U.S. NAVAL ACADEMY
COMPUTER SCIENCE DEPARTMENT
TECHNICAL REPORT

Clustering Systems with Kolmogorov Complexity and MapReduce

Troisi, Louis R

USNA-CS-TR-2011-01

June 2, 2011

USNA Computer Science Dept. ∘ 572M Holloway Rd Stop 9F ∘ Annapolis, MD 21403

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **02 JUN 2011**	2. REPORT TYPE	3. DATES COVERED

4. TITLE AND SUBTITLE **Clustering Systems with Kolmogorov Complexity and MapReduce**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **United States Naval Academy,Computer Science Department,572M Holloway Rd,Annapolis,MD,21402**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR’S ACRONYM(S)
	11. SPONSOR/MONITOR’S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

In the field of value management, an important problem is quantifying the processes and capabilities of an organization’s network and the machines within. When the orga- nization is large, ever-changing, and responding to new demands, it is difficult to know at any given time what exactly is being run on the machines. Accordingly, one could lose track of approved or, worse, not approved or even malicious software, as the machines become employed for various tasks. Moreover, the level of utilization of the machines may affect the maintenance and upkeep of the network. Our goal is to develop a tool that can cluster the machines on a network, in a meaningful way, using different attributes or features, and it does so autonomously, in an efficient and scalable system. The so- lution developed implements, at its core, a streaming algorithm that in real-time takes meaningful operating data from a network, compresses it, and sends it to a MapReduce clustering algorithm. The clustering algorithm uses a normalized compression distance to measure the similarity of two machines. The goal for this project was to implement the solution and measure the overall effectiveness of the clusters. The implementation was successful in creating a software tool that can compress, determine the normalized compression distance and cluster the machines. More work however, needs to be done in using our system to extract more quantitative meaning from the clusters generated.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES **19**	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**			

Clustering Systems with Kolmogorov Complexity and MapReduce

Louis R Troisi
Dept. Computer Science, U.S. Naval Academy
572M Holloway Rd, Annapolis, MD 21402
louistroisi7@gmail.com

Abstract

In the field of value management, an important problem is quantifying the processes and capabilities of an organization's network and the machines within. When the organization is large, ever-changing, and responding to new demands, it is difficult to know at any given time what exactly is being run on the machines. Accordingly, one could lose track of approved or, worse, not approved or even malicious software, as the machines become employed for various tasks. Moreover, the level of utilization of the machines may affect the maintenance and upkeep of the network. Our goal is to develop a tool that can cluster the machines on a network, in a meaningful way, using different attributes or features, and it does so autonomously, in an efficient and scalable system. The solution developed implements, at its core, a streaming algorithm that in real-time takes meaningful operating data from a network, compresses it, and sends it to a MapReduce clustering algorithm. The clustering algorithm uses a *normalized compression distance* to measure the similarity of two machines. The goal for this project was to implement the solution and measure the overall effectiveness of the clusters. The implementation was successful in creating a software tool that can compress, determine the normalized compression distance and cluster the machines. More work however, needs to be done in using our system to extract more quantitative meaning from the clusters generated.

1 Introduction

Within the value management field, there is a need to know what resources, hardware and software, are available to match the needs of the organization. The difference between needed resources and existing resources should be minimized. All organizations desire to minimize error when estimating current needs, or predicting the needs for future growth. This is increasingly difficult when the organization expands. For hardware, it is relatively easy to quantify what exists, as it rarely changes, ignoring degradation. Software, however, can change rapidly and determination of software capabilities tends to be rather ethereal. Moreover, the value isn't just measured in capability, it's also about assessing what components are installed and what kind of effect does that create on the system from a myriad of performance and cost perspectives. For instance: CPU clock rate, power consumption, percent of processor utilized, are just a few things that should be tracked in order to understand if some machines on the network are under a heavy load and could be better load balanced to prevent misuse and provide better long-term results.

The objective of this work is to develop an autonomous system that can determine the capabilities of a network without having to label the components within it beforehand (unsupervised learning). The system allows the user to quickly gather some facts (attributes) about the machines in operation, cluster the machines based on the attributes, and use that model to classify the machines using the gathered data. This goal shouldn't be affected by scale, as networks can be for a company like Yahoo, with tens of thousands of machines each of those with multiple nodes, so the system should be able to work at the "enterprise" level.

This paper focuses on the clustering aspect of the system. The first challenge in developing the system was comparing machines with very different software configuration and load. We chose as similarity metric between machines the *Normalized Compression Distance* or NCD [2]. NCD offers the ability to quantify the similarity between any two objects, based on a description of the objects. The object description can include any set of attributes, making NCD a flexible choice for similarity metric. In our experiments, we used as description the software run-tables of the machines.

The second challenge was choosing a clustering algorithm that works for objects in a non-Euclidean space, and does not need much prior knowledge. Most partitioning clustering algorithms, like K-means, require the number of clusters in advance. We implemented an agglomerative hierarchical clustering algorithm that clusters similar objects together and the final result is a dendogram.

Finally, we aim to build a scalable system, so we implemented our clustering algorithm in the MapReduce framework [4].

Our system is able to autonomously cluster the machines in a network. However, more work needs to be done in creating a more efficient and scalable clustering algorithm as well as classifying new machines.

The rest of the paper is organized as follows: we introduce some background information in Section 2. We detail the goals of this work in Section 3 and describe the implementation in Section 4. We present the experimental results in Section 5 and discuss related work in Section 7. We conclude in Section 8.

2 Background

This section introduces some of the building blocks for our system.

2.1 Cluster analysis

Cluster analysis or data segmentation is concerned with the segmentation of various objects into subsets or clusters such that the objects within a cluster are much more closely related to all the objects within the cluster versus the other objects that do not fall within the same cluster. An important step in cluster analysis is choosing a definition for similarity to quantify the similarity or distance between objects [5]. The clusters are then defined based on this similarity metric. There are two major types of clustering algorithms: partition-based and hierarchical. The algorithms most commonly used for each type are K-means and agglomerative, respectively. Briefly, partitioning involves picking cluster centers in a distance matrix and objects are assigned to the cluster with the closest center. Cluster centers are then re-computed and the process is iterated until some stopping criteria is reached. In hierarchical clustering, a binary tree is constructed, where the nodes of the tree are clusters, with the leaves being the objects to be clustered (clusters of size 1) and the root being a cluster containing all the objects. In agglomerative clustering, construction starts at the leaves. In each step, the 2 most similar clusters are grouped together to form the cluster at the higher level, until there is only one cluster containing all objects. Section 4 discusses the agglomerative hierarchical clustering algorithm implemented in our system.

2.2 Normalized Compression Distance

As discussed in the previous section, an important part of cluster analysis is choosing a similarity metric that captures the similarity between two objects. *Normalized compression distance* or NCD offers us the ability to quantify the relationship between objects (including machines on a network) such that one can determine that while objects a, b and c are similar, b is best described as the same type of object as c by maximizing the total number of attributes described. This means that NCD attempts to use a full description of each object, compressed, giving it the maximum number of possible descriptors to distinguish between objects without the burden of an enormous table of attributes. NCD is a practical application of *Kolmogorov Complexity* named after Andrei Kolmogorov, due to his study in computing randomness[8]. Kolmogorov Complexity itself is an offspring in the study of randomness. Kolmogorov postulated that an object (whether a string or a picture)'s randomness is determined by the minimum amount of instructions or code required to compute or generate that object. For example, the object '1010101010' could be simplified as '10' repeated 5 times. Declaring an object s to be totally random requires $C(s) \geq s$, where $C(s)$ is the complexity required to describe object s[8]. Knowing this, the distance metric for Kolmogorov Complexity between two objects is the amount of supplemental information needed to describe x given y. Using the example above, if given '1010101010' as x and '10' as y the inference $x = y * 5$ can be made. This knowledge leads to the formula for NCD, which attempts to formalize the description previously given, as

$$NCD(x,y) = \frac{C(x,y) - min\{C(x), C(y)\}}{max\{C(x), C(y)\}} \tag{1}$$

Most accurately, NCD is an approximation of an optimality measure of $C(s)$ discovered by Vitányi which is described in *Algorithmic Clustering of Music Based on String Compression:*

> Roughly speaking, two objects are deemed close if we can significantly "compress" one given the information in the other, the idea being that if two pieces are more similar, then we can more succinctly describe one given the other[3].

2.3 Hadoop

There are two major benefits to using Hadoop suite in terms of maximizing scalability: MapReduce and HDFS (Hadoop File System). Both are based on a series of papers at Google to explain how accomplishes the task as the largest search engine by improving scalability. This allows them to maintain and process terabytes of data in a redundant and fail-safe package. The MapReduce framework was developed at Google and was introduced in 2004 by Dean et al. [4]. The paradigm accomplished many goals that frustrated previous attempts at large-scale distributed computing. It could be accomplished on commodity machines and was very scalable. Secondly, utilizing the *map* function from programming languages like Scheme or Haskell allows users to run a function on a key-value pair and output an intermediate key-value pair. The *reduce* function combines all pairs with the same key to do some combining or other function associated with *reduce* in functional programming[4]. Formulaically MapReduce can be described as :

$$Map : (k1, v1) \rightarrow list(k2, v2) \tag{2}$$

$$Reduce : (k2, list(v2)) \rightarrow list(v3) \tag{3}$$

The independence of the map as a functional method allows no side effects (meaning that each key-value pair is independent of the other pairs in the list so you don't have to worry about a logical processing order) so multiple key-value pairs could be "mapped" at the same time allowing a distributed system that was intuitive to any user who has used a functional programming language. Many problems have been solved at Google once they attempted to make it fit into the MapReduce format[4]. After the MapReduce paper in 2004, a software project named Hadoop started by Cloudera, with Yahoo, developed an open-source software alternative to the proprietary version developed at Google. This allowed the Google system to become openly and freely available as the design was an attempt to replicate the system[12].

HDFS or Hadoop File System is also based on a Google paper, *The Google file system*. HDFS aims to replicate the functionality of GFS in an open source environment based on the article[10, 12]. Without delving to heavily into the architecture, it allows for extremely large files stored across commodity hardware. The files are append or read-only mostly with a write once and never expect for overwrites[10]. This may sound limited, but webcrawlers, log files on a server, the system this report is describing, all use the write once append many style of document creation[10].

3 Goals

The goal of this project is to develop a system that allows us to flexibly cluster various forms of text data without regard for structure or a requirement in terms of attributes. To summarize:

Flexible as the attributes or input changes, the system should not require drastic reconfiguration to work effectively

Scalable the system must be able to scale or otherwise adapt to the need of both large and small networks

Effective the observed network must be accurately described and in such a way that useful comparisons in the future can be made, otherwise clustering will be difficult and generally inaccurate

3.1 Flexible

The way this project achieves flexibility is through the use of NCD as the metric to determine the similarity between objects. The fact that NCD relies on compression creates a benefit because, as a rule, a compressed file could have originally been anything so it now becomes context-free; when compressed file a is compared in a predicate fashion such as a in terms of b you can make determinations about the relationship between a and b .

3.2 Scalable

Creating an efficient implementation of our algorithm through streaming software, MapReduce and utilizing the Hadoop File System improves the scalability of the system. The distributed nature of the HDFS and MapReduce should allow Tera and PetaByte level storage possible[10]. Additional confidence in using Hadoop comes from the many organizations that rely on or use a similar framework to Hadoop to provide a personalized and quickly retrievable user experience like Twitter, Facebook, Stumbleupon[12].

3.3 Effective

Effective, is a summarization of two ideas, efficient and accurate; if the system lacks one of these two it becomes either useless or limited in scale. All consideration is taken towards determining where the most accurate hierarchical chaining of clusters will be found which is discussed further in the results and conclusion about the work done in this course.

4 Methods

4.1 System Model

In this paper, we describe and implement a model system fitting the requirements described in Section 3. Our goal is to cluster machines in a network. We use as description of a machine the software run-table extracted off each machine in the network, converted into a format very similar to the ps aux command available in Unix, by a Java Messaging Service. This description is then compressed using LZW [8], a common compression algorithm, in the IBM's Infosphere Streams system. The now compressed run-tables are then compared in to determine the pair-wise NCD values. These values are then stored into a symmetrical similarity matrix containing each machine in the network. Our MapReduce clustering algorithm uses the similarity matrix as its input to produce clusters. Figure 4.1 shows the data flow in our system.

4.2 Design Considerations

We implemented in the MapReduce framework a hierarchical algorithm based off the traditional algorithm described in Section 2.1. The dendrogram created by the clustering algorithm becomes the model for a classifier, so that further machines can be "learned" based off the seed model.

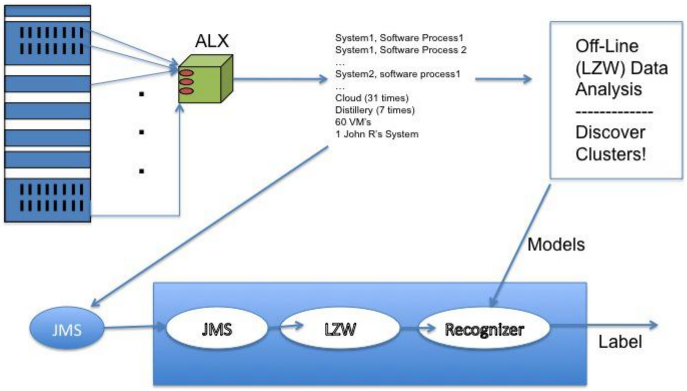

Figure 1: A visual description of the system starting from the machines through the processing engine

For this project, a hierarchical agglomerative algorithm was determined to be the preferred choice. The main problem with a partition based clustering scheme like K-means or K-medoids is that K, the final number of clusters, must be defined. The project specifications were such that if we were to develop a system that used a partition based scheme we would be relying on some beforehand knowledge of the system when choosing K.

In hierarchical clustering, a complete dendogram, or binary tree is produced. In agglomerative clustering, at each step, the two closest clusters are merged. There are many ways of defining the "closest" clusters. Linkage is based on the notion of intercluster dissimilarity, meaning that the differences between points in a cluster have to be considered in choosing a definition for "closest" clusters. In single linkage, the two closest points of different clusters are used as distance between clusters. Complete linkage uses the two objects farthest apart within two different clusters in an attempt to prevent "chains" or elliptical clusters versus the intuitive spherical clusters. Average linkage uses the average distance between any two points in different clusters. We define the distance between two clusters as being the distance (NCD value) between their medoids [7], where the medoid of each cluster is the machine with the lowest average NCD to all the other machines in the cluster. The linkage used aims to maximize the benefits of each type of linkage, with a possible small increase in time cost, by measuring the distance between the Euclidean average of each cluster in comparison with other clusters, and merging the clusters with the lowest distance.

4.3 Implementation

We implemented an iterative MapReduce hierarchical clustering algorithm. Figure 2 shows the pseudo-code for our algorithm. The input for the algorithm is a comma-separated-values CSV file containing the matrix with the pair-wise NCD values between machines. The algorithm, as we implemented it, contains two maps and a single reduce. Our algorithm begins with the one time use map1 function for pulling data from the CSV file and emitting (NULL, c) key-value pairs with a singleton cluster c for each machine. This is designed to initialize the data structures used by the algorithm. Once the initial data structures are created, the algorithm iterates until a desired completion criteria is met, for example when total dendrogram is completed (until all objects are merged into one cluster). Each iteration executes a map and a reduce. In this second Map function, each cluster determines the cluster which is most similar to. In the reduce phase, the overall two closest clusters are determined and merged. The reduce then outputs the new clusters and a tuple describing the clusters that were merged. The output of the reduce function is used as the input of the map function in the next iteration.

4.3.1 Data Structures

The data structures used by our algorithms are briefly described next.

A *machine* as showed in Figure 3 represents the objects that are clustered.

A *cluster* as shown in Figure 4 represents the cluster of "similar" objects.

A *triple* as shown in Figure 5 describes the 2 clusters that merged in a particular iteration of the agglomerative clustering algorithm.

4.3.2 Map1

As seen in Figure 2, the first map is used only once in implementation. The main goal for each map1 instance spawned by Hadoop is to take a single line from the input CSV file and convert the data into clusters containing one machine. The pseudo-code for map1 is given in Figure 6.

```
//read in tokens from the comma separated list
//use these tokens to generate singleton clusters
map1:<machineID, list of NCD to all other machines> -->
     <null,singleton cluster c>;

while{!done}
{
 //for-each cluster, determine the cluster "closest" to it
 //if this is not the first iteration, distances to
 //other clusters might need to be updated

 //output a null key and the updated cluster
 map2: <(oldCluster1, oldCluster2, newMergedCluster), c> -->
     <null, (c, c-closest)>;

 //find the 2 closest clusters
 //merge the two clusters

 //output a description of the clusters that merged as a key and
 //the possibly new clusters as the value
 reduce: <null, list of (c, c-closest)> -->
    list of <(oldCluster1, oldCluster2, newMergedCluster), updated c>;

}
```

Figure 2: MapReduce agglomerative hierarchical clustering algorithm

```
machine{
 machineID; //unique id for each machine
 ncds; // list of NCD values from this machine to all other machines }
```

Figure 3: Machine data structure

```
cluster{
 clusterID; //unique id for the cluster
 machines; //list of machines in cluster
 medoidID; //machineID of the machine in the cluster that has
    //the smallest average NCD to all the other machines in the cluster
 medoids; //list of medoidIDs for all the other clusters
 closestMedoidID; //medoidID of the closest cluster}
```

Figure 4: Cluster data structure

```
triple{
 mID1; //old medoidID of one of the merged clusters,
 mID2; //old medoidID of the other merged cluster,
 newmID; //medoidID of the new combined clustered }
```

.

Figure 5: Triple data structure

```
Input: <line number, row in the similarity matrix>
Output: <triple(−1,−1,−1), a cluster of one machine>
Pseudo−code Map1:

//each line of the file corresponds to an individual machine
parse the comma–separated values in the line

use those values to fill in the data about the individual machine;
//machineID for this machine is the input line number

//each cluster has one individual machine in the
//initial step of agglomerative clustering
//clusterID is the machineID
//medoidID is the machineID
//medoids contains the list of all other machine IDs
//closestMedoidID is computed
output a key−value pair with a dummy triple key (−1,−1,−1) and
the cluster containing one machine as value
```

Figure 6: Map1: read from input file and initialize data structures

4.3.3 Map2

The second map function described in Figure 7 is executed in each iteration of the clustering algorithm. In this map, each cluster updates its list of current clusters based on the previous merge, and re-computes (if needed) the new "closest" cluster, meaning what cluster is it most similar with currently. The code executed for each cluster depends on what occurred in the previous merge (reduction). The map outputs a null key ensuring that all ⟨Key,Value⟩ pairs are properly sent to the same reducer for determining the clusters to merge. The value output by the map is the input cluster, with the internal values (medoids and closestMedoidID) updated.

4.3.4 Reduce

The reduce function has as input the list of all the clusters. The reduce determines the two closest clusters and merges them (the cluster with the smaller id is enlarged to encompass all the objects in the two clusters, and the cluster with the larger id is eliminated). The reduce outputs a list of key (a triple describing the merged clusters)-value (cluster) pairs.

```
Input: <triple(oldMedoid1,oldMedoid2,newMedoid), cluster c>
Output: <null, cluster c with updated values>
Pseudo-code Map 2:

//if this is first iteration of clustering algorithm,
//just output the cluster, since it already has the correct closestMedoidID
if (oldMedoidID == -1){
 output <null, c>;
 return;
}

//if this is not the first iteration
//update the current list of medoids
//and re-compute closestMedoidID if needed
switch {
 case 1: if this cluster increased in size during the last iteration
        //oldMedoid2 represents the cluster removed due to merge
           remove oldMedoid2 from c.medoids
           compute the new closest cluster medoidID c.closestMedoidID
 case 2: if this cluster was not involved in the merge during last iteration
         and oldMedoid1 = newMedoid
           remove oldMedoid2 from c.medoids
           compute the new closest cluster medoidID c.closestMedoidID
 case 3: if this cluster was not involved in the merge during last iteration
         and oldMedoid2 = newMedoid
           remove oldMedoid1 from c.medoids
           compute the new closest cluster medoidID c.closestMedoidID
 case 4: if this cluster was not involved in the merge during last iteration
         and oldMedoid1 != newMedoid and oldMedoid2 != newMedoid
           remove oldMedoid1 from c.medoids
           remove oldMedoid2 from c.medoids
           add newMedoid to c.medoids
           compute the new closest cluster medoidID c.closestMedoidID
}
output the modified cluster c: <null, c>;
```

Figure 7: Map2: For each cluster, update the cluster list based on previous merge and find the current cluster closest to it

The pseudo-code for the Reduce function is given in Figure 8.

5 Results

In reference to the goals outlined in the paper, the project was successful. It was possible to cluster the machines using the methodology describe above. However, some of the goals of the larger project which involved utilizing the model created to classify new machines, were for reached. A large part of this is due to the difficulties in properly implementing the agglomerative clusters in MapReduce, which limited the amount of testing possible. This suggests that perhaps the idea of agglomerative clustering, while an important type of clustering, might be of limited use in a MapReduce paradigm.

5.1 Discussion

The project achieved its theoretical goals: to cluster data using Kolmogorov Complexity and do it without emphasis on features and utilizing an efficient method. In reference to the three goals discussed: flexibility, scalability and effectiveness, the project achieved results in all three areas. That does not mean there were not some shortcomings, but overall the project had shown success and just as importantly, potential for more success if continued further.

One important thing to note is that the algorithm ran in $O(n)$ time as shown in Figure 5. This shows good promise that this implementation would scale well for larger data sets. This was a pleasant surprise, as we expected that the use of relatively small datasets would make the overhead cost of Hadoop a large factor. Due to the time it took to implement the clustering algorithm in the MapReduce framework, limited time was available to tune the system, and tuning the system is noted as an important part to most efficiently use Hadoop [12].

6 Future Work

More work needs to be done to increase the runtime efficiency of the MapReduce implementation as well as to determine a good heuristic for the stopping point of the agglomerative clustering algorithm. One way of improving the MapReduce clustering algorithm would be to implement a self organizing algorithm such as a Quality Threshold algorithm [6] as it has several advantages: the clusters must be of a minimum size defined by the data or, if necessary, by the user, the number of clusters is not specified a priori, and individual ele-

```
Input: <null, list of all clusters: clusterList>
Output: <triple(oldMedoid1,oldMedoid2,newMedoid), cluster c>
Pseudo-code Reduce:

//find the two closest clusters
minNCD = 1;
foreach(cluster c in clusterList) {
 //if distance between cluster and its nearest neighbor < minimum change the
 if (c.medoidID.ncds(closestMedoidID) < minNCD){
  minNCD = c.medoidID.ncds(closestMedoidID);
  //remember clusters to be merged
  oldMedoid1 = c.medoidID;
  oldMedoid2 = c.closestMedoidID;
 }
}

//merge the two closest clusters
c1 = cluster with medoid oldMedoid1;
c2 = cluster with medoid oldMedoid2;
c1.machines = c1.machines + c2.machines;

//remove c2 from the list of clusters
clusterList     = clusterList - c2;

//determine the medoid for the enlarged cluster
current average = 1;
foreach(machine m in c1.machines){
 determine its average ncd against all other machines in the cluster
 if(the average is smaller than the current average){
  //make that machine the new medoid
  c1.medoidID = m.machineID;
  set the current average to the average calculated for the current center
 }
}

//output
foreach(cluster c in clusterList){
 output <triple(oldMedoid1,oldMedoid2,c1.medoidID), c>
}
```

Figure 8: Reduce: Find the two closest clusters and merge them

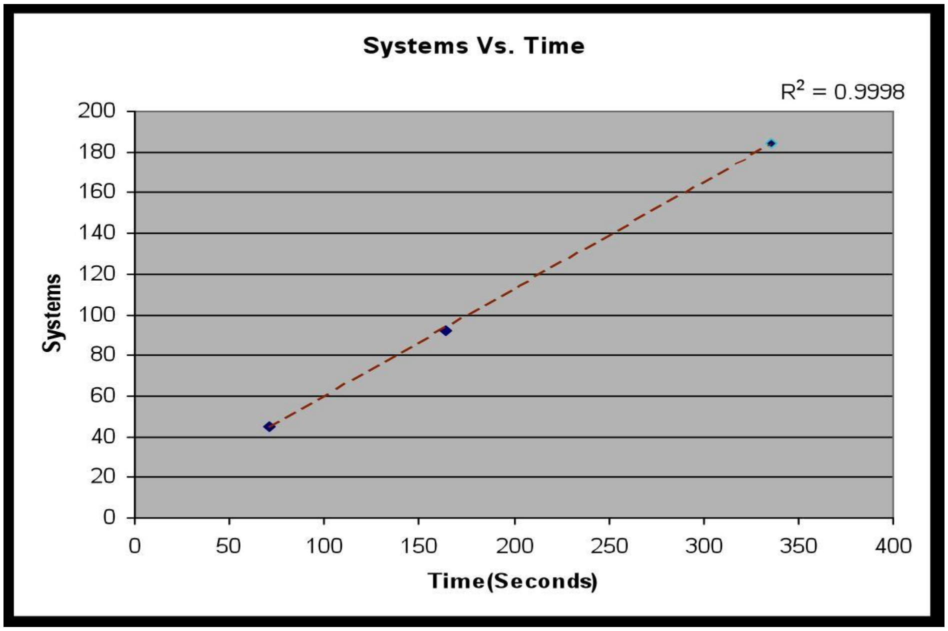

Figure 9: A display of runtimes for various size sets of data. Note the linear runtime

ments that don't cluster are removed, thereby removing some of the noise that could make clustering difficult.

Another algorithm worth implementing in the MapReduce environment would be the OPTICS algorithm [1] as it provides many of the benefits of the Quality Threshold algorithm while producing a dendrogram like in hierarchical clustering. A dendogram could be an effective model to use for classification and reveal more about the structure of similarities between machines in the network.

Another direction for future work would be to label the machines when placed on the network and then as they evolve over time use that to determine how different in behavior (or NCD) two machines of the same type could become. This is an idea in response to the fact that when a machine is purchased and set-up, it is often quite obvious what its intended purpose or use is. Measuring the evolution of the machines would allow better analysis of the evolving needs of an enterprise.

Last, a desired future goal would be to develop a method to approximate an ideal number of clusters K and use a partition clustering algorithm. In the MapReduce framework, a partitioning based scheme holds a natural edge when implemented due to the intuitiveness and the ease of programming versus a hierarchical method due to the need to iterate n times with n objects. However, a caution towards using that method is the need for the user to declare K. Until an effective way to estimate the optimal K exists, using K-means and NCD in areas where the desired K is uncertain can be prohibitively expensive.

7 Related Work

The biggest contributor to the shape of this research and whose work inspired its use is Paul Vitànyi, as he developed the NCD metric and then used it for clustering applications in a variety of fields [2, 3]. There are a few differences between the focus of his work and this application. This project is developed to be part of a working environment and scalability is an important concern. The method developed in *Clustering by Compression* [2] describes a technique for clustering described as Quartet Clustering. Quartet Clustering is a hierarchical method designed to explicitly show the attributes of NCD. It is a multi-pass algorithm that, after several random passes for initial clustering determines the "most accurate" dendrogram which becomes the result. It is a important distinction to be made as in [3] sets of data as large as 60 objects are used while this project operated on datasets several times larger.

Another important development to this software system is the work of Pike et al.[9]. They have implemented a MapReduce oriented language designed to deal with large-scale data sets by automating analysis for things like filtering, aggregation. However, they don't implement a cluster analysis in their query language.

Finally, the work of Stephanie Wehner [11] is closely related with ours in terms of environment and usage. She incorporates techniques in Kolmogorov Complexity across a network with Snort to determine by behavior if a worm has occurred previously or if it is a new version of a pre-existing worm that would have gone undetected by a signature-based system. She also discusses the use of NCD in determining all forms of traffic anomalies as compression effectively highlights general modal shifts in traffic patterns. The difference is that she is attempting supervised learning as she is attempting to classify worms based on previously seen behaviors and attributes versus unsupervised learning and then classification which is the focus of this project.

8 Conclusion

The system designed proves that it can successfully cluster machines on a network while generally adhering to the constraints imposed on the system to maintain the goals of flexibility, scalability and effectiveness. The results of running the algorithm on networks of different sizes suggest that out algorithm can handle much larger networks without any serious degradation in performance.

However, giving meaning to a compressed file, even in respect to another compressed file, is difficult, and therefore the task of giving some valuable interpretation to the result produced by clustering may require some knowledge of the inputs. This could constrain the project to less flexibility than the initial level of flexibility desired.

9 Acknowledgements

I would like to thank Dr. Adina Crainiceanu who guided me through the research process and implementation, and Dave Rapp for his continuing support of my computing interests.

References

[1] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics: ordering points to identify the clustering structure. *SIGMOD Rec.*, 28:49–60, June 1999.

[2] R Cilibrasi and P M B Vitanyi. Clustering by compression. *IEEE Trans. Inf. Theory, IEEE Trans. Inf. Theory*, 51(4):1523–1545, April 2005.

[3] Rudi Cilibrasi, Paul Vitányi, and Ronald De Wolf. Algorithmic clustering of music based on string compression. *Comput. Music J.*, 28:49–67, December 2004.

[4] J Dean and S Ghemawat. Mapreduce: Simplified data processing on large clusters. In *In OSDI04: Sixth Symposium on Operating System Design and Implementation*, 2004.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[6] Laurie J. Heyer, Semyon Kruglyak, and Shibu Yooseph. Exploring expression data: Identification and analysis of coexpressed genes. *Genome Res.*, 1999.

[7] L. Kaufman and P.J. Rousseeuw. *Finding Groups in Data An Introduction to Cluster Analysis*. Wiley Interscience, New York, 1990.

[8] W. Kirchherr, M. Li, and P. M. B. Vitànyi. The Miraculous Universal Distribution. *Mathematical Intelligencer*, 1997. bibliographical data to be processed – The Mathematical Intelligencer Vol: 19 Nr: 4 Coden: maindc Issn: 0343-6993 pages: 7–15 1997 – springer – 8.

[9] R Pike, S Dorward, R Griesemer, and S Quinlan. Interpreting the data: Parallel analysis with sawzall. *Sci. Program*, 13:277–298, 2005.

[10] G Sanjay, G Howard, and L Shun-Tak. The google file system. In *Proceedings of the Symposium on Operating Systems Principles*, 2003.

[11] S Wehner. Analyzing worms and network traffic using compression. *Journal of Computer Security*, 15(3), April 2007.

[12] Tom White. *Hadoop: The Definitive Guide*. O'Reilly, first edition edition, june 2009.