CS6240 -Assignment 2

Rahul Pandey

GitHub: https://github.com/2020-F-CS6240/homework-2-rahulpandeycs

Analysis Code: https://github.com/2020-F-CS6240/homework-2-rahulpandeycs/blob/master/src/main/java/twitter/CountAnalysisTwitterReduceSideJoinWith2HopPathCount.java

RS Join Code: https://github.com/2020-F-CS6240/homework-2-rahulpandeycs/blob/master/src/main/java/twitter/TwitterReduceSideJoinWithMaxFilter.java

Rep Join code: https://github.com/2020-F-CS6240/homework-2-rahulpandeycs/blob/master/src/main/java/twitter/TwitterReplicatedJoin.java

Problem Analysis

MapReduce pseudo-code for the program you used to determine the cardinality (and maybe data volume) of Path2:

Mapreduce PsuedoCode:

The map reduce code for analysis is run on full edges.csv data set. The data is input to map task and during each task we emit two values to count both incoming and outgoing edges from a user. For input edge (userX, userY) we emit (userX, “B”) and (userY, “A”), with flag “A” for incoming and “B” for outgoing. Later during the reduce phase we receive incoming and outgoing edges for each key as we mapped them during the map phase, the multiplication of these two values for each key gives us the two Hop path count for edges (X,Y,Z);

```
//Global counter defined to keep count of 2Path Sum
enum numberOfPath2 {
         COUNT;
}

Mapping Phase:

map(input){
 String[] followerFollowing =   input.split(“,”) // Split input row by “,” and get
an array with two values

  emit(followerFollowing[1], ”A”);
  emit(followerFollowing[0], ”B”);
}
```

Now during the reduce phase we count total incoming and outgoing edges for each user and then multiply it with each other to get the total path2 count.

```
reduce(key, ListValues[val1, val2...]){
    localEdgeIncoming = 0;
    localEdgeOutgoing = 0;

    for each value in ListValues{
        if value.charAt(0) == ‘A’  localEdgeIncoming++;
        else if value.charAt(0) == ‘B’  localEdgeOutgoing++;
    }
    currentCount =  localEdgeIncoming* localEdgeOutgoing
    increase count for global counter  numberOfPath2 by currentCount;
    write this output to file as (key, currentCount) // To check on possible output
volume
}
```

Map Reduce Analysis of Twitter data set:

	RS join input	RS join shuffled	RS join output	Rep join input	Rep join file cache	Rep join output
Step 1 (join of Edges with itself)	Cardinality : **85331845** Volume : **1319465938**	Cardinality **170663690** Volume of data sent from Mappers to Reducers: **1916604980**	Total cardinality : **953138453592** Volume of output: (Considering 8 bytes data for each record) **7625107629000**	Total cardinality : **85331845** Volume of input: **1319465938**	Total cardinality: **(85331845)***(**number Of Workers)** Volume of data broadcast to all machines: **(1319465938*** Number of workers)	Total cardinality: **953138453592** Volume of output: **7625107629000**

Step 2 (join of Path2 with Edges)	Total cardinality: **953138453592** Volume of input: **7625107629000**	Total cardinality :(UPPERBOUND) **953223785437** Volume of data sent from Mappers to Reducers: **size**(edges. csv) + input Volume = ~**7626427095000**	Good upper bound for total Cardinality: **1** Volume of output: 1	Total cardinality: (UPPERBOUND) **953138453592** Volume of input: **7625107629000**	Total cardinality: **85331845*(no. Of Workers)** Volume of data broadcast to all machines: **(1319465938*** no. of workers)	Good upper bound for total Cardinality: **1** Volume of output: **1**

The Step1 RS Join output volume is 8bytes for each record * (total cardinality) I.e **7625107629000**
The Step2 RS Join total volume sent from mapper is **size**(edges.csv) + input Volume(**7625107629000**) = ~**7626427095000**
The Step2 RS Join cardinality is **cardinality**(edges.csv) **85331845** + cardinality of path2(**953138453592**) = ~**953223785437**

Join Implementation:

Map-Reduce pseudo-code for MAX-filter (Already present as part of Map phase)

```
long MAX_FILTER;

isWithinFilter(String[] followerFollowing){ //Max filter code that is used within the MAP phase
   follower = followerFollowing[0]
   following = followerFollowing[1]

  return follower < MAX_FILTER (AND)  following < MAX_FILTER // Return true if both follower and following are less than MAX_FILTER

}
```

Map-Reduce pseudo-code for RS-join

//To count the number of triangles we do it in steps: First we need to join edges table on itself and complete the Path (X,Y,Z) for all input pairs (X,Y) and (Y,Z). We do this my taking multiple file inputs of edges.csv and passing to below map and reduce function to generate all possible (X,Y,Z) pairs such that X follows Y and Y follows Z.

```
private static long MAX_FILTER_VALUE = 50000;
```

```
//Global counter defined to keep count of triangle Sum
enum numberOfTriangles {
         TRIANGLE_COUNT;
}

//Mapping phase:

map(input, output){  //For input row (X,Y),  First Mapper which emits (Y, AXY) and
(X, AXY)

 String[] followerFollowing =   input.split(“,”) // Split input row by “,” and get
an array with two values


 isWithinFilter(followerFollowing) { // Returns true if userIds are within the
MAX_FILTER VALUE   defined

emit(followerFollowing[1],”A”+input);
emit(followerFollowing[0],”B”+input);
}
```

```
//Reduce Phase:

Define listA, listB

reduce(key, listVal[val1, val2….]){

   for each val in listVal:
      if val.charAt(0) == ‘A’
          listA.add(val.substring(1))  //Add the input to listA
      else if val.charAt(0) == ‘B’
          listB.add(val.substring(1))  //Add the input to listB

   executeJoinLogic(key)
}



executeJoinLogic(key){

    for each valA in listA:
        for each valB in listB:
          emit(NULL, valA + “,” + valB.split(“,”)[1]) // EMIT (X,Y,Z) and write as
intermediate output

}
```

//After Processing the input file now we have intermediate output with Existing XYZ paths, now we need to make sure path ZX also exists so that make the triangle complete. For this we will use multiple inputs. The intermediate file and our original input file.

For this step now we will have two mappers, one mapper will read intermediate file created in step 1 and emit (X,Z) as the key, the other will read edges.csv and emit (X,Z) for each input record (Z,X). This will insure that both the emits with same key reach the reducer and we can be assured that path ZX exists for this pair. For example:

(X,Y), (Y,Z)   Exists.
We want to be sure (Z,X) also exists in edges.csv, so when we read a record from intermediate output we emit (X,Z) as the key stating we have path (X,Y,Z) for this (X,Z) pair.

When we read edges.csv input to this mapper is path (Z,X) to match it with previous mapper key so as to reach same reducer we need to emit (X,Z) I.e
(userId2,userId1)

//Job Mapper & Reducer Part 2

//Mapper 1: (Input is intermediate path (X,Y,Z))

```
map(input, output){

 String[] followerFollowing =   input.split(“,”) // Split input row by “,” and get
an array with three values (X,Y,Z)


// From (X,Y,Z) Now we emit , (X,Z) as the Key
emit(followerFollowing[0] + “,” + followerFollowing[2], “P2”);
// Emitted ( followerFollowing[0] + “,” + followerFollowing[2]) as the key and “P2”
flag as value to be used in reduce phase.
}
```

//Mapper 2: (Input is edges.csv path (Z,X))

```
map(input, output){

 String[] followerFollowing =   input.split(“,”) // Split input row by “,” and get
an array with tow values (Z,X)

  isWithinFilter(followerFollowing){ //if values within maxFilter
 //Input to this mapper is edge (Z,X), to match with the earlier key we need to emit (X,Z)
emit(followerFollowing[1] + “,” + followerFollowing[0], “ZXEdge”);
// Emitted ( followerFollowing[1] + “,” + followerFollowing[0]) as the key and
“ZXEdge” flag as value to be used in reduce phase.
}
}
```

Now the above values from mapper will be input to the reducers and we need to count both values separately and our triangle count will be multiplication of both counts as we got all values for which (X,Y,Z) exists.

```
Initialize globalCount =0
reduce(key, ListValues[val1, val2...]){

  localXZCount = 0
  localZXCount = 0

  for each val in ListValues :
       if(val == “P2”) localXZCount++;
       else if(val == “ZXEdge”) localZXCount++;
   }
   globalCount += localXZCount*localZXCount;

}

cleanup(){
    increase count for global counter  numberOfTriangles by globalCount;
}
```

Map-Reduce pseudo-code for Rep-join

//The map reduce Replicated Join uses the cache to store file and is a map only. For Replicated join we are first going to setup our distributed cache in the @Setup method and once we have the edges.csv table in the cache we proceed to map phase and try to establish path X,Y,Z for each user such that X follows Y, Y follows Z and Z follows X.

The setup method is used to load file into cache that will be available to all mappers during the map phase. For our scenario we will define a map **pathXYZMap** to store each user and the list of users that user is following.
We define *pathXYZMap* as HashMap<String, Set<String>> pathXYZ;

private static long MAX_FILTER_VALUE = 130000;

```
//Global counter defined to keep count of triangle Sum
enum numberOfTriangles {
         TRIANGLE_COUNT;
}


//Input to setup is the edges.csv file
setup(){
   For each file in Cache:
       while all lines are read for each file (line = CurrentLine):
          String[] following = line.split(“,”)
          isWithinFilter(following){ // If the data is within defined MAX Filter
             setOfUsersXFollows = pathXYZMap.get(following[0])
             setOfUserXFollows.add(following[1])
             pathXYZMap.put(following[0], setOfUserXFollows) //store each user and
the list of users that user is following.
         } }
```

//Mapping phase:

```
map(input) {  // Input to the map phase is the edges.csv file

   String[] followerFollowing = input.split(“,”);
   if(!isWithinFilter(followerFollowing)) do nothing;

   userX = followerFollowing[0];
   userY = followerFollowing[1];

//// Check for each input user if path (X->Y), (Y->Z), (Z->X)
if(pathXYZ contains key for userY) { // We will get the list of users Y follows

       zUsersThatYFollows = pathXYZMap.get(userY)
       for each zUser in  zUsersThatYFollows:
           xUsersThatZFollows = pathXYZMap.get(zUser);
           if(xUserThatZFollows contains userX) { // We have established a
path   // X,Y,Z such that X follows Y, Y follows Z, Z follows X
              increase count for global counter  numberOfTriangles by 1;
           }
   }
}
```

Configuration	Small Cluster Result	Large Cluster Result
RS-join, MAX = 50000	Running time: 22mins Triangle count: **12029907**	Running time: 14mins Triangle count: **12029907**
Rep-join, MAX = 130000	Running time: 21mins Triangle count: **63045888**	Running time: 20mins Triangle count: **63045888**

Syslogs Link:

Analysis: https://github.com/2020-F-CS6240/homework-2-rahulpandeycs/blob/master/logs/Path2_MRAnalysis/AWS_RUNLOG_ANALYSIS.TXT

Reduce Side Join:

Run 1 (Small Cluster): https://github.com/2020-F-CS6240/homework-2-rahulpandeycs/blob/master/logs/MR_RSJoinLogs/Run1SmallCluster/AWS_6Machines_RUN.txt

Run 2 (Large Cluster): https://github.com/2020-F-CS6240/homework-2-rahulpandeycs/blob/master/logs/MR_RSJoinLogs/Run2LargeCluster/syslog.txt

Replicated Side Join:

Run 1 (Small Cluster): https://github.com/2020-F-CS6240/homework-2-rahulpandeycs/tree/master/logs/MR_ReplicatedJoinLogs/Run1SmallCluster

Run 2 (Large Cluster): https://github.com/2020-F-CS6240/homework-2-rahulpandeycs/blob/master/logs/MR_ReplicatedJoinLogs/Run2LargeCluster/syslog.txt

Output Folder Link: (Including output file)

Analysis: https://github.com/2020-F-CS6240/homework-2-rahulpandeycs/tree/master/outputAnalysis

Reduce Side Join:

Run 1 (Small Cluster): https://github.com/2020-F-CS6240/homework-2-rahulpandeycs/tree/master/outputRS
Run 2 (Large Cluster): https://github.com/2020-F-CS6240/homework-2-rahulpandeycs/tree/master/outputRS2

Replicated Side Join:

Run 1 (Small Cluster): https://github.com/2020-F-CS6240/homework-2-rahulpandeycs/tree/master/outputRJ
Run 2 (Large Cluster): https://github.com/2020-F-CS6240/homework-2-rahulpandeycs/tree/master/outputRJ2