Akshay Syal

Link to Log files:
https://northeastern-my.sharepoint.com/:f:/r/personal/syal_ak_northeastern_edu/Documents/CS6240%20Large%20Scale%20Parallel%20Data%20Processing?csf=1&web=1&e=tImwt2

Write a brief report about your findings, answering the following questions:

1. **[10 points] Write a MapReduce program to implement EXACT (exact cardinality of length-2 paths in the Twitter graph using the product trick), using a single job. Instead of using a second job for adding up the products of the number of incoming and outgoing edges over all nodes, use a global counter. Show the pseudo-code for this program.**

```
Map (edge)
        follower_node, followed_node = edge.split(“,”)
        emit(follower_node,”o”)
        emit(followed_node,”i”)

Reduce (node,edgeTypeList)
        Outgoing_edge = 0
        Incoming_edge = 0

        For edgeType in edgeTypeList:
                If edgeType==’i’
                        Incoming_edge++
                Else
                        Outgoing_edge++

        Product = Incoming_edge*Outgoing_edge
        Global_Counter.increment(Product)
```

2. **[10 points] Show the link to the source code for this program in your Github Classroom repository.**
   https://github.com/CS6240/hw2-AkshaySyal/blob/main/src/main/java/wc/Exact.java

3. **[2 points] Run this program locally (no need to spend money on AWS) and report the cardinality of Path2 it computed on the full Twitter edges dataset.**
   953,138,453,592

4. **[10 points] Write a MapReduce program to implement APPROX (approximate cardinality of length-2 paths in the Twitter graph) that ideally uses a single job as well. Recall that this program uses a parameter MAX to remove edges and then**

performs the join of step 1 of the Naïve algorithm on the reduced edge set. Show the pseudo-code for this program.

APPROX algorithm

I implemented RS join. Since both nodes that map receives can act as a join key so I emit both of them. Since there's only one dataset I simulate the presence of two datasets by tagging each record with "S" or "E".

"S" signifies that the path will start from the node that is returned as value in the map phase.

"E" signifies that the path will end at the node that is returned as value in map phase.

The key emitted in the map phase represents the middle node of any 2 length path.
For eg
For 1,2:
emit(1, ("E",2)) and emit(2,("S",1))

In the reduce phase I get the key (middle node) and list of nodes (tagged as Start or End). I segregate the nodes into two lists (Start_List and End_List).

I join each Start node with the End node with the key node in the middle. I increment the global counter with 1 and emit the joined tuple.

Pseudocode

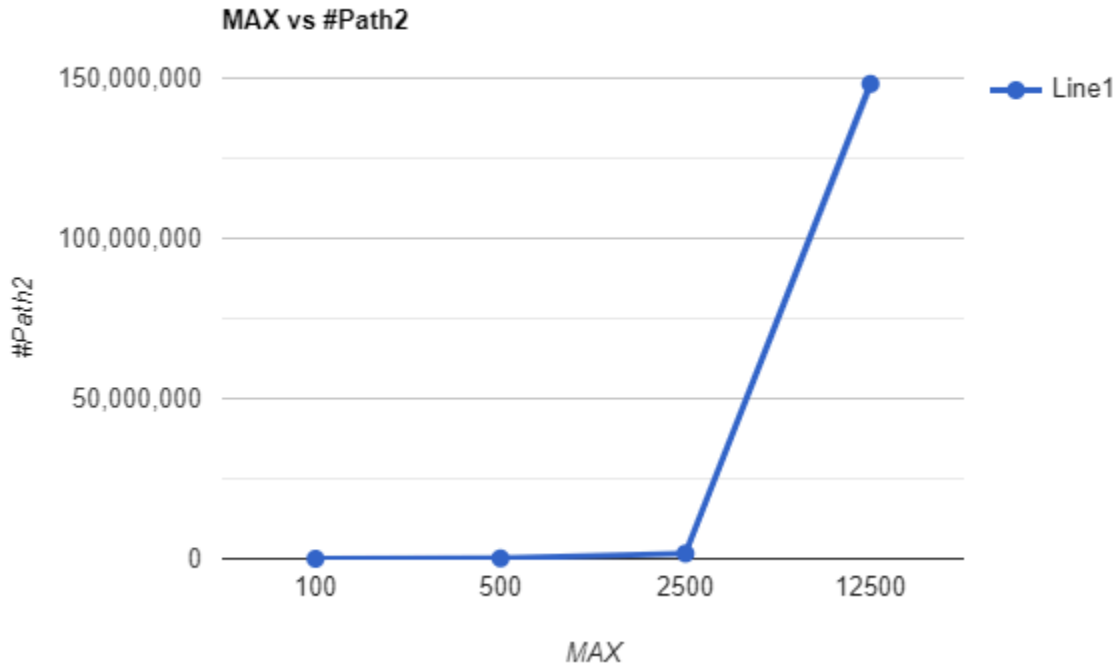
```
MAX = context.getConfiguration().get("MAX")

map(edge)
        follower_node, followed_node = edge.split(",")
        if follower_node.value < MAX and followed_node.value < MAX
                emit(follower_node,("E",followed_node))
                emit(followed_node,("S",follower_node))
reduce(node, records)
S_list = []
E_list = []

for record in records
        if record[0] == "S"
                S_list.append(record[1])
        elif record[0] == "E"
                E_list.append(record[1])
```

```
for start_node in S_list
        for end_node in E_list
                Global_Counter.increment(1)
                emit(start_node, node, end_node)
```

5. **[10 points] Show the link to the source code for this program in your Github Classroom repository.**

   Approx:
   https://github.com/CS6240/hw2-AkshaySyal/blob/main/src/main/java/wc/Approx.java

6. **[2 points] Run this program locally (no need to spend money on AWS) for a geometric sequence of MAX values such as 100, 500, 2500, and so on. Show a plot of your observations as a graph with MAX on the x-axis and the corresponding Path2 cardinality on the y-axis. As you increase MAX, look carefully at the increasing Path2 cardinality, and stop increasing MAX when you believe the next larger value will overwhelm your local resources (CPU, memory, or disk space).**



MAX	#Path2
100	1543
500	133,416
2,500	1,636,183
12,500	148,228,913

7. **[2 points] Look at the above graph and try to find the trend, i.e., manually "fit a curve" to the graph. Then use this trend to estimate the y-value when setting the x-value to the largest user ID in nodes.csv. Report this estimate and compare it to the Path2 cardinality you obtained from the exact approach. Is it close?**

   $y = (y_2 - y_1 / x_2 - x_1)x$

   $y_2 =$ 148,228,913 $x_2 =$ 12,500
   $y_1 =$ 1,636,183 $x_1 =$ 2,500

   $y = 14{,}659.273\ x$

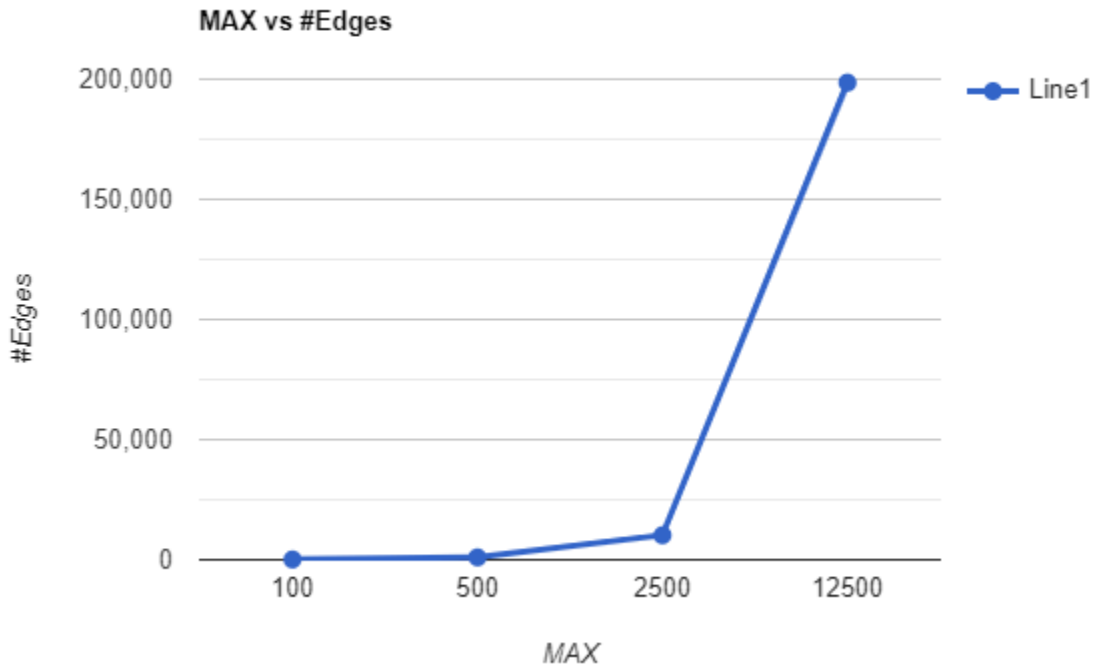
   Keeping x = 11,316,811

   y (Estimate #Path2) = 165,896,221,938 (approx)

   But Exact #Path2 = 953,138,453,592

   Hence the estimate is not close.

8. **[2 points] The MAX threshold is simple, but potentially problematic because it depends on user IDs, which are artificial values. For example, assume the data has users with IDs 1-10 and 1000-1100. Setting MAX=20, we get the edges for the first 10 users. Doubling MAX to 40 does not add any edges, and this remains unchanged until we reach MAX > 1000 when suddenly a lot of new edges for users 1000-1100 are included. The corresponding trend graph with MAX on the x-axis and the number of included edges on the y-axis would look like an extreme step function. Now assume the IDs are modified by replacing all numbers in range 1000-1100 with numbers in range 11-111. Then the trend would look very different, even though the Twitter graph did not change. Find out the situation for the Twitter dataset by creating and showing a plot with MAX on the x-axis and the number of input edges that remain after applying the MAX filter on the y-axis. Use the same MAX values as in the previous question. (You can run this program locally, not on AWS.)**



MAX	#Edges
100	140
500	1004
2500	10,217
12,500	198,567

As described in the question the trend looks like an extreme step function. Lots of new edges are added as MAX increases from 2500 to 12,500.

9. **[10 points] Show the pseudo-code for your triangle-counting program using RS-join. It is allowed to use more than 1 MR job. Make sure your program has MAX as a parameter to control job cost.**

   Consider path2 result to be P of schema (start, mid, end) and edges to be E of schema (start, end). Joining keys are (P.end, E.start) and (P.start, E.end). Hence the Map phase will handle one of the equality conditions [here P.end == E.start] and the Reduce phase will handle the other equality condition [P.start == E.end].

Algo RS-join consist of 2 steps:

1. Map phase (different map function for handling path2 and edges)
2. Reduce phase

Algorithm:

Job 1: Edges Join Edges to derive Path2 (similar to Q4)

```
MAX = context.getConfiguration().get("MAX")
map(edge)
        follower_node, followed_node = edge.split(“,”)
        if follower_node.value < MAX and followed_node.value < MAX
                emit(follower_node,(“E”,followed_node))
                emit(followed_node,(“S”,follower_node))

reduce(midNode, records)
        S_list = []
        E_list = []

        for record in records
                if record[0] == “S”
                        S_list.append(record[1])
                elif record[0] == “E”
                        E_list.append(record[1])

        for start_node in S_list
                for end_node in E_list
                        emit(start_node, midNode, end_node)
```

Job 2: Path2 Join Edges to get number of triangles

Map_path2 (path)

```
        start,mid,end = path.split(“,”)

        // I dont want paths going to reducers that are loops like 2,1,2
        if (start != end):
                emit(end, (start,mid) )
```

Map_Edges(edge)

```
    start, end = edge.split(“,”)
    emit(start, end)
```

Reduce(node, records)

```
Path2_List = []
Edges_List = []

for record in records:

    if record.length == 2:
        Path2_List.add(record)

    else
        Edges_list.add(record)

for path in Path2_list:
    for endNode in Edges_list:
        if path.start == endNode
            Global_Counter.increment(1)
            emit(path.start, path.mid+node+endNode)
```

After both jobs are done

```
return Global_Counter.getValue() / 3
```

10. **[10 points] Show the link to the source code for this program in your Github Classroom repository.**
    https://github.com/CS6240/hw2-AkshaySyal/blob/main/src/main/java/wc/RSJoin.java

11. **[4 points] Run RS-join on EMR using 1 master and 4 worker nodes (all cheap machines as for HW 1). Based on your analysis, decide if you need to use the MAX filter for the RS-join program or if you can run it on the full Twitter edge set. If you use MAX filter, set it to a value for which you estimate that your program will finish within 15 minutes to 1 hour. You may need to try 2-3 “interesting” MAX values to get into the desired range. (Note: It is fine to submit results for a program running longer than 1 hour. We set the upper limit to protect you from spending too much money: If your job is still not done after 1 hour, how long are you willing to wait? If you are not sure, terminate it and set a much smaller MAX value.)**

a. **Report the MAX value you used.**
b. **Report the triangle count obtained for this MAX value.**
c. **Report the running time of your program on EMR.**
d. **Show a link to your syslog file created by the run for which you reported these numbers.**

12. [2 points] Run the same RS-join program for the same MAX value on a larger EMR cluster with 1 master and 8 workers (same instance type as above).

a. **Report the running time on the larger cluster.**
b. **Compute the ratio between running time on the small vs the large cluster. Is this a good speedup?**

13. [10 points] Show the pseudo-code for your triangle-counting program using Rep-join. It is allowed to use more than 1 MR job but try to solve the problem with a single job. Make sure your program has MAX as a parameter to control job cost.

As per the hint mentioned in the assignment, by performing multiple join steps in a single job we can do Rep-Join in a single job. Edges.csv will be loaded into File Cache and will be referred to as Edges in pseudocode.

Algorithm:

```
MAX = context.getConfiguration().get("MAX")
setup()
        H = new hashMap

        for edge in Edges
                if edge.startNode.value < MAX and edge.endNode.value < MAX
                        H.insert(edge.startNode, edge.endNode)

map(edge)

// edge = startNode, endNode coming from Edges.csv
// input edge (1,2) <> 2:3 and 2:1
if edge.startNode.value < MAX and edge.endNode.value < MAX
        for each hmapEndNode in H.lookup(edge.endNode)

                // Avoiding loop like paths (2,1,2)
                if hmapEndNode  == edge.startNode
                        pass
                else
                        // intermediate_path = edge.startNode, edge.endNode, hmapEndNode
                        // Completing the triangle
                        for each hmapStartNode in H.lookup(hmapEndNode)
                                if hmapStartNode == edge.startNode
                                        Global_Triangle_Counter.increment(1)
```

emit(edge.startNode, edge.endNode+hmapEndNode)

After the jobs
return Global_Triangle_Counter.getValue() / 3

14. [10 points] Show the link to the source code for this program in your Github Classroom repository.

https://github.com/CS6240/hw2-AkshaySyal/blob/main/src/main/java/wc/RepJoin.java

15. [4 points] Run Rep-join on EMR using 1 master and 4 worker nodes (the same instance type as for RS-join). Based on your analysis, decide if you need to use the MAX filter for the Rep-join program or if you can run it on the full Twitter edge set. If you use MAX filter, set it to a value for which you estimate that your program will finish in 15 minutes to 1 hour. You may need to try 2-3 "interesting" MAX values to get into the desired range. (Note: It is fine to submit results for a program running longer than 1 hour. We set the upper limit to protect you from spending too much money: If your job is still not done after 1 hour, how long are you willing to wait? If you are not sure, terminate it and set a smaller MAX value.)

a. **Report the MAX value you used.**
b. **Report the triangle count obtained for this MAX value.**
c. **Report the running time of your program on EMR.**
d. **Show a link to your syslog file created by the run for which you reported these numbers.**

16. [2 points] Run the same Rep-join program for the same MAX value on a larger EMR cluster with 1 master and 8 workers (same instance type as above).

a. **Report the running time on the larger cluster.**
b. **Compute the ratio between running time on the small vs the large cluster. Is this a good speedup?**