Intermediate Project Report

Group 30

Team - Javesh Monga, Vivin Wilson, Virat Goradia

Git Repository - https://github.ccs.neu.edu/group-30/mr-project

Project Overview

As we are a team of 3 members, we are doing a 1 -person task as our first task which is to find the single source shortest path for a given edges dataset. Our Analysis goal here is to prove the six degree of separation experiment using Twitter follower dataset. The theory according to this wikipedia article suggests that any two users will be connected with 6 or less users between them. We were able to see this in practice by implementing single source shortest path algorithm where distances to any user from our source user was less than or equal to 6.

As the single source shortest path is based on Breadth First Search algorithm, each new MapReduce Job indicates that a new level/breadth in the user relations tree is being processed. Thus, according to the six degrees theory we must never get more than 7 MR jobs in our executions where the 7th job marks the convergence of our execution due to no change in the results from the 6th job. We were able to observe exactly this and thus could prove the six degree theory.

For our second task we will be implementing k means clustering in MapReduce which is a 2-person task and will be implemented in the next leg of our project.

Input Data

For the first task our input dataset is the Twitter edges data, which consists of relationships between two users where one is the follower of the other.

Represented as (a,b) meaning that a follows b.

Task 1 -Single Source Shortest Path (Spark) (1-person task)

Overview

Implemented single source shortest path algorithm in Spark, the algorithm is implemented from scratch without use of special graph libraries like GraphX. The optimization related to removing any vertex with infinite distance has been applied to get a more efficient solution.

Pseudo-code

```
 lines = read input from input text file
  graph = lines.map { s =>
     val parts = s.split(",")
     (parts[0], parts[1])
   }
   .filter(x => x[1] <= MAX && x[2] <= MAX)
    .distinct().groupByKey().cache()

   var distances = {set distance to source node to 0 and other nodes to infinity}
   var temp = distances

   while(true) {
         temp = graph.join(temp)
        .filter(x => x[2][2] != infinity)
        .flatMap(x => x[2][1]
          .map(y => (y, x[2][2] + 1)))

      distances1 = temp.union(distances).reduceByKey((x, y) => getMin(x, y))

      done = distances.join(distances1)
         .map { case (x, y) => y._1 == y._2 }
         .reduce((x, y) => x && y)


       if (!done) distances = distances1 else break
     }
save distances to file
```

Algorithm and Program Analysis

Algorithm Steps:
1. Read the input as RDD.
2. A. Split the input on “,” and group by key where key is our part(0) after split.
   B. Filter out the nodes which have values above the given threshold parameter.
   C. Remove duplicates if any(using distinct())
   D. Get the adjacency list of a node (using groupByKey)
   E. Cache this graph RDD created for reusing later.
3. Create the initial distance RDD with source having the distance of 0 and all other nodes with the distance as positive infinity.
4. A. Join the graph RDD and the distance RDD.
   B. FIlter out the nodes which have a distance of -1 (We do this as our optimization step since we will be joining this RDD with our distance RDD which already has all the nodes with distance of positive infinity. Thereby, we reduce the unnecessary checks and reduce our intermediate results formed).
   C. Add the edge distance(1 in our case) to the nodes discovered so far.
5. Union the current distances found with the initial distance and get the minimum distance for each node.
6. Check if the distances have been updated in any of the nodes, or whether more nodes have been added to current set, or both. If true, update the distances RDD with this current distance RDD and iterate through steps 4,5 again until the check at this step(viz. step 6) fails.
7. Save the distance RDD formed after all the iterations as our output. This gives us the shortest path from source node given to all target nodes the source can reach.

Experiments

Executed on Twitter Edges Dataset with Source vertex = 3 and changing total number of users via Max Threshold. This tells us how many MR Jobs are executed before convergence when looking from a fixed source vertex and a growing graph of users. The number of MR jobs tells us the max number of connections between any two users and thus helps us prove the small world theory.

Configuration (all m4.large)	**Small Cluster**	**Large Cluster**
Max Threshold = 1000	Running Time = 1 Minute Total MR Jobs = 7	Running Time = 1 Minute Total MR Jobs = 7
Max Threshold = 2500	Running Time = 2 Minutes Total MR Jobs = 7	Running Time = 2 Minutes Total MR Jobs = 7
Max Threshold = 5000	Running Time = 45 Minutes Total MR Jobs = 7	Running Time = 25 Minutes Total MR Jobs = 7

Log Files

Small Cluster
M = 1000 https://github.ccs.neu.edu/group-30/mr-project/blob/master/log_m_1000_s_3/j-V8H3TGZ3V0RD/steps/s-2H8Y6U8BQP7YM/stderr
M = 2500 https://github.ccs.neu.edu/group-30/mr-project/blob/master/log_m_2500_s_3/j-1GJ15Z8A1D6HY/steps/s-1R1FJBTGE3B33/stderr
M = 5000 https://github.ccs.neu.edu/group-30/mr-project/blob/master/log_m_5000_s_3/j-29O1ONYR6VZX8/steps/s-LWI8A139ZYDI/stderr

Large Cluster
M = 1000 https://github.ccs.neu.edu/group-30/mr-project/blob/master/log_large_m_1000_s_3/j-18BQJN45FYS9D/steps/s-1DCG8KY5PBRSO/stderr
M = 2500 https://github.ccs.neu.edu/group-30/mr-project/blob/master/log_large_m_2500_s_3/j-2ZPHSUNR9H0D8/steps/s-160D466FVH0OW/stderr
M = 5000 https://github.ccs.neu.edu/group-30/mr-project/blob/master/log_large_m_5000_s_3/j-1TF7QH3XGBY7I/steps/s-I0WX7QA6SA4G/stderr

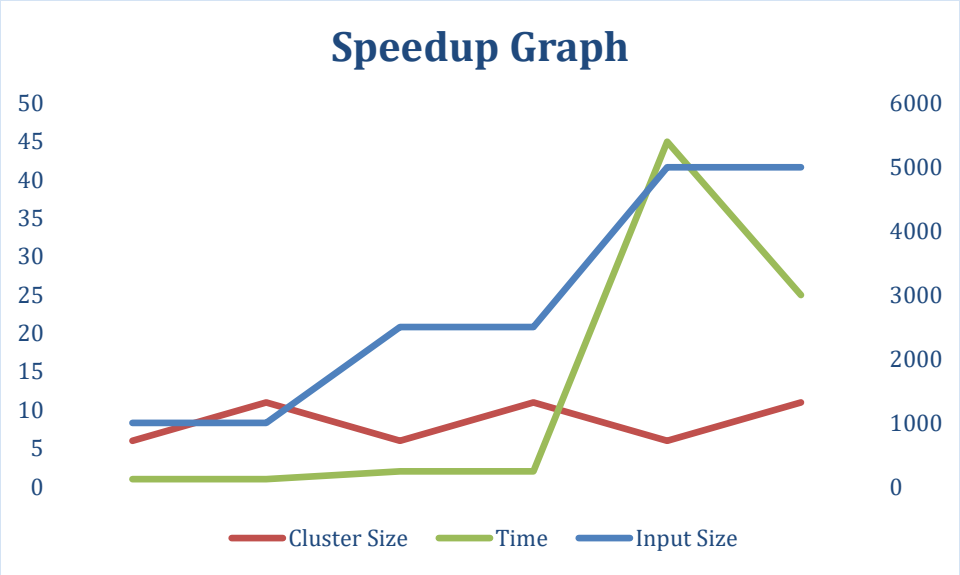
Link to Output Files

Small Cluster
M = 1000 https://github.ccs.neu.edu/group-30/mr-project/tree/master/output_m_1000_s_3
M = 2500 https://github.ccs.neu.edu/group-30/mr-project/tree/master/output_m_2500_s_3
M = 5000 https://github.ccs.neu.edu/group-30/mr-project/tree/master/output_m_5000_s_3

Large Cluster
M = 1000 https://github.ccs.neu.edu/group-30/mr-project/tree/master/output_large_m_1000_s_3
M = 2500 https://github.ccs.neu.edu/group-30/mr-project/tree/master/output_large_m_2500_s_3
M = 5000 https://github.ccs.neu.edu/group-30/mr-project/tree/master/output_large_m_5000_s_3

Speedup

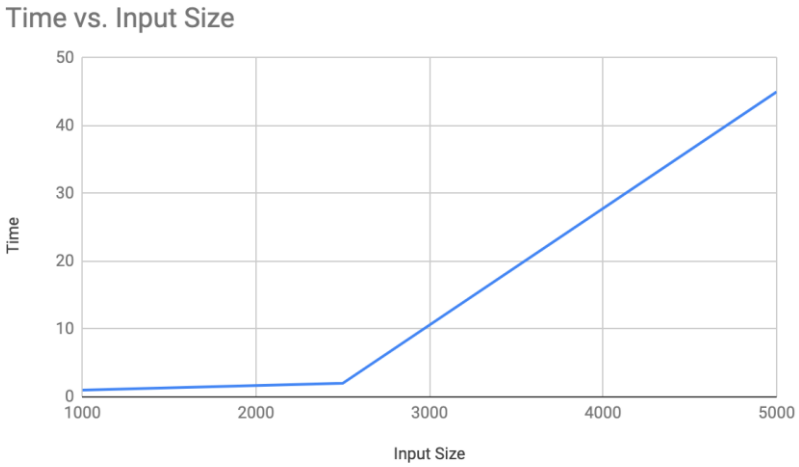
Input Size	**Time taken by small cluster (6 nodes)**	**Time taken by large cluster (11 nodes)**
1000	1 minute	1 minute
2500	2 minutes	2 minutes
5000	45 minutes	25 minutes



As seen from the above graph, for a given input size, the time taken by large cluster decreases (or at the most remains the same) as compared to the time taken by the small cluster, but never increases. Hence, we achieve good speedup. We believe, using more optimization techniques going further, we will be able to achieve a higher speedup.

Scalability

Input Size	Time
1000	1 min
2500	2 min
5000	45 min



Result Samples

Sample output data:
(231,4)
(330,4)
(132,-1)

Each record represents the nodeID and minimum distance from node 3
Example : (231,4) => nodeID : **231**, Distance from node 3: **4** (3-> a-> b-> c-> 231, where a,b,c are intermediate nodes in the path from 3 to 231)

Task 2 - K Means Clustering (MapReduce) (2-person task)

Overview

We planned to divide our entire 3-team project into two tasks.
1) A 1-person task for computing single source shortest path using Scala. (To be submitted as our Intermediate Project).
2) A 2-person task for K-Means clustering using MapReduce.

We decided to do this so that we could get a better understanding of both Scala and MapReduce projects. Doing our task 1 and learning various optimization techniques using Scala, we believe we will get a better insight now when we work on the MapReduce program in understanding the depth of parallel processing.

Task 2: 0 % progress

Conclusions

Task 1 is 100% done, we have implemented a correct algorithm and are currently working on spark optimization to improve performance. We are looking into using the same partitioner for pair RDDs that our program runs a join on.
From our various experimental runs, we see that every node in the dataset that is reachable from node 3 in less than or equal to 5 hops, this is in accordance to the small world theory that we set to explore.