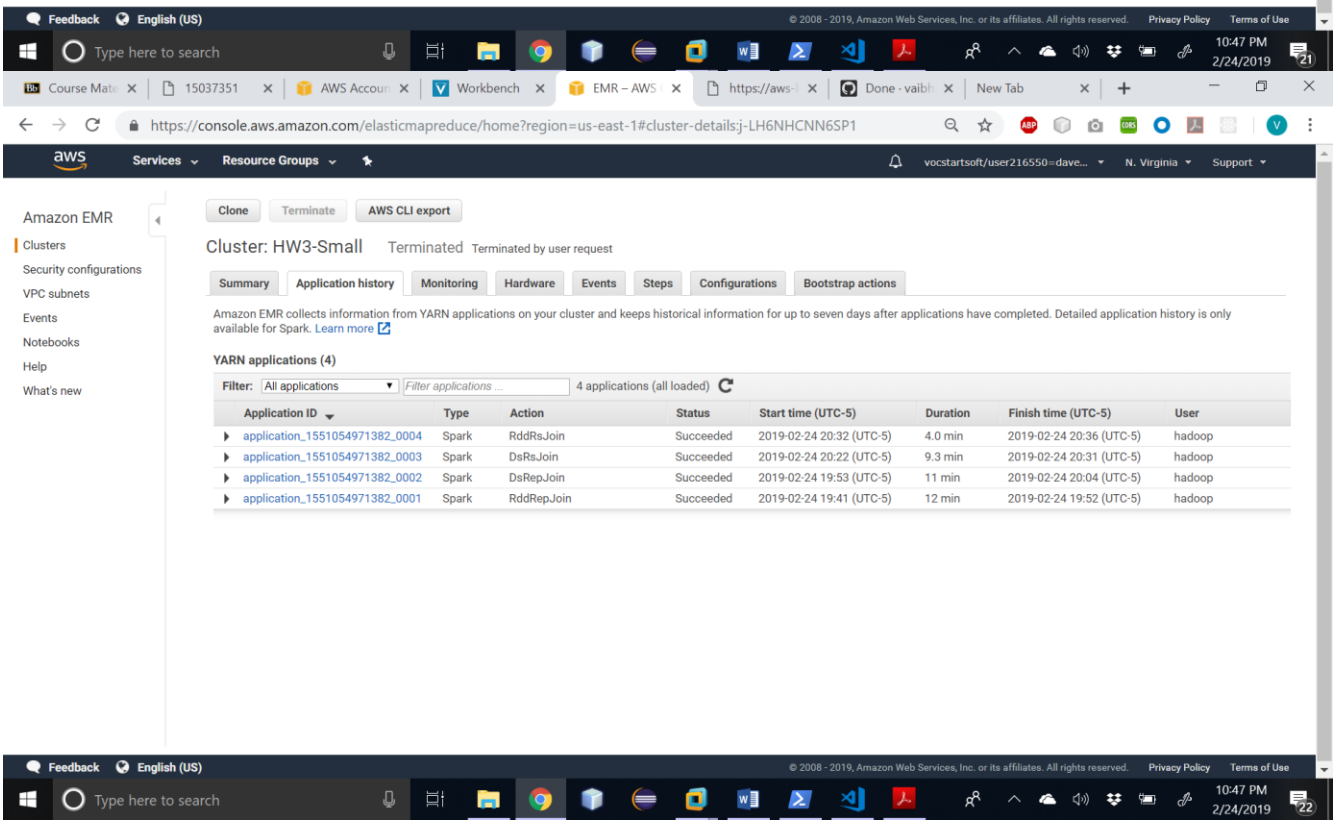
https://github.ccs.neu.edu/vaibhavdave5/parallelDataProcessing/blob/master/SocialTriagleSpark/Spark-Demo/answer.txt

Configuration	Small Cluster 4 machines	Large Cluster 7 machines
RS-R	Time =4.0min MaxF = 10000 Answer =520296	Time =2.5min MaxF = 10000 Answer =520296
RS-D	Time = 9.3 min MaxF = 10000 Answer =520296	Time = 7.4 min MaxF = 10000 Answer =520296
Rep-R	Time = 12 min MaxF = 500 Answer = 136	Time = 10 min MaxF = 500 Answer = 136
Rep-D	Time = 11min MaxF = 500 Answer = 136	Time = 11min MaxF = 500 Answer = 136

All the logs can be found here:

https://github.ccs.neu.edu/vaibhavdave5/parallelDataProcessing/tree/2bce0094419d184ce235908335b92fe0e351a157/SocialTriagleSpark/Spark-Demo/Logs


https://console.aws.amazon.com/elasticmapreduce/home?region=us-east-1#cluster-details:j-B5X77YOOR2Y8
Services
Resource Groups
N. Virginia
Support
Amazon EMR
Clusters
Security configurations
VPC subnets
Events
Notebooks
Help
What's new
Clone
Terminate
AWS CLI export
Cluster: HW3Large
Terminated
Terminated by user request
Summary
Application history
Monitoring
Hardware
Events
Steps
Configurations
Bootstrap actions
Amazon EMR collects information from YARN applications on your cluster and keeps historical information for up to seven days after applications have completed. Detailed application history is only available for Spark. Learn more
YARN applications (4)
Filter: All applications
Filter applications ...
4 applications (all loaded)
Application ID
Type
Action
Status
Start time (UTC-5)
Duration
Finish time (UTC-5)
User
application_1551059137217_0004 Spark DsRepJoin Succeeded 2019-02-24 21:13 (UTC-5) 11 min 2019-02-24 21:24 (UTC-5) hadoop
application_1551059137217_0003 Spark RddRepJoin Succeeded 2019-02-24 21:02 (UTC-5) 10 min 2019-02-24 21:12 (UTC-5) hadoop
application_1551059137217_0002 Spark DsRsJoin Succeeded 2019-02-24 20:53 (UTC-5) 7.4 min 2019-02-24 21:00 (UTC-5) hadoop
application_1551059137217_0001 Spark RddRsJoin Succeeded 2019-02-24 20:50 (UTC-5) 2.5 min 2019-02-24 20:52 (UTC-5) hadoop
Feedback
English (US)
© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Type here to search
10:47 PM
2/24/2019


https://console.aws.amazon.com/elasticmapreduce/home?region=us-east-1#cluster-details:j-LH6NHCNN6SP1
Services
Resource Groups
N. Virginia
Support
Amazon EMR
Clusters
Security configurations
VPC subnets
Events
Notebooks
Help
What's new
Clone
Terminate
AWS CLI export
Cluster: HW3-Small
Terminated
Terminated by user request
Summary
Application history
Monitoring
Hardware
Events
Steps
Configurations
Bootstrap actions
Amazon EMR collects information from YARN applications on your cluster and keeps historical information for up to seven days after applications have completed. Detailed application history is only available for Spark. Learn more
YARN applications (4)
Filter: All applications
Filter applications ...
4 applications (all loaded)
Application ID
Type
Action
Status
Start time (UTC-5)
Duration
Finish time (UTC-5)
User
application_1551054971382_0004 Spark RddRsJoin Succeeded 2019-02-24 20:32 (UTC-5) 4.0 min 2019-02-24 20:36 (UTC-5) hadoop
application_1551054971382_0003 Spark DsRsJoin Succeeded 2019-02-24 20:22 (UTC-5) 9.3 min 2019-02-24 20:31 (UTC-5) hadoop
application_1551054971382_0002 Spark DsRepJoin Succeeded 2019-02-24 19:53 (UTC-5) 11 min 2019-02-24 20:04 (UTC-5) hadoop
application_1551054971382_0001 Spark RddRepJoin Succeeded 2019-02-24 19:41 (UTC-5) 12 min 2019-02-24 19:52 (UTC-5) hadoop
Feedback
English (US)
© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use
Type here to search
10:47 PM
2/24/2019