CS 6240- Large-scale Parallel Data Processing
Homework 2
Name: Srikanth Babu Mandru

Words used in document:
PATH2 – number of paths of length 2
TRIANGLES– number of TRIANGLES in twitter edges dataset
user1 – user at first position in edge
user2 - user at second position in edge
MR job - MapReduce job

Problem Analysis:

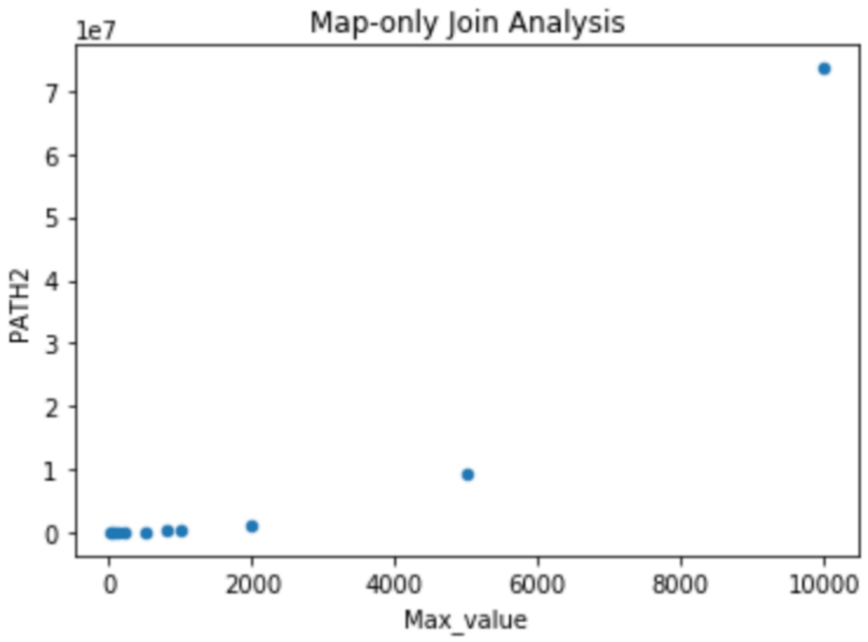
Steps followed to estimate the cardinality of PATH2:

For analysis, I have executed the Implementations of counting Triangles and noted the (max value, PATH2) pairs for both Reduce side Join and Replicated Join implementations. The pairs collected are plotted and is as shown in the figures given in "Graphs of Analysis" part below.

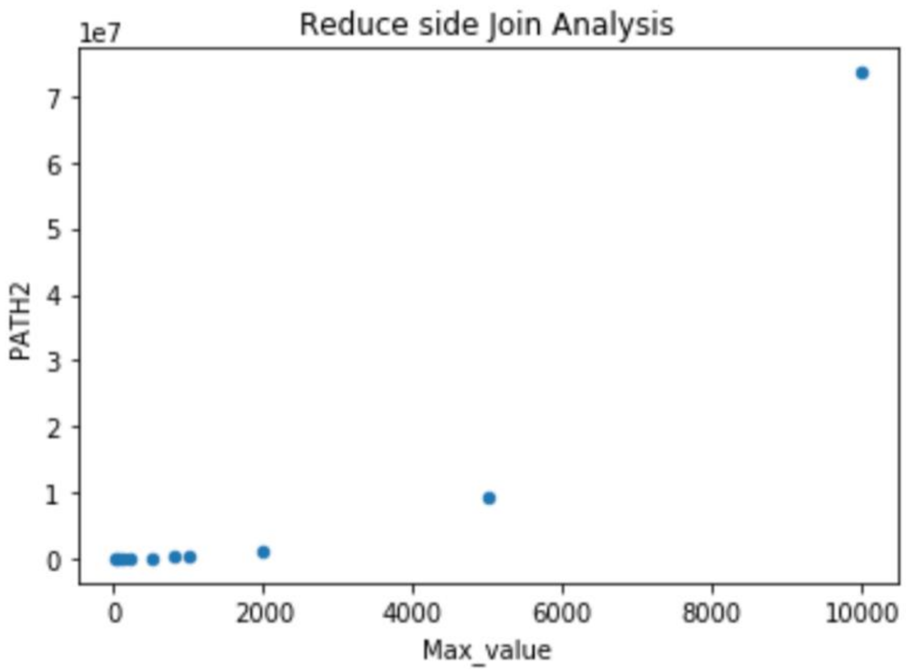
Table of analysis:
In below table, I have filled the values based on the exponential growth in values (observed from the graphs below) and made educated guess for bytes by looking at the log files. For Rep join, I have implemented only one MapReduce job.

	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)	Records: 85331845 Bytes: 28818535605	Records: 2 * 85331845 (since passing both file two times) Bytes: 2 * 85331845 * 8 (since 2 characters are passed)	Records: exp(85331845) Bytes: nearly exp(85331845) * 32	Records: 85331845 Bytes: 28818535605	Records: 0 Bytes: 0	Records: out = $\frac{\exp(85331845)}{\sqrt{85331845}}$ Bytes: out * 32
Step 2 (join of Path2 with Edges)	Records: exp(85331845) Bytes: nearly exp(85331845) * 32	Records: exp(85331845) Bytes: nearly exp(85331845) * 32 (since triple is passed)	Records: out = $\frac{\exp(85331845)}{\sqrt{85331845}}$ Bytes: out * 32	NA	NA	NA

Graphs of analysis:
Number of PATH2 for different values of "max" value in Replicated Join implementation of counting traingles is as below:



Number of PATH2 for different values of "max" value in Reduce Side Join implementation of counting traingles is as below:



Traingles Count through Joins Mapreduce Implementation:

(1) Reduce-side Join Map-Reduce Implementation:

The pseudo-code for triangle count program constitute of three map-reduce (MR) jobs. The functionalities of MR jobs are as follows:

- First MR job filter will filter initial data with "max" value.
- Second MR job joins two datasets of "edges" with the user2 as the key from one dataset and user1 as the key from other dataset.
- Third MR job joins the output dataset from second MR job and output dataset from first MR job.

Pseudo-code:

```
// First MR job for filtering with "max" value
    class MAPPER
        method Map (key, value)
            MAX = input the max value
            users[]  ← split "value" delimited by ","
            if (users[0] < MAX & users[1] < MAX )  // filter out the users more than "max"
                EMIT (users[0], users[1] )
    class REDUCER
        method Reduce (user u, <list of users>)
            for each c ∈ <list of users>:
                EMIT (u, c)

// second MR job for joining two datasets

    class MAPPER
        method Map (key, value)
            users[]  ← split "value" delimited by ","
            value_out = "A" + users[0]
            EMIT ( users[1], value_out )
            value_out2 = "B" + users[1]
            EMIT ( users[0], value_out2 )


    class REDUCER
        Initialize listA, listB
        method Reduce (term t, <list of values>)
            For each c ∈ <list of users> :
                user_value[]  ← split "value" delimited by ","

                if (user_value[0] == A ) {
                    listA.add(user_value[1] + "," + user_value[2] )
                }
                else {
                listB.add(user_value[1] + "," + user_value[2] )
                }
            For each a ∈ listA
                For each b ∈ listB
```

```
                        A_in ← Split “a” delimited by “,”
                        EMIT (A_in[0], b)
                        Increment (PATH2_counter) by 1
```

// Third MR job for joining output of MR job 2 with output of MR job of filtering and count traingles

```
    class MAPPER1
        method Map (key, value)
            users[] ← split “value” delimited by “,”
            value_out ← “A” + value
            EMIT (users[2] , value_out)

    class MAPPER2
        method Map (key, value)
            users[] ← split “value” delimited by “,”
            value_out ← “B” + value
            EMIT (users[0] , value_out)

    class REDUCER
        Initialize listA, listB
        method Reduce (term t, <list of values>)
            For each c ∈ <list of users> :
                user_value[]  ← split “value” delimited by “,”

                if (user_value[0] == A ) {
                    listA.add(user_value[1] + “,” + user_value[2] )
                }
                else {
                listB.add(user_value[1] + “,” + user_value[2] )
                }
            For each a ∈ listA
                For each b ∈ listB
                    A_in ← Split “a” delimited by “,”
                    B_in ← Split “a” delimited by “,”
                    If (A_in[0] == B_in[1])
                        EMIT (a , B_in[0])
                        Increment (TRIANGLES_counter) by 1
```

(2) Replicated Join (Map-Only) Map-Reduce Implementation:

Basic idea of solution:

Create the hash for edges dataset and broadcast it to all the mappers while executing the MAP phase on the splits of edges dataset.

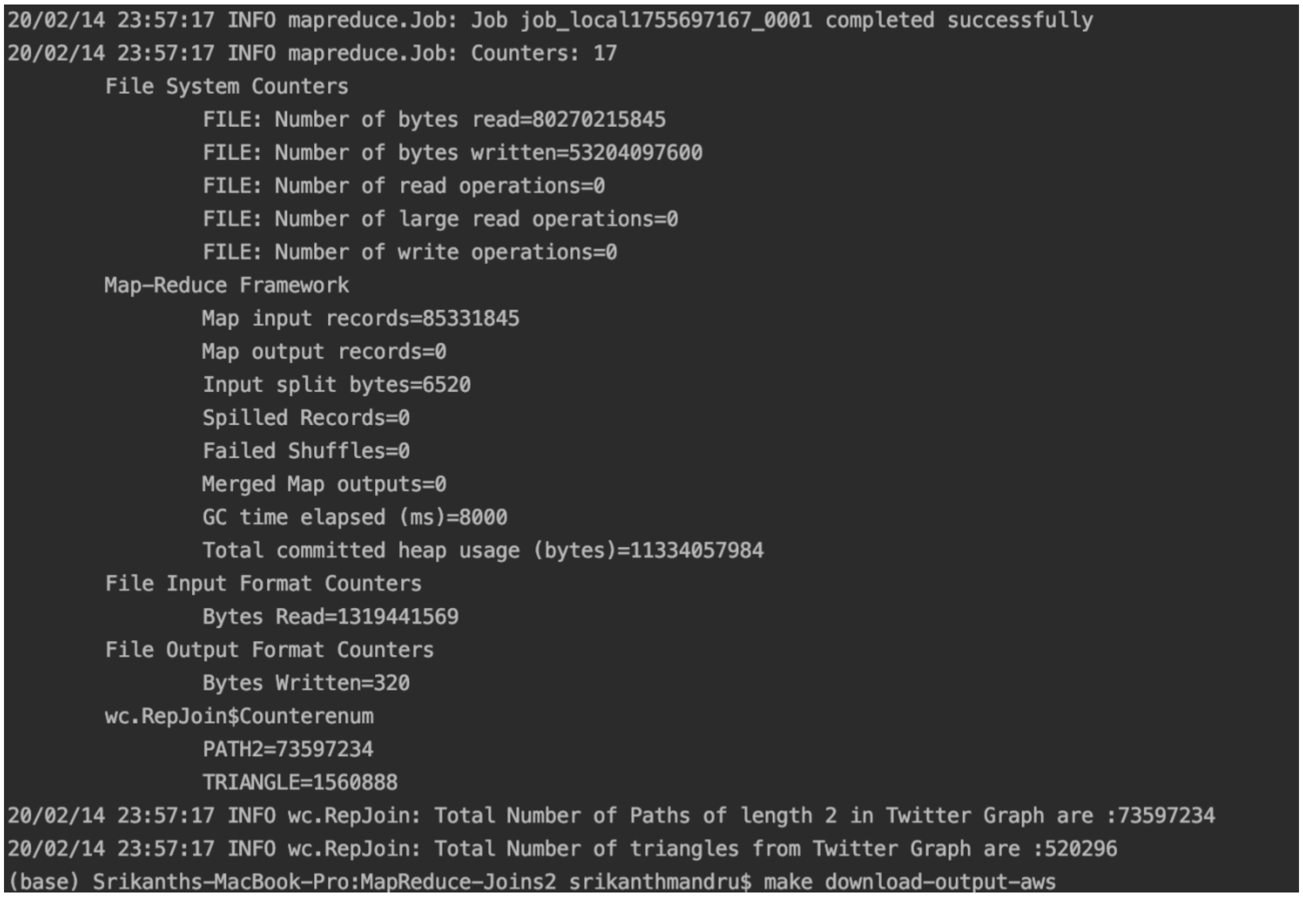
Pseudo-code:

```
Class Mapper {
    // Index H maps a join attribute value to all “edges” with user2 value

    Initialize hashIndex H

    method setup() {
    // Load data set “edges” from the distributed file cache into H, indexing on join attribute “user 2”.

        H = new hashMap
        Line ← Read each line from input file
        user_ids[] ← Split “line” delimited by “,”
        if (user_ids[0] < max  & user_ids[1] < max )
            users_set ← H.get(user_ids[0])
            H.put(user_ids[0], users_set)
    }

    method map(key, value) {
    // join the edges with hashIndex “H” two times. While joining recursively, check if first and last
    //users are same and if same, Increment the number of “TRIANGLES”

    user_ids[] = split “value” delimited by “,”
    if (user_ids[0] < max & user_ids[1] < max ) {
        user2_set ←  H.get(user_ids[1])
        for each user2 ∈ user2_set:
            Increment (PATH2_counter) by 1
            user3_set ← H.get(user2)
            for each user3 ∈ user3_set:
                if (user3 == user_ids[0])
                    Increment (TRIANGLES_counter) by 1
    }
    method cleanup () { clean up H }
    }
```

Output:
Output can be inferred from the syslog files provided in the "log files path" directory mentioned below:

Configuration	**Small Cluster Result**	**Large Cluster Result**
RS-join, MAX = 10000	Running time: 22 minutes, Triangle count: 520296, PATH2: 73597234	Running time: 18 minutes, Triangle count: 520296, PATH2: 73597234
Rep-join, MAX = 10000 (local) **(look at Graph of analysis part to know the results obtained)**	Running time: 40 minutes, Triangle count: 520296, PATH2: 73597234	Running time: 40minutes, Triangle count: 520296, PATH2: 73597234

For Replicated Join, I ran the job on local machine and I am providing the results correspondingly in above table. I obtained the correct results on local machine. Below is the screenshot of the log statements after executing the job locally:


```
20/02/14 23:57:17 INFO mapreduce.Job: Job job_local1755697167_0001 completed successfully
20/02/14 23:57:17 INFO mapreduce.Job: Counters: 17
        File System Counters
                FILE: Number of bytes read=80270215845
                FILE: Number of bytes written=53204097600
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
        Map-Reduce Framework
                Map input records=85331845
                Map output records=0
                Input split bytes=6520
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=8000
                Total committed heap usage (bytes)=11334057984
        File Input Format Counters
                Bytes Read=1319441569
        File Output Format Counters
                Bytes Written=320
        wc.RepJoin$Counterenum
                PATH2=73597234
                TRIANGLE=1560888
20/02/14 23:57:17 INFO wc.RepJoin: Total Number of Paths of length 2 in Twitter Graph are :73597234
20/02/14 23:57:17 INFO wc.RepJoin: Total Number of triangles from Twitter Graph are :520296
(base) Srikanths-MacBook-Pro:MapReduce-Joins2 srikanthmandru$ make download-output-aws
```

For aws run, I am getting the error as follows and have tried a lot figure out to error. I have provided the syslog from aws run for replicated join in "HW2/logs/Replicated Join/" directory.

Log files path:
I have provided log files in the following directory separately for “Reduce side Join” and “Replicated join”.

HW2/logs/

Output files path:

HW2/output/   ; where ‘#’ represents the run number

Report path:
HW2/Srikanth_Mandru_HW2.pdf