Friend Recommendation System Using Map-Reduce and Spark: A Comparison Study

A.M. Abhishek Sai, Gottimukkala Sahil, Boddu Sasi Sai Nadh
Department of Computer Science and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri, India
amenu4aie20101@am.students.amrita.edu,
amenu4aie20128@am.students.amrita.edu,
amenu4aie20117@am.students.amrita.edu

Kalla Likhit Sai Eswar, Nisha K S
Department of Computer Science and Engineering
Amrita Vishwa Vidyapeetham
Amritapuri, India
amenu4aie20137@am.students.amrita.edu,
nishaks@am.amrita.edu

K.S. Reddy Banu Prakash, A.D. Mahesh
Department of Electronics and Communication Engineering
Amrita Vishwa Vidyapeetham
Amritapuri, India
amenu4ece20028@am.students.amrita.edu, amenu4ece20035@am.students.amrita.edu

2023 4th International Conference on Innovative Trends in Information Technology (ICITIIT) | 978-1-6654-9414-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICITIIT57246.2023.10068723

***Abstract*—**

Connecting with other people online is common practice and huge amounts of data are generated each day by increasing web activity. This type of data collection can be used to recommend friends on social media. We have utilized Map Reduce and Spark to analyze the vast amount of data. A Friend Recommendation system has been implemented using Map Reduce and Spark. Furthermore, we compared both Distributed Computation techniques in order to determine the optimum solution. We found that spark computation is 16 times faster than Hadoop Map-Reduce computation for Friend Recommendation System. Spark proves to be more efficient than map-reduce in terms of time efficiency.

***Index Terms*—Map-Reduce, Spark, Recommendation System, Hadoop**

I. INTRODUCTION

Social networking sites aim to increase their users' interaction by recommending them to like-minded users. On Facebook, they do this with a feature known as "People You May Know" [1]. Our primary goal is to link individuals who have mutual friends. Two people that share a lot of mutual friends and are indirectly connected will be recommended to each other by our recommender engine. Mathematically, if both nodes are two-edge distant from each other, we can figure out how many unique paths exist between the two nodes (with exactly one node in each direction). Based on the number of distinct paths, the results are ordered in descending order. This process can then be repeated in the same way for each additional node. Hence, each of us can compute the top connections using MapReduce. We employ an algorithm that accounts for the fact that there is a greater probability of a recommendation between two persons if they have more common friends. Upon completion, a person's output is a list of people ranked from highest to lowest priority.

The use of recommender systems increases click-through rates, raising revenue for organizations, hence, recommendation engines have positive effects on customer experiences that lead to higher satisfaction and retention [2]. Instead of having to wade through thousands of box sets and movie titles, Netflix provides a curated selection of items that are most likely to appeal to you. This feature provides a better user experience. Netflix was able to achieve lower cancellation rates as a result of this, saving the corporation roughly a billion dollars per year [3]. Amazon recommendation method was first used approximately 20 years ago, and it is now being used in various industries such as finance and tourism.

II. RELATED WORKS

In the present times, many studies have shown different types of recommendation systems that are been developed. Chen et al gave an idea of a people recommendation system within a commercial social network. They displayed four

978-1-6654-9414-4/23/$31.00 ©2023 IEEE

algorithms, two of which are based on content similarities and two of which are based on social relationship data. While algorithms based on content similarity are better at making new friends, algorithms based on social relationship data are better at identifying related contacts. This method takes into account both interactions with the information and contact with other users to produce effective recommendations [4].

Using the Hadoop Framework, Jai Prakash Verma et al proposed a recommendation system to recommend movies using the vast quantity of data that can be found on the internet in the form of opinions, comments, ratings, and reviews on any item [5]. Lakshay Nagpal et al proposed the use of the Hadoop framework to develop enhanced MapReduce-based data processing. They have analyzed a large amount of data in a very short period of time by using an open source technology offered by Apache Software Foundations [6].

According to shared interests, connections between users, and other variables, Gupta et al proposed Twitter user suggestions. According to the authors, Twitter is represented as a graph, with users serving as the vertices and directed edges showing connections between users. Using the Stochastic method of approach for link-structure analysis, the system makes suggestions. The approach was built on Hadoop, a highly scalable technology, the authors also address scalability difficulties [7]. Guy et al, presented a people recommendation system for the IBM Fringe social network. This solution takes into account enterprise data which includes co-membership project data, organizational chart relationships, and also paper and patent co-authorship [8].

III. Map Reduce vs Spark

A programming tool called Hadoop MapReduce handles large data sets in a distributed, parallel manner. Developers do not have to take fault tolerance or job distribution into consideration when designing the massively parallelized types of operators. MapReduce faces a bit of complication with the sequential multi-step process that is basically required to complete a task. After each step, MapReduce writes the processed data back to HDFS after retrieving it from the cluster and processing it. Disc I/O delays making the MapReduce tasks slow, as in each step it basically requires a disc read and write [9]. The scheduling of several jobs on the Hadoop cluster will be handled by the job tracker. It is a multi-threaded program. [10].

MapReduce is a programming paradigm used in Hadoop and represents an execution engine, a distributed file system for storage, and a resource management system, YARN, to manage shared resources [11] [12]. It is typical for Hadoop systems to use several execution engines, such as Presto, Spark, and Tez.

Spark is a scalable, quick, and adaptable data processing engine that leverages big data and pre-existing data platforms. The foundation of Spark is the idea of a resilient distributed dataset (RDD), which is a group of independent components that may be processed in parallel to speed up reading and writing tasks [13].

An open-source option for the distributed method of processing huge data sets is Apache Spark [14]. For quick analytical type queries on data of any size, it combines in-memory caching and quick query execution. It offers development APIs in Scala, Java, R, and Python, and enables the feature of code reusability across a range of workloads, including interactive queries, batch processing features, machine learning, real-time analytics, and graph processing [15] [16]. Some of the companies that use it include Zillow, FINRA, DataXu, Yelp, the Urban Institute, and CrowdStrike. Apache Spark, which has 365,000 meetup members in 2017, has become a well-known framework for processing large amounts of data distribution [17].

The major reason for the creation of Spark is to alleviate the drawbacks of MapReduce by performing the method of processing in memory, minimizing the number of job stages, and also to reuse the data across numerous concurrent processes. As Spark executes operations, reads input into memory, and outputs the results in one step, it drastically speeds up execution. An in-memory cache is used by Spark in order to significantly speed up machine learning algorithms that run the same function on the same dataset frequently [18] [19]. In Spark, DataFrames, an abstraction over Resilient Distributed Datasets (RDD), is used to re-use data when combined with Resilient Distributed Datasets (RDD) [20]. In particular, Spark is significantly faster than MapReduce for machine learning and interactive analytics due to a large reduction in latency.

IV. Algorithm

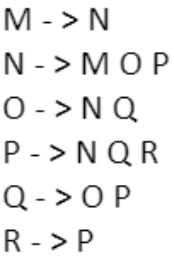

Fig. 1. Input

Let us consider a sample input as in Fig. 1. The first letter of every line indicates the Source node (user) and the rest indicates their direct friends.

A. Mapping Phase

We emit a key-value pair with the values "ID of the real node, Concatenation of "-1" and a special character (dollar character), the ID of the neighbour node" and its opposite for each pair of the real node and its neighbour. A direct link between the real node and its neighbours is indicated by the number "-1."

(M, -1$ N)	(O , 1$ P)	(N , 1$ Q)	(N , 1$ R)	(Q , -1$ P
(N, -1$ M)	(M , 1$ P)	(Q , 1$ N)	(Q , 1$ N)	(O , 1$ P)
(N , -1$ O)	(P , 1$ M)	(P , -1$ N)	(Q , 1$ R)	(P , 1$ O)
(N, -1$ P)	(P , 1$ O)	(P , -1$ Q)	(R , 1$ Q)	(R, -1$ P)
(M, 1$ O)	(O , -1$ N)	(P , -1$ R)	(R , 1$ N)	
(O , 1$ M)	(O , -1$ Q)	(N, 1$ Q)	(Q , -1$ O)	

Fig. 2. Mapping

We emit a key-value pair with the values "ID of the real node, Concatenation of "1" and a special character (dollar character), the ID of the neighbour node" and its opposite for each pair of two distinct neighbours. The reference number "1" is used to denote that neighbours can become friends because they reside on the same node.

B. Shuffle and Sort Phase

M ->	N ->	O ->	P ->	Q ->	R ->
-1$N	-1$M	1$M	1$M	1$N	1$N
1$O	-1$O	1$N	-1$N	1$N	-1$P
1$P	-1$P	-1$P	1$O	-1$O	1$Q
	1$Q	-1$P	1$O	-1$P	
	1$Q	1$Q	-1$Q	1$R	
	1$R		-1$R		

Fig. 3. Shuffle and Sort

The shuffle and sort phase is followed by the map phase. During this step, Hadoop sorts each key internally. The values for each key are then grouped and passed to the same reducer. In this case, we may be able to suggest a suitable recommendation for that key based on a list of values matching its matching key, since both values are sent to the same reducer.

C. Reduce Phase

M -> O(1) , P(1)
N -> Q(2) , R(1)
O -> P(2) , M(1)
P -> O(2) , M(1)
Q -> N(2) , R(1)
R -> N(1) , Q(1)

Fig. 4. Reducing

The reduction phase is followed by the shuffle and sort phase. Now we have a key and a set of values. The node id and the corresponding "1" or "-1" are taken from the list of values. "1" indicates the extracted node and key had a mutual friend, while "- 1" indicates they are already connected in the network graph. Then, add up how many friends the key and all extracted nodes have in common. If a pair of them has a "-1" relationship with one another, exclude that pair from our calculations because "-1" denotes a direct connection, indicating they are already friends.

After aggregation, finally rank the nodes according to their aggregated sum in descending order. It shows our best recommendations of friends for that node (or key). We might be able to improve our current system and reduce the amount of data that needs to be transferred over the network by adding combiners to perform some aggregation before the shuffle and sort phase.

V. IMPLEMENTATION

A. Map Reduce

Using two Mappers and two Reducers, we simultaneously run two jobs, with the output of the first job serving as the input for the second.

Concatenation of -1 for direct friends and 1 for mutual friends is done in mapper class. The next step is to tokenize each line after converting it to a string when the input is given. Then insert the tokens in an array after iterating through them. The source node (primary user) would be the first element in the array, followed by the primary user's friends. So first combine primary user and friend with -1. Then create all possible combinations and append 1 to the friend list.

We repeatedly iterate through the output of mapper 1, which would take the form AB1, BC-1, and so on, in the first reducer, where A, B, and C are users. The next step is to disregard outputs with a value of -1 and only take into account outputs with a value of 1. Calculate the repeated number of friend combinations and output it. The result of the prior map-reduce process, which would take the form of (AB 1, CD 2) and so on, is now sent into the second mapper, where A, B, C, D are the users and 1,2 represents a number of mutual friends between them.

Now, map this input once more into key-value pairs. Repeat the process with the new input, mapping the primary user (source node) with multiple counts to the other friend (neighbour). It would take the form of (A, 1B) (C, 2D).

In the Second Reducer, take the input from the second mapper and now sort it in the form from highest to lowest priority of mutual friends. For example, if A is having 5 mutual friends with B, 10 mutual friends with C, and one mutual friend with D, then the order should be of form C, B, and D.

B. Spark

First, initialize the context and spark configuration. Then, divide each line of the text file into ”words” by any white space. Later, each line must be mapped to the form: **(user_id, [friend_id_0, friend_id_1, ...]).**

Map each ”friend ownership” to multiple instances of ((user_id, friend_id), VALUE).

VALUE = 0 indicates that user_id and friend_id are already friends.

VALUE = 1 indicates that user_id and friend_id are not friends.

For example, the value “(0, [1, 2, 3])“ will get mapped to:

[((0,1), 0), ((0,2), 0), ((0,3), 0), ((1,2), 1), ((1,3), 1), ((2,3), 1),]

To find the number of common friends between two users, filter all user pairings that are already friends, then add up all the ”1” values. For example ((0, 1), 21)” encodes the fact that users 0 and 1 have 21 friends in common. Because of this, user 1 should be recommended to user 0, and vice versa. For each input, this method will return two recommendations in a List.

After creating the suggestion objects and grouping them by key, the recommendations should be sorted and reduced. A person with more and fewer mutual friends should receive the suggestion.

VI. EXPERIMENTATION

```
0 1 2 3
1 0 2 4 5 6
2 0 1 4 6 7 8
3 0 4 7
4 1 2 3 5 8
5 1 4
6 1 2
7 2 3
8 2 4
```

Fig. 5. Input Format

Fig. 5 is the sample input file. The first number/letter indicates the source node(user) and the next number/letter indicates their direct friends. The same pattern follows in every line. For example, in 0 1 2 3, 0 is the user, and 1,2,3 are its direct Friends.

```
04  3
05  1
06  2
07  2
08  1
13  2
17  1
18  2
23  3
25  2
31  2
32  3
35  1
38  1
40  3
46  2
47  2
```

```
50  1
52  2
53  1
56  1
58  1
60  2
64  2
65  1
67  1
68  1
70  2
71  1
74  2
76  1
78  1
80  1
81  2
83  1
85  1
86  1
87  1
```

Fig. 6. Output of Reducer1

Fig. 6 is the result after the First Map-Reducer operation. The first number represents the source node (user), the second represents the recommended friend, and the third is the total number of friends that they have in common. For example, 04 3, where 0 is the source node, 4 is the suggested friend, and 3 is the total number of mutual friends they both have.

```
0  ->  6  4  7  8  5
1  ->  3  8  7
2  ->  3  5
3  ->  2  1  5  8
4  ->  0  6  7
5  ->  2  8  6  3  0
6  ->  4  0  7  5  8
7  ->  4  0  1  6  8
8  ->  1  0  3  5  6  7
```

Fig. 7. Final Output of Map Reduce

Fig. 7 is the result after the Second Map-Reducer operation. The first column of every line indicates the user, while the remaining indicates recommended friends in order of high priority to low priority (i.e. from having more friends in common to having fewer friends in common). For example, in $1 -> 3\ 8\ 7$, 1 is the user and 3,8,7 are recommended mutual friends, with 1 and 3 having more mutual friends compared to other recommendations.

```
(1, [3, 8, 7])
(3, [2, 1, 5, 8])
(2, [3, 5])
(0, [4, 6, 7, 5, 8])
(4, [0, 6, 7])
(5, [2, 0, 3, 6, 8])
(6, [0, 4, 5, 7, 8])
(7, [0, 4, 1, 6, 8])
(8, [1, 0, 3, 5, 6, 7])
```

Fig. 8. Final output of spark

Fig. 8 is the result of the recommendation system using Spark Framework. The user is shown in the first column of every line, while the remaining columns list recommended

friends based on their mutual friends, from high priority to low priority (i.e. from having more friends in common to having fewer friends in common).

VII. Results

```
Launched map tasks=1
Launched reduce tasks=1
Data-local map tasks=1
Total time spent by all maps in occupied slots (ms)=18248
Total time spent by all reduces in occupied slots (ms)=19751
Total time spent by all map tasks (ms)=18248
Total time spent by all reduce tasks (ms)=19751
Total vcore-milliseconds taken by all map tasks=18248
Total vcore-milliseconds taken by all reduce tasks=19751
Total megabyte-milliseconds taken by all map tasks=18685952
Total megabyte-milliseconds taken by all reduce tasks=20225024
```

Fig. 9. Time Taken by Map-Reduce 1

Fig. 9 depicts the time taken by First Map Reduce Operation to Recommend Friends. The mapping function takes 18248 milliseconds whereas reduce function takes 19751 milliseconds. The total time taken to complete the first map-reduce function is 37999 milliseconds.

```
Launched map tasks=1
Launched reduce tasks=1
Data-local map tasks=1
Total time spent by all maps in occupied slots (ms)=16586
Total time spent by all reduces in occupied slots (ms)=16178
Total time spent by all map tasks (ms)=16586
Total time spent by all reduce tasks (ms)=16178
Total vcore-milliseconds taken by all map tasks=16586
Total vcore-milliseconds taken by all reduce tasks=16178
Total megabyte-milliseconds taken by all map tasks=16984064
Total megabyte-milliseconds taken by all reduce tasks=16566272
```

Fig. 10. Time Taken by Map-Reduce 2

Fig. 10 depicts the time taken by the second Map Reduce Operation to Recommend Friends. The mapping function takes 16586 milliseconds whereas reduce function takes 16178 milliseconds. The total time taken to complete the first map-reduce function is 32764 milli seconds.

```
Launched Tasks=1
Total time spent by all tasks in occupied slots (ms)=4400
Total time spent by all tasks (ms)=4400
Total vcore-milliseconds taken by all tasks=4400
```

Fig. 11. Time Taken by Spark

Fig. 11 depicts the time taken by spark operation to Recommend Friends. Time taken to complete the recommendation task is 4400 milliseconds.

Spark operation is 16 times faster than map-reduce operation in terms of total computation time, taking only 4400 milliseconds compared to 70,763 milliseconds for map-reduce operation. In terms of computation time, we can therefore conclude that spark is more effective than map-reduce.

VIII. Conclusion

In this study, we have successfully used Spark and Map-Reduce operations to construct a friend recommendation system. For the current recommender system to provide optimal friend recommendations, a huge quantity of data must be processed. The recommender system is developed in a distributed cluster setting using Hadoop since the input data set is large. We have worked on and made a recommendation system based on the idea of "mutual friends," and our analysis shows that spark is faster than map-reduce. The output was also sorted, and it ranks friends from high importance to low priority depending on their friendships with one another. When used on the same dataset, map-reduce and spark provide results that are identical.

References

[1] ""facebook"." [Online]. Available: https://www.facebook.com/

[2] "Saravanan, s. "design of large-scale content-based recommender system using hadoop mapreduce framework." 2015 eighth international conference on contemporary computing (ic3). ieee, 2015."

[3] "Syed, mr patrick robert cheron dr, and adeel ahmed. "profit optimization via process enhancement through ai controllers."."

[4] "Chen, jilin, et al. "make new friends, but keep the old: recommending people on social networking sites." proceedings of the sigchi conference on human factors in computing systems. 2009."

[5] "Verma, jai prakash, bankim patel, and atul patel. "big data analysis: recommendation system with hadoop framework." 2015 ieee international conference on computational intelligence communication technology. ieee, 2015."

[6] "Nagpal, lakshay, and nikhil khurana. "design of friend recommender system using apache hadoop." 2017 ieee international conference on intelligent techniques in control, optimization and signal processing (incos). ieee, 2017."

[7] "Gupta, pankaj, et al. "wtf: The who to follow service at twitter." proceedings of the 22nd international conference on world wide web. 2013."

[8] "Guy, ido, inbal ronen, and eric wilcox. "do you know? recommending people to invite into your social network." proceedings of the 14th international conference on intelligent user interfaces. 2009."

[9] "Dean, jeffrey, and sanjay ghemawat. "mapreduce: Simplified data processing on large clusters." (2004)."

[10] "A. m. abhishek sai, d. reddy, p. raghavendra, g. y. kiran and r. v. r, "producer-consumer problem using thread pool," 2022 3rd international conference for emerging technology (incet), 2022, pp. 1-5, doi: 10.1109/incet54531.2022.9824031."

[11] "Condie, tyson, et al. "mapreduce online." nsdi. vol. 10. no. 4. 2010."

[12] "Jayan, anandu, and bhargavi r. upadhyay. "rc4 in hadoop security using mapreduce." 2017 international conference on computational intelligence in data science (iccids). ieee, 2017."

[13] "Zaharia, matei, et al. "spark: Cluster computing with working sets." 2nd usenix workshop on hot topics in cloud computing (hotcloud 10). 2010."

[14] "Shyam, r., et al. "apache spark a big data analytics platform for smart grid." procedia technology 21 (2015): 171-178."

[15] "Vimalkumar, k., and n. radhika. "a big data framework for intrusion detection in smart grids using apache spark." 2017 international conference on advances in computing, communications and informatics (icacci). ieee, 2017."

[16] "Ku, abinesh kamal, and shiju sathyadevan. "intrusion detection system using big data framework." (2006)."

[17] "Salloum, salman, et al. "big data analytics on apache spark." international journal of data science and analytics 1.3 (2016): 145-164."

[18] "Shi, juwei, et al. "clash of the titans: Mapreduce vs. spark for large scale data analytics." proceedings of the vldb endowment 8.13 (2015): 2110-2121."

[19] “Gopalani, satish, and rohan arora. ”comparing apache spark and map reduce with performance analysis using k-means.” international journal of computer applications 113.1 (2015).”
[20] “Monica, velamuri, and narendra kumar parambalath. ”shuffle phase optimization in spark.” 2017 international conference on advances in computing, communications and informatics (icacci). ieee, 2017.”