CS6240, MW, Assignment1, Hardik Shah

https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-1

Map-Reduce Implementation

- Pseudo Code

Let Follower be an object (userId1, userId2) such that userId1 follows userId2, and userId2 has userId1 as follower.

```
map(Follower f){
      followerString = convert f to string
      userIds = split(followerString, ",")
     userId_that_has_one_follower = userIds[1]
     emit(userId_that_has_one_follower, 1)
}

reducer(userId_that_has_one_follower, counts []){
     sum = 0
     For each c in counts:
         add c to sum
     emit(userId_that_has_one_follower, sum)
}
```

- Explanation
    1. Program reads the edges.csv file and read input line by line. Each line is of format userId1,userId2
    2. The input file is split into various chunks according to the Hadoop framework. By default, it creates the same number of map tasks as input splits.
    3. Each map task receives a bunch of input records to execute.
    4. For each input record, the map task calls the map function which is shown in the pseudo-code.
    5. Each map function call, then receives one input record.
    6. Map function converts the input to string, splits the record to separate them by “,” inorder to get both user ids. The aim of the program is to find

the followers counts of userIds that has one follower. According to the input, the second record is the userId that has first record as follower.

7. Hence, we emit the second user record with count 1.
8. Each map task calls map function of each input record.
9. Once, all map tasks are completed, the hadoop framework shuffles the map output to group all record with same userId and sort each grouping by the key. (key is userId, value is the count)
10. Now reducer tasks are created. Each reducer task is responsible for set of keys.
11. The reducer calls the reduce function for each input key record.
12. Hence, reduce function received input key and iterable of counts
13. The function then adds each of the counts and emits (userId, total)
14. The total output files are equal to the total reducers.

Spark Scala Implementation

- Pseudo Code

Let sc be the spark context which contains all the spark configuration and initializes and object. Let Follower be an object (userId1, userId2) such that userId1 follows userId2, and userId2 has userId1 as follower.

```
main(inputFile, outputFile) {
      val textFile = sc.readFromFile(inputFile)
      val userRDD = getUserRDD(textFile)
      val counts = reduce(userRDD)
      counts.saveToFile(outputFile)
}

getUserRDD(textFile) {
      val user_with_one_follower = textFile.map(line =>
line.split(",")(1))
      val userRDD = user_with_one_follower.map(user => (user,1))
     userRDD
}

reduce(userRDD) {
     userRDD.reduceByKey((count1, count2) => count1 + count2)
}
```

- toDebugString output:

Spark run 1:

```
19/09/19 16:10:02 INFO root: (20) ShuffledRDD[3] at reduceByKey at
TwitterFollowers.scala:24 []
 +-(20) MapPartitionsRDD[2] at map at TwitterFollowers.scala:20 []
    |   s3://hardiks-bucket/input MapPartitionsRDD[1] at textFile at
TwitterFollowers.scala:18 []
    |   s3://hardiks-bucket/input HadoopRDD[0] at textFile at
TwitterFollowers.scala:18 []
```

Spark run 2:

```
19/09/19 23:14:40 INFO root: (20) ShuffledRDD[3] at reduceByKey at
TwitterFollowers.scala:24 []
 +-(20) MapPartitionsRDD[2] at map at TwitterFollowers.scala:20 []
    |   s3://hardiks-bucket/input MapPartitionsRDD[1] at textFile at
TwitterFollowers.scala:18 []
    |   s3://hardiks-bucket/input HadoopRDD[0] at textFile at
TwitterFollowers.scala:18 []
```

Running Time Measurements

- Time to run Map-Reduce program
    1. Run1: 1 min 54 sec according to syslog
    2. Run2: 1 min 50 sec according to syslog
- Time to run Spark Scala program
    1. Run1: 1 min 11 sec according to stderr logs
    2. Run2: 1 min 11 sec according to stderr logs

-

	Data
Input to Mappers	1319473149 bytes = 1.319 Gb
Mapper to Reducer	961483442 bytes = 961.48 Mb
Reducer to output	67641452 bytes = 67.41 Mb

Why Map-Reduce Program has good or bad speedup?

According to the logs, total of split of input file was 20 and according to the default setting of one map task per input split, there were 20 map tasks created. These 20 map tasks were processed and passed to 9 reduce tasks which is equal to 9 output files that are generated.
The flow from mapper to reducer is a barrier primitive since, all the reduce tasks have to be executed only after all the map tasks are completed.
Since, most of the tasks run in parallel and since (1)one input split doesn't have to depend on others and (2)there is no particular part of the program that is sequential, the program has a pretty good speedup.
However, the only problem that might slow down the speedup is limited number of containers per node. Let's say each node has 3 containers to execute task. No matter how good the split be, there can be only 3*5 i.e 15 tasks that can be in parallel. The other tasks have to wait in the Yarn queue. Same goes for reducers.

Log File links:

1. Hadoop run 1: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-1/tree/master/logs/hadoop-run-1
2. Hadoop run 2: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-1/tree/master/logs/hadoop-run-2
3. Spark run 1: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-1/tree/master/logs/spark-run-1
4. Spark run 2: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-1/tree/master/logs/spark-run-2

Link to Output Directories:

1. Map-Reduce 1st output: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-1/tree/master/output1_hadoop
2. Map-Reduce 2nd output: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-1/tree/master/output2_hadoop
3. Spark-Scala 1st output: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-1/tree/master/output1_spark
4. Spark-Scala 2nd output: https://github.ccs.neu.edu/cs6240-f19/shardik95-Assignment-1/tree/master/output2_spark