# Distributed Community Detection Using Redis

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

***Abstract*—This paper discusses a method for the detection of communities using a parallel version of the label propagation algorithm to allow scaling in terms of memory and processing power on commodity hardware. The unification of primary memory is achieved with a Redis cluster allowing memory to be scaled horizontally.**

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

## I. Introduction

### A. Graph

A Graph is an ordered set $G = (V, E)$ where $V$ is the vertex set and $E$ is the edge set. 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 un-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 have limited our discussions to simple graphs which lack multi edges and self loops.

### B. Communities

Community structure in graphs can be loosely stated as a collection of nodes having a high edge density among themselves as compared to those with outside the collection. The structure is very prominent in social networks which highlight common interests, hobbies, friend groups etc. depending on the context of the graph. Community detection can be thought of as graph clustering in a much general sense, the difference being that clustering has the number of clusters in the result to be an input of the clustering algorithm while community detection has no such restriction [1] Detections of these communities become a memory intensive task as the size and order of graph increases to millions and beyond.

### C. Label Propagation algorithm

Label propagation algorithm (LPA) [2] is of particular interest in this paper. In LPA, initially, each node is assigned a unique label which signifies the group it belongs to, then several iterations are run in which the following is done; every vertex in an iteration joins the group to which a major number of its neighbors belong to. Iterations are performed until a state of equilibrium is reached.

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. The memory on the distributed system is unified by Redis which abstracts the discrete memory units on the cluster.

## II. Literature Review

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.

Martin Rosvall and Carl T. Bergstrom [4] used random walks and compression of the information transfer involved in the walks for detection of communities.

Charith Wickramaarachchi et al. [5] employed a parallel variation of the Louvian algorithm to discover communities achieving a significant performance improvement than its sequential counterpart.

## III. Redis

Redis [6] 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[7]. The vertex number is used as the key and hence leads to a uniform distribution of the keys across cluster which is approximately $(|V|\ no.of Redis instances)$.

## 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

The algorithm is a slightly modified version of the original LPA and uses a client-server model. For every iteration the algorithm involves shuffling and partitioning of the vertex set into $(n)$ partitions where n is the number of workers on the network; these partitions are sent to the workers. The workers then detect communities in the afore mentioned partition using LPA and update the labels of the vertices. It then reports the greatest count of the nodes updated among all iterations of LPA it ran to the master. The master then decides whether another iteration of the algorithm should be run depending upon a minimum updated nodes threshold($th$) value. In LPA, ties amongst equally popular neighbors are broken randomly but in our algorithm this is not the case, instead if a vertex under consideration already belongs to one of the equally popular neighbors then it continues to do so. This leads to oscillations in some cases which can be controlled by setting a threshold of updated nodes for further iterations of LPA to continue.

**Algorithm 1** Master Algorithm

1: **repeat**
2: shuffle vertex set $V$
3: partition $V$ into $P_n$ sets
4: send each worker a partition $P_n$
5: accept greatest updated vertices count from workers as $R_n$
6: $updt \leftarrow \Sigma R_n$
7: **until**
8: $updt < th$

**Algorithm 2** Worker Algorithm

1: accept partition $P_n$ from master
2: partition $P_n$ into $W_m$ sets
3: fetch vertex information $v, \forall v \in W_m$ from Redis
4: run LPA $\forall W_m$ parallel on each processor core
5: report greatest updated vertices count to master

## VI. Evaluation

The algorithm was tested on 3 SNAP datasets [8] with 88234, 396160 and 1049866 edges on the following setup

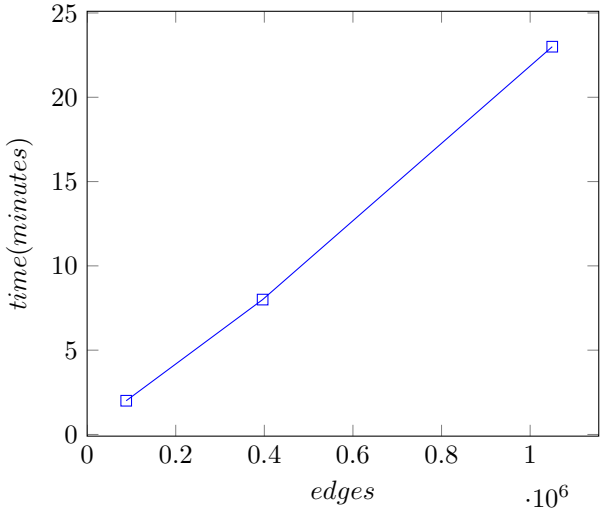
- 5 computers
- 1 master
- 4 workers
- 4 Redis instances (1 per worker)

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* |

*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] 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.

[2] 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.

[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] C. Wickramaarachchi, M. Frincu, P. Small, and V. K. Prasanna, "Fast parallel algorithm for unfolding of communities in large graphs," in *2014 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6, Sept 2014.

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

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

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