Executing Social Network Tasks Using MapReduce On Hadoop

Anumita Srivastava*, Sonali Sharma*

*Department of Computer Science
University of Georgia
Athens, Georgia 30602
anumita@uga.edu
sss63549@uga.edu

***Abstract*—In this project, we will implement social networking tasks such as finding mutual friends and making friend suggestions based on the number of mutual friends two people have using Map reduce on Hadoop. We will implement a cluster based Hadoop system and find the variation in performance parameters based on the number of nodes in the cluster.**

I. Introduction

In todayś time where the entire world is hooked to various social networking platforms, it is obvious that the amount of data to be handled is enormous. On Facebook alone, 2.5 billion content items are shared shared every day. There are approximately 2.7 billion likes generated per day. 300 million photos are uploaded per day. This translates to more that 100 petabytes of disk space in one of Facebookś largest clusters. 105 terabytes of data scanned via Hive, Facebook's Hadoop query language, every 30 minutes 70,000 queries executed on these databases per day. More than 500 terabytes of new data ingested into their databases every single day.

With so much content being generated on a daily basis, there has to be a programming model and some implementation to process and generate large data sets in a parallel distributed way. MapReduce is a program model that handles distributed computing. It is a processing technique based on Java.

It has two main tasks namely, map and reduce. Map is given an input data set that it converts into a (key,value) pairs. The reduce task takes these output tuples from the map function and then combines them to form smaller tuples.

MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after

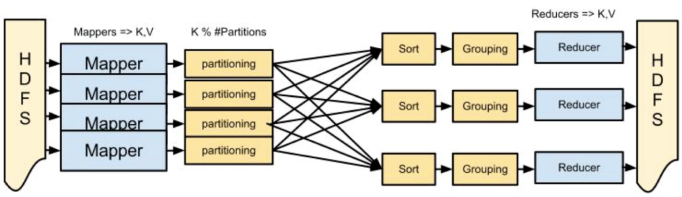

Figure 1. MapReduce Architecture

the map job.

A huge advantage of using Mapreduce is that it makes it easy to scale data processing on multiple nodes. The data processing primitives in this model are called mappers and reducers which might seem non trivial at first but when an application is written that has to be run over hundereds of thousands of machines in a cluster, all it requires is a configuration change. The scalability it provides is what makes the Mapreduce model so attractive. Figure1 shows a typical run of MapReduce on Hadoop.[Hadoop]

II. Problem Statement

To execute Social Network Tasks Using Map Reduce On Hadoop

III. Our approach

We have implemented two basic social networking tasks.

1) Finding Mutual Friends
2) Making friend suggestions

A. Finding Mutual Friends

The first part of the project is to find mutual friends amongst two people who are on Facebook. Friendships on Facebook are symmetric, i.e, if person 1 is friends with person 2, then automatically person 2 is friends with person1. But there are situations where the two people who are friends with each other might have mutual friends who they are not currently connected with.

1) Map Phase: So our program uses MapReduce to find mutual friends that two people will have. Consider the following example.
Consider 5 people 1,2,3,4,5. Their friends are stored as Person-[List of friends]. The friend list then looks like :
1 - 2 3 4
2 - 1 3 4 5
3 - 1 2 4 5
4 - 1 2 3 5
5 - 2 3 4
Every single line here would be taken as an input to the mapper. For each person, the mapper will give a [key,value] pair as the output.
Here key = person, friend and value = list of friends. The next step is to sort the keys so that we get a sorted order of friends. This is done to ensure that all pairs of friends will go to the same reducer. At the end of the map function, we get the following output:

1) For map(1 - 2 3 4) :
   (1 2) - 2 3 4
   (1 3) - 2 3 4
   (1 4) - 2 3 4

2) For map(2 - 1 3 4 5) :
   (1 2) - 1 3 4 5
   (2 3) - 1 3 4 5
   (2 4) - 1 3 4 5
   (2 5) - 1 3 4 5

This is done for all the input arguments. As discussed above, we then group the [key,value] pairs based on their keys. The output for persons A and B after grouping would be:

1) (1 2) - (1 3 4 5) (2 3 4)

2) (1 3) - (1 2 4 5) (2 3 4)

3) (1 4) - (1 2 3 5) (2 3 4)

4) (2 3) - (1 2 4 5) (1 3 4 5)

5) (2 4) - (1 2 3 5) (1 3 4 5)

6) (2 5) - (1 3 4 5) (2 3 4)

2) Reduce Phase: The reducer will intersect the lists of values and output the same key with the result of the intersection.
For example, ((1 2) - (1 3 4 5) (2 3 4)) taken as input to reducer, will intersect the list and provide the result of intersection as (1 2) : (3 4).
This means that 1, 2 have friends 3 and 4 as mutual friends. We can find mutual friends between 3,4 and 5 similarly.[Steve Krenzel]

B. Giving Friend Recommendations

The second part of the project is using MapReduce for making friend recommendations based on whether 2 people having common friends but not being friends on their own. An example of a friend network is shown in Figure2.[DB Tsai]

The input is an adjacency list that has the person's ID that is a unique identifier that every user has, and a list of his friends. The input looks like:
0 - 1 2 3
1 - 0 2 3 4 5
2 - 0 1 4
3 - 0 1 4
4 - 1 2 3
5 - 1 6
6 - 5
An example would be recommending user 4,5 to 0 as 0 is not friends with 4,5 but they have mutual friends 1,2 and 4; also 0 and 5 have a 1 as a mutual friend.

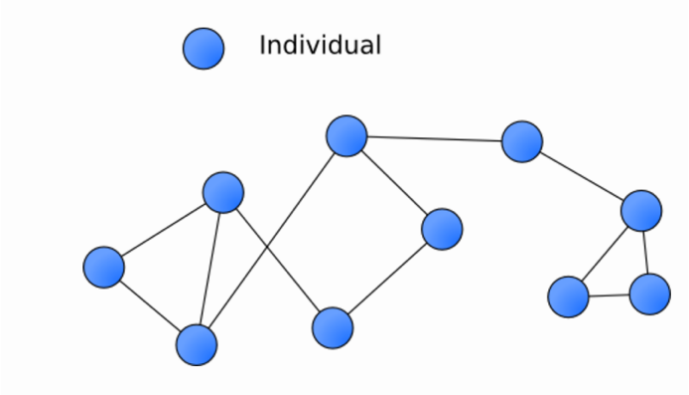

Figure 2. An example of a friend network on Facebook

1) Map Phase: Each line would be taken as an input to the mapper. For each person, the mapper will give a [key,value] pair as the output. For example, 0 has friends 1,2,3.
This would be turned to [1, 2], [2, 1], [2, 3], [3, 2], [1, 3] and [3, 1] having 0 as the mutual friend.
The [key,value] pair would be [1, r=2; m=0],[2, r=1; m=0],[2, r=3; m=0]...
where r = recommended friend and m = mutual friend. The following steps are executed in the map phase:

1) Emit [fromUser, r=toUser; m=-1] for all toUser.For n toUser commands, we emit n records

to describe the fromUser and toUser are friends already. Thus we set m=-1 because for the emitted key and r, a connection already exists.

2) Emit [toUser1, r=toUser2; m=fromUser] for all the possible combination of toUser1, and toUser2 from toUser with mutual friend as fromUser. This emits n(n-1) records.
3) A total of $n\hat{2}$ records are emitted after the map phase.

2) Reduce Phase: We just sum up how many mutual friends they have between the key, and r in the value in the reduce phase. If any of them has mutual friend value as -1, we don't make the recommendation since they are already friends.[DB Tsai]

3) Hashing: We have used Consistent hashing for MapReduce We make a dictionary of key,value pairs across a cluster of n nodes. This will be used to handle the input and output from the MapReduce job. We will use a naïve hashing method. Let's say that our cluster has n nodes. The nodes will be numbered from 0 ... n-1. We will store a key, value pair (k,v) on the node which matches the hashing function : hash(k) mod n. The distribution of keys will be uniform across the n nodes.

IV. Results

The input data has 50,000 users with each user having a unique identifier. A typical entry in the input data set looks like:
7 0,31993,40218,40433,1357,21843
This means that user with ID 7 is friends with users with IDs 0,31993,40218,40433,1357 and 21843. This data was taken from [Snap]. We evaluated our system based on two aspects:

1) Increasing the number of nodes on cluster
2) Cross Validation

A. Effect of increase in nodes in cluster on Run time

We executed both of the aforementioned tasks with initially 2 nodes in the cluster.
We found that the time required for execution of the program was around 28 seconds. We then increased the number of nodes in the cluster. Our observation was that the time required for executing the program was not affected by the increase in clusters much.

No. of Clusters	Task1	Task2
2	31s	31s
4	29s	30s
6	28s	28s

We observed that increasing the number of nodes in the cluster did not have very significant changes on the overall RunTime of the program. With two cluster nodes, we got the final output in around 32 seconds which was reduced to 28 seconds when we had 6 cluster nodes. This could be because the input data set has mere 50,000 entries. There would be a more pronounced change if the input data set was bigger.

B. Cross Validation

As discussed above, we used naive hashing to make a dictionary for the key,value pairs. The hash function we used was hash(k) mod n, where k

= key. In this case, key will be the unique identifier that every user has.

1) Task 1: Finding Mutual Friends: We used MapReduce to find mutual friends amongst two people. An example of the input would be :
18320 - 6,3238,9822,9841,9847,18279,18556,18581, 18587,19539,20525,20542,20553,24140
and
20553 - 6,3010,9463,18306,18320,18886,20456, 24173,40414,41782,417941801,42019,43213
The output is [18320 20553]:[6]

Another input would be:
9504 - 52,1245,1249,2793,4981,2204,9436, 12598,13643,18576,20529,22300,29840,30162, 30263,30269,30280,41784,41970,42967,43171, 43318,18327,18628,39174,44584
and
9822 - 52,102,171,2554,5099,5588,7699,9770 ,9788,9820,503,9850,13166,13654,13929,17998, 18320,18578,26369,35477,35698,35702,12667, 17034,18327,20604,40047,45181
The output is found out to be [9504, 9822] : [52, 18327]
This means that between Person 9504 and 9822, the mutual friends are Person 52, 18327.

We now repeat this procedure after first using naive hashing. We will use the hash function described above to direct the key to the node that it is assigned to after hashing. Once this is done, we will execute the MapReduce model on the [key,value] pair.

We find that we get the same result for the aforementioned input for keys 9504 and 9822. The result was same as above, [9504, 9822] : [52, 18327]. We did this for all the key,value pairs and found that each time, we got the same output after the reduce function.

2) Task 2: Making Friend Recommendations: We first run the MapReduce program to find the number of friends that can be recommended to a user based on his friends list.
Given the following input: 4 - 0,8,14,15,18,27,72,80,15326,19068,19079,24596, 42697,46126,74,77,33269,38792,38822.
14 - 0,4,19,19079,42697,444,42748

42697 - 4,14,2908,6152,6158,6902, 8768,11214,12333,13117,13156,17438,18523, 19032,19058,19079,19113,20441,23207,23297, 27888,29622,29884,42692,39169,42713,42718, 42719,42736,42742,46263,42717,42732,42748, 42750,42760

The output is :
4 2908(2: [0,42697]), 42748 (3: [0,14,42697])

We then repeated this procedure with hashing and checked the results for the same input. We found for this particular input from data set, the output was
4 2908(2: [0,42697]), 42748 (3: [0,14,42697])
The output remained the same for all the [key,value] pairs.

Thus we cross validated our results with comparing the output values for MapReduce where the key,value pairs were first passed through MapReduce without a hashing function and then with.

V. Conclusion

We implemented two Social networking tasks, i.e., finding mutual friends between two users and giving friend recommendations to users using MapReduce on Hadoop. We found that the due to the comparatively smaller size of input data set, there is not much difference in Run times when the number of nodes are increased in the cluster. There is a very small difference in the run time. We then checked our results after MapReduce by doing cross validation. We implemented a naive hashing technique and then compared the results after MapReduce with and without hashing. We found that we got the same output for both the techniques. This proves that the program works efficiently and gives consistent results for finding mutual friends and giving friend recommendations.

Acknowledgment

The authors would like to thank Dr. Lakshmish Ramaswamy (Professor, Department of Computer Science), for his support and encouragement.

References

[Steve Krenzel] MapReduce *http://stevekrenzel.com/finding-friends-with-mapreduce*

[Michael Nielsen] Consistent hashing, *http://michaelnielsen.org/blog/consistent-hashing/*

[DB Tsai] *https://www.dbtsai.com/assets/blog/2013/01/soc-LiveJournal1Adj.txt*

[DB Tsai] *Hadoop M/R to implement "People You Might Know" friendship recommendation*

[Hadoop] *http://hadoop.apache.org/*

[Facebook] *Under the Hood: Scheduling MapReduce jobs more efficiently*

[Snap] *https://snap.stanford.edu/data/egonets-Facebook.html* **The data set has been taken from Stanford's "Stanford Network Analysis Project"