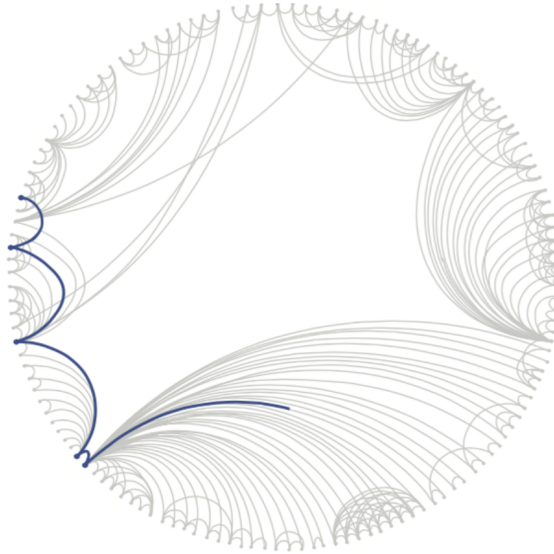


# Friend Recommendation Based on Hadoop MapReduce

Chao Chen, Linjing Wang, Youzhi Qu  
University of Southern California

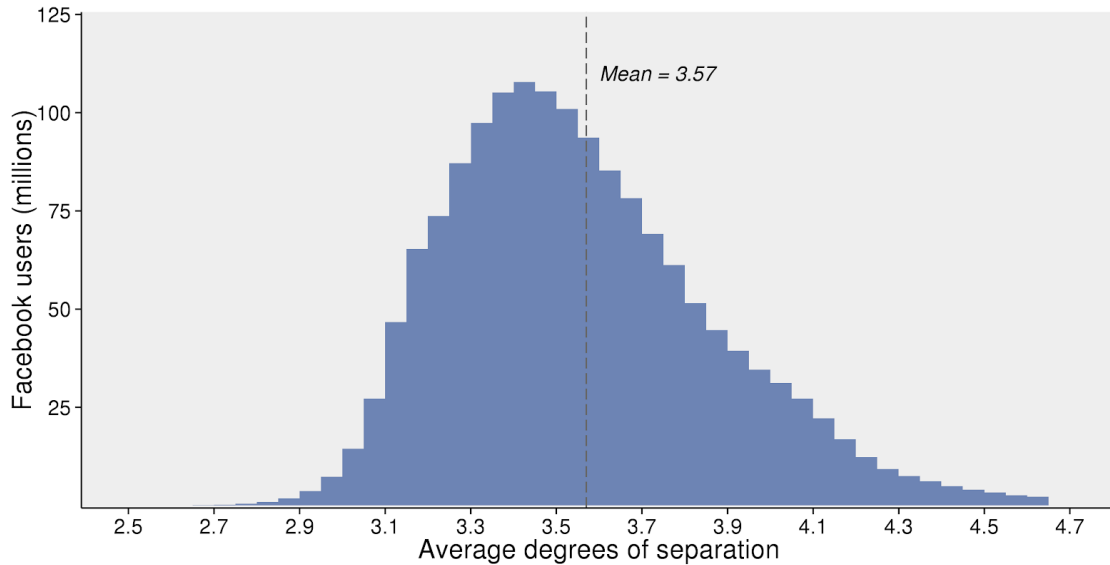
## 1. Introduction

As virtual communities, the main idea of social network is collaboration, creation and sharing. Unlike traditional media, social media are actually changing the way we interact with each other. Through a social network, people who even do not know each other can talk together, share their mood and ideas, make new friends. Indeed, the achievement of an individual is a product of the relationships he or she has. Therefore, leveraging what exists in the network can enhance the influence of an issue and improve the efficiency of problem solving.



*Figure 1.1*  
*Facebook friend graph*

According John Guare, all living individuals in the world are six or fewer steps away from each other. This idea is known as six degrees of separation. In February 2016, Facebook published the Facebook friend graph(Figure 2.1) and determined the average degrees of separation was 3.57(Figure 2.2). "Each person in the world (at least among the 1.59 billion people active on Facebook) is connected to every other person by an average of three and a half other people."



*Figure 1.2*

*Estimated average degree of separation between all people on Facebook.*

In summary, the world is more closely connected and it is really important to uncover the power of second and third degree connections in a network. Start by identifying the first-degree connection, which are people who are direct friends of an individual. One can only maintain a finite circle of friends. Next, there are some people who an individual might know, which means those friends of friends are second-degree connections.

With the popularity of the network, the data that servers need to process is getting bigger and bigger. The most straightforward approach for recommendation is to iterate all users. For each user, compare his/her friends to all other users for suggestion. This approach is feasible if applied asynchronously because users add friends at different time. However, If the data set that needs to be calculated at one time is too large, a more scalable approach is required.

Since the scalability of data, MapReduce of Apache Hadoop is becoming a very popular open-source framework. With a parallel, distributed algorithm(Figure 2.3) on a cluster, MapReduce can implement for processing huge amount of data in short, unlike the traditional data processing system.

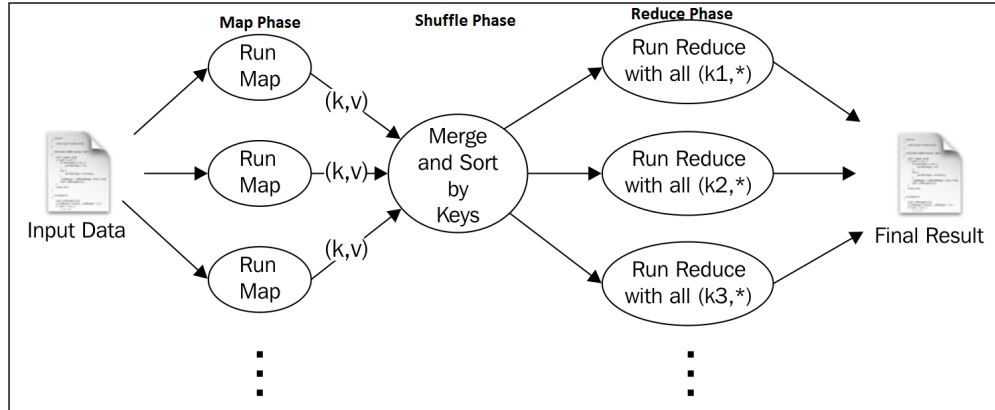


Figure 1.3

In final consideration, according the scalability of social media, such as Twitter, Facebook, LinkedIn and Instagram, this report will discuss a very popular use of big data processing — friend recommendation based on MapReduce.

## 2. Scope of Report

### *Specific problem*

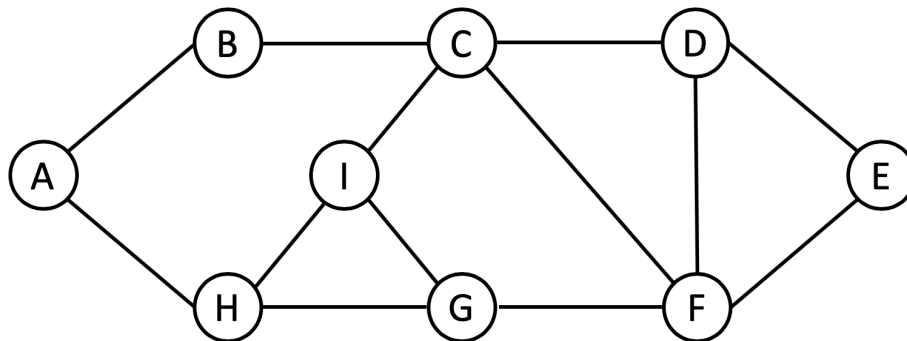


Figure 2.1

The association between users in a social networking site can be abstracted as a single undirected graph. Let's take the figure 3.1 as an example to introduce how to recommend friends to user . As we can see in the picture, connected vertices represent users who know each other. Suppose we need to recommend friends to users I. user I is connected to C, H and G, then we know they are user I's friends. Among them, we can find user I's second-degree contacts who are friends of user I's friends like user A, B, D and F. As shown in Figure 3.1, user F and user I have two common friends C and G. User A, B and D only have one common friend with user I. Therefore, user F has a higher recommendation priority than user A, B and D.

According to the above analysis, we use second-degree connections of user I as his recommended friends, and we vote for each possible friend of user I to select the best recommendation. In fact, the result of the second connection can only get the relationship chain

of a certain user on the social networking site, and those second-degree connections based on the voting election is users required in the friend recommendation function.

### **3. Research Methodology**

#### ***MapReduce***

MapReduce is a distributed computing model proposed by Google. It is mainly used in the search field to solve the calculation problem of massive data. MapReduce has two phases: Map and Reduce. Users only need to implement mapper and reducer functions to implement distributed computing. Mapper is responsible for breaking up complex tasks into several simple tasks. The simple task has three meanings: First, the scale of data or calculation is much smaller than the original task; the second, the principle of near calculation, that is, the task will be allocated to the node where the required data is stored; the third is that these small tasks can be calculated in parallel, with no dependence among all tasks. Reducer is responsible for summarizing the results of the map phase.

Hadoop is an open-source software framework written in Java. It consists of two parts, the storage part and the other part of the data processing part. The storage part is called Hadoop Distributed File System (HDFS), and the processing part is called MapReduce (Kaushik, 2015).

#### ***Advantages of MapReduce Programming*** (Kaushik, 2015)

- Scalability - Hadoop is a platform that is highly scalable
- Cost-effective solution - Hadoop's highly scalable structure also implies that it comes across as a very cost-effective solution for businesses that need to store ever growing data dictated by today's requirements
- Flexibility - Business organizations can make use of Hadoop MapReduce programming to have access to various new sources of data and also operate on different types of data, whether they are structured or unstructured.
- Fast - Hadoop uses a storage method known as distributed file system, which basically implements a mapping system to locate data in a cluster. The tools used for data processing, such as MapReduce programming, are also generally located in the very same servers, which allows for faster processing of data.
- Security and Authentication
- Parallel processing - dividing tasks that allows their execution in parallel
- Availability and resilient nature
- Simple model of programming

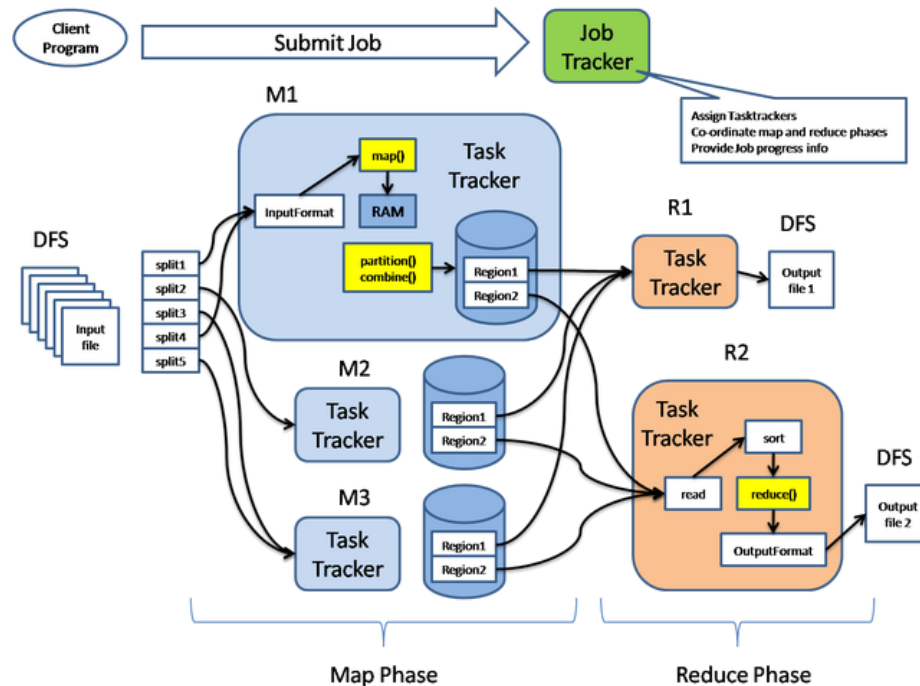


Figure 3.1

#### 4. Detailed Implementation

Taking Figure 3.1 as an example, the input file input01.txt (Fig. 5.1) saves information that the users are following each other. Each line has two user IDs, separated by commas, indicating that the two users are direct friends.



Figure 4.1  
input01.txt

The friend recommendation algorithm requires two different MapReduce. In the first round of MapReduce, for the first part (Map procedure), if the input is "H, I", then the output is key=H, value="H, I" and key=I, value="H, I". The former indicates that user I can find his

second-degree connection through user H, and the latter indicates that user H can find his second-degree connection through user I.

The input of the first round of Reduce process is, for example, key = I, value = {"H, I", "C, I", "G, I"}. In fact, the input of Reduce is all the people who pay attention to the users represented by Key. If the users H, C, and G are friends that are in contact with each other, the relationship between the users H, C, and G may be a second-degree friend. Specifically, corresponding to Figure 3.1, H and C are second-degree friends, G and C are second-degree friends, but G and H are not second-degree friends, because they are concerned about each other.

To summary, the first round of MapReduce processing is to output the pairs of friends who are following each other as first-degree friends ("deg1friend"), and all the possible friend pairs who may be second-degree friends as second-degree friends ("deg2friend").

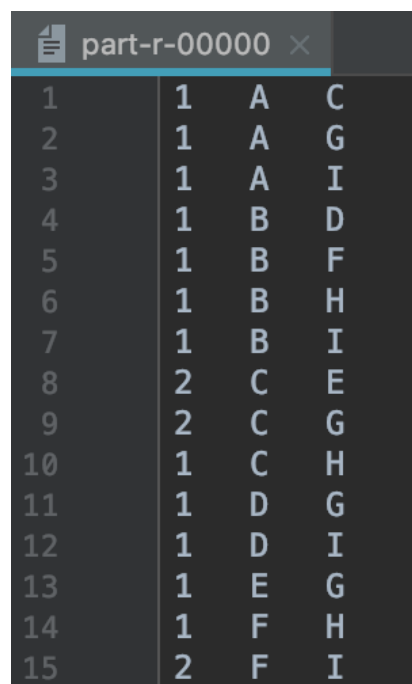
part-r-00000						
1	A	H	deg1friend	26	D	F deg1friend
2	A	B	deg1friend	27	E	F deg1friend
3	B	H	deg2friend	28	C	F deg1friend
4	A	B	deg1friend	29	F	G deg1friend
5	B	C	deg1friend	30	D	E deg2friend
6	A	C	deg2friend	31	D	G deg2friend
7	B	C	deg1friend	32	E	G deg2friend
8	C	F	deg1friend	33	C	D deg2friend
9	C	D	deg1friend	34	C	E deg2friend
10	C	I	deg1friend	35	C	G deg2friend
11	B	F	deg2friend	36	F	G deg1friend
12	B	D	deg2friend	37	G	I deg1friend
13	B	I	deg2friend	38	G	H deg1friend
14	F	I	deg2friend	39	F	I deg2friend
15	D	F	deg2friend	40	F	H deg2friend
16	D	I	deg2friend	41	H	I deg2friend
17	D	F	deg1friend	42	H	I deg1friend
18	D	E	deg1friend	43	A	H deg1friend
19	C	D	deg1friend	44	G	H deg1friend
20	E	F	deg2friend	45	A	I deg2friend
21	C	F	deg2friend	46	A	G deg2friend
22	C	E	deg2friend	47	G	I deg2friend
23	D	E	deg1friend	48	H	I deg1friend
24	E	F	deg1friend	49	C	I deg1friend
25	D	F	deg2friend	50	G	I deg1friend
26	D	F	deg1friend	51	C	H deg2friend
				52	C	G deg2friend
				53	G	H deg2friend

Figure 4.2  
Output file 1 (after MapReduce 1)

The second round of MapReduce input needs to be based on the output of the first round of MapReduce. In the second round of MapReduce, for each friend pair, by determining whether

their tag is a direct friend ("deg1friend") or a second friend ("deg2friend"), the true connection of one friend pair can be confirmed. If they have the "deg1friend" tag, then it is impossible that they could be second-degree friends; if there are "deg2friend" tags, no "deg1friend" tag, then they are second-degree friends. In addition, in particular, the number of "deg2friend" tags that a friend pair has is the number of votes they may become second-degree friends, that is, the number of common friends they have.

For instance, according to the output file 1, the relationship between the user D and the user F is displayed as "deg2friend" on the 15th line, but as "deg1friend" on the 17th line. Therefore, user D and user F are direct friends. Another example, for the pair of user C and user E, there are two records in output file 1 as "deg2friend" on line 22 and line 34. Also, there is no "deg1friend" tag between them. Hence, user C and user E get 2 votes which indicate the possibility that they might know each other.

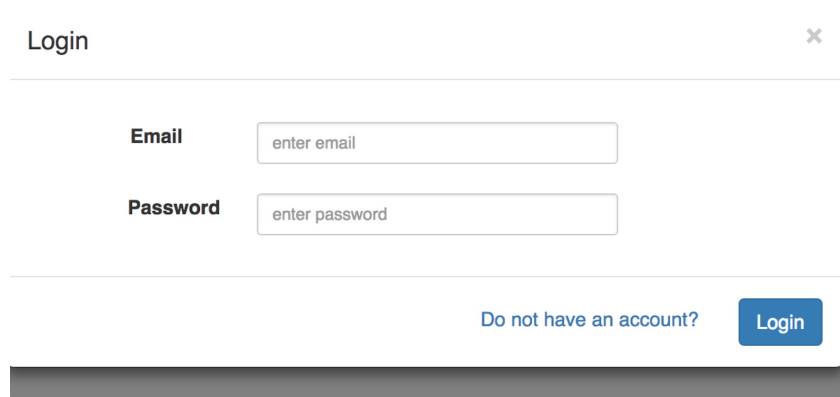


1	1	A	C
2	1	A	G
3	1	A	I
4	1	B	D
5	1	B	F
6	1	B	H
7	1	B	I
8	2	C	E
9	2	C	G
10	1	C	H
11	1	D	G
12	1	D	I
13	1	E	G
14	1	F	H
15	2	F	I

*Figure 4.3*  
*Output file 2(after MapReduce II)*

Also, the web UI of this system is implemented by using ReactJS, Bootstrap4, Html5, and Css3.

In Figure 4.4, it shows the login component. A user can login his account by typing email address and the associated password.

A login form titled "Login" with a close button (X) in the top right corner. It contains two input fields: "Email" with a placeholder "enter email" and "Password" with a placeholder "enter password". Below the password field, there is a link "Do not have an account?" and a blue "Login" button.

Login

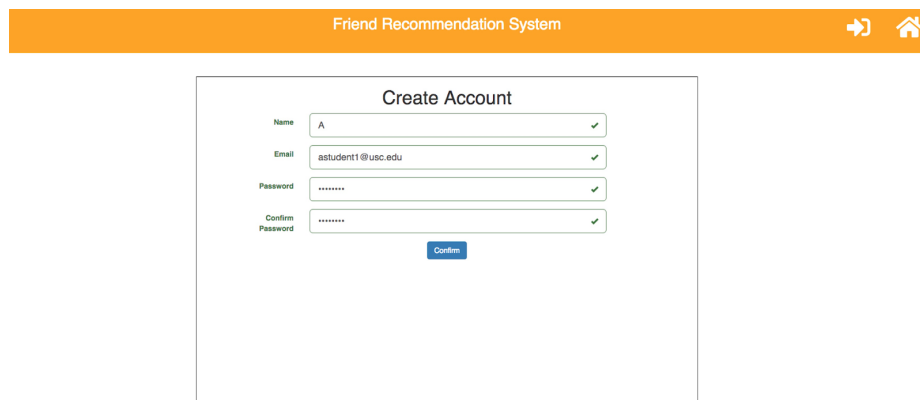
Email enter email

Password enter password

Do not have an account? Login

*Figure 4.4*

In Figure 4.5, it shows the Account Creation component which allows a user to register a new account. The user information will be save in the backend.

The interface for the "Friend Recommendation System" features an orange header bar with the system name and navigation icons. Below the header is a "Create Account" form. The form includes fields for Name (containing "A"), Email (containing "astudent1@usc.edu"), Password (masked with "\*\*\*\*\*"), and Confirm Password (masked with "\*\*\*\*\*"). Each field has a green checkmark icon to its right. A blue "Confirm" button is located at the bottom of the form.

Friend Recommendation System

Create Account

Name A ✓

Email astudent1@usc.edu ✓

Password \*\*\*\*\* ✓

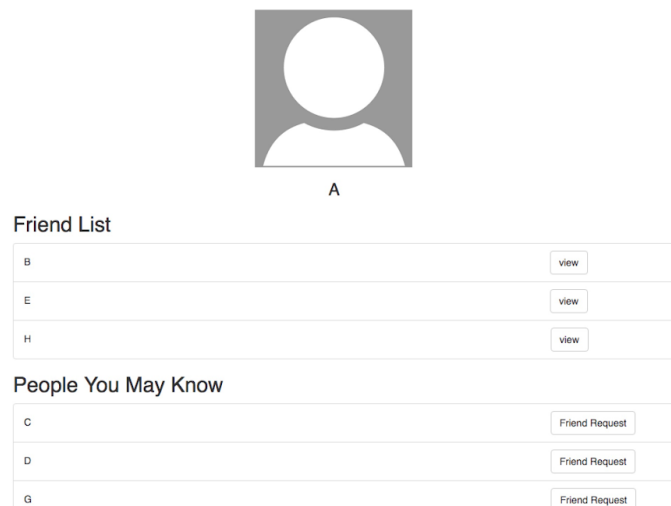
Confirm Password \*\*\*\*\* ✓

Confirm

*Figure 4.5*

Figure 4.6 shows the user profile page. The data of user's friends and recommended friends are fetched from backend (configuration file or database in the next version). User can view his friends' pages and send friend requests to the users in "people you may know" list.



*Figure 4.6*

## 5. Conclusion

### 5.1 Conclusion

MapReduce programming model has been widely used in various fields, because the model is easy to use and a lot of problems are suitable for calculating in this way. We have learned many things from this project. First, network bandwidth is a valuable resource. We need to reduce the amount of data transmission by reading data from local disks and storing intermediate data in local disk. Second, it is a good way to recommend friends to users by iterating all users asynchronously because people adding friends in different time. Third, the speed of different machines to process data can be different. Execution speed will be impacted by slow machine, machine failure and data loss. Redundant execution is a good way to reduce this influence.

### 5.2 Recommendation

Indeed, currently there are many implementations based on MapReduce idea such as Apache Hadoop or Spark, etc. Spark is also a popular open-source framework developed by Apache. Familiar with Hadoop, Spark can also process data in parallel across a cluster. The main difference is that Spark works in RAM and uses Resilient Distributed Datasets(RDDs). Spark solves some of the flaws and shortcomings of Hadoop.

- Based on RDD abstraction, Spark makes the code of data processing logic neatly.
- Hadoop only provides two operations, Map and Reduce. However, Spark provides multiple transformations and actions. Many basic operations such as Join, GroupBy have been implemented in RDD transformations and actions.

- When Spark works, a Job can contain multiple conversion operations of RDD. Multiple stages can be generated during scheduling, and if the partitions of RDD of multiple map operations are unchanged, they can process in the same Task.
- The intermediate result of spark is placed in memory. If the memory is full, it will be written to the local disk instead of HDFS.
- Those conversion in the same partitions constitute the pipeline running in a Task. The conversions of the different partitions require Shuffle, which are divided into different Stages. Each stage needs to wait for the previous Stage to complete before it can be processed.
- Spark improves the performance of iterative calculations by caching data in memory.

Hadoop MapReduce	Spark
Linear processing of Huge data sets.	Fast data processing
	Iterative processing
	Near real-time processing
Economical solution	Graph processing
	Machine learning
	Joining datasets

Although Hadoop Mapreduce and Spark are both big-data frameworks, they serve different purposes or projects. The fact is that they have a symbiotic relationship with each other. Also, they all have many features that each other deserves to learn. The trend of technology development is that Hadoop MapReduce will be replaced by a new generation of big data processing platform, and Spark is currently the most widely recognized and supported in the new generation of big data processing platforms. The perfect big-data scenario is that Hadoop and Spark can work together on the same team.

## 6. Author contributions

L.W. proposed the project topic - Friend Recommendation System. C.C. did a research on how to apply MapReduce to this problem by using Java on Linux and Mac. Y.Q. configured the Hadoop MapReduce, and L.W. and Y.Q. developed the first version of Friend Recommendation System by using Hadoop MapReduce. C.C. refined and modified the Mapper and Reducer functions' code in the second MapReduce phases and implemented the web UI. L.W. did a research on Apache Spark and compared it with Hadoop MapReduce. All authors discussed and prepared the final report.

## REFERENCE

- 1) Berryman, J. (2018). *Friend Recommendations using MapReduce*. [online] OpenSource Connections. Available at: <https://opensourceconnections.com/blog/2013/07/04/friend-recommendations-using-mapreduce/> [Accessed 12 Dec. 2018].
- 2) Bhagat, S., Burke, M., Diuk, C., Filiz, I. and Edunov, S. (2018). *Three and a half degrees of separation*. [online] Facebook Research. Available at: <https://research.fb.com/three-and-a-half-degrees-of-separation/> [Accessed 12 Dec. 2018].
- 3) Boss, J. (2018). *4 Ways To Uncover The Power Of 2nd And 3rd Degree Connections In Your Network*. [online] Forbes.com. Available at: <https://www.forbes.com/sites/jeffboss/2015/07/20/4-ways-to-uncover-the-power-of-2nd-and-3rd-degree-connections-in-your-network/#64cac5ba3b97> [Accessed 12 Dec. 2018].
- 4) En.wikipedia.org. (2018). MapReduce. [online] Available at: <https://en.wikipedia.org/wiki/MapReduce> [Accessed 12 Dec. 2018].
- 5) En.wikipedia.org. (2018). *Six degrees of separation*. [online] Available at: [https://en.wikipedia.org/wiki/Six\\_degrees\\_of\\_separation#cite\\_note-facebook2016-39](https://en.wikipedia.org/wiki/Six_degrees_of_separation#cite_note-facebook2016-39) [Accessed 12 Dec. 2018].
- 6) Fasale, A. (2013). *Friend Recommender In MapReduce*. [online] Bigobject.blogspot.com. Available at: <http://bigobject.blogspot.com/2013/08/friend-recommender-in-mapreduce.html> [Accessed 12 Dec. 2018].
- 7) Hess, K. (2018). *Hadoop vs. Spark: The New Age of Big Data*. [online] Datamation.com. Available at: <https://www.datamation.com/data-center/hadoop-vs.-spark-the-new-age-of-big-data.html> [Accessed 12 Dec. 2018].
- 8) Pal, K. (2015, October 5). *Advantages of Hadoop MapReduce Programming*. Retrieve From <https://www.tutorialspoint.com/articles/advantages-of-hadoop-mapreduce-programming>

## APPENDIX

### Critical code

```
public class deg2Friend {
    //map1
    public static class Map1 extends Mapper<Object, Text, Text, Text>
    {
        private Text map1_key = new Text();
        private Text map1_value = new Text();

        @Override
        protected void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {
            String[] eachterm = value.toString().split(",");
            if (eachterm.length != 2) {
                return;
            }

            if (eachterm[0].compareTo(eachterm[1]) < 0) {
                map1_value.set(eachterm[0] + "\t" + eachterm[1]);
            }
            else if (eachterm[0].compareTo(eachterm[1]) > 0) {
                map1_value.set(eachterm[1] + "\t" + eachterm[0]);
            }

            map1_key.set(eachterm[0]);
            context.write(map1_key, map1_value);

            map1_key.set(eachterm[1]);
            context.write(map1_key, map1_value);
        }
    }
    //reduce1
    public static class Reduce1 extends Reducer<Text, Text, Text, Text>
    {
        @Override
        protected void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {
            Vector<String> hisFriends = new Vector<String>();

            for(Text val : values)
            {
                String[] eachterm = val.toString().split("\t");
                if (eachterm[0].equals(key.toString())) {
                    hisFriends.add(eachterm[1]);
                    context.write(val, new Text("deg1 friend"));
                }
            }
        }
    }
}
```

```

        if (eachterm[1].equals(key.toString())) {
            hisFriends.add(eachterm[0]);
            context.write(val, new Text("deg1 friend"));
        }
    }

    for(int i = 0; i < hisFriends.size(); i++)
    {
        for(int j = 0; j < hisFriends.size(); j++)
        {
            if (hisFriends.elementAt(i).compareTo(hisFriends.elementAt(j)) < 0) {
                Text reduce_key = new Text(hisFriends.elementAt(i)+"\t"+hisFriends.elementAt(j));
                context.write(reduce_key, new Text("deg2 friend"));
            }
        }
    }
}
}
}

```

//map2

```
public static class Map2 extends Mapper<Object, Text, Text, Text>
```

```
{
```

```
    @Override
```

```
    protected void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {
```

```
        String[] line = value.toString().split("\t");
```

```
        if (line.length == 3) {
```

```
            Text map2_key = new Text(line[0]+" \t"+line[1]);
```

```
            Text map2_value = new Text(line[2]);
```

```
            context.write(map2_key, map2_value);
```

```
        }
```

```
    }
```

```
}
```

//reduce2

```
public static class Reduce2 extends Reducer<Text, Text, Text, Text>
```

```
{
```

```
    @Override
```

```
    protected void reduce(Text key, Iterable<Text> values, Context context)
        throws IOException, InterruptedException {
```

```
        boolean isdeg1 = false;
```

```
        boolean isdeg2 = false;
```

```
        int count = 0;
```

```
        for(Text val : values)
```

```
        {
```

```
            if (val.toString().compareTo("deg1 friend") == 0) {
```

```

        isdeg1 = true;
    }
    if (val.toString().compareTo("deg2friend") == 0) {
        isdeg2 = true;
        count++;
    }
}

if ((!isdeg1) && isdeg2) {
    context.write(new Text(String.valueOf(count)),key);
}
}
}
//main
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf,args).getRemainingArgs();
    if (otherArgs.length != 3) {
        System.err.println("Usage: Deg2friend <in> <temp> <out>");
        System.exit(2);
    }
    Job job1 = new Job(conf, "Deg2friend");
    job1.setJarByClass(deg2Friend.class);
    job1.setMapperClass(Map1.class);
    job1.setReducerClass(Reduce1.class);
    job1.setOutputKeyClass(Text.class);
    job1.setOutputValueClass(Text.class);

    FileInputFormat.addInputPath(job1, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job1, new Path(otherArgs[1]));

    if (job1.waitForCompletion(true)) {
        Job job2 = new Job(conf, "Deg2friend");
        job2.setJarByClass(deg2Friend.class);
        job2.setMapperClass(Map2.class);
        job2.setReducerClass(Reduce2.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job2, new Path(otherArgs[1]));
        FileOutputFormat.setOutputPath(job2, new Path(otherArgs[2]));
        System.exit(job2.waitForCompletion(true)? 0 : 1);
    }
    System.exit(job1.waitForCompletion(true)? 0 : 1);
}
}
}

```

## Console log

```
Linjing:2.8.5 linjingwang$ hadoop jar /Users/linjingwang/Documents/2018F596/FinalProj/deg2Friend.jar
deg2Friend.deg2Friend ~/deg2friend/input01 ~/deg2friend/temp ~/deg2friend/output01
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.hadoop.security.authentication.util.KerberosUtil
(file:/usr/local/Cellar/hadoop/2.8.5/share/hadoop/common/lib/hadoop-auth-2.8.5.jar) to method
sun.security.krb5.Config.getInstance()
WARNING: Please consider reporting this to the maintainers of
org.apache.hadoop.security.authentication.util.KerberosUtil
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
18/11/29 20:18:32 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using
builtin-java classes where applicable
18/11/29 20:18:32 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
18/11/29 20:18:34 INFO input.FileInputFormat: Total input files to process : 10
18/11/29 20:18:34 INFO mapreduce.JobSubmitter: number of splits:10
18/11/29 20:18:34 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1543551291773_0001
18/11/29 20:18:35 INFO impl.YarnClientImpl: Submitted application application_1543551291773_0001
18/11/29 20:18:35 INFO mapreduce.Job: The url to track the job:
http://Linjing.local:8088/proxy/application_1543551291773_0001/
18/11/29 20:18:35 INFO mapreduce.Job: Running job: job_1543551291773_0001
18/11/29 20:18:43 INFO mapreduce.Job: Job job_1543551291773_0001 running in uber mode : false
18/11/29 20:18:43 INFO mapreduce.Job: map 0% reduce 0%
18/11/29 20:18:58 INFO mapreduce.Job: map 60% reduce 0%
18/11/29 20:19:05 INFO mapreduce.Job: map 70% reduce 0%
18/11/29 20:19:06 INFO mapreduce.Job: map 100% reduce 0%
18/11/29 20:19:07 INFO mapreduce.Job: map 100% reduce 100%
18/11/29 20:19:07 INFO mapreduce.Job: Job job_1543551291773_0001 completed successfully
18/11/29 20:19:07 INFO mapreduce.Job: Counters: 49
    File System Counters
        FILE: Number of bytes read=13465
        FILE: Number of bytes written=1762300
        FILE: Number of read operations=0
        FILE: Number of large read operations=0
        FILE: Number of write operations=0
        HDFS: Number of bytes read=30120
        HDFS: Number of bytes written=13114
        HDFS: Number of read operations=33
        HDFS: Number of large read operations=0
        HDFS: Number of write operations=2
    Job Counters
        Launched map tasks=10
        Launched reduce tasks=1
        Data-local map tasks=10
        Total time spent by all maps in occupied slots (ms)=104552
        Total time spent by all reduces in occupied slots (ms)=6932
        Total time spent by all map tasks (ms)=104552
```

Total time spent by all reduce tasks (ms)=6932  
Total vcore-milliseconds taken by all map tasks=104552  
Total vcore-milliseconds taken by all reduce tasks=6932  
Total megabyte-milliseconds taken by all map tasks=107061248  
Total megabyte-milliseconds taken by all reduce tasks=7098368

#### Map-Reduce Framework

Map input records=834  
Map output records=146  
Map output bytes=13167  
Map output materialized bytes=13519  
Input split bytes=1386  
Combine input records=0  
Combine output records=0  
Reduce input groups=79  
Reduce shuffle bytes=13519  
Reduce input records=146  
Reduce output records=193  
Spilled Records=292  
Shuffled Maps =10  
Failed Shuffles=0  
Merged Map outputs=10  
GC time elapsed (ms)=187  
CPU time spent (ms)=0  
Physical memory (bytes) snapshot=0  
Virtual memory (bytes) snapshot=0  
Total committed heap usage (bytes)=2306867200

#### Shuffle Errors

BAD\_ID=0  
CONNECTION=0  
IO\_ERROR=0  
WRONG\_LENGTH=0  
WRONG\_MAP=0  
WRONG\_REDUCE=0

#### File Input Format Counters

Bytes Read=28734

#### File Output Format Counters

Bytes Written=13114

18/11/29 20:19:07 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032  
18/11/29 20:19:07 INFO input.FileInputFormat: Total input files to process : 1  
18/11/29 20:19:08 INFO mapreduce.JobSubmitter: number of splits:1  
18/11/29 20:19:08 INFO mapreduce.JobSubmitter: Submitting tokens for job: job\_1543551291773\_0002  
18/11/29 20:19:08 INFO impl.YarnClientImpl: Submitted application application\_1543551291773\_0002  
18/11/29 20:19:08 INFO mapreduce.Job: The url to track the job:  
[http://Linjing.local:8088/proxy/application\\_1543551291773\\_0002/](http://Linjing.local:8088/proxy/application_1543551291773_0002/)  
18/11/29 20:19:08 INFO mapreduce.Job: Running job: job\_1543551291773\_0002  
18/11/29 20:19:19 INFO mapreduce.Job: Job job\_1543551291773\_0002 running in uber mode : false  
18/11/29 20:19:19 INFO mapreduce.Job: map 0% reduce 0%  
18/11/29 20:19:24 INFO mapreduce.Job: map 100% reduce 0%



18/11/29 20:19:29 INFO mapreduce.Job: map 100% reduce 100%  
18/11/29 20:19:30 INFO mapreduce.Job: Job job\_1543551291773\_0002 completed successfully  
18/11/29 20:19:30 INFO mapreduce.Job: Counters: 49

#### File System Counters

FILE: Number of bytes read=13506  
FILE: Number of bytes written=342477  
FILE: Number of read operations=0  
FILE: Number of large read operations=0  
FILE: Number of write operations=0  
HDFS: Number of bytes read=13247  
HDFS: Number of bytes written=723  
HDFS: Number of read operations=6  
HDFS: Number of large read operations=0  
HDFS: Number of write operations=2

#### Job Counters

Launched map tasks=1  
Launched reduce tasks=1  
Data-local map tasks=1  
Total time spent by all maps in occupied slots (ms)=2622  
Total time spent by all reduces in occupied slots (ms)=3019  
Total time spent by all map tasks (ms)=2622  
Total time spent by all reduce tasks (ms)=3019  
Total vcore-milliseconds taken by all map tasks=2622  
Total vcore-milliseconds taken by all reduce tasks=3019  
Total megabyte-milliseconds taken by all map tasks=2684928  
Total megabyte-milliseconds taken by all reduce tasks=3091456

#### Map-Reduce Framework

Map input records=193  
Map output records=193  
Map output bytes=13114  
Map output materialized bytes=13506  
Input split bytes=133  
Combine input records=0  
Combine output records=0  
Reduce input groups=72  
Reduce shuffle bytes=13506  
Reduce input records=193  
Reduce output records=21  
Spilled Records=386  
Shuffled Maps =1  
Failed Shuffles=0  
Merged Map outputs=1  
GC time elapsed (ms)=12  
CPU time spent (ms)=0  
Physical memory (bytes) snapshot=0  
Virtual memory (bytes) snapshot=0  
Total committed heap usage (bytes)=419430400

#### Shuffle Errors

BAD\_ID=0  
CONNECTION=0  
IO\_ERROR=0  
WRONG\_LENGTH=0  
WRONG\_MAP=0  
WRONG\_REDUCE=0

File Input Format Counters

Bytes Read=13114

File Output Format Counters

Bytes Written=723