Distributed Community Detection Using Distribued Key-Value Stores

Mufaddal Dewaswala
P.E.S Institute of Technology
Hosur Rd, Konappana Agrahara, Electronic City,
Bengaluru 560100

***Abstract*—Real world networks like social interaction networks, computer networks, country road networks etc. all exhibit clustering or groups of densely connected nodes, the interpretation of which can be a group of friends or a local area network or a settlement respectively. The interpretation of the clusters or more aptly called communities depends on the context of the graph and the information represented by the vertices and edges. The identification of these communities is an NP-hard optimization problem. The problem is magnified owing to the large size of the graphs which can reach billions of edges depending on the information it models. These problems require a scale-able approach to solving them and hence a distributed approach to detection of communities allows not only allows easy scaling of processing power but also scaling of memory. We propose a solution which builds on Label Propagation Algorithm (LPA), a near linear time community detection algorithm, which by parallelising, we build a scale-able version which leverages distributed computing. The core of the solution relies on key-value database clusters to provide the much important horizontal memory scalibility to fit graphs billions of edges in size while running on commodity hardware.**

***Index Terms*—Community Detection, Label Propagation Algorithm, Redis, Distributed Computing**

I. Introduction

A. Graph

In graph theory, a Graph G is defined as an ordered pair $G = (V, E)$ where V is the vertex set and E is the edge set where $E \subseteq \{(a, b) : a \in V and\ b \in V\}$. A pair of vertices$(u, v) \in E$ if an edge exists between u and v. The edge set E can be ordered or unordered depending on whether the graph is directed or undirected. $|V|$ is the cardinality of the vertex set called the $order$ of the graph, while $|E|$ is called the $size$ of the graph. In this paper we limit our discussions to undirected graphs which lack multi edges and self loops.

B. Communities

1) Definition: Community structure in graphs can be loosely stated as a collection of nodes having a high edge density among themselves as compared to with those outside the collection. Intuitively, a community within a graph has more edges between the vertices as compared to the edges the members of the community have with the vertices not in the group.

An empirical mathemathical description of a community is given by defining a measure $\delta(A)$ and $n_c = |A|$ (total edges in A), defined on some graph A. The definition is as follows :

$$\delta(A) = \frac{\#\ edges\ in\ A}{n_c(n_c - 1)/2}$$

The equation is the ratio of the number of edges in the graph to the total number of all different combinations of edges possible. For a community in a graph G represented by sub graph C, $\delta(C)$ should be much larger than $\delta(G)$

2) Characteristics: Communities may or may not be present in graphs depending on what and how a data set is modelled. Also, the mere presence of communities does not necessarily mean that it conveys some meaning about that group of vertices or the graph. Depending on the domain and application communities and community detection have several uses some of which are summarized by Mini Singh Ahuja et al.[1]

- Detection of suspicious events in telecom networks
- Refactoring software packages
- Product recommendation systems
- Prediction of epidemic spread in communities with overlapping structure
- Terrorist groups detection in social networks
- Lung cancer detection

C. Community Detection

Community detection can be thought of as graph clustering in a much general sense, the difference being that clustering

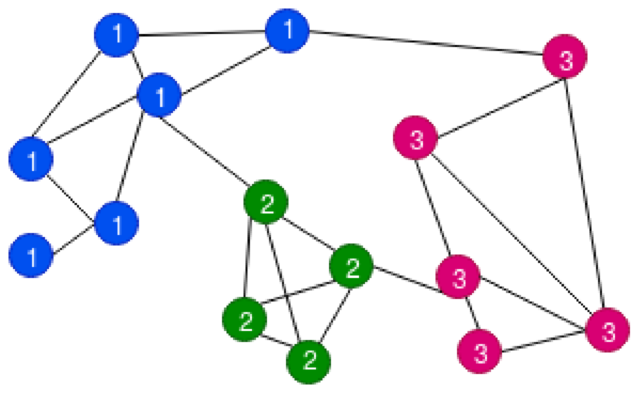


Fig. 1. Example of a graph with 3 communities

has the number of clusters in the result to be an input of the clustering algorithm while community detection has no such restriction that is, community detection is unsupervised [2] Detections of these communities become a memory intensive task as the size and order of graph increases to millions and beyond.

II. Literature Review

A. Edge Betweenness Measure

Several methods exist for detection of communities in graphs. GirvanNewman [3] used edge betweenness to reveal the underlying community structure by successively removing edges with high edge betweenness. The approach defines edge betweeness by generalising the measure of vertex betweeness where vertex betweenness is defined as the number of times some vertex i occurring in the shortest path between a pair of any vertices in the graph, hence, edge betweeness is defined as the number of pairs of vertices which have their shortest path running along this edge. For a communities loosely connected, the edges connecting the communities will have a high edge betweeness, thus by removing these edges, the underlying community structure will be revealed.

B. Coding theory

Martin Rosvall and Carl T. Bergstrom [4] developed an algorithm based on information theory and coding theory which used random walks and compression of the information transfer involved in the walks for detection of communities in weighted and directed graphs. The theory behind information flow being that if information within a group of node flows easily and quickly then it is a well connected structure and hence reduces the problem of detection of communities in a network to a coding theory problem.

C. Constrained LPA

Michael J. Barber et al [5] built on Tib ely et. al [6] formal proof that LPA is mathematically equivalent to the minimization of the Hamiltonian for a kinetic Potts model. They expressed the original LPA algorithm as an optimizing procedure of an objective function H. To improve the efficacy of LPA by removing undesirable solutions, they devised methods to optimize H by constraining it by subtracting a weighted function called the penalty function G(v)from the objective function. They discussed the various possible functions G(v) to effect different optimizations on LPA.

D. Label Propagation algorithm

LPA was proposed by Raghavan et. al. [7] for the detection of disjoint communities in networks. The simplicity of the algorithm along with its intuitive working and a linear running time makes it an attractive choice further exploration and enhancements. Its working makes it a choice for parallelization. The algorithm takes an iterative bottom up approach to build communities. It begins with all vertices in the graph having their own labels. At every iteration, every vertex in the graph adopts the most popular label amongst its neighbours. This continues until an equilibrium is reached where there are no changes amongst the nodes. Ties are broken by choosing a label on random. The algorithm has a non-deterministic nature and hence there might be slight or large variations depending on the graph. Fig 2 shows the algorithm

It becomes difficult to implement some of the algorithms for a distributed system to achieve true parallelism. LPA on the other hand can be easily adapted for a near parallel approach to community detection, allowing every iteration of the algorithm to be run parallel on partitions of the vertex set.

Algorithm 1 Label Propagation Algorithm

```
    Input $V, \sigma$
    Output $V$
 1: updt = $\infty$
 2: while $updt > \sigma$ do
 3:     updt = 0
 4:     for $\forall v \in V$ do
 5:         $newLabel = max(\{(l, count(l, v.adj()))\})$
 6:         if v.getLabel() $\neq$ newLabel then
 7:             v.setLabel(newLabel)
 8:             $updt = updt + 1$
 9:         end if
10:     end for
11: end while
```

E. PLPA on Xeon Phi Architecture

Khlopotine et. al [8] used xeon phi platform for parallelising LPA. They described 2 algorithms for PLPA and PLPA-M for processing graphs. The latter was devised to handle the 1.8 billion edge Friendster graph. PLPA worked similar to LPA with an additional active edge data structure $A(v)$ to optimize the running time of LPA. The implementation was done in C++ using the OpenMP library. A programming model called the offload programming model was used and synchronization and locking issues were kept to a minimum by reducing access to shared resources and buffering required data locally. It wasn't possible to work with the Friendster network with all data structures in memory, therefore they opted to go for modified version of PLPA which partitioned the graph and ran processed each partition. The shortcomings of their methods was that the implementation was bound to a very specific platform.

III. Distributed key-value Stores

A. Redis

Redis [9] is a NoSQL in memory database in which data is stored as key-value pairs. The key functionality provided by Redis is a distributed database though clustering. A Redis sever is ran on every computer in the cluster and are connected to form a Redis cluster. This cluster stores information about the vertices, its neighbors and labels which is then retrieved and updated during the detection of communities.

The cluster uses a checksum as a hash function given by $CRC16(key)\%16384$ to map keys to slots sharded across the cluster[10]. The vertex number is used as the key and hence leads to a uniform distribution of the keys across cluster which is approximately $(|V|\ no.of Redisinstances)$.

IV. Terminology

- *Master* : A computer on the network which is responsible for co-coordinating the detection among the workers by sending partitions of the vertex set, initiating detection and handling the responses from the workers after the end of every iteration. A single master exists on the network.
- *Worker* : A system on the network which performs community detection on the vertex partition sent by the master. It also hosts an instance of a Redis server. One or more workers exist on the network.

V. Methodology

Our approach relies on initially partitioning of the graph into nearly connected subgraphs and sending them to workers nodes on the network for futher processing. Until the threshold condition is met for stopping the algorithm, the master will instruct the worker nodes to run LPA on its subset. The worker nodes on receipt of the message will first acquire the labels of vertices which are adjacent to the the vertices in its subset. If the vertex is not in its subset then it obtains it from the node which does. The worker then paritions the its vertex set depending on the hardware threads it has and runs the algorithm on these partitions on seperate threads. After running an iteration, it reports the iteration the to the master node.

The approach is based on 2 important observations, First, the LPA algorithm as originally described requires that the vertex being updated, opts the label that the majority of it neighbours have and this majority is calculated from the labels of the neighbours in the current and the previous iteration. This is done to prevent oscillations of labels that would otherwise occur in bipartite graphs if only the labels from the previous iteration were considered. We prevent this oscillation to an extent by requiring that at least one neighbour of every vertex exists in the same partition. Second, Khlopotine et. al. used a list L(v) on which the threads locked to retrieve node label info [8], this would cause other threads to block, we instead allow finer granularity by allowing locking on a per vertex basis.

VI. Evaluation

The algorithm was tested on 3 SNAP datasets [11] with 88234, 396160 and 1049866 edges on the following setup

- 5 computers
- 1 master
- 4 workers
- 4 Redis instances (1 per worker)

Worker Setup

- Intel Penitum 4 (3.2 Ghz, 1 core) x86
- 1 GB RAM

Name	Edges	Detection Time
facebook_combined	88234	2 *minutes*
CA-AstroPh	39616	8 *minutes*
com-dblp.ungraph	1049866	23 *minutes*

Algorithm 2 Master Algorithm

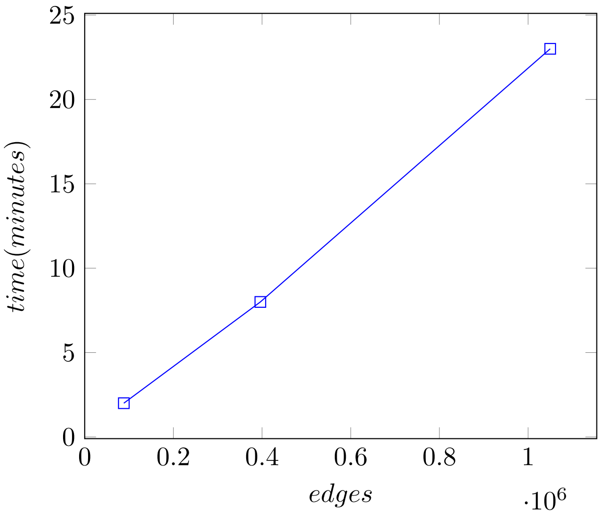
```
    Input $G(V,E)$, $W$, $\sigma$

    Initialization
1:  Partition Vertex set $V$ into $P = \{V_1, V_2 ... V_n\}$ disjoint sets
    such that $\Sigma\ deg(V_i) \approx |E|/n$, $i < 0 < n$ and $\forall v \in V$ if
    $deg(v) > 0$ then $\exists (v,u), u \in V$
2:  send worker $n$ partition $V_n$
3:  for $\forall V_i \in P$ do
4:      Send $V_i$ to $w_i$
5:  end for
    Execution
1:  while $updt > \sigma$ do
2:      for $\forall w_i \in W$ do
3:          send start message to $w_i$
4:      end for
5:      $sum\_updt = 0$
6:      for $\forall w_i \in W$ do
7:          $sum\_updt = sum\_updt + updt(w_i)$
8:      end for
9:  end while
```

Algorithm 3 Worker-i Algorithm for

```
    Initialization
1:  accept partition $V_i$ from master
2:  partition $V_i$ into $W = \{V_{i1}, V_{i2} ... Vik\}$ disjoint sets where
    k is no. of hardware threads
    Execution
1:  fetch vertex neighbour label information $adj(v), \forall v \in P_v$
    from neighbour
2:  updt = 0
3:  for $\forall V_{ij} \in W$, on thread $j$ do
4:      for $\forall v \in V_{ij}$ do
5:          newLabel = $max(\{(l, count(l, v.adj()))\})$
6:          if v.getLabel() $\neq$ newLabel then
7:              v.setLabel(newLabel)
8:              updt = updt+1
9:          end if
10:     end for
11: end for
12: Send $updt$ to master
```

$Fig.A$: Plot of no. of edges vs detection duration



VII. Conclusion

The program was run on commodity hardware on a 100 Mbit/s LAN . From the graph, we observe that the detection time varies nearly linearly with the total edges. Hence, scalability in terms of memory is achieved without compromise on detection time.

Acknowledgment

The author would like to thank Prof. Sreenath M.V, Assistant Professor, Department of ISE, PESIT-BSC for his encouragement and guidance during the course of the project.

References

[1] N. Mini Singh Ahuja, Jatinder Singh, "Practical applications of community detection," *International Journal of Advanced Research inComputer Science and Software Engineering*, vol. 6, pp. 412–415, April 2016.

[2] S. Papadopoulos, Y. Kompatsiaris, A. Vakali, and P. Spyridonos, "Community detection in social media," *Data Mining and Knowledge Discovery*, vol. 24, pp. 515–554, May 2012.

[3] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[4] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proceedings of the National Academy of Sciences*, vol. 105, no. 4, pp. 1118–1123, 2008.

[5] M. J. Barber and J. W. Clark, "Detecting network communities by propagating labels under constraints," *Phys. Rev. E*, vol. 80, p. 026129, Aug 2009.

[6] G. T. ely and J. K. esz, "Community detection in social media," *Physica A: Statistical Mechanics and its Applications*, 2008.

[7] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Phys. Rev. E*, vol. 76, p. 036106, Sep 2007.

[8] A. B. Khlopotine, A. V. Sathanur, and V. Jandhyala, "Optimized parallel label propagation based community detection on the intel(r) xeon phi(tm) architecture," in *2015 27th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 9–16, Oct 2015.

[9] "Redis." [Online; accessed 28-January-2018].

[10] "Redis cluster specification." [Online; accessed 28-January-2018].

[11] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." http://snap.stanford.edu/data, June 2014.