1. **Pseudo-code for 5 Twitter-follower-count Programs**

* RDD-G

텍스트이(가) 표시된 사진

자동 생성된 설명

* RDD-R

텍스트이(가) 표시된 사진

자동 생성된 설명

* RDD-F

텍스트이(가) 표시된 사진

자동 생성된 설명

* RDD-A

텍스트이(가) 표시된 사진

자동 생성된 설명

* DSET

텍스트이(가) 표시된 사진

자동 생성된 설명

1. **Source code for 5 Twitter-follower-count Programs**

* RDD-G: <https://github.com/CS6240/hw-3-jill666666/blob/master/src/main/scala/fc/RDD-G.scala>
* RDD-R: <https://github.com/CS6240/hw-3-jill666666/blob/master/src/main/scala/fc/RDD-R.scala>
* RDD-F: <https://github.com/CS6240/hw-3-jill666666/blob/master/src/main/scala/fc/RDD-F.scala>
* RDD-A: <https://github.com/CS6240/hw-3-jill666666/blob/master/src/main/scala/fc/RDD-A.scala>
* DSET: <https://github.com/CS6240/hw-3-jill666666/blob/master/src/main/scala/fc/DSET.scala>

1. **Aggregation and Shuffle**

\* ‘toDebugString’ method logs the RDD lineage graph. Using this we can find out how the shuffle, partition, and parallelism work. Indentation indicates the shuffle boundary.

\*\* Value members referenced from: <https://spark.apache.org/docs/latest/api/scala/index.html>

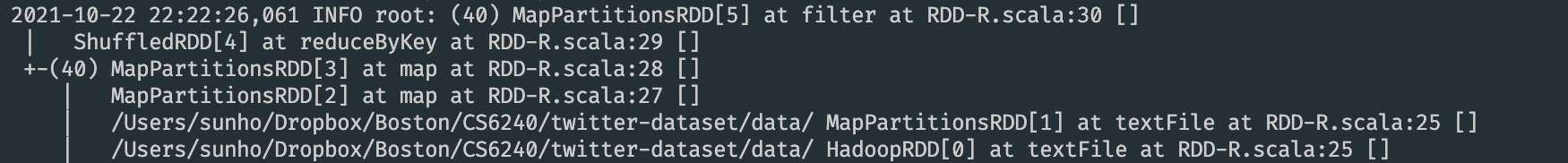
1. RDD-G

텍스트이(가) 표시된 사진

자동 생성된 설명

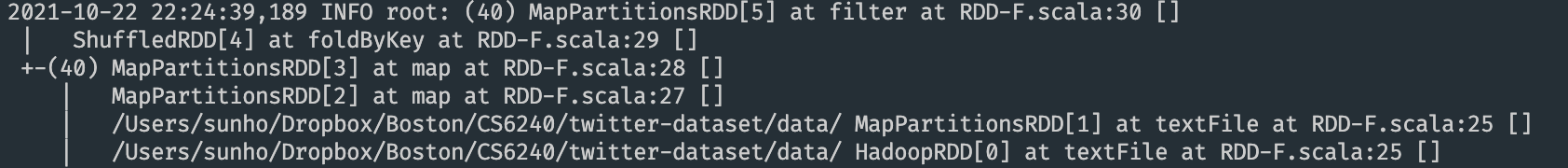
* First of all, ‘groupByKey()’ reshuffles the data. Afterward, we use ‘RDD.sum()’ equivalent in order to sum up the data. Shuffling happens before the aggregation. The operation is very expensive due to the shuffling.
* Ref: [Apache Spark ScalaDoc groupByKey()](https://spark.apache.org/docs/latest/api/scala/org/apache/spark/rdd/PairRDDFunctions.html#groupByKey():org.apache.spark.rdd.RDD[(K,Iterable[V])])

1. RDD-R



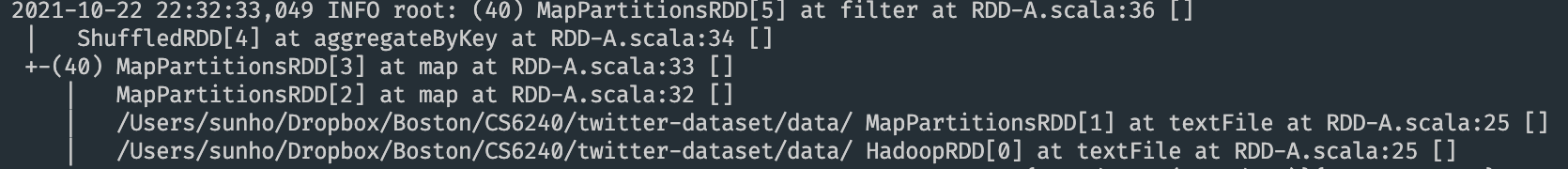
* Here, unlike ‘groupByKey()’, the ‘reduceByKey()’ work as a combiner. Before shuffling the data, it can combine values for each key on each partition. Basically, it works similarly to a ‘combiner’ in MapReduce.
* Ref: [Apache Spark ScalaDoc reduceByKey()](https://spark.apache.org/docs/latest/api/scala/org/apache/spark/rdd/PairRDDFunctions.html#reduceByKey(func:(V,V)=%3EV):org.apache.spark.rdd.RDD[(K,V)])

1. RDD-F



* ‘foldByKey()’ merges the values for each key using the combine functions. But this does not cause shuffle.
* Ref: [Apache Spark ScalaDoc foldByKey()](https://spark.apache.org/docs/latest/api/scala/org/apache/spark/rdd/PairRDDFunctions.html#foldByKey(zeroValue:V)(func:(V,V)=%3EV):org.apache.spark.rdd.RDD%5B(K,V)%5D)

1. RDD-A

 텍스트이(가) 표시된 사진

자동 생성된 설명

* ‘aggregateByKey()’ use the given combine functions to aggregate both partition and the value pairs, but it will not cause shuffle.
* Ref: [Apache Spark ScalaDoc aggregateByKey()](https://spark.apache.org/docs/latest/api/scala/org/apache/spark/rdd/PairRDDFunctions.html#aggregateByKey[U](zeroValue:U)(seqOp:(U,V)=%3EU,combOp:(U,U)=%3EU)(implicitevidence$3:scala.reflect.ClassTag[U]):org.apache.spark.rdd.RDD[(K,U)])

1. DSET

텍스트이(가) 표시된 사진

자동 생성된 설명

* Above is the physical plan we get by running ‘counts.explain()’ where variable count can be attained by following the steps on the DSET pseudo-code. The ‘groupBy(col)’ doesn’t combine, but instead reshuffle the data. Afterward, ‘agg(col)’ and ‘sum(col)’ aggregate the data.
* Ref: [Apache Spark ScalaDoc groupBy()](https://spark.apache.org/docs/latest/api/scala/org/apache/spark/sql/Dataset.html#groupBy(col1:String,cols:String*):org.apache.spark.sql.RelationalGroupedDataset)

1. **Pseudo-code for Triangle-counting Programs**
2. RS-R

텍스트이(가) 표시된 사진

자동 생성된 설명

1. RS-D

텍스트이(가) 표시된 사진

자동 생성된 설명

1. Rep-R

텍스트이(가) 표시된 사진

자동 생성된 설명

1. Rep-D

텍스트이(가) 표시된 사진

자동 생성된 설명

1. **Source code for Triangle-counting Programs**

* RS-R: <https://github.com/CS6240/hw-3-jill666666/blob/master/src/main/scala/tc/RS-R.scala>
* RS-D: <https://github.com/CS6240/hw-3-jill666666/blob/master/src/main/scala/tc/RS-D.scala>
* Rep-R: <https://github.com/CS6240/hw-3-jill666666/blob/master/src/main/scala/tc/Rep-R.scala>
* Rep-D: <https://github.com/CS6240/hw-3-jill666666/blob/master/src/main/scala/tc/Rep-D.scala>

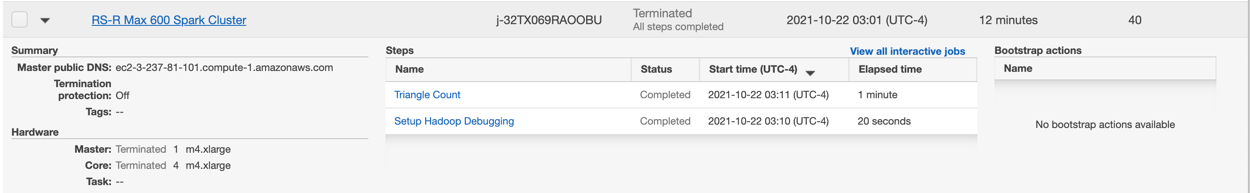
1. **Run Triangle-counting Programs on EMR (1 Master & 4 Workers)**

\* The MAX\_VALUE values from HW2 were 600, 2,400, and 6,000, but after receiving a feedback that the running time is too short, the values for HW3 have been updated to 600, 2,400, and 42,000. We also use MAX\_VALUE of 12,000 to compare between the Spark and MapReduce implementations, which will be discussed in Question 8.

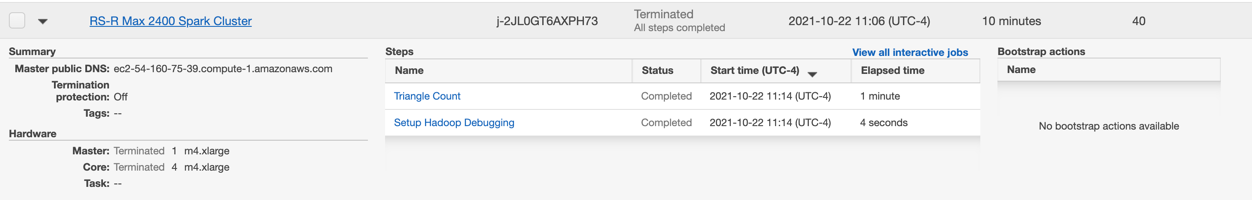
\*\* Due to out-of-memory error, for Rep-D, the greatest MAX\_VALUE has been set to 6,000, instead of 42,000 (Also tested with values 28,000 and 12,000, but same errors have been occurred).

1. **RS-R**

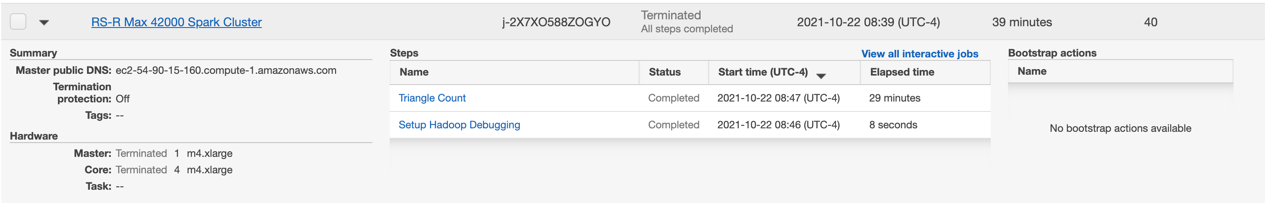
* MAX: 600 / Running Time: 1 min 20 seconds / Triangle Count: 572 / Machine Type: 1 Master (m4.xlarge) & 4 Workers (m4.xlarge)



* MAX: 2,400 / Running Time: 1 min 4 seconds / Triangle Count: 8,798 / Machine Type: 1 Master (m4.xlarge) & 4 Workers (m4.xlarge)

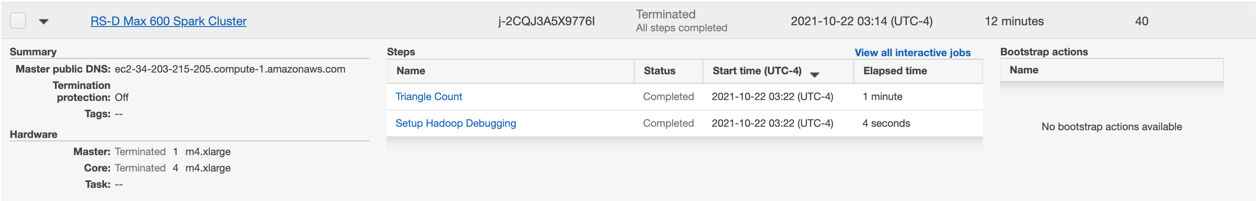


* MAX: 42,000 / Running Time: 29 mins 8 seconds / Triangle Count: 4,912,313 / Machine Type: 1 Master (m4.xlarge) & 4 Workers (m4.xlarge)

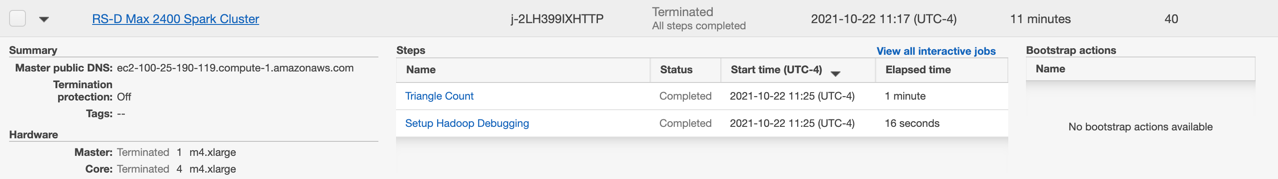


1. **RS-D**

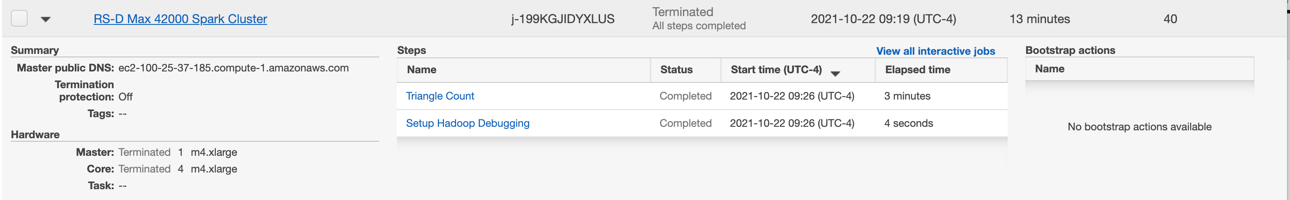
* MAX: 600 / Running Time: 1 min 4 seconds / Triangle Count: 572 / Machine Type: 1 Master (m4.xlarge) & 4 Workers (m4.xlarge)



* MAX: 2,400 / Running Time: 1 min 16 seconds / Triangle Count: 8,798 / Machine Type: 1 Master (m4.xlarge) & 4 Workers (m4.xlarge)

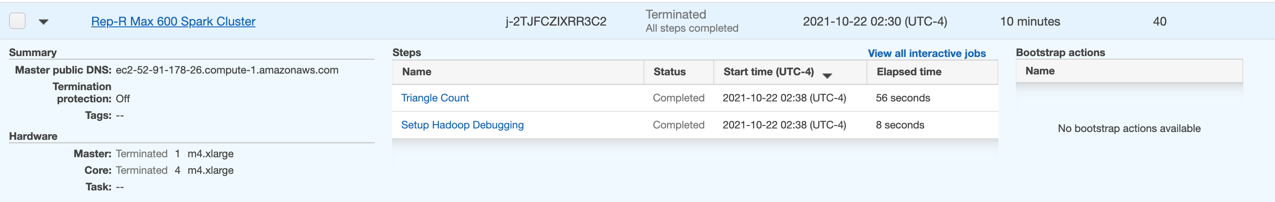


* MAX: 42,000 / Running Time: 3 min 4 seconds / Triangle Count: 4,912,313 / Machine Type: 1 Master (m4.xlarge) & 4 Workers (m4.xlarge)

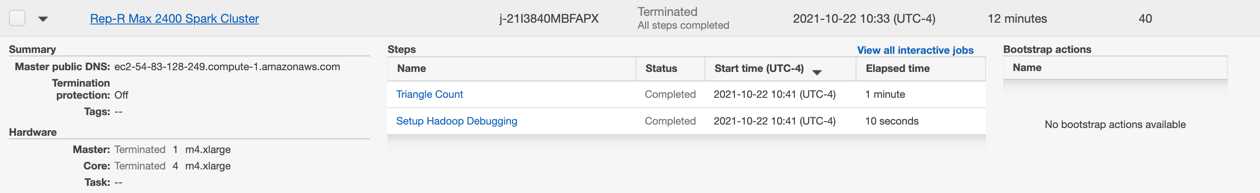


1. **Rep-R**

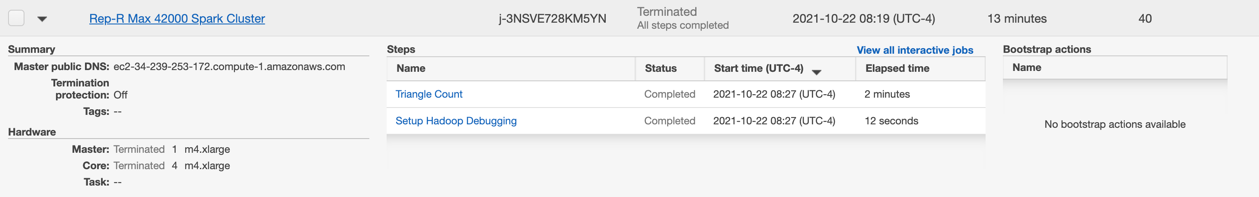
* MAX: 600 / Running Time: 1 min 4 seconds / Triangle Count: 572 / Machine Type: 1 Master (m4.xlarge) & 4 Workers (m4.xlarge)



* MAX: 2,400 / Running Time: 1 min 10 seconds / Triangle Count: 8,798 / Machine Type: 1 Master (m4.xlarge) & 4 Workers (m4.xlarge)

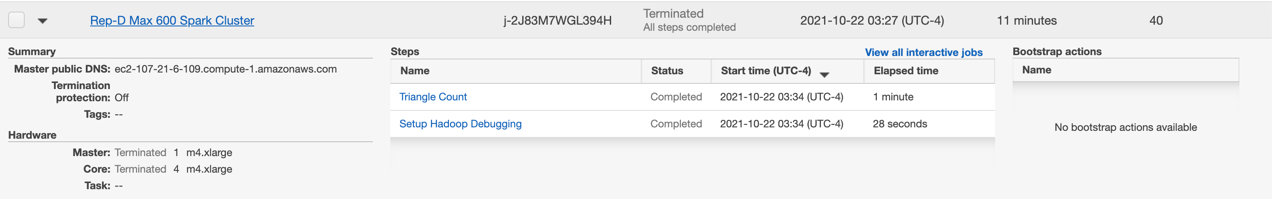


* MAX: 42,000 / Running Time: 2 mins 12 seconds / Triangle Count: 4,912,313 / Machine Type: 1 Master (m4.xlarge) & 4 Workers (m4.xlarge)

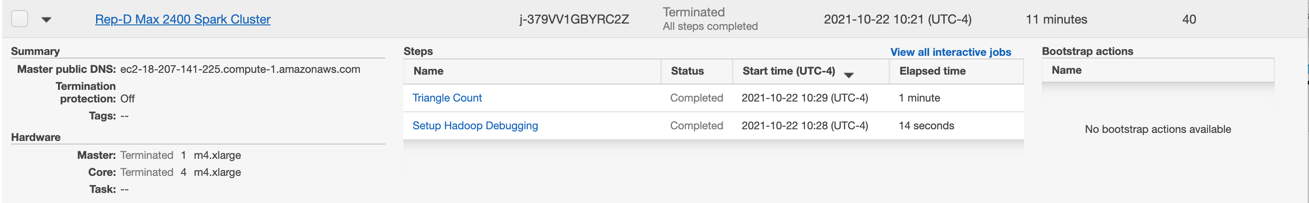


1. **Rep-D**

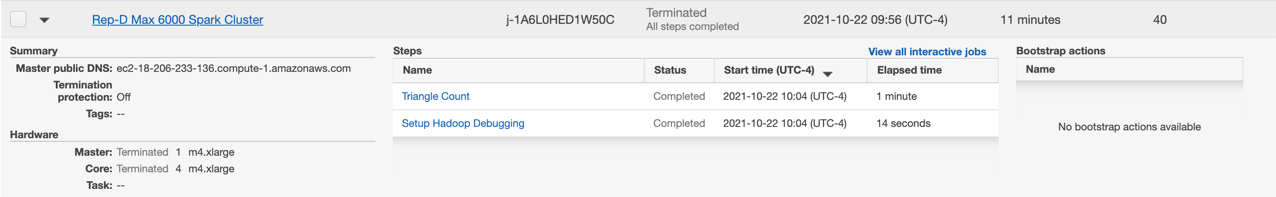
* MAX: 600 / Running Time: 1 min 28 seconds / Triangle Count: 572 / Machine Type: 1 Master (m4.xlarge) & 4 Workers (m4.xlarge)



* MAX: 2,400 / Running Time: 1 min 14 seconds / Triangle Count: 8,798 / Machine Type: 1 Master 1 (m4.xlarge) & 4 Workers (m4.xlarge)



* MAX: 6,000\*\* / Running Time: 1 min 14 seconds / Triangle Count: 131,654\*\* / Machine Type: Master 1 (m4.xlarge) & Workers (m4.xlarge)



1. **EMR Output File std-err (1 Master & 4 Workers)**

* RS-R: <https://github.com/CS6240/hw-3-jill666666/tree/master/aws-outputs/RS-R/1M4W>
* RS-D: <https://github.com/CS6240/hw-3-jill666666/tree/master/aws-outputs/RS-D/1M4W>
* Rep-R: <https://github.com/CS6240/hw-3-jill666666/tree/master/aws-outputs/Rep-R/1M4W>
* Rep-D: <https://github.com/CS6240/hw-3-jill666666/tree/master/aws-outputs/Rep-D/1M4W>

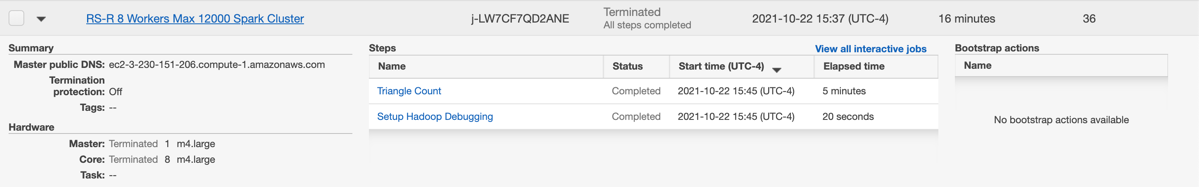
1. **Run Triangle-counting Programs on EMR (1 Master & 8 Workers)**

\* The MAX\_VALUE values from HW2 were 600, 2,400, and 6,000, but after receiving a feedback that the running time is too short, the values for HW3 have been updated to 600, 2,400, and 42,000. We also use MAX\_VALUE of 12,000 to compare between the Spark and MapReduce implementations.

\*\* Every implementations **except** Spark Rep-D has the same setup, MAX\_VALUE and machine type, as well as triangle count result. As mentioned in the Question 6, due to out-of-memory error, Rep-D program has been ran using the MAX\_VALUE of 6,000. The details including the comparison with the MapReduce are shown below.

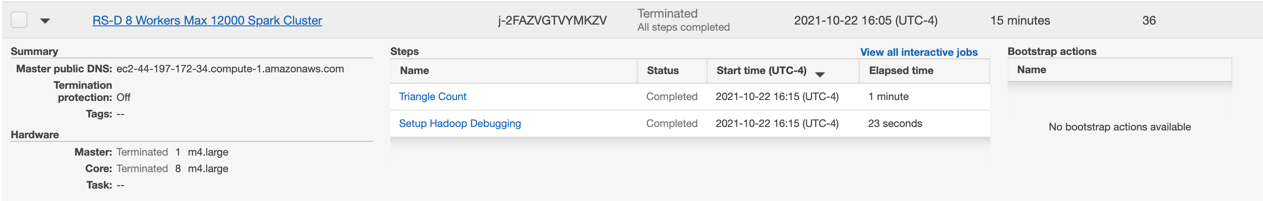
1. **Spark (HW3)**
2. **RS-R**

* MAX: 12,000
* Running Time: 5 mins 20 seconds
* Triangle Count: 856,482
* Machine Type: 1 Master (m4.large) & 8 Workers (m4.large)



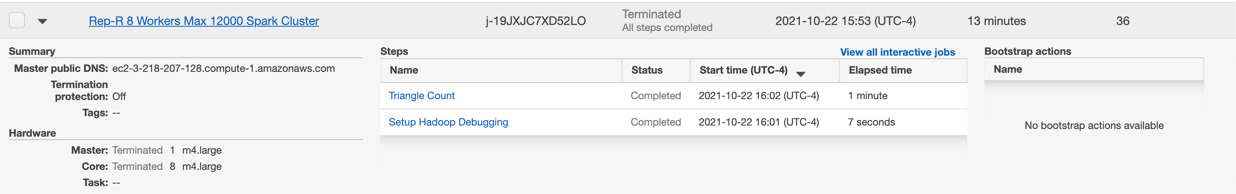
1. **RS-D**

* MAX: 12,000
* Running Time: 1 min 23 seconds
* Triangle Count: 856,482
* Machine Type: 1 Master (m4.large) & 8 Workers (m4.large)



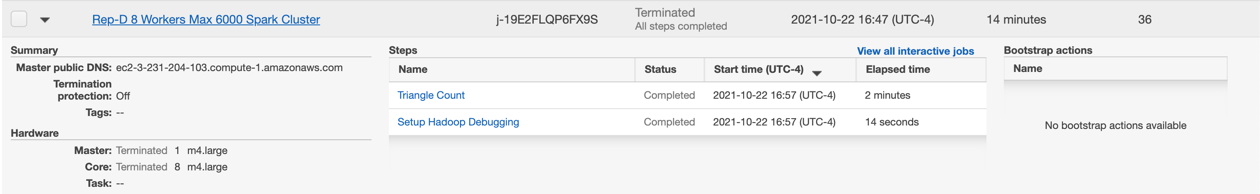
1. **Rep-R**

* MAX: 12,000
* Running Time: 1 min 7 seconds
* Triangle Count: 856,482
* Machine Type: 1 Master (m4.large) & 8 Workers (m4.large)



1. **Rep-D\***

* MAX: 6,000\*
* Running Time: 2 mins 14 seconds
* Triangle Count: 131,654\*
* Machine Type: 1 Master (m4.large) & 8 Workers (m4.large)



1. **MapReduce (HW2)**
2. **Reduce-side Join**

* MAX: 12,000
* Total Running Time: Approximately 5 mins 30 seconds
* Triangle Count: 856,482
* Machine Type: 1 Master (m4.large) & 8 Workers (m4.large)

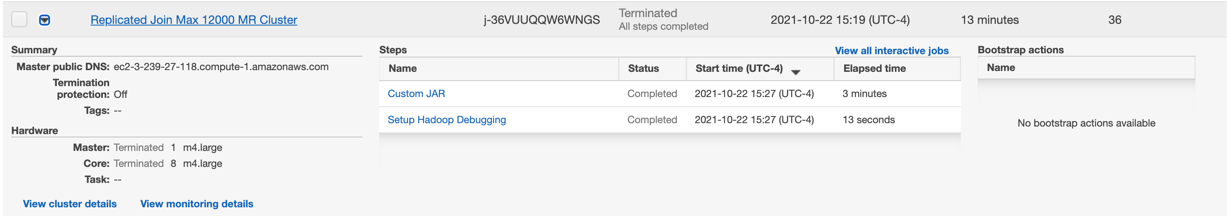
텍스트이(가) 표시된 사진

자동 생성된 설명

* ‘Path2 Join Max 12000 MR Cluster’ stores output Path2 in the S3 bucket, and ‘Reduce-side Join Max 12000 MR Cluster’ take it as an input and calculates triangle count.

1. **Replicated Join**

* MAX: 12,000
* Running Time: 3 mins 13 seconds
* Triangle Count: 856,482
* Machine Type: 1 Master (m4.large) & 8 Workers (m4.large)



1. **EMR Output File std-err (1 Master & 8 Workers)**

* RS-R: <https://github.com/CS6240/hw-3-jill666666/tree/master/aws-outputs/RS-R/1M8W>
* RS-D: <https://github.com/CS6240/hw-3-jill666666/tree/master/aws-outputs/RS-D/1M8W>
* Rep-R: <https://github.com/CS6240/hw-3-jill666666/tree/master/aws-outputs/Rep-R/1M8W>
* Rep-D: <https://github.com/CS6240/hw-3-jill666666/tree/master/aws-outputs/Rep-D/1M8W>