Social Network Analysis at Scale in Cloud

Xiaolan Cai
xiaolan.cai@colorado.edu

❑ Introduction

Social network analysis has drawn a lot of attentions in the past decades. Initially sociologist have started to study how social networks is formed and how people are connected. Nowadays social networks like Facebook, Twitter is used to spread information like news, ideas. And also for marketing purpose, for example we may also want to study how to maximize the influence of advertisement by choosing proper targets.

Social networks normally can be modeled as graph structured models, with vertices/node representing an individual and edge representing the relationship between individuals.

Meantime big scale data mining tools is also well developed, and distributed computing is significantly improving the speed of data mining by parallely executing subtasks divided from one single big problem with distributed systems in scalable ways. And social network analysis normally includes massive amount of graph-structured data to processing, and how to leverage scalable distributed computing work to handle graph-structured data is indeed an interesting topic.

In this work we are interested in a specific problem of SNA, called influence maximization[1], and we want to use currently state-of-art distributed computing tools Hadoop and Spark, as MongoDB as the database, to examine how well we can process the graph-structured data in terms of quality and speed.

Maximum influence in Social network is an interesting topic of SNA (social network analysis). In [1] this problem is formally defined: Given a Graph N (V, E), and choose

an ignition k number of vertices set S, what is the maximum number of influence we can achieve? In [1], two Basic Diffusion Models are considered specifically **Linear Threshold** and **Independent Cascade** models. In this work, we consider only Independent Cascade(IC) models for simplicity.

In this model, we initially start with a set of active nodes A_0, in each step an active nodes v will have a chance to flip his any inactive neighbor w into active one with probability p_{vw}. In the whole process, node v only have a single chance to activate any of his inactive node w, meaning he cannot make any further attempts no matter he is successful in flip w or not. And the process stops when there is no mare activation are possible [1].

Pseudo code for Algorithm 1: Greedy Hill-Climbing Algorithm

1: $A = \Theta$
2: **for** $iter = 1 : k$ **do**
3: $a = arg\ max_a (\delta (A \cup a) - \delta (A))$
4: $-> maximizing\ the\ spread\ of\ chose\ set.$
5. $A = A \cup a$
6: return A

Finding the optimal solution is **NP-hard problem**, and in [1], an approximation solution is proposed, a natural greedy strategy obtains a solution that is provably 63% of optimal for several classes of models. However, the computation cost is still pretty heavy. For each iteration of the greedy algorithm, it simulates the influence of each inactive node, and chooses the nodes that bring maximum influence. Thus the simulating number needed is proportional to the size of the graph in each iteration.

It is a challenging task to process massive data not even to say graph structured data. In this work, we are trying to apply large scale graph data processing framework to simulate the propagation model, and implement the algorithm to overcome the drawback of such graph related algorithm.

Overall, in this project, we are trying to conduct experiments in the purpose of finding:

- Can we make the greedy algorithm of Influence Maximization scalable in order to apply it to large scale graph data? How is the running time of the program?

- How is the performance compared with other heuristic algorithms: PageRank, Connected Component in terms of quality in the same setup?
    - Quality is measured by the spread of the solution found(initial active set of vertices) under Independent Cascade model.

There are many kinds of large scale graph processing frameworks, such as GraphLab [7]: which shares state in parallel vertex computation. And Pregel [8]: which is used for passage passing. In Pregel, each vertex is a computing node and it is based on BSP (Bulk Synchronous Parallel) model. We choose Spark GraphX which has Pregel-like API, that allows user to only implement the vertex program and communication strategy. In our work, we also need to modify existing algorithm to make it suitable for vertex programing, which is requested by most graph based commutating framework. We need provide a vertex program that is run by each vertex in parallel.

❑ Related Work

We have investigated several popular social network application of graph algorithms for this project as well as some work that has been done on large scale graph analytics platforms.

In paper[1], the problem of maximizing influence in social networks is first been raised and proved that it's an NP-hard problem to find the optimal solution. But they have proposed an approximation algorithm. They have proved that their method can perform better than others such as Random Selection, High-degree or central nodes in estimating spread. But in each iteration of the algorithm, Monte Carlo simulations need to be run, which will add lots of computation overhead, therefore, in our work, we will apply Spark GraphX.

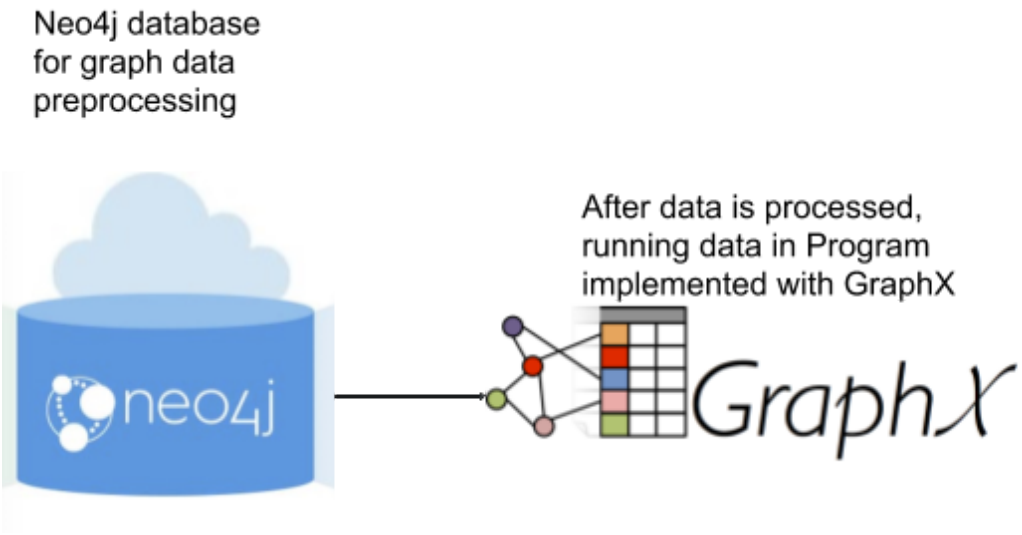
Another related work is using PageRank in analyzing social networks [2]. PageRank is originally designed in Google's web search engine by ranking the website based on its importance. It has also been used in other domains such as paper citation networks analysis. The importance of a page is determined by the sum of all PageRank scores of pages pointing to it. The prestige score of a page should be shared pages that it points to. PageRank is comparably simple algorithm that can be implemented in GraphX. We are interested in running PageRank on our data and observe the result.

Another heuristic algorithm of our interest is Connected Component in [3]. A connected component of an undirected graph is a maximal set of nodes such that each pair of nodes is connected by an edge. Connected components form a partition of the set of graph vertices, meaning that connected components are non-empty, they are pairwise disjoints, and the union of connected components forms the set of all vertices. Equivalently, we can say that the relation that associates two nodes if and only if they belong to the same connected component is an equivalence relation, that is it is reflexive, symmetric, and transitive. In real undirected networks, we usually find there is a large node that fills most of the network - usually more than half and not infrequently over 90% - while the rest of the network is divided into a large number of small components disconnected from the rest. Connected component will be implemented in GraphX as well. We will run the connected component on our data and observe the result.

Facebook has done numerous research work on real world graphs. In[4], they have done a quantitative and qualitative measurement on Apache Giraph. Facebook uses Apache Giraph in most of its Facebook-scale production workloads. In their experiment, they used three different algorithms: PageRank, Connected Components, and Triangle Counting. They chose these three algorithms is because they have different features. For PageRank, the amount of computation and communication is constant across iterations. For Connected components, the amount of computation and communications will reduce across iterations. And for Triangle counting, the amount of communications is comparably heavy. But the computation concept of storing graph and sending messages are similar. In our work, we choose to use GraphX because it has more features and it is much more easier to develop. And we will investigate the greedy algorithm in [1] in addition to the algorithms in GraphX lib.

❑ System Design

The basic parameters in the system are the number of workers, the number of processing threads, and the total memory allocated. Cluster setup: 3~10 physical machines, each node of 64GB RAM, 8 core Xeon D1548 2GHz CPU, Apache Spark 1.3.0



➔ Problem Faced

We have met continuous problem when deploying system in the Cloud. Experiments are run on CloudLab.us. We have setup cluster with different configurations but not all configurations can successfully run the program. When we use 60GB memory of 10 physical nodes with IC probability = 0.001, it becomes too complicated and will fail. As we increase the number of nodes in the cluster, the probability of failure will increase as well.

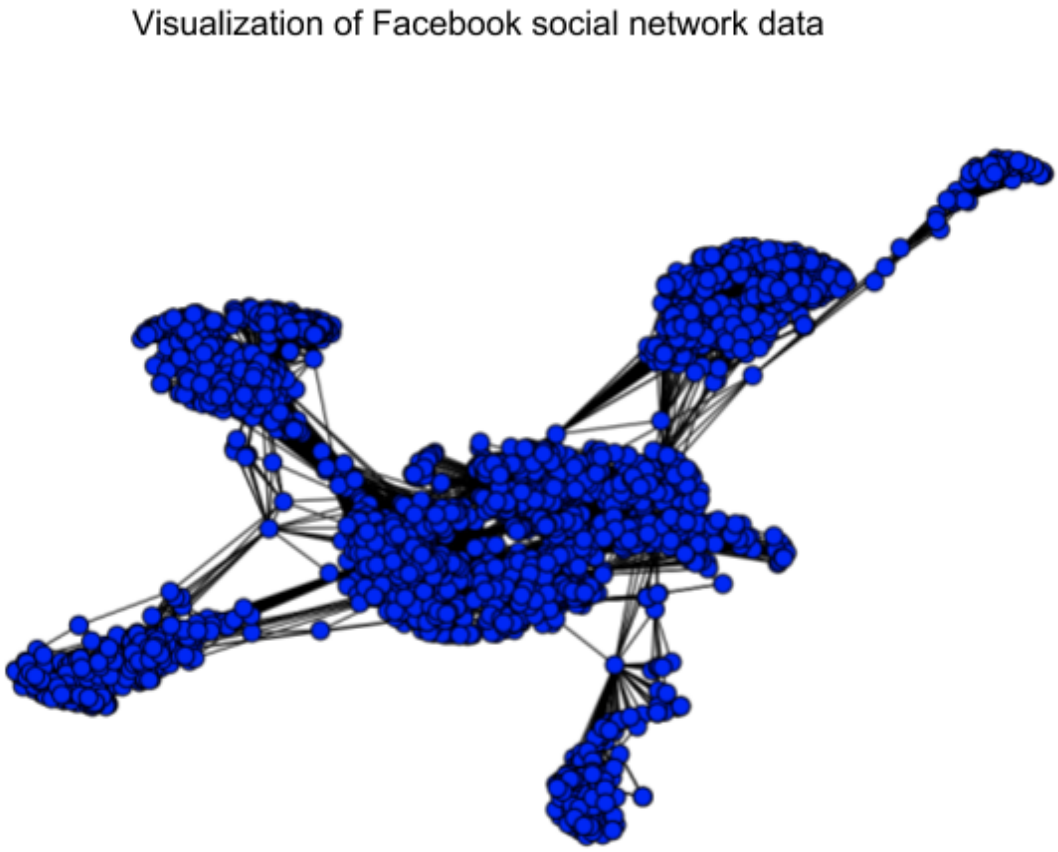
➔ Dataset

The dataset we used is Stanford SNAP: *http://snap.stanford.edu/data/ego-Facebook.html* The data files consisted of 5 kinds of files.

nodeId.edges : The edges in the ego network for the node 'nodeId'. Edges are undirected for facebook, and directed (a follows b) for twitter and gplus. The 'ego' node does not appear, but it is assumed that they follow every node id that appears in this file. **nodeId.circles** : The set of circles for the ego node. Each line contains one circle, consisting of a series of node ids. The first entry in each line is the name of the circle. **nodeId.feat** : The features for each of the nodes that appears in the edge file. **nodeId.egofeat** : The features for the ego user. **nodeId.featnames** : The names of

each of the feature dimensions. Features are '1' if the user has this property in their profile, and '0' otherwise. This file has been anonymized for facebook users, since the names of the features would reveal private data.

The dataset files of my interest is nodeId.edges containing totally 4038 nodes, and 80234 edges. The edges to node ratio is about 20. We can say the edges or nodes are densely connected or clustered. The graph data is represented in table view and I found that undirected graph structure may include duplicate data such as edge 1234 5678 is same as edge 5678 1234. We need to do data cleaning and remove the duplicated data. Below is the visualization of the data we used. We can see how the vertice are formed in clusters.



❑ Evaluation/Findings

In this session, the result of running the algorithms will be shown.

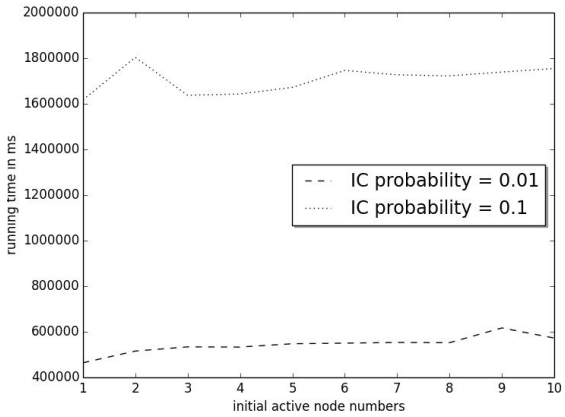


Figure 1

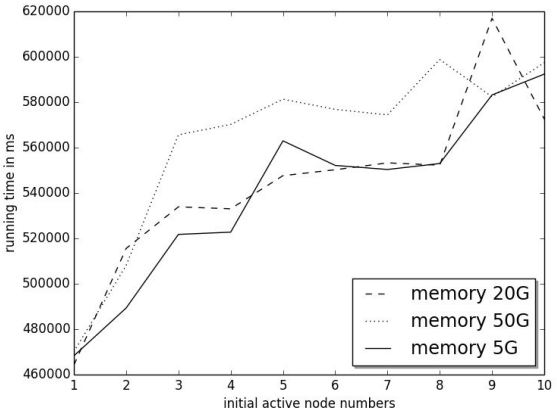


Figure 2

In the figures, x axis means the vertex number in the initial set and corresponding y axis represent the running time to spread the whole graph data. One vertex means we choose one vertex randomly into the set, run each iteration with simulation, the total time it cost to spread and use greedy algorithm to choose the node that spread to most nodes. Likewise, ten vertice means we choose ten nodes that spread to most nodes using greedy algorithm.

In Figure 1, when IC propagation probability is 0.01 has much better performance in terms of speed. So we set the probability to 0.01 for the rest of experiments.

In Figure 2, we investigate if the CPU memory will play major role in affecting the performance. We configure cluster setup with machines with 20GB, 50GB and 5GB CPU machines. And we can find that there is not much difference regarding the running time, which means this algorithm does not require more memory and are therefore not sensitive to how efficient memory management is.

In Figure 3, we run the program with same dataset in four clusters with different number of slave nodes. We find at first running time is decreased when machines in the cluster increased from 3 machines to 5 machines, but without huge changes when increase further.

The fluctuation is because of uncertainty of the simulation since we use random number as the probability to flip vertex status.

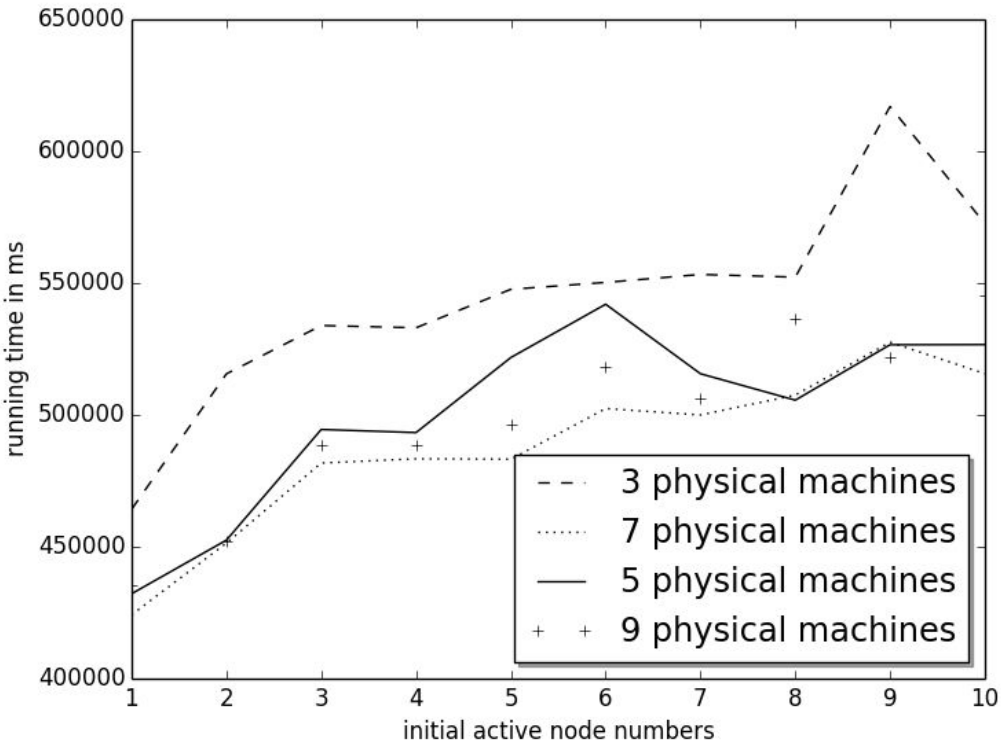


Figure 3

❑ Review of Team Member Work

- I am working independently for this project.

❑ Conclusion

Overall, my project work is a measurement work of Spark GraphX in a social network analysis application. I studied the influence maximization and its current solution in details, including advantages in accuracy and disadvantages in terms of scalability. I find that the greedy

algorithm has scalability issue. Firstly, it requires thousands of iteration. Secondly, this algorithm cannot be run well in parallel. That is why the increase in the number of machines doesn't make big difference. And Spark use parallelism to break large program into small tasks and running simultaneously, therefore it can take up to hours to run the program.

On the other hand, Spark GraphX programming API offers numerous features that is to simplify application development and is overall flexible to use and easy to implement such as SQL integration, however regarding scalability and performance, it is not good enough to support production workloads level for graph processing. Even for graph size that GraphX can handle, it can take up too much time to run a complex graph algorithm and its performance is not stable.

➔ Future work

SNA is an vast topic and developing quickly. For future work, studying more algorithms and other large scale graph platform such as Apache Giraph developed by Facebook, would be of interest.

❑ References

[1] David Kempe, Jon Kleinberg, and Eva Tardos. 2003. Maximizing the spread of influence through a social network. In Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining . ACM, pages 137–146.

[2] Wang R., Zhang W., Deng H., Wang N., Miao Q., Zhao X. (2013) Discover Community Leader in Social Network with PageRank. In: Tan Y., Shi Y., Mo H. (eds) Advances in Swarm Intelligence. ICSI 2013.

[3] G. Malewicz and M. Austern and A. Bik and J. Dehnert and I. Horn and N. Leiser and G. Czajkowski. Pregel: a System for Large-Scale Graph Processing. Proceedings of the 2010 ACM SIGMOD International Conference on Management of data, 2010.

[4] The Dang Huynh. Extension of PageRank and application to social networks. Web. Université Pierre et Marie Curie - Paris VI, 2015. English.

[5] https://www.sci.unich.it/~francesc/teaching/network/components.html

[6]https://code.fb.com/core-data/a-comparison-of-state-of-the-art-graph-processing-systems/