Benchmarking Graph database/processing systems for Agglomerative Clustering

Ashish Jain
ashishjain@cs.umass.edu
School of Computer Science
University of Massachusetts Amherst

Vijay Pasikanti
vijaykp@cs.umass.edu
School of Computer Science
University of Massachusetts Amherst

Gerome Miklau
miklau@cs.umass.edu
School of Computer Science
University of Massachusetts Amherst

Abstract

Graph data is getting increasingly popular in, e.g. text processing, social networks etc. Over the years it has lead to the development of various graph databases and graph processing systems. A recent study by [8] benchmarks the graph and relational databases. However, according to our knowledge there has been no study on benchmarking graph databases and graph processing systems to compare their performance. Therefore our proposal is novel in terms of a benchmark study. In this paper we take one of the most expensive data mining technique i.e. clustering, to benchmark the two systems. Since personal social networks are big and cluttered, clustering is an effective technique to organize and find out community structure within the social network graphs. We are using Neo4j and GraphLab as our graph database and graph processing system respectively.

Keywords: Clustering, graph database, graph processing, Neo4j and GraphLab

1. Introduction

There has been great deal of discussion recently on the subject of graph processing. With the explosion of social networks like Facebook, Twitter and other complex, inter-dependent datasets. Graph storage and processing has emerged as one of the new computational challenges in the field of big data. Since the social data with connections between people provides a graph like structure, finding efficient ways to process it and discovering close-knit clusters in these networks is of fundamental interest to us.

Currently, users in Facebook, Google+ and Twitter identify their circles either manually, or in a nave fashion by identifying friends sharing a common attribute. Here in this paper we study the problem of automatically find communities of related users by utilizing the graph properties. It is based on the intuition that a cluster is a collection of individuals with dense friendships patterns internally and sparse friendship externally. Utilizing such graph properties requires traversal of entire graph and extract out important information during the course

of traversal. In this fashion, we would be able to judge the performance of two systems under consideration based on their efficiency of accessing and processing graph network.

Based on our clustering technique described in Section 5, we will measure the performance of Neo4j and GraphLab in terms of running time over various size and complex structure of datasets. This analysis will help us in understanding whether storing graph like structure in database (Neo4j) provides any significant advantage over dynamically loading the graph from raw file formats and processing it using a graph processing system.

2. Related Work

Hierarchical clustering algorithm has been studied ([1][2][3]) for over decades now. Also there have been many of its successful implementations for various applications ([4][5]). In addition to that, many researchers have evaluated different types of hierarchical algorithms but most of the results were presented on smaller datasets.

As social networks gained popularity and datasets became available through various sources like Stanford SNAP library [12], various works on clustering or analyzing social networks has been published [9][10]. All of these works used graph like structure stored in simple raw files. However, as the data sizes grow for network graphs, these algorithms are implemented on graph databases and there is a very little literature [6] on the implementation and evaluation of clustering algorithms on large datasets using graph database systems.

Some work on performance comparison on the relational and graph databases has been done by [8] but they evaluated different tasks i.e. page ranking and shortest path distance on SNAP social network data. They showed that relational databases perform better in comparison to graph database for the above tasks. However when dealing with social network data, SQL is certainly not a preferred choice. Therefore a comprehensive performance comparison of some popular graph database and graph processing system is required.

In this paper, we propose a clustering method over a synthetic graph like network. Each node is treated as a person in social network and edges represent relationship with other people in the network. Since it is a synthetic graph, we dont have node specific properties, therefore we utilize some graph properties described in below mention sections.

3. Problem Statement

This paper addresses the following two problems:

- Clustering a synthetic social network graph on graph databases and graph processing systems to detect communities. For accomplishing this task, we have implemented standard agglomerative clustering with various cluster quality measures to form clusters.
- We have done performance evaluation of graph databases/processing systems in terms of their time of execution and run-time memory usage while running the clustering algorithms. It will provide benchmark numbers and an opportunity to select an appropriate system.

- We have also studied the scalability performance of these graph database/processing systems as data grows in orders of magnitude and complexity using clustering as the task.

4. Data Generation

Social network datasets are generated synthetically with varying sparsity and size. `Networkx` [16], a python graph data generator package is used for generating datasets using the following two step approach.

1. Generate small size communities (undirected graphs) with a fixed number of nodes and edges with a given triangle formation probability using `netwotkx`'s `powerlaw_cluster_graph` graph generator. `powerlaw_cluster_graph` takes three arguments viz. number of nodes, number of random edges for each node and probability of adding a new triangle after adding a random edge. Varying number of random edges per node and triangle formation probability, density of the graph changes. For example, for a dataset with 100,000 edges, 10 clusters are created each with total nodes of 400, number of average edges per node are 25 and triangle formation probability of 0.8.

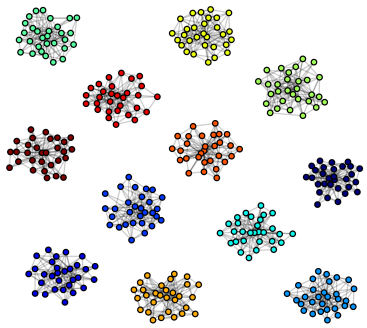

Figure 1: Initial Clusters

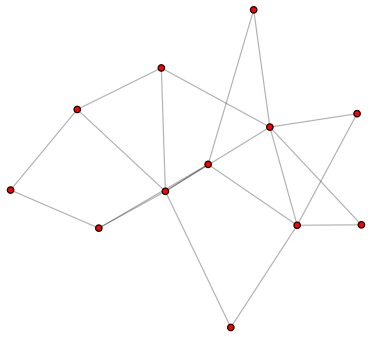

Figure 2: Template used for adding random edges

2. After the creation of small clusters like shown above, create a template shown in the figure 2 using a powerlaw cluster graph generator with number of nodes as the number of clusters and add edges between the nodes of one cluster to another cluster from the template using reduced edge addition probabilities (shown in figure 3).

3. Repeat the above step by generating random templates and random edge addition probabilities until the graph becomes a social network.

5. Agglomerative Clustering

Agglomerative Clustering [15] is a method of cluster analysis which seeks to build a hierarchy of clusters from "bottom-up" approach. In this approach, each node in the graph starts with a cluster of its own and pairs of clusters are merged as one moves up the hierarchy. In order to decide which clusters should be combined, a measure of similarity between sets of clusters is required. Two measures are used in our work to combine single node clusters and multi-node clusters:

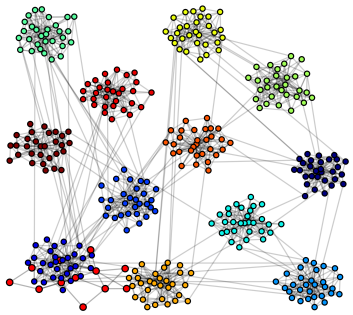

Figure 3: Social network after adding random edges between groups

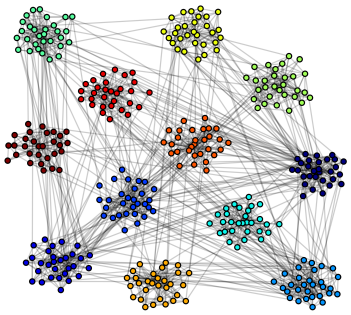

Figure 4: Social network after adding more random edges

5.1. Structural Similarity Index

Let $G = (V, E, w)$ be a weighted undirected network and $w(e)$ be the weight of the edge e. For a node $u \in V$, we define $w(\{u, u\}) = 1$. The structure neighborhood of a node u is the set $\Gamma(u)$ containing u and its adjacent nodes which are incident with a common edge with $u : \Gamma(u) = \{v \in V | \{u, v\} \in E\} \bigcup \{u\}$. The structural similarity [11] between two adjacent nodes u and v is then

$$\sigma(u, v) = \frac{\sum_{x \epsilon \in (u) \cap \in (v)} w(u, x).w(v, x))}{\sqrt{\sum_{x \epsilon \in (u)} w^2(u, x)} \sqrt{\sum_{x \epsilon \in (v)} w^2(v, x)}}$$

The above structural similarity index can be replaced by other similarity definitions such as Jaccard similarity. Our internal testing results showed that the structural similarity index is better.

5.2. Clustering Coefficient

Clustering coefficient [14] is a measure of degree to which nodes in a graph tend to cluster together. Our study of social network graphs suggested that nodes in communities with higher clustering coefficient tend to have high edges per node and are tightly connected.
Let $G = (V, E, w)$ be a graph consists of a set of vertices V and set of edges E between them. An edge e_{ij} connects vertex v_i with vertex v_j. The neighborhood N_i for a vertex v_i is defined as its immediately connected neighbors as follows:

$$N_i = \{v_j : e_{ij} \in E \wedge e_{ij} \in E\}$$

The local clustering coefficient C_i for a vertex v_i is then given by the proportion of links between the vertices within its neighborhood divided by the number of links that could possibly exist between them. An undirected graph has the property that $e_i j$ and $e_j i$ are considered identical. Therefore, if a vertex v_i has k_i neighbors then $k_i(k_i - 1)/2$ edges could exist among the vertices within the neighborhood. Therefore, local clustering coefficient for directed graphs:

$$C_i = \frac{2|e_{jk} : v_j, v_k \epsilon N_i, e_{jk} \epsilon E|}{k_i(k_i - 1)}$$

and clustering coefficient for the whole network is given as the average of the local clustering coefficients of all the vertices n: 0

$$C = \frac{1}{n}\sum_{i=1}^{n} C_i$$

6. Implementation

Clustering algorithm is implemented in python using two pass approach and has two separate interfaces for Neo4j [17] and GraphLab [18] for querying node information. Each pass is described below.

- **First pass**: Start breadth first search with 20 randomly sampled nodes from the database. Request neighbor list of each node using queries provided by Neo4j and Graphlab. As we get neighbor list of a node, calculate structural similarity index of the node with respect to already processed nodes and if similarity index is more than a tuned value of index threshold, combine the nodes into a cluster. This is repeated with each node as soon as we get its neighbor list. Query used for requesting neighbor list:
  - Neo4j:

    ```
    START b = node:community('id:given\_node') MATCH b-[:KNOWS]-a return a
    ```

    The index `community` is created on nodes during data insertion phase.
  - GraphLab:

    ```
    graph = graphlab.Graph()
    neighbor_list = graph.get_edges(src_ids=[str(nodeId)])
    ```

- **Second pass**: At the end of the first pass, we see a list of clusters generated using structural similarity index calculations explained above. To begin with, sort the list of clusters with respect to the number of nodes in the group from large to small clusters. Iteratively pick up the clusters with smallest size (i.e. one at the end of the cluster list) and try to combine it with the (large) clusters on the top of the list. For each pair of clusters, clustering coefficient (C_{ij}) of the combination of clusters is calculated and compared it with the clustering coefficient of individual clusters. When C_{ij} is greater than a tune factor (k) of C_i and C_j, clusters are merged into one, i.e.

  $$C_{ij} > kC_i \text{ and } C_{ij} > kC_j.$$

  This process is repeated until there are no more clusters that can be merged.

7. Evaluation

7.1. Datasets

As mentioned before datasets are generated using `networkx` library and there are two types of datasets, one sparse and two dense. Sparsity is measured based on the clustering coefficient of the graph. Sparse random graphs have a small clustering coefficient while the dense graphs have large clustering coefficient.

Dataset	Total Edges	Dense (nodes)	Sparse(nodes)
1	20,000	900	2,000
2	50,000	2,100	5,000
3	100,000	4,000	9,500
4	200,000	8,000	19,000
5	500,000	20,000	47,000
6	1,000,000	40,000	91,000

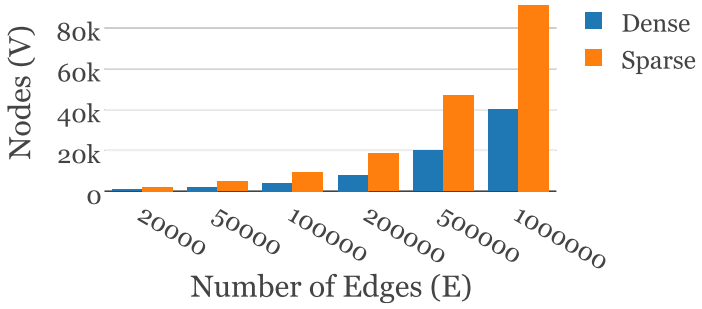

Figure 5: Dataset Statistics

We have created six groups of datasets each with two graph datasets with a fixed number of edges. One graph is a sparse graph while the other one is dense. Below is the table of all the datasets generated for testing.

The **average degree distribution** for the sparse graph is 12 edges/node while for dense graph is 25 edges/node.

7.2. Setup

The following are the set of machines used for evaluation:

- Neo4j: Amazon EC2 instance with 4 vCPUs and 7.5GB memory with 20GB disk space.
- GraphLab: Amazon EC2 instance with 4 vCPUs and 7.5GB memory with 20GB disk space.

7.3. Evaluation Measure

All the above datasets are run on both Neo4j and GraphLab systems and performance evaluation is done using **Execution Time**. Time of execution is calculated starting from the beginning of the clustering algorithm till the end.
Execution time does not include the time taken for the dataset to load into the database systems. On an another note, GraphLab is a graph processing system which loads the dataset into memory and is really quick, for example, a dataset of 100,000 edges takes around 5 seconds to load into memory, whereas Neo4j stores each node in its database on a persistent storage. Therefore it takes quite a long time as compared to GraphLab.

8. Results

8.1. Performance

This section presents results of performance comparison between Neo4j and GraphLab.

- Figure 6 and 7 show the comparison between Neo4j and GraphLab with varying number of edges in the dataset. GraphLab performs better than Neo4j over the entire range of dataset generated.
- Similar comparison with the number of nodes per dataset is shown in plots 8 and 9. Over the small dataset, the relative difference between GraphLab and Neo4j is high, however it reduces as the data scales up.
- Figure 10 and 11 show that the execution time for a sparse dataset is always smaller than dense datasets. This is expected because dense graphs have high number of neighbors which lead to higher query processing time.

8.2. Cluster Purity

This section introduces one of the external criteria of clustering quality. Purity is a simple and transparent evaluation measure. To compute purity, each cluster is assigned to the class which is most frequent in the cluster, and then the accuracy of this assignment is measured by counting the number of correctly assigned documents and dividing by N. Here, class is the set of clusters we already know in the synthetic input data for clustering and N is the total number of nodes in the graph. Formally

$$purity(\Omega, C) = \frac{1}{N} \sum_{k} max_j |w_k \bigcap c_j| \tag{1}$$

where $\Omega = \{w_1, w_2, ..., w_k\}$ is the set of clusters and $C = \{c_1, c_2, ..., cj\}$ is the set of classes. High purity is easy to achieve when the number of clusters is large - in particular, purity is 1 if each document gets its own cluster. However, we are getting number of resulting clusters almost equal to the number of gold standard clusters. Hence this measure is reasonable here to judge performance of our clustering algorithm.
NOTE: We know the number of actual clusters in the synthetic data generated from networkx. Therefore we will call it as gold standard data.

- Performance of our implementation is better on dense graph as compared to sparse graph, which is expected considering nodes are scattered in sparse graph.
- Average clustering purity for dense graph having number of edges $> 200,000$ is 0.92.
- Average clustering purity for sparse graph having number of edges $> 200,000$ is 0.84.
- For smaller datasets having edge counts between $20,000$ to $100,000$, we are achieving nearly perfect clustering. Clustering coefficient for them is > 0.98 with same number of clusters/ classes in output as in the gold data.

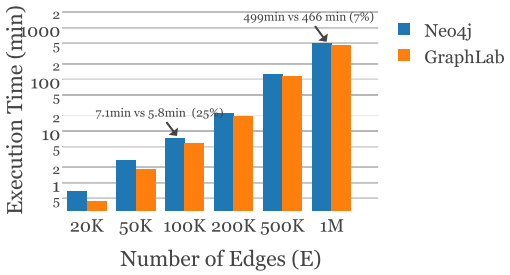

Figure 6: Dense datasets - Neo4j vs GraphLab

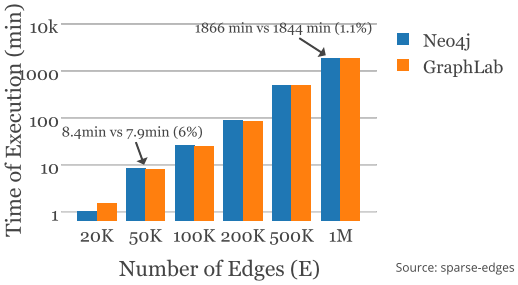

Figure 7: Sparse datasets - Neo4j vs GraphLab

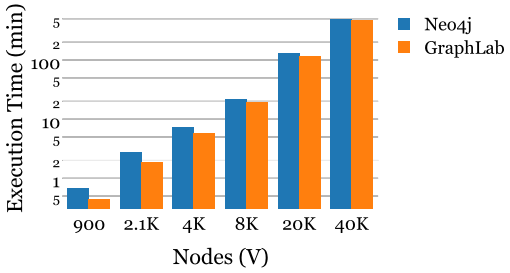

Figure 8: Dense datasets - Neo4j vs GraphLab

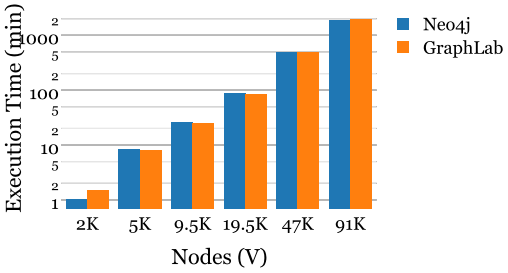

Figure 9: Sparse datasets - Neo4j vs GraphLab

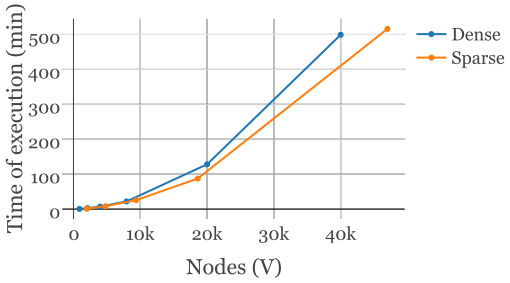

Figure 10: Neo4j - Dense vs Sparse datasets

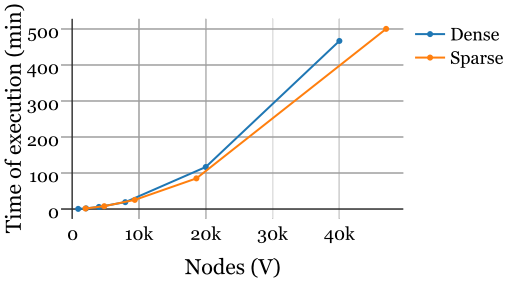

Figure 11: GraphLab - Dense vs Sparse datasets

8.3. Clustering on real world dataset: Facebook

This section presents clustering performance of real world social network Facebook [13] from Stanford Large Network Dataset Collection [12].

Figure 12: Facebook Dataset - Before Clustering

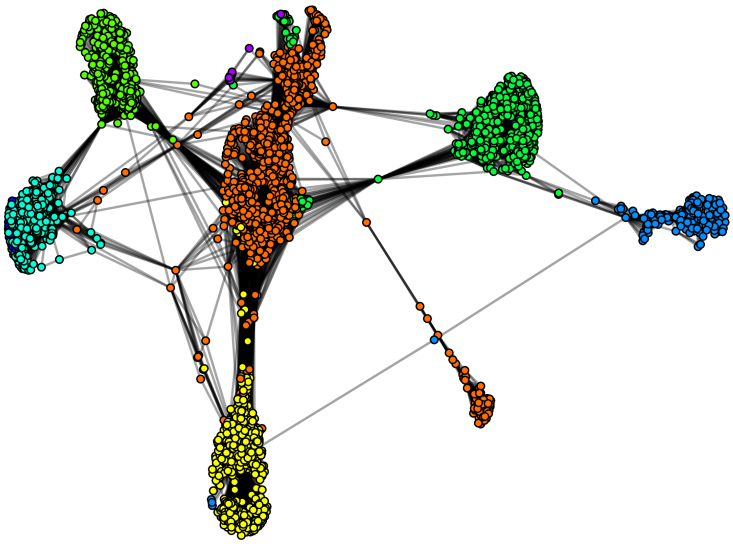

Figure 13: Facebook Dataset - After Clustering (Neo4j/GraphLab)

Facebook Dataset Details:

- Nodes: 4039
- Edges: 88234
- Average clustering coefficient: 0.6055
- Number of triangles: 1612010

Facebook Dataset Results:

- Total number of clusters detected: 8
- Total Execution Time (Neo4j): 11.3 minutes
- Total Execution Time (GraphLab): 9.5 minutes
- List of clusters with number of nodes in them:
  Cluster-1 Nodes:1368
  Cluster-2 Nodes:770
  Cluster-3 Nodes:753
  Cluster-4 Nodes:590
  Cluster-5 Nodes:316
  Cluster-6 Nodes:211
  Cluster-7 Nodes:21
  Cluster-8 Nodes:10
- Figure 12 shows the graph of unclustered Facebook dataset.
- Figure 13 shows the graph clusters with each color representing one cluster.

9. Conclusion

- As we can observe from the above results for synthetic data, GraphLab outperforms Neo4j on almost all datasets. However there is not huge difference between the performance of the two systems. Neo4j's performance is very close to GraphLab's.
- As data scales up, absolute difference between times of execution of Neo4j and GraphLab increases but relative difference between them decreases. For big data processing tasks where the whole data doesn't fit into memory, graph databases (Neo4j) are the clear choice.
- The tests on small dataset of Facebook shows robustness of our implementation. We were able to identify communities within the real world social network data to quite an extent with very little noise.
- We conclude that Neo4j provides a good combination of persistent storage and computational power. If we do not consider the high cost of loading data into the Neo4j, it is comparable to GraphLab in graph processing performance. However, we haven't explored the multiprocessing aspect of these two systems, which would be an interesting future work.

References

[1] William H. E. Day, Herbert Edelsbrunner "Efficient algorithms for agglomerative hierarchical clustering methods".

[2] Chris Fraley "Algorithms for model-based Gaussian hierarchical clustering".

[3] [3] CF Olson "Parallel algorithms for hierarchical clustering".

[4] M Steinbach, G Karypis, V Kumar "A comparison of document clustering techniques".

[5] Y Zhao, G Karypis "Evaluation of hierarchical clustering algorithms for document datasets".

[6] S Guha, R Rastogi, K Shim "CURE: an efficient clustering algorithm for large databases".

[7] J. McAuley and J. Leskovec. "Learning to Discover Social Circles in Ego Networks".

[8] http://istc-bigdata.org/index.php/benchmarking-graph-databases.

[9] N. Mishra, R. Schreiber, I. Stanton, R. Tarjan Clustering Social Networks.

[10] P.Held, K.Dannies Clustering on Dynamic Social Network Data.

[11] J Huang, H Sun "A Structural Clustering Algorithm for Detecting Hierarchical Communities in Networks"

[12] http://snap.stanford.edu/data

[13] http://snap.stanford.edu/data/egonets-Facebook.html

[14] http://en.wikipedia.org/wiki/Clustering_coefficient

[15] http://en.wikipedia.org/wiki/Hierarchical_clustering

[16] http://networkx.github.io

[17] http://www.neo4j.org

[18] http://graphlab.com