<u>Project Report – Team Cinderella</u>

Network attack analysis via *k*-means clustering

Chandni Pakalapati
cp6023@rit.edu

Priyanka Samanta
ps7723@rit.edu

Project Overview

Growth in network-based services and sensitive information over networks have grown tremendously over past years. Securing the network from attacks has become critical. We intend to detect the anomaly in the network traffic data and help preventing the attacks. For the same as a part of our research investigation we would build a model that would help in detecting network anomalies and further parallelize this building procedure.

Networks are prone to different kinds of attacks. We can categorize the records in the following category:

- DoS (Denial of Services): This kind of attack denies legitimate requests made by any client to the system.
- Probing or Surveillance attacks: This kind of attack collects information about the network state.
- User-to-root: It is the unauthorized access by the user to log in as a local super user or root.
- Remote-to-local: This attack explains the unauthorized access to local machines from remote machines.

We intend to build our detection model using k-Means Clustering and NSL-KDD Cup 1999 Dataset.

Dataset Link: https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

For getting the best clustering model, we are considering only 10% of the actual dataset avaialbale at: http://kdd.ics.uci.edu/databases/kddcup99/kddcup.data_10_percent.gz

Computational Problem

k-Means clustering:

- Is an unsupervised learning technique that helps in finding an inherent structures in underlying data

- Is an algorithm that partitions the independent data points into K clusters based on their Euclidian Distance from chosen centroids of the clusters. Euclidean distance is the measure of the dissimilarity between pair of objects. It is computed as

$$d(x, y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$ where x,y are data points with m attributes [1]

- Is an NP hard problem. Time complexity is defined as ***O(i*k*d*n)*** [i: number of iterations, k: number of clusters, d: dimensionality of the data, n: total number of data points]

As k-Means algorithm is a NP hard problem with an extremely high time complexity, our goal is to design and implement a parallelized version of this clustering model or ***parallelize k-Means algorithm*** and compare the sequential and parallel versions of this model along with their performances.

Research Paper Analysis

In the following section we would be providing an analysis of 3 research papers that we studied for our project work. The analysis is divided under following sections:

- Title of the paper
- Problem statement addresses in the paper
- Approach to solve the problem
- Contribution of the paper
- Drawbacks of the paper
- Takeaway from the paper for our implementation

Analysis of Research Paper 1

Title: K-Means Clustering Approach to Analyze NSL-KDD Intrusion Detection Dataset
Author: Vipin Kumar, Himadri Chauhan, Dheeraj Panwar
Journal Name: International journal of soft computing and engineering
ISSN: 2231-2307 **Volume** -3, **Issue** -4,
Date: 09/01/2013
Page Number: 1-4
URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.413.589&rep=rep1&type=pdf

<u>**Problem Statement:**</u> The paper addresses the following problems:

i. Detection of Network Intrusion by performing unsupervised data mining technique of clustering on unlabeled data.
ii. Analysis of the NSL-KDD dataset using K-Means Clustering.
iii. Clustering the dataset into normal and the following network attack types:

- DoS (Denial of Services)
- Probe
- R2L
- U2R

Approach to the problem: k-Means is a simple, unsupervised clustering algorithm that is used to cluster unlabeled data. k-Means algorithm : [2]

Initialization of k prototypes ($w_1, \ldots, w_k$) [k is the number of clusters]
Each cluster C_i is associated with prototype w_i
Repeat
 for each input data record d_i, where i ϵ {1,… n},
 do
  Assign di to the cluster C_j* with nearest prototype w_j
 done
 for each cluster C_j, where j ϵ {1,…, k},
 do
  Update the prototype wj to be the centroid of all samples currently in C_i,
  so that $w_i = \sum d_i \epsilon C_i \ / \mid C_i \mid$
 done

Paper's Contribution: Understanding of the NSL-KDD dataset.

Records in the database consists of the following 41 attributes and a class variable determining the type of the attack.

Duration	src_bytes	dst_bytes	land	wrong_fragment	urgent
num_failed_logins	logged_in	num_compromised	root_shell	su_attempted	num_root
num_shells	num_access_files	num_outbound_cmds	is_host_login	is_guest_login	count
serror_rate	srv_serror_rate	rerror_rate	srv_rerror_rate	same_srv_rate	diff_srv_rate
dst_host_count	dst_host_srv_count	dst_host_same_srv_rate	dst_host_diff_srv_rate	dst_host_same_src_port_rate	dst_host_srv_diff_host_rate
dst_host_srv_serror_rate	dst_host_rerror_rate	dst_host_srv_rerror_rate	protocol_type	service	Flag
hot	num_file_creations	srv_count	srv_diff_host_rate	dst_host_serror_rate	***Class***

Table1: List of attributes

Class variable contains the type of network traffic. It may be normal or one of the following attack types:

DoS	Back, Land, Neptune, Pod, Smurf, teardrop, Mailbomb, Processtable, Udpstorm, Apache2, Worm
Probe	Satan, Ipsweep, Nmap, Portsweep, Mscan, Saint
R2L	Guess_Password, Ftp_write, Imap, Phf, Multihop, Warezmaster, Xlock, xsnoop, Snmpguess, Snmpgetattack, Httptunnel, Sendmail, Named
U2R	Buffer_overflow, Loadmodule, Rootkit, Perl, Sqlattack, Xterm, Ps

Table2: Attack Types

Drawback: This implementation of k-Means Clustering algorithm has following drawbacks:

- It requires the entire dataset to be stored in the main memory.
- Involves high computational complexity. [O(n*k*i*d) n=number of data records, k=number of clusters, i= number of iterations, d= dimensionality of the record]

Takeaway for our project implementation:

- Knowing the attributes to work with.
- Understanding the types of attacks and their belongings to one of the 4 attack categories.
- Prototype to be considered is the centroid of the cluster.
- With the cluster size of 4, the optimal result can be achieved.
- Using Euclidean distance metric to compute the dissimilarity between the data records.

Analysis of Research Paper 2

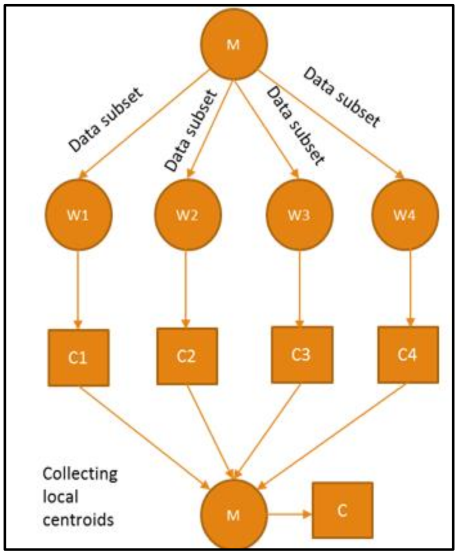
Title: Parallelizing k-means Algorithm for 1-d Data Using MPI
Author: Ilias K. Savvas and Georgia N. Sofianidou
Journal Name: 2014 IEEE 23rd International WETICE Conference
Date: 2014
Page Number: 179 – 184
URL: http://ieeexplore.ieee.org.ezproxy.rit.edu/stamp/stamp.jsp?tp=&arnumber=6927046

Problem Statement: Reducing the computational complexity of the K-Means by implementing parallel K-means.

Approach to the problem: Parallelizing k-Means

- Follows the Single Instruction Multiple Data (SIMD) paradigm.
- Works on the concept of master and worker nodes.
- Uses MPI (Message Passing Interface) for communication between the master and worker nodes.

Paper’s Contribution: Parallel K-Means Algorithm [3]

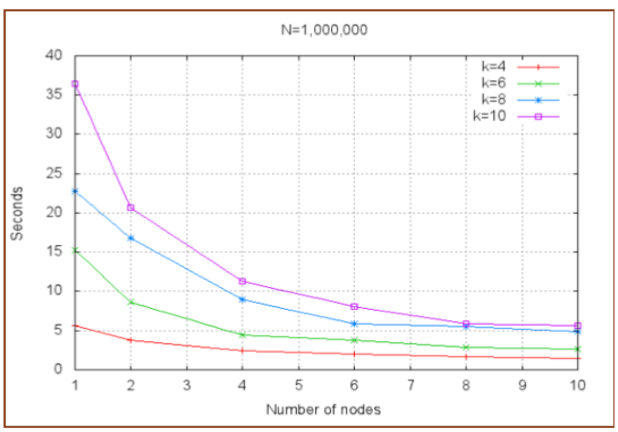


- Parallel K-Means Algorithm [2]
  Master node, M finds the number of available worker nodes, assume: (N – 1)
  M: splits input dataset D into D/(N – 1) subsets
  M: transfers data to worker nodes
  for all Worker nodes w_i, w_i where $i \in \{1, 2, 3,\ldots,N-1\}$, do in parallel
  Receive the data chunk from the master
  Perform K-Means clustering
  Send the local centroids and the number of data records assigned to each one of them to M
  end for
  M receives centroids and the corresponding data records assigned to each of the centroids.
  M sorts the centroid list
  M calculates global centroids by applying the weighted arithmetic mean and produce the final clustering.

Fig 1: Working Model as per the paper 2

Results obtained by parallel k-Means

- Parallel K-Means efficiency increases with an increasing number of clusters.



- Parallel K-Means is highly scalable.

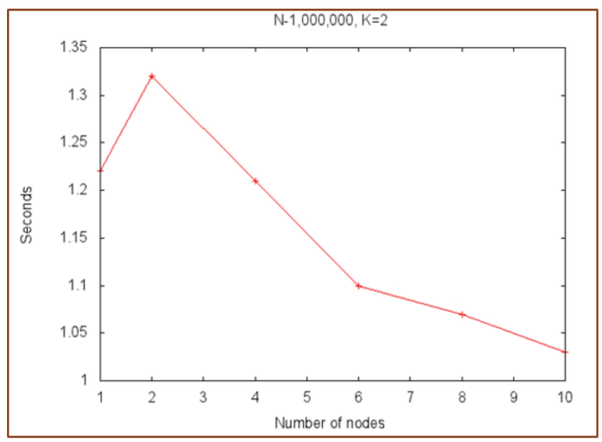


Fig 2: Increasing K and number of nodes and With K=2 increasing number of nodes [3]

Drawback: Message passing overhead is involved in sending and receiving information across the master and worker nodes.

Takeaway for our project implementation:

- Improved performance of K-Means clustering by divide and merge mechanism (parallelizing).

- The results obtained are same to the results generated from the original K-Means clustering.

Analysis of Research Paper 3

Title: High performance parallel k-means clustering for disk-resident datasets on multi-core CPUs
Author: Ali Hadian, Saeed Shahrivari
Journal Name: The Journal of Supercomputing
ISSN: 0920-8542
Date: 08/2014
Page Number: 845 – 863
URL: http://link.springer.com.ezproxy.rit.edu/article/10.1007%2Fs11227-014-1185-y

Problem Statement: Implementation of parallel K-Means clustering utilizing the maximum capabilities of multiple cores of a computer.

Approach to the problem:

- The algorithm takes a parallel processing approach utilizing the available cores in the CPU.
- Divide the dataset into multiple chunks.
- Cluster each of the chunks.
- Aggregate the chunk clustering and make the final cluster.
- This algorithm does only single pass over the dataset.

Paper's Contribution:

Concepts of ***master thread*** and ***chunk-clustering thread*** are integral part of the algorithm.

Algorithm: Master thread [4]

```
Input: K, chunk_size, dataset

Initialize chunks_queue                      [queues shared between threads]
Initialize centroids[]                       [centroids shared between threads]
while Not End_Of_Dataset do
      new_chunk←load_next_chunk(data_set, chunk_size)
      enqueue(new_chunk, chunks_queue)
      if (chunks_queue is full) then
            wait(chunks_queue)               [wait until a consumer thread dequeues a chunk]
      end if
end while
while chunks_queue is not empty do
      wait(chunks_queue)                     [wait for all chunks in queue to be clustered]
end while
C←Cluster [using K-means++ clustering]
```

Algorithm: Chunk Clustering thread [4]

```
Input: chunks_queue, centroids[], K(Number of clusters for each chunk)

while queue is not empty & main thread is alive do
      chunk_instances←Dequeue(queue)
      C←Cluster
      centroids[]←centroids[]UC
      if (queue is empty) then
            wait(queue)                    [wait until a consumer thread dequeues a chunk]
      end if
end while
```

- With the clustered chunks, the master thread loads the centroid list.
- With the centroid list the master thread performs k-means++ clustering to find the final cluster.

<u>Results obtained:</u>

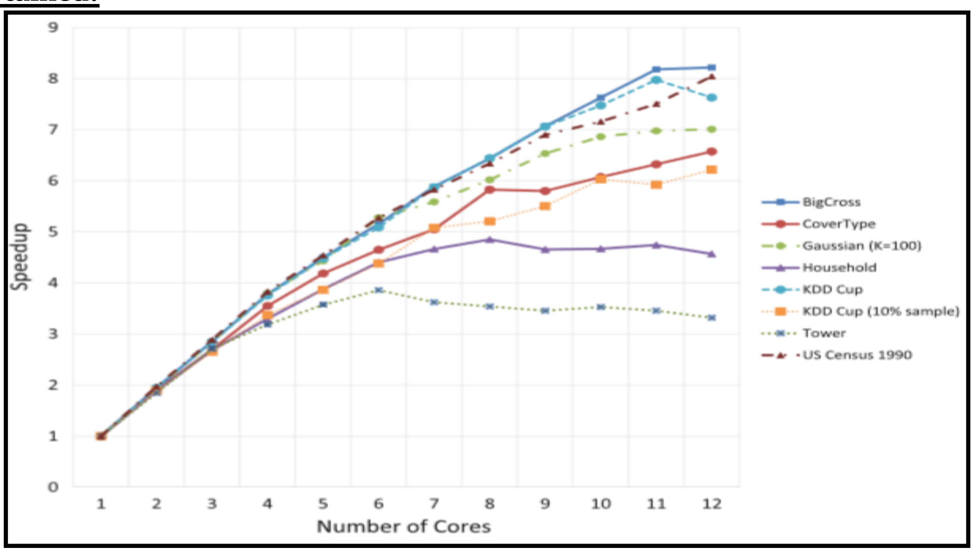


Fig 3: Speedup of the algorithm [4]

<u>Takeaway for our project implementation:</u>

- Utilizing multiple cores in the machine to cluster the dataset at the fastest.

Data preprocessing for our program

k-Means clustering is based on the concept of grouping similar objects together. Distance metric, Euclidean Distance is used to compute the dissimilarity between 2 objects.
Our dataset has 41 dimensions and are of varied datatypes like integer, double, float, strings and long. It was needed to normalize the datasets before we could compute Euclidean distance. Normalization is the process of transforming the data to make it comparable. For our project we used min-max normalization.

Steps followed for normalizing the dataset

1. Strings were mapped to unique integers. This step was essential because we had no basis to compute dissimilarity between 2 strings. (For eg. Protocol attribute)
2. For every attribute do
   i. Compute the minimum and maximum value.
   ii. Subtract the minimum value from all the values of the attribute.
   iii. Divide every value of the attribute by (maximum-minimum).
   iv. Generates data in the range of 0-1.
3. Done. Normalized data obtained.

Original data sample:
0,tcp,http,SF,215,45076111111111,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,0,0,0.00,0.00,0.00,0.00,0.00,0.00,0.00,0.00

Normalized data sample:
0,0.5,0.0307692,0,2.61042e-07,0.00105713,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0.0156556,0.0156556,0,0,0,0,1,0,0,0.0352941,0.0352941,1,0,0.11,0,0,0,0,0

normalizeData.sh script has been used to normalize the dataset.

Design and operation of sequential program

Our design of the sequential version of the code is depicted in the below flowchart. For developing this model, we would consider the cluster size to be 4. Though there are 4 different attack types to verify along with normal traffic, we would only consider in identifying the attack types: DoS, Probe, U2R and normal. This is because the dataset we are considering has a negligible instances of R2L attack type. Additionally from the analysis of research paper 1, we understand that taking the cluster size as 4 would help us to generate the best clustering model.

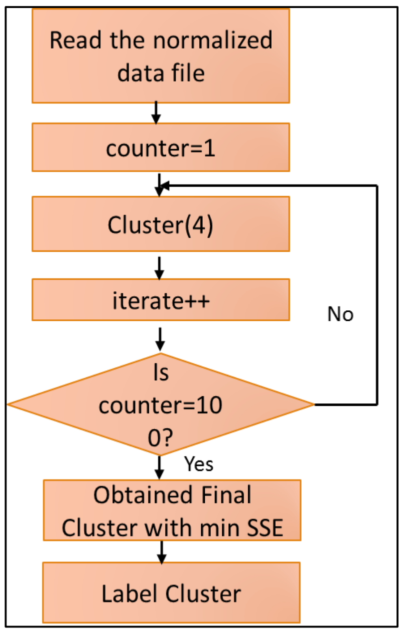


Fig 4: Sequential flow of the code

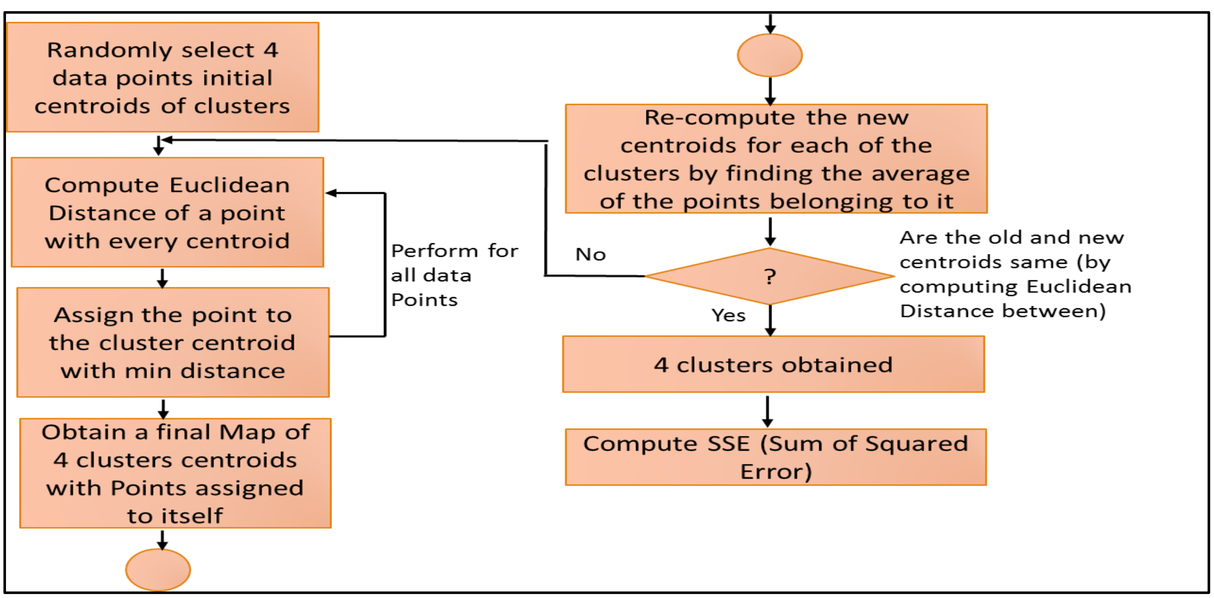


Fig 5: Design of the cluster method.

Operations performed:

1. Creation of a Features Class that would contain all the 41 attributes of the data along with the getters and setters. Feature represents the type of our data point.
2. Creation of the main class kMeansClusteringSeq. Within this class we performed the following operations:
    a. Initialization of the data structures.
    b. Reading the data from the input file and storing in the ArrayList of the object type Features.
    c. Iterate over 100 times and call the cluster method
        i. Using random number generator and seed, k initial data points are selected as the initial centroids and stored in ArrayList<Features>.
        ii. Euclidean distance is computed for every point with all the centroids using the function getDistance().
        iii. Data point is assigned to the cluster with the centroid that generated least distance.
        iv. After all the data points have been assigned to the clusters, new k centroids are computed using the function getNewCentroid().
        v. For every cluster, new centroid is computed by computing the mean of the data points assigned to the cluster.
        vi. Difference between old and new centroids and computed and checked if centroid_distancewithEachOther=0.
        vii. If distance is 0, clustering model is obtained for the current iteration, else go to step ii.
        viii. Compute the Sum of Squared Error. This is the sum of the square of the distance of every point from its respective centroids. Lower the error, better is the clustering model. As it implies the intra cluster is less an inter cluster distance is more.
        ix. Check if the SSE value obtained from the current iteration is the minimum till now. If minimum store the SSE value and the corresponding clustering model.
    d. Perform the labelling for the final clusters obtained using the function findAttackMappings(). In this method, we group all the sub-attacks into five major attack types including normal as shown in Table2. We label the cluster in the order based on the number of occurrences of each of the attack types in our dataset. If attack type DoS has the maximum occurrences in the dataset, then we would first look for the cluster that has maximum instances of DoS and label the cluster as DoS and so on.

Note: SSE calculation is mandatory as the initialization of the centroid is random, thus if we just run for 1 time, we are not sure how good the cluster model is. Thus minimum we should iterate over 100 times and calculate the SSE for respective model. And we should consider the model to be best, for which we obtain the minimum SSE.

Design and operation of parallel program

After several changes to our parallel model, we fixed our model to be like the following:

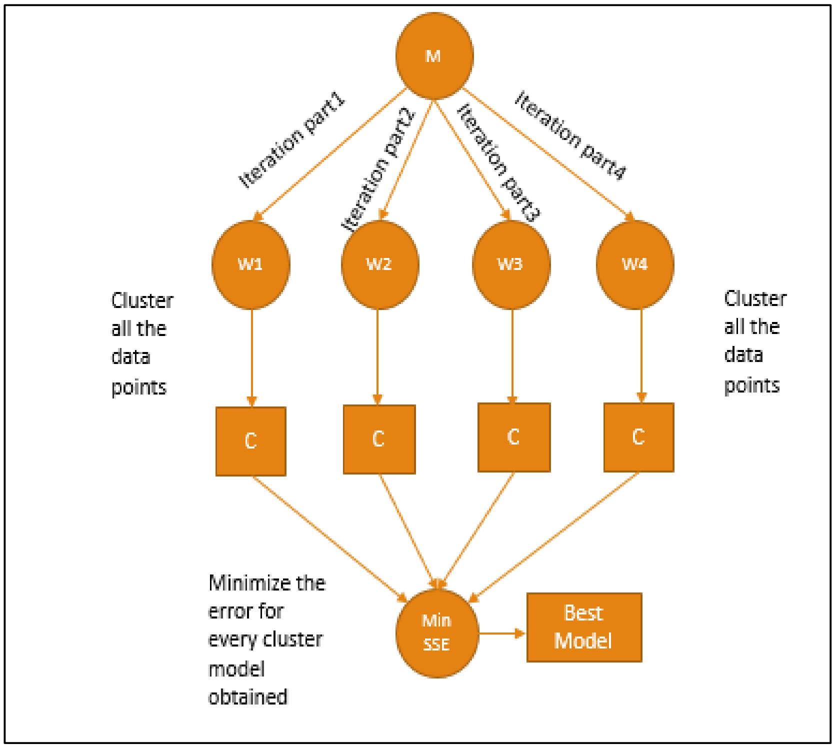


Fig 6: Parallel model of k-Means clustering

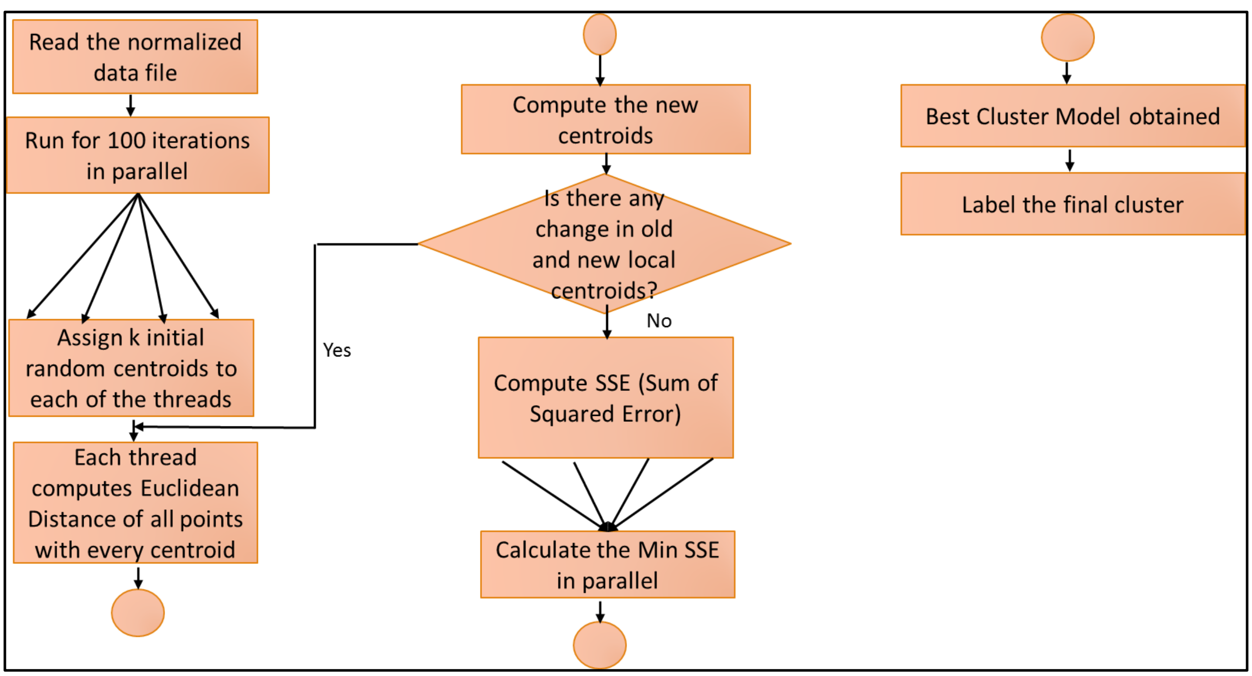


Fig 7: Flow of the parallel model

Operations performed:

1. We will use the same Feature class as created for implementing the sequential version
2. Create a reduction class as SSEVbl.java. This class is used to create the reduction variable for our program that would be used to compute the minimum SSE value (Sum of the Squared Error)
3. Creation of the main class kMeansClusteringSmp. Within this class we performed the following operations:
    a. Initialization of the data structures.
    b. Reading the data from the input file and storing in the ArrayList of the object type Features.
    c. Parallelize the 100 iterations among the threads.
        i. Using random number generator and seed, k initial data points are selected as the initial centroids and stored in ArrayList<Features>.
        ii. Euclidean distance is computed for every point with all the centroids using the function getDistance().
        iii. Data point is assigned to the cluster with the centroid that generated least distance.
        iv. After all the data points have been assigned to the clusters, new k centroids are computed using the function getNewCentroid().
        v. For every cluster, new centroid is computed by computing the mean of the data points assigned to the cluster.
        vi. Difference between old and new centroids and computed and checked if centroid_distancewithEachOther=0.
        vii. If distance is 0, clustering model is obtained for the current iteration, else go to step ii.
        viii. Once clustering model is created, SSE is computed and minimum SSE is found using the reduction variable. That computes the minimum SSE in parallel for all the threads iterations.
    d. End of parallel for, the clustering model corresponding to the minimum SSE is used and it is labelled using the method finalAttackMappings().

Developer’s manual : Code compilation

Follow the below steps to compile the code.

1. Unzip the folder kMeansClustering.zip
2. From the folder named SourceCode copy the following java files and place it under a specific path of your choice on your system:
    - Features.java
    - kMeansClusteringSeq.java
    - kMeansClusteringSmp.java
    - SSEVbl.java
3. Use the following commands to set the environment variable in the terminal:
   bash shell: export CLASSPATH=.:/var/tmp/parajava/pj2/pj2.jar
   csh shell: setenv CLASSPATH .:/var/tmp/parajava/pj2/pj2.jar
4. To compile the codes execute the command: javac *.java

User’s manual : Code execution

Copy the data file named DataSet5 to the location where the java files are present. This is the normalized 10% of the dataset that is ideal for obtaining a good clustering model.

To execute the sequential version of the code execute the below command:
java pj2 kMeansClusteringSeq DataSet5 2000
Usage: java pj2 kMeansClusteringSeq <input file> <seeds>

To execute the parallel version of the code execute the below command:
java pj2 threads=2 kMeansClusteringSmp DataSet5 2000
Usage: java pj2 threads=k kMeansClusteringSmp <input file> <seeds> [Here k=no of threads]

Strong scaling performance of data

We performed strong scaling for our program with 5 different problem sets. Below table shows the speed up and efficiency that we achieved from this strong scaling.

No of data points	K	T(msec)	SpeedUp	Efficiency
98805	seq	33096		
	1	32194	1.028	1.028
	2	16925	1.955	0.978
	4	10434	3.172	0.793
	8	7116	4.651	0.581
296412	seq	159069		
	1	158463	1.004	1.004
	2	88698	1.793	0.897
	4	48014	3.313	0.828
	8	34059	4.67	0.584
395217	seq	150400		
	1	148155	1.015	1.015
	2	85425	1.761	0.881
	4	44893	3.35	0.838
	8	30520	4.928	0.616
444619	seq	152602		
	1	155015	0.984	0.984
	2	81991	1.861	0.931
	4	47270	3.228	0.807
	8	32571	4.685	0.586
494021	seq	143454		
	1	142111	1.009	1.009
	2	79837	1.797	0.899
	4	46901	3.059	0.765
	8	33324	4.305	0.538

Graphs to interpret the strong scaling performance:

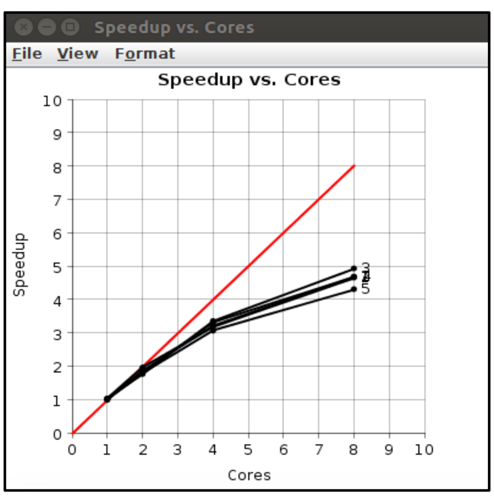


Fig 8: Speedup vs. Cores

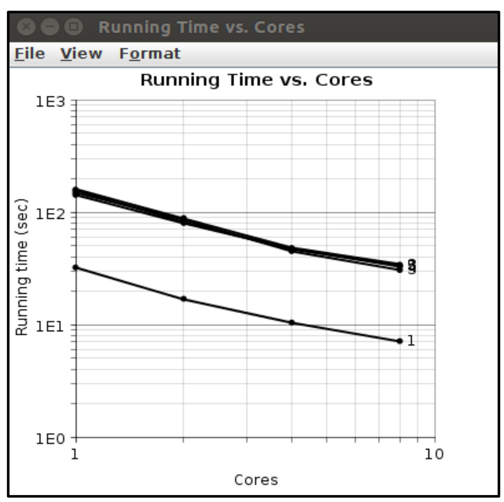


Fig 9: Running time vs. Cores

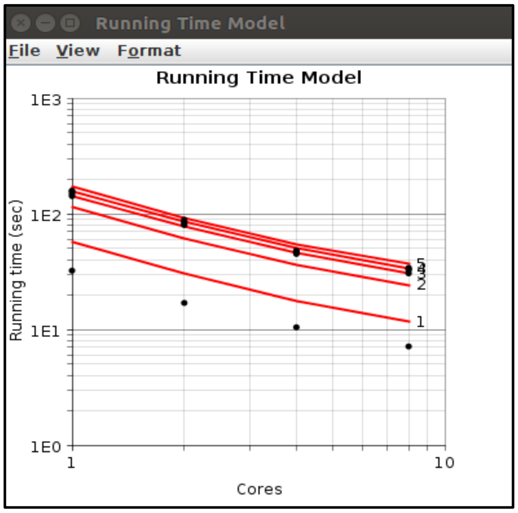


Fig 10: Running Time Model

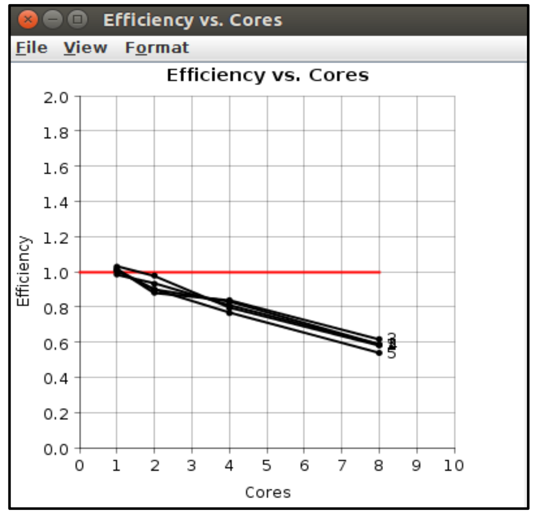


Fig 11: Efficiency vs. Cores

Reasoning for non-ideal strong scaling

```
Running time model
T (sec) = (a + bN) + (c + dN)K + (e + fN)/K
a = 1.26871
b = 2.64364e-05
c = 0.0889871
d = 5.22762e-07
e = 27.0474
f = 0.000259684
normsqr = 5008.21
```

Label	N	K	T	Spdup	Effic	SeqFr
1	9.88e+04	seq	33096			
	9.88e+04	1	32194	1.028	1.028	0.121
	9.88e+04	2	16925	1.955	0.978	
	9.88e+04	4	10434	3.172	0.793	
	9.88e+04	8	7116	4.651	0.581	

Label	N	K	T	Spdup	Effic	SeqFr
2	2.96e+05	seq	159069			
	2.96e+05	1	158463	1.004	1.004	0.057
	2.96e+05	2	88698	1.793	0.897	
	2.96e+05	4	48014	3.313	0.828	
	2.96e+05	8	34059	4.670	0.584	

Label	N	K	T	Spdup	Effic	SeqFr
3	3.95e+05	seq	150400			
	3.95e+05	1	148155	1.015	1.015	0.079
	3.95e+05	2	85425	1.761	0.880	
	3.95e+05	4	44893	3.350	0.838	
	3.95e+05	8	30520	4.928	0.616	

Label	N	K	T	Spdup	Effic	SeqFr
4	4.45e+05	seq	152602			
	4.45e+05	1	155015	0.984	0.984	0.084
	4.45e+05	2	81991	1.861	0.931	
	4.45e+05	4	47270	3.228	0.807	
	4.45e+05	8	32571	4.685	0.586	

Label	N	K	T	Spdup	Effic	SeqFr
5	4.94e+05	seq	143454			
	4.94e+05	1	142111	1.009	1.009	0.101
	4.94e+05	2	79837	1.797	0.898	
	4.94e+05	4	46901	3.059	0.765	
	4.94e+05	8	33324	4.305	0.538	

Fig 12: Running time model for Strong Scaling

1. Sequential version took slightly longer time to complete than parallel version for threads=1.
2. With the number of threads increasing, the efficiency decreased. Because the sequential portion of the code becomes significant. We can also see this from the running model as obtained above. The term a+bN signifies the sequential portion of the code. The sequential portion comprises of the thread initialization, reduction of the threads and labelling of the clusters. Additionally, the time taken by each thread to form clusters cannot be reduced further because internally a considerable time is taken to form clusters in every iteration.
3. With our model, efficiency did not improve beyond threads = 4. We have done a detailed analysis on the time spent by each thread for every iteration in the while loop. We observed that this time varies for every iteration because of the random initialization of the centroids. (Refer to the file TimeTakenInEachIterationToConverge.xlsx for the time records)

Weak scaling performance of data

We performed weak scaling with 5 different data sets for our program. For weak scaling we considered our problem size in the terms of the number of iterations that is performed in parallel to cluster the data points and get the minimum SSE (Sum of Squared Error).

No of data points	K	N	T(msec)	SizeUp	Efficiency
98805	seq	100	33096		
	1	100	32399	1.022	1.022
	2	150	26353	1.884	0.942
	4	280	26519	3.494	0.874
	8	550	30222	6.023	0.753
296412	seq	100	159069		
	1	100	157129	1.012	1.012
	2	200	166956	1.906	0.953
	4	320	150642	3.379	0.845
	8	600	174468	5.47	0.684
395217	seq	100	150400		
	1	100	149105	1.009	1.009
	2	180	148529	1.823	0.912
	4	650	282919	3.455	0.864
	8	1000	272958	5.51	0.689
444619	seq	100	152602		
	1	100	151440	1.008	1.008
	2	150	120522	1.899	0.95
	4	300	131315	3.486	0.872
	8	800	218734	5.581	0.698
494021	seq	100	143454		
	1	100	140887	1.018	1.018
	2	200	153645	1.867	0.934
	4	380	154315	3.533	0.883
	8	600	156761	5.491	0.686

Graphs to interpret the weak scaling performance:

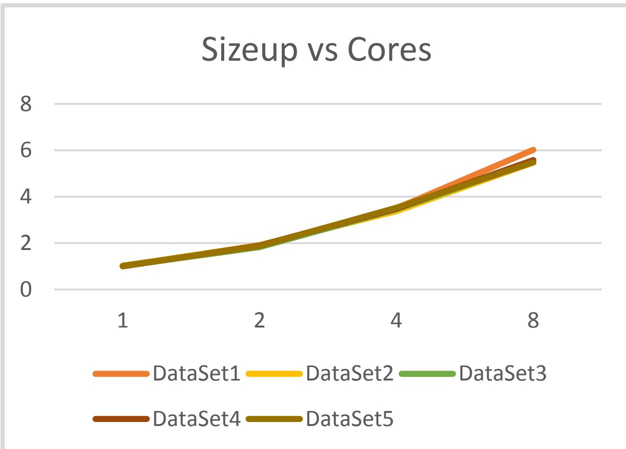


Fig 13: Sizeup vs. Cores

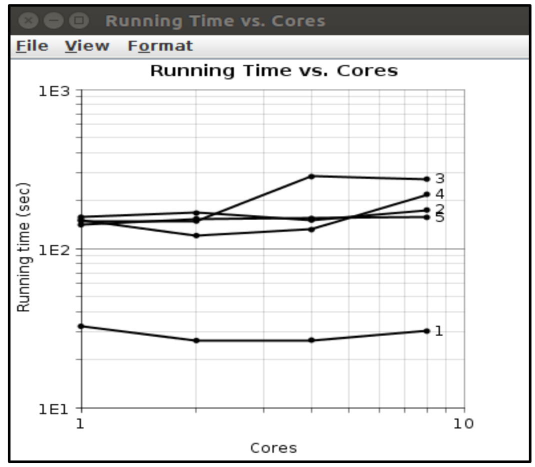


Fig 14: Running time vs. Core

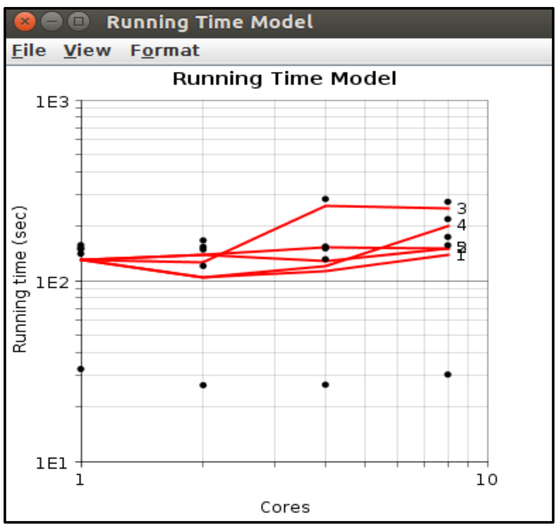


Fig 15: Running Time Model

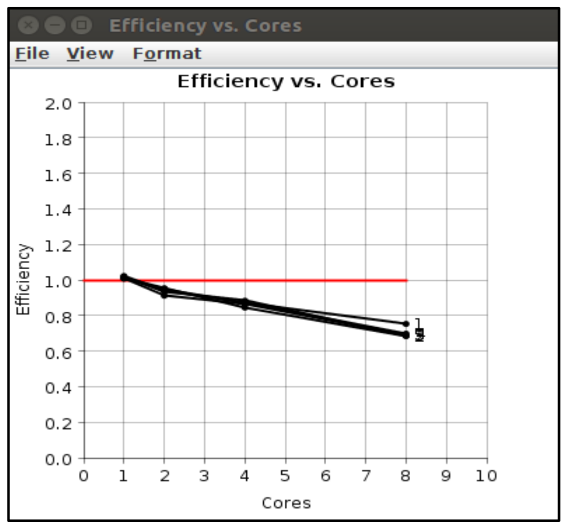


Fig 16: Efficiency vs. Cores

Reasons for non-ideal weak scaling:

```
T (sec) = (a + bN) + (c + dN)K + (e + fN)/K
a = 0.00000
b = 0.101414
c = 0.00000
d = 0.00000
e = 0.00000
f = 1.19362
normsqr = 40558.3
```

Label	N	K	T	Spdup	Effic	SeqFr
1	1.00e+02	seq	33096			
	1.00e+02	1	32399	1.022	1.022	0.313
	1.50e+02	2	26353	1.884	0.942	
	2.80e+02	4	26519	3.494	0.874	
	5.50e+02	8	30222	6.023	0.753	

Label	N	K	T	Spdup	Effic	SeqFr
2	1.00e+02	seq	159069			
	1.00e+02	1	157129	1.012	1.012	0.065
	2.00e+02	2	166956	1.906	0.953	
	3.20e+02	4	150642	3.379	0.845	
	6.00e+02	8	174468	5.470	0.684	

Label	N	K	T	Spdup	Effic	SeqFr
3	1.00e+02	seq	150400			
	1.00e+02	1	149105	1.009	1.009	0.068
	1.80e+02	2	148529	1.823	0.911	
	6.50e+02	4	282919	3.455	0.864	
	1.00e+03	8	272958	5.510	0.689	

Label	N	K	T	Spdup	Effic	SeqFr
4	1.00e+02	seq	152602			
	1.00e+02	1	151440	1.008	1.008	0.067
	1.50e+02	2	120522	1.899	0.950	
	3.00e+02	4	131315	3.486	0.872	
	8.00e+02	8	218734	5.581	0.698	

Label	N	K	T	Spdup	Effic	SeqFr
5	1.00e+02	seq	143454			
	1.00e+02	1	140887	1.018	1.018	0.072
	2.00e+02	2	153645	1.867	0.934	
	3.80e+02	4	154315	3.533	0.883	
	6.00e+02	8	156761	5.491	0.686	

Fig 17: Running time model for Weak Scaling

1. As we can see from the running model, the sequential term a+bN has considerable effect, the second term (c+dN)k is 0 and has got no impact on the efficiency. But the third term (e+fN)/k (the parallelizable portion) is of higher magnitude, which cannot be parallelized any further beyond threads=4.
2. Subsets of the data sets considered for weak scaling are not ideal for clustering. There convergence rate also varies not allowing us to get an ideal performance.

Future Work

- We can use a test model and evaluate the accuracy of the clustering model we built. With this we can guarantee that parallelizing the model construction assures quality of the output along with efficiency (in time).
- This approach could be used in parallelizing other machine learning algorithms.

What we learned from the project?

- How to approach any NP hard problem and try to parallelize it. We understood which portion of the code we should try to parallelize to obtain the optimal result.
- With the advancement in the project, we found ways to improve our models by building on the last block.
- We understood how k-Means clustering works.
- For machine learning algorithms, use of proper data set is essential.

Team member's contribution to the project

- Chandni did an analysis on research paper 1, working on design and implementation of sequential and parallel code.
- Priyanka did an analysis on research paper 2, research paper 3 and worked on the design and implementation of sequential and parallel code. Have also worked with the shell scripts used for data normalizing and automating the run for strong and weak scaling.
- Chandni and Priyanka worked together on deciding the project topic, making presentations and writing final report.

References

[1] Visualize Network Anomaly Detection by Using K-Means Clustering Algorithm
Riad,A. M. and Elhenawy,Ibrahim and Hassan,Ahmed and Awadallah,Nancy
doi: 10.5121/ijcnc.2013.5514
International Journal of Computer Networks & Communications
Vol. 5 ,195-208 , 2013

[2] K-Means Clustering Approach to Analyze NSL-KDD Intrusion Detection Dataset
Vipin Kumar, Himadri Chauhan, Dheeraj Panwar
Journal Name: International journal of soft computing and engineering
ISSN: 2231-2307 Volume -3, Issue -4,

Date: 09/01/2013
Page Number: 1-4
URL:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.413.589&rep=rep1&type=pdf

[3] Parallelizing k-means Algorithm for 1-d Data Using MPI
Ilias K. Savvas and Georgia N. Sofianidou
Journal Name: IEEE 23rd International WETICE Conference
Date: 2014
Page Number: 179 – 184
URL: http://ieeexplore.ieee.org.ezproxy.rit.edu/stamp/stamp.jsp?tp=&arnumber=6927046

[4] High performance parallel k-means clustering for disk-resident datasets on multi-core CPUs
Ali Hadian, Saeed Shahrivari
Journal Name: The Journal of Supercomputing
ISSN: 0920-8542
Date: 08/2014
Page Number: 845 – 863
URL: http://link.springer.com.ezproxy.rit.edu/article/10.1007%2Fs11227-014-1185-y

----------------------------------- X -----------------------------------