Cloud Computing Project Report: Community Detection in Massive Network Using MapReduce

Md Maruf Hossain
Dept. of Computer Science
University of North Carolina at Charlotte
Charlotte, USA
mhossa10@uncc.edu

Community detection in a massive network is a common problem in graph algorithm. On the other hand, community detection is important to analyze the graph and extract knowledge. Without information extraction, the graph is useless. So, it always a great concern what is the better way to detect community in the network. But, the size of the graph is increasing day by day. To handle this massive graphs extra effort required. MapReduce is well known to handle the big data and distribution system. In this project, I choose MapReduce for community detection. I am trying to make it simple because it is a class project and I am the only person to contribute to this task.

A. Methodology

There are lots of popular methods [1] [2] [3] [4] exist to detect community in the large network. I preferred the Louvain method because of its popularity. There are lots of sequential and parallel implementation of this algorithm exist, but I think no try on the Hadoop distributive system. In this perspective, it is a novel idea. So, we can say it is a challenging project.

1) Louvain Method: The Louvain Method is the most popular method to extract communities from the large network, first proposed by Blondel *et al.* [1]. It is a greedy optimization method that optimizes the *modularity* of a partition of the network, where *modularity* is defined as the fraction of edges that falls within the partitions minus the expected fraction that would be within the partition if the edges were distributed randomly. When a network has a dense connection between nodes and modules then it has a high modularity. On the other hand, a low modularity represents a sparser connection between node and module. The Louvain Method is a greedy bottom-up multilevel algorithm which uses modularity as the objective function [5]. In each pass of the *Move Phase*, nodes are repeatedly moved to neighboring communities such that the locally maximal increase in modularity is achieved until the communities are stable. The graph goes through a *Coarsening Phase* where each community is collapsed into a single vertex. The coarsened graph is then recursively processed.

Let's say, A graph of n number of nodes is denoted by $G = (V, E)$ where V and E represent the vertex and edge set respectively. Edges are represented by $\{u, v\}$ with the edge weight $\omega : E \to \mathbb{R}^+$. We used ζ to represent the community set and C for a single community and vol [5] method is for the volume of a node or community. To represent the neighbor set and community number, we used N and $\mathbb{N}$ respectively.

So, if we want to move a node u from community C to community D then the modularity difference will be,
$\Delta mod(u, C \to D) = \frac{\omega(u,D\setminus\{u\})-\omega(u,C\setminus\{u\})}{\omega(E)} + \frac{(vol(C\setminus\{u\})-vol(D\setminus\{u\})).vol(u)}{2.\omega(E)^2}$

Algorithm 1 Louvain Method: Move Phase

Input: graph $G = (V, E, \omega)$, communities $\zeta : V \to \mathbb{N}$
Result: communities $\zeta : V \to \mathbb{N}$

repeat
  for $u \in V$ **do**
    $\delta \leftarrow \max_{v \in N(u)}\{\Delta mod(u, \zeta(u) \to \zeta(v))\}$
    $C \leftarrow \zeta(\arg\max_{v \in N(u)}\{\Delta mod(u, \zeta(u) \to \zeta(v))\})$
    if $\delta > 0$ **then**
      $\zeta(u) \leftarrow C$
    end if
  end for
until ζ stable
return ζ

B. Procedure

To support, community detection using *Hadoop* framework we need a need a graph structure that can fulfill our expectation. The best way we can feed *Hadoop Mapper* class is to provide all the information for a vertex in a single line. That is why I choose `METIS` graph format, in this format first line will give you the number of vertices and edges with the information of weighted or non-weighted. In this project, I only consider the non-weighted graph that means every edge has the same priority. In rest of the file, every line represents a vertex and it contains all the neighbors of that vertex. But we need a little bit more information. That is why we need a preprocessing task to handle our necessity.

1) Preprocess: In this task, I modified every line with addition two information, default edge weight, and community. Initially, I assigned every vertex in a singleton community, that is means every vertex is itself a community. And then I assigned default edge weight 1.0 for every edge. And then save all the information in the *Hadoop* file system and use it as an input file for the main task.

2) Implementation: We need a graph structure to represent the graph, that is why I need a graph builder. In our algorithm, we have to major task move and coarsen. That is why we need two jobs to handle this two task. The job that handles move will try to find out the appropriate community for every vertex. This is an iterative task, that is why we need to maintain a looping system that can repeat the task. We need to find out an appropriate threshold value to terminate the job. This is come up by the experimental value, I tried different value and then able to get a minimum iteration value that can give the cluster with good modularity. To measure the performance of the project I am going to use modularity [6]. Modularity is a quality metrics often used to evaluate the quality of community detection algorithms. So, I implemented a method that can take a set of communities of the graph and give the modularity of the graph. Now, the second job will handle the coarsening, coarsening means after one set of move task performance we want to treat every community as a vertex and perform the move task again in this coarsen graph. In this project, I applied the coarsening only two times in the graph.

Github link of the project: Louvain Method in Hadoop

3) Challenges of The Project to Incorporate With Hadoop: There is a two main limitation of *hadoop* system for graph processing,

- No message passing.
- Not suitable for iterative computing.

The problems I faced during the project, no vertices can communicate with each other and it is very much required for this kind of algorithm. The reason behind this is when I am trying to assign a vertex to a new community it requires to inform all of its neighbor about this information. To handle this scenario I used `GSON` library from Google that I converted the graph object as text and passed it to the *Mapper*. In the *Mapper* setup method, I retrieve the graph object again. So, *Mapper* class updated this object as well generate the intermediate value. The second problem is that it is an iterative process algorithm and *Hadoop* can perform once the Map-Reduce task. To handle this case, you need to maintain a loop and repeat the job every time. It cost lots of time that makes this kind of framework inappropriate for graph processing. This is a class project and I choose *Hadoop* system by myself but I gain some knowledge that next time I have to choose Apache *Giraph* or Apache *Hama*. Apache *Giraph* is a framework of *Pregel* that is a distributed system for graph processing and every vertex can pass messages to its neighbor in every super step.

C. Dataset

I used cluster datasets from DIMACS as a graph for the project. Here is the table of the result sets.

TABLE I
LIST OF GRAPHS USED IN THE EXPERIMENT

Graph	Nodes (V)	Edges (E)	Modularity
SmallWorld	100,000	499,998	0.79564
As-22july06	22,963	48,436	0.760376
G_n_pout	100,000	501,198	0.5393
Email	1,133	5,451	0.72648

D. Responsibility

It is an individual project for me, so I am the only person responsible for the whole project.

E. Experimental Setup and Evalution

To experiment on my project, I used `Cloudera` on my personal computer. The main reason for this is the heavy traffic of the DSBA Hadoop Cluster. That is why I used the small size of the graph.
In this experiment, I find out that graph `As-22july06` performs better in comparison of the parallel *Louvain* [5] method. Obviously, I am talking about the modularity not execution time because of their performance a lot better than this one. That is why I do not mention the time complexity. In the PLM [5] the modularity for As-22july06 is about 0.68 but in this algorithm, it provides 0.760376.
So, we can conclude that you can earn better modularity for some graph in the *Hadoop* system but it is a little bit complex for the graph processing.

REFERENCES

[1] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[2] Usha Nandini Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, 76(3):036106, 2007.

[3] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. In *International symposium on computer and information sciences*, pages 284–293. Springer, 2005.

[4] Venu Satuluri, Srinivasan Parthasarathy, and Yiye Ruan. Local graph sparsification for scalable clustering. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 721–732. ACM, 2011.

[5] Christian Staudt and Henning Meyerhenke. Engineering parallel algorithms for community detection in massive networks. *IEEE TPDS*, 27:171–184, 2016.

[6] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.