Distributed Graph-Based Clustering for Network Intrusion Detection

Corbin McNeill  
Computer Science Department  
Wheaton College, IL  
Wheaton, Illinois 60187  
corbin.mcneill@my.wheaton.edu

Enyue Lu  
Department of Computer Science  
Salisbury University  
Salisbury, Maryland 21801  
ealu@salisbury.edu

Matthias Gobbert  
University of Maryland, Baltimore County  
Department of Mathematics and Statistics  
Baltimore, Maryland 21250  
gobbert@umbc.edu

Abstract—In order to process large volumes of network traffic data and quickly detect intrusions, we use parallel computation frameworks for Network Intrusion Detection Systems (NIDS). Additionally, we model network data as graphs to highlight the interconnected nature of network traffic and apply unsupervised graph-based clustering to flag anomalies as network intrusions. We particularly examine the effectiveness of barycentric clustering on graph network traffic models for intrusion detection. The clustering algorithms have been implemented in Hadoop MapReduce for the purpose of rapid intrusion detection. Furthermore, k-nearest neighbor graphs (kNNGs) are used to optimize the clustering process. We find that across various data sets, unsupervised graph-based clustering is able to exceed a 92% intrusion detection accuracy and that kNNGs can effectively optimize the network traffic graphs for larger data sets.

I. INTRODUCTION

Network Intrusion Detection Systems (NIDS) have been used in order to monitor network traffic. One type of intrusion detection is anomaly detection, which uses unsupervised methods to detect anomalous network intrusions.

Methods of graph-based anomaly detection have been explored and applied to network intrusion detection; however, no parallel computation approaches exist. We examine the effectiveness of graph-based clustering in Hadoop MapReduce for network intrusion detection.

II. NETWORK DATA

The KDD-99 cup’s test data set was used for all conducted experiments. The KDD-99 data set simulates Local Area Network (LAN) traffic, [1] and contains over 310,000 network records, a majority of which are network intrusions. [2]

In order to perform unsupervised anomaly detection on the test data set, intrusions must be statistical anomalies within the data set. Many Denial of Service (DOS) were removed by filtering the data set to only include records of network connections that made successful logins.

The filtered data set contained 53,645 records of which roughly 20% were network intrusions. All experiments used a random sampling of records from this filtered data set.

III. GRAPH CREATION

In order to model network traffic as a graph, we construct a similarity graph. In this weighted undirected graph, vertices represent network connection records. Edges weights are outputs of a similarity functions representing the similarity (or lack of similarity) between the two records that edges connect. Edge weights are calculated by the radial basis function measure:

\[ w_{i,j} = \frac{1}{\exp(||\vec{v}_i - \vec{v}_j||^2)} \]

where \( \vec{v}_x \) is the data vector associated with the \( x^{th} \) network record and the \( x^{th} \) graph vertex. [1, p. 25] This similarity measure is inspired by the Euclidean distance between points in space. All continuous network record features are scaled to the range \([0,1]\) and symbolic features are assigned a distance of 0 if identical, otherwise 1. By this calculation, network records with high similarity will be connected by an edge with a higher weight.

This graph model can employ different edge densities ranging from a complete graph to a sparse graph.

A. Complete Graph

The complete graph approach creates \( \frac{|V|(|V|-1)}{2} \) edges, and any graph clustering algorithm which accounts for all edges will therefore have a running time of \( O(|V|^3) \), where \(|V|\) is the number of network records. This makes such an approach inconvenient for processing large numbers of network records.

Complete graphs can be created by the following algorithm in MapReduce.

Algorithm 1 Full Graph Creation

1: procedure MAP(v1) \triangleright match vertices
2: vertices ← all vertices from distributed cache
3: for \( v2 \) in vertices do
4: \hspace{1em} if \( v1 > v2 \) then
5: \hspace{2em} emit: \((v1, v2)\)
6: procedure REDUCE(v1, list{vadj}) \triangleright calculate weight
7: for \( v2 \) in vadj do
8: \hspace{1em} emit: \((v1, v2, weight(v1, v2))\)

B. K Nearest Neighbor Graph

K Nearest Neighbor Graphs (kNNGs) can serve an approximation of complete graphs for graph clustering. [3] A kNNG...
is a graph where edges exist only between every vertex and its $k$ nearest neighbors, where $k$ is some constant. This allows the graph to store only edges which will have a significant effect during clustering. This approach will create graphs that contain no more than $k|V|$ edges.

KNNGs were formed from fully connected graphs using two additional MapReduce jobs. The first map function bins every edge by each vertex. The first reduce emits each vertex’s $k$ highest weighted edges. The second map function is the identity function. The second reduce function removes duplicate edges.

Table I shows the extent to which KNNGs reduce the number of edges for graph clustering algorithm to consider.

<table>
<thead>
<tr>
<th>Iterations</th>
<th>Graph Edges Remaining</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Full Graph: 497k</td>
</tr>
<tr>
<td></td>
<td>kNNG: 32k</td>
</tr>
<tr>
<td>1</td>
<td>Full Graph: 348k</td>
</tr>
<tr>
<td></td>
<td>kNNG: 20k</td>
</tr>
<tr>
<td>2</td>
<td>Full Graph: 322k</td>
</tr>
<tr>
<td></td>
<td>kNNG: 13k</td>
</tr>
<tr>
<td>3</td>
<td>Full Graph: 300k</td>
</tr>
<tr>
<td></td>
<td>kNNG: 9k</td>
</tr>
<tr>
<td>4</td>
<td>Full Graph: 282k</td>
</tr>
<tr>
<td></td>
<td>kNNG: 5k</td>
</tr>
<tr>
<td>5</td>
<td>Full Graph: 277k</td>
</tr>
<tr>
<td></td>
<td>kNNG: 3k</td>
</tr>
</tbody>
</table>

**TABLE I** The number of edges remaining after various barycentric clustering iterations. $|V| = 1000$ and $k = 50$.

**IV. Clustering**

After similarity graphs were formed, barycentric clustering was used to partition network records into various clusters. Barycentric clustering is an algorithm for clustering undirected graphs motivated by the physical concept of vertices, connected by springs, iteratively affecting each others positions. Barycentric clustering identifies edges with abnormally low edge weights removes them. [4] After enough iterations, multiple connected components will separate. These remaining connected components partition the network records into closely related clusters.

Barycentric clustering was implemented in Hadoop MapReduce to distribute storage and computation. [5]

**V. Intrusion Classification**

Once clusters have been formed, classification algorithms are used to determine which clusters contain intrusive network records and which clusters contain normal network records.

An optimal classification strategy appeared to be classifying every vertex record within the largest connected component as non-intrusive and classifying all other vertex records as intrusive.

**VI. Detection Accuracy**

The effectiveness of graph-based clustering and intrusion classification can be evaluated by via detection accuracy. Detection accuracy is calculated as:

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

where $TP$ is the number of true positives, $FN$ is the number false negatives, etc. In general terms, accuracy is the percent of properly classified records.

Fig. 1 shows the detection accuracy of different data sets during graph clustering. Table II highlights the detection accuracy of KNNGs after 2 iterations and full graphs after 5 iterations. The selection of these number of iterations allowed the highest detection accuracy.

**VII. Conclusion**

We find that graph-based clustering algorithms serve as an effective method for unsupervised network intrusion detection with detection accuracy consistently above 92%. Additionally graph creation and graph clustering can be distributed via Hadoop MapReduce and optimized with KNNGs in order to quickly detect network intrusions. KNNG optimizations appeared effective for larger data sets.

**ACKNOWLEDGMENT**

This work is funded by the National Science Foundation under the Research Experience for Undergraduates Program.

**REFERENCES**