PARALLELIZATION OF K-MEANS AND DBSCAN CLUSTERING ALGORITHMS ON A HPC CLUSTER

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

HUNAIN DURRANI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTOR ENGINEERING

JANUARY 2013

Approval of the thesis:

A PARALLEL IMPLEMENTATION AND VERIFICATION OF K-MEANS AND DBSCAN CLUSTERING ALGORITHMS ON A HPC CLUSTER

submitted by **Hunain Durrani** in partial fulfillment of the requirements for the degree of **Master of Science in Computor Engineering, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Ahmet Coşar
Supervisor, **Computer Engineering Dept., METU** _____

Examining Committee Members:

Prof. Dr. Ismail Hakkı Toroslu
Computer Engineering Dept., METU _____

Assoc. Prof. Dr. Ahmet Coşar
Computer Engineering Dept., METU _____

Assoc. Prof. Dr. Veysi İşler
Computer Engineering Dept., METU _____

Umut Tosun, MS
Siemens EC, Ankara _____

Assoc. Prof. Dr. Uğur Güdükbay
Computer Engineering Dept., BILKENT _____

Date: January 14, 2013

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Hunain, Durrani

Signature:

ABSTRACT

PARALLELIZATION OF K-MEANS AND DBSCAN CLUSTERING ALGORITHMS ON A HPC CLUSTER

Durrani, Hunain

M.Sc., Computer Engineering

Supervisor: Assoc. Prof. Dr. Ahmet Coşar

January 2013, 32 pages

The amount of information that must be processed daily by computer systems has reached huge quantities. It is impossible, or would be prohibitively expensive, to build such a powerful supercomputer that could process such large data in the required time limits. Cluster computing has emerged to address this problem by connecting hundreds of small computers using ultra-fast switches so that their combined computational power and parallel processing techniques make it possible to quickly solve many difficult problems. In fact, cloud computing has emerged to market the data processing power collected in cluster computing centers with hundreds of thousands of computers and allow the customers to purchase additional data processing power, storage, memory, and communication capacity when needed.

Data mining has been one of the most favorite topics for all the researchers as it's the technique that helps large scale business to extract useful data from the heap of irrelevant data. In this era of big data stores parallel implementation of data mining is the basic tool of all the large scale businesses.

In this research, parallel versions of two popular clustering algorithms, K-Means and DBSCAN, are developed and it is experimentally shown that their performance continues to improve even as the input data size keeps increasing, making these parallel implementations ideally suited to parallel computing environments.

Keywords: clustering, data mining, k-means, dbscan, parallel processing.

ÖZ

DBSCAN VE K-MEANS KÜMELEME ALGORİTMALARININ BİR HPC KÜMESİ ÜZERİNDE PARALELLEŞTİRİLMESİ

Durrani, Hunain

Yüksek Lisans, Bilgisayar Mühendisliği Ana Bilim Dalı

Tez Yöneticisi: Doç. Dr. AhmetCoşar

Ocak 2013, 32 sayfa

Günlük olarak bilgisayar sistemleri tarafından işlenmesi gereken bilgi miktarı çok büyük miktarlara erişmiştir. Bu kadar çok datayı istenen sürede işleyebilecek bir super bilgisayarın inşa edilmesi hem çok zor hem de çok pahalı olacaktır. Küme hesaplama bu problem çözmek için yüzlerce küçük bilgisayarın çok-hızlı anahtarlar ile birbirine bağlanmasıyla ortaya çıkmıştır ve toplam işlemci gücü ve parallel işleme teknolojileri kullanılarak bir çok zor problemin çabucak çözülmesini sağlamıştır. Aslında bulut hesaplama küme hesaplama merkezlerinde toplanan yüzbinlerce bilgisayarın parallel bilgi işleme güçlerinin pazarlanmasını sağlamış ve müşterilerin gerek oldukça ek bilgi işleme gücü, veri depolama yeri, bellek, ve komünikasyon kapasitesi satın almalarına izin vermiştir.

Veri madenciliği büyük ölçekli işletmelerin çoğu ilişkisiz verilerin içinden yararlı olanlarının çıkartılmasını sağlayan bir teknik olarak araştırmacıların en çok çalıştığı konulardan biri olmuştur. Büyük veri depolarının yaygınlaştığı günümüzde veri madenciliğinin parallel gerçekleştirilmesi bütün büyük işletmelerin temel aracıdır.

Bu çalışmada, iki popular kümeleme algorittmasının, K-Means ve DBSCAN, parallel sürümleri geliştirilmiş ve deneysel olarak very miktarı arttıkça başarımdaki iyileşmenin sürdüğü gözlenerek bu parallel sürümlerin parallel ortamlar için çok uygun olduğu gösterilmiştir.

AnahtarKelimeler: kümeleme, veri madenciliği, k-means, dbscan, paralel işleme

To My Family

This thesis is dedicated to my parents who were always praying for my success and also worked hard to bring me up to who I am today.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Ahmet Coşar, whose encouragement, guidance and support throughout my thesis work enabled me to complete this work. His careful advices, constructive criticisms, and encouraging me to continue my research have aided me to complete this thesis successfully. I would also like to thank Dr. Pınar Karagöz for her guidance that showed me how to go on the right way. I would also like to add thanks for Dr. Murat Manguoğlu for his valuable advices. Finally, I would like to thank Mr. Umut Tosun for his helps in debugging my parallel code and suggestions to improve the achieved parallelism.

Finally, I would pay my greetings to my parents who have supported me throughout my education for my success.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

For future planning, gathering useful information from existing data has always been an important task for human beings. However, day by day the past data in the data warehouse is increasing with a high rate due to which it becomes difficult for an individual to extract the required knowledge by manual techniques. It is also difficult to extract knowledge from that bundle of data even by using computing technologies that are usually based on single sequential processors. In order to fulfill such information extraction tasks there is a need for developing a technique that can provide the results much more efficiently and without losing accuracy, within a limited amount of time.

In the following sections the rapidly increasing rate of Internet usage and its threats are discussed, giving the problem definition and why it's required to develop parallel algorithms for clustering massive amount of data, such as Internet traffic data, in order to extract valuable information and identify malicious threats that can be discovered in network traffic data. At the end of the chapter a brief description of the road map of this document will be given.

1.1 The Threats Coming Through Internet

According to Moore's law **[4]**, the size of Internet doubles every year and so the amount of the threats to the large networks grows exponentially with increasing Internet usage. As a side effect of this growth, the network traffic data also increases rapidly and it is not possible to identify which part of network traffic data belongs to normal traffic and which part might contain malicious attacks and/or traffic. Hackers are the bigger threats nowadays they are dangerous both locally and they also pose threats to remote computer systems over Internet. For example, someone can upload a virus or a trojan into our computer or someone can hack our network, steal and misuse our financial data which might cause a big threat for the integrity and confidentiality of trade secrets and/or reputation of our organization. From this perspective, the development of *fast*, *adaptive* and *accurate* anomaly detection algorithms attract more research.

1.2 Importance of Parallel Data Mining Methods

Data mining aims at processing data in order to filter or discover the desirable rules or patterns in a database. It is a multi-disciplinary field, which combine research areas such as machine learning, statistics, and high performance parallel computing. More than often, data mining tasks have to operate on very large sets of data and require a lot of data storage and processing resources. The amount of data arriving from various sources can be huge in size varying from gigabytes to terabytes, and this further increases the need to develop efficient parallel data mining algorithms that can run on a distributed architecture system such that there will be sufficient capacity both for storing and also processing data in parallel, possibly even reading data from many disks (in parallel) attached to nodes of a cluster computer. Another important reason for dividing the work on data is the memory limitations of a single processor that would be typically around 4GB. Many modern database management systems use parallel architectures both for storing data and also for assigning each query request to one of the many query processing nodes, all working in parallel. Therefore, the task of processing data which is already distributed to many nodes/computers over a network can be ideally done in parallel obtaining maximum speed-up. Since data mining tasks often require processing of huge amounts of data, parallel implementation for common data mining operations are very much in demand.

The common techniques for implementing parallelism in data mining operations are *"task parallelism"* and *"data parallelism"*, according to [15] in *task parallelism* the search space is

distributed into portions among separate processors whereas in *data parallel* approach the data set is distributed over the available processors so that they can be processed in parallel by many processors.

CHAPTER 2

DATA MINING CONCEPTS

Data mining is a technology used for extracting hidden information from massive datasets, so that we can use this extracted information to make predictions about future events and/or behaviors. It is an attractive technique since it helps large scale businesses and companies to extract and concentrate only on the part of data that it requires to make decisions and plans for future. Since data mining tools are able to better predict future trends and behaviors, large scale business companies can make proactive and knowledge-driven decisions. The term "*knowledge discovery*", as used in the phrase "Knowledge Discovery from Data" (KDD), is taken as a synonym for *data mining*. The processes of KDD consist of the following steps;

1) Cleaning noise and other irrelevant information from useful data
2) Integrating the data by reducing multiple occurrences of the same data item to one.
3) Selecting the relevant data in order to perform required analysis for some specific tasks.
4) Determining the data patterns occurring in the selected data.
5) Presenting the extracted data to the end user so that it is easy for the user to find the desired information. This can be done by various data presentation tools.

Figure 1 **[6]** presents and describes the flow of data mining steps. The data can be obtained from any database repository; a data warehouse of any large scale business organization can be used to perform data mining for extracting meaningful information from that data. The data which is available over World Wide Web can also be helpful in mining the useful information. Any type of text formatted file such as *comma separated values* (".csv") files can also be used as sources of input in order to perform KDD operations on relatively meaningful and useful data.

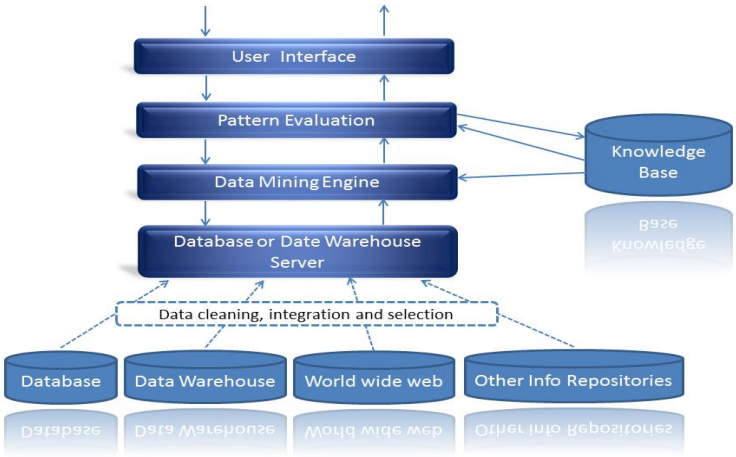

Figure 1. Architecture of a typical data mining system [6].

2.1 Clustering Approaches

We are in an era in which the access to the Internet data has become so easy that we can reach that data with a few clicks, and due to the increasing numbers of Internet users, the data traffic over

Internet is also increasing in parallel. In Table 1.1 taken from **[10],** the statistics for world Internet usage and population are given from 2000 to 2010.

Table 1. World Internet usage and population statistics [Usage2011].

World Regions	Population (2010 Est.)	Internet Users (2000)	Internet Users (2011)	Penetration (%)	Growth 2000-2010	% of All Users
Africa	1,013,779,050	4,514,400	110,931,700	10.9 %	2,357.3 %	5.6 %
Asia	3,834,792,852	114,304,000	825,094,396	21.5 %	621.8 %	42.0 %
Europe	813,319,511	105,096,093	475,069,448	58.4 %	352.0 %	24.2 %
Middle East	212,336,924	3,284,800	63,240,946	29.8 %	1,825.3 %	3.2 %
N. America	344,124,450	108,096,800	266,224,500	77.4 %	146.3 %	13.5 %
Caribbean	592,556,972	18,068,919	204,689,836	34.5 %	1,032.8 %	10.4 %
Australia	34,700,201	7,620,480	21,263,990	61.3 %	179.0 %	1.1 %
TOTAL	**6,845,609,960**	**360,985,492**	**1,966,514,816**	**28.7 %**	**444.8 %**	**100.0 %**

From the numbers if Table 1, it is quite clear that with such massive amounts of data it is very difficult, if not impossible, for a human to manually extract any meaningful information from this data for further use.

Cluster analysis or simply named as *clustering* is a very useful data mining method which groups sets of "similar" (using a distance metric) objects in a multidimensional space into so-called "clusters". The clusters obtained from the clustering operation will contain a subset of all objects that more similar to objects in the same cluster rather than the objects in other clusters. The phenomenon of clustering is depicted in Figure 2, where part (*a*) shows the input dataset and the four discovered and output clusters are shown as circles in (*b*). The data that are not assigned to any cluster are treated as noise.

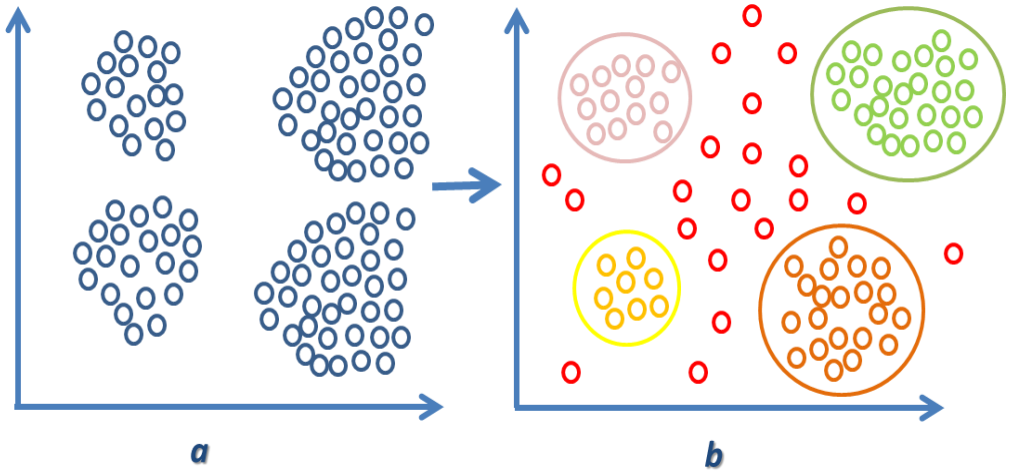

Figure 2. Clustering phenomenon.

As seen from Figure 2, Cluster analysis greatly helps in grouping together the objects of similar behaviour from a multidimensional data space. Clustering a phenomenon follows the idea that if a rule is valid for an object in the data space it is also possible that the same rule is applicable to other objects in the space that are similar to it. This is how clustering helps in tracing dense and sparse regions in the data space and can discover the hidden relationships between data objects.

Therefore, from all of the above discussion we can say that clustering is the process of data mining in which objects with similar attributes are grouped in a cluster to form a single group of objects (also called a "class"). In order to cluster data using data mining we can use the following two approaches:

1) Supervised Clustering
2) Unsupervised Clustering

The objective of supervised clustering is that it is applied on a classified example and its task is to identify the objects with high probability density with respect to a single class. Furthermore, in supervised clustering the number of output clusters are kept as small as possible and the objects are assigned to their respective clusters on the basis of minimum distance from its assigned cluster obtained by using the given distance formula.

In unsupervised clustering we don't have any prior knowledge of the actual number of clusters. This type of clustering can be performed by attempting, repeatedly, to cluster for several ***C*** (the number of clusters) values which can be costly in terms of the CPU time. Unsupervised clustering, then, performs some measurements in order to select the best partitioning of the data. Another approach that can be used for performing unsupervised clustering can be the process of performing several passes over the dataset, the algorithm proceeds by seeking one cluster at a time and removes it from the data set in order to add it to the similar clusters **[1]**. A threshold value used by this technique can widely vary for different size of data sets which in turn reduces the validity of the output taken from this type of technique.

As in **Compatible Cluster Merging [2]** unsupervised clustering can also be performed by starting clustering with a default number of clusters and during the process similar clusters are merged together and the suspicious ones are eliminated, this process continues until the algorithm comes up with the requested number of clusters. **Competitive Agglomeration [3]** is one of the recent approaches used for unsupervised clustering, in this approach the dataset is partitioned into *N* distinct

clusters, each clusters “competes” with its neighbors for the data points and the winning clusters eliminate the losing clusters, thus trying to obtain exactly *N* clusters.

In the following section we will be talking about data mining algorithms, K-Means and DBSCAN.

2.2 K-Means Clustering Approach

K-Means is one of the simplest clustering algorithms which cluster the given dataset under ***K*** number of clusters also named as *centroid*; this K number of clusters and cluster center points are provided to the algorithm as starter clusters. The dataset objects are gathered under the selected centroids on the basis of having the smallest distance between the cluster centroids and the dataset point. After all the datasets are gathered under their respective centroids, the process of recalculation of new centroids is started giving as output a new set of centroids. Then, these new centroids are used for gathering the dataset objects under the newly created centroids. The process of recalculation of centroids and the reassembling of dataset under these centroids is performed within a loop which ends when no new centroid can be generated in the recalculation of centroids phase.

2.3 The K-Means Algorithm

K-Means algorithm is composed of the following steps;

```
INPUT
        K: number of centroids
        D: set of data points
OUTPUT
        C: vector of centroids
        CL[ ][ ]: list of data point indices for each cluster center

PREVC= [-1]*K // initialize list of previous centroids
for(i=0;i<K; i++):
    C[i] = randomly_select_centroid (D)
While (! PREVC.equals(C) ):
    PREVC= C
    for(j=0;j< |D|; j++):
        nearest= find_nearest_centroid(D[j], C)
        CL[nearest].append (j) // add data point D[j] to nearest cluster
    for(i=0;i<K;i++):
        NC[i]= calculate_new_centroid(CL[i], D)
```

K-Means algorithm is the simplest clustering algorithm but its biggest drawbacks are the usage of Euclidean distance for cluster identification and requiring the number of clusters as an input. Hence, K-Means cannot identify the number of clusters by itself and also because of the random initialization of cluster centroids it might not be able to generate clusters of some shape.

2.4 Density Based Spatial Clustering of Applications with Noise (DBSCAN)

DBSCAN was introduced by Ester, et al. [8] in 1996 to discover the clusters and noise in a spatial database. DBSCAN generates groups of similar data objects from larger space of data, and these groups are named as *clusters* whereas those data points not belonging to any cluster are named as *noise*. According to the definition of DBSCAN in Ester, etsal. [8], a subset of similar objects is called

cluster C with respect to Epsilon (i.e. "radius") and minimum number of points (minPts) in database, D, of points if : [8]

i) $\forall$p, q: if $p \in$C and *q* is density reachable from *p* wrt. *Eps* and *MinPts*, then q $\in$C. (Maximality).
ii) $\forall$p, q$\in$C: *p* is density connected to *q* wrt. *Eps* and *MinPts*. (Connectivity).

The data points other than those found in any one of discovered clusters are called noise if: [8]

i) Let $C_1,\ldots, C_\kappa$ be the clusters of the database D with respect to parameters Eps_i and $MinPts_i$ where i = 1,2,3,….,k, then we define the noise as the set of points in the database D not belonging to any cluster C_i, i.e. noise=$\{n \in D \mid \forall i: n \notin C_i\}$.

Here *Eps* is the radius that delimits the neighborhood area of a point and *MinPts* is the minimum number of points that must exist in the Eps-neighborhood. To obtain the ideal result from DBSCAN we should provide the accurate values for *Eps* and *MinPts* but it is one of the problems in DBSCAN to identify the accurate *Eps* and *MinPts*. However, there can be a heuristic method to determine the values for Eps and MinPts.

2.5 The Algorithm

The DBSCAN algorithm randomly chooses a point *p* as a cluster center and finds all those data points which are neighboring to *p* and which are "*density-reachable*". The notion of "density reachability" is determined by using the *Eps* parameter that is used to compare with the calculated distance of a data point from *p*. If the distance is smaller than *Eps*, then the data point is included in the cluster of data points around *p*. If *p* is a *core point*, this procedure will generate a cluster with respect to *Eps* and *MinPts* which must also satisfy the condition that the number of "*density reachable*" points to *p* is more than or equal to *MinPts*. If *p* is a *border point*, no points are *density-reachable* from *p* and DBSCAN algorithm will continue looking for new clusters using the next point in the database. It is also possible that even after two (or more) clusters are discovered, DBSCAN may decide to merge clusters to the other using the same *Eps* and *MinPts* parameters. The two clusters are separated into two separate clusters only if the distance between all possible pairs of points *X* (in cluster-1) and point *Y*(in cluster-2), are larger than *Eps*. This can be implemented with a recursive call to DBSCAN algorithm that tries to identify clusters. This recursive call must be handled carefully in order to minimize the data communication overhead caused by recursive calls.

2.6 DBSCAN Pseudo Code

The pseudo code for DBSCAN is given below:**[16]**

```
DBSCAN ( D , eps, MinPts) // D: Dataset     eps: cluster members distance
  clusterID=0
  foreach ( unvisited P in Dataset)
        P.visited = True
        NeigborPts= regionQuery(P, eps)
        If sizeof(NeigborPts)< MinPts
              P.mark= NOISE
        Else
              C= ++clusterID
              ExpandCluster (P, NeigborPts, C, eps, MinPts)
        End if
  endfor
```

The algorithm accepts the dataset, *eps* and *MinPts* as its inputs. In the algorithm setofPoints.get(i) returns the i-th point in the dataset. Here, ExpandCluster is the most important method. In this method the point is being expanded i.e. its neighbors are being calculated and decision of noise or normal point is given.

ExpandCluster (P, NeighborPts, C, eps, MinPts)

```
  Add P to cluster C
  C.clusterID= C
  Foreach P' in NeghborPts
        If P'.visited == FALSE
           P'.mark= VISITED
           NeigborPts' = GetNeighbors(P', eps)
           if sizeof (NeighborPts') >= MinPts then
                 NeigborPts= NeighborPts UNION NeighborPts'
            If P'.clusterID == NOTASSIGNEDYET
                 Add P' to cluster C
                 P'.clusterID= C
```

GetNeighbors(P, eps)

```
  return all points within P's eps-neighborhood
```

In the above algorithm the call to *GetNeighbors(P, epsilon)* method returns the neighbors of the provided point, P, on the basis of *epsilon* value.

CHAPTER 3

PARALLEL PROGRAMMING WITH MPI

Message Passing Interface (MPI) is an API used for programming parallel applications that are able to communicate among themselves from different computers. MPI is a language independent communication protocol that supports both Point-to-Point and collective communication. MPI interface provides synchronization and communication functionality among the processes with language bindings for example language syntax. MPI routines that are specifically used for communication are divided into two categories:

1) Point-to-Point Communication
2) Collective Communication

3.1 Point-to-Point communication Routine

Message Passing Interface (MPI) provides a standard library of messaging functions that are used for performing point-to-point messaging operations between nodes of a cluster computer. The messages may identify a cluster node as the destination/sender of a message it is trying to send/receive and also optionally may select a subset of messages by using "message tags" that are assigned to messages by senders. It is also possible to perform broadcast operations where all nodes involved in the broadcast call the same broadcast function. The send/receive operations could have other features such as:

1) Synchronous send
2) Blocking Send / Blocking Receive
3) Non-Blocking Send / Non-Blocking Receive
4) Buffered Send
5) Combined Send / Receive
6) Ready Send

It is possible to match any kind of send operation with any type of receive operation. In an ideal world, every send operation would be matched with exactly one receive operation and if the receiver of an MPI send operation is not available then the sender must be able to store the contents of the send operation until the receiver is ready to fulfill its part. For example what will happen to the data in the following two situations where both send and receive operations are not synchronized.

1) A node performs a send few seconds before the receiver becomes ready to accept it
2) Several nodes perform send operations and multiple messages arrive at the same receiving node meaning the receiver must choose and accept only one of these messages

In the situations mentioned above MPI maintains a system buffer that holds the data in transition as shown in the Figure 3 below.

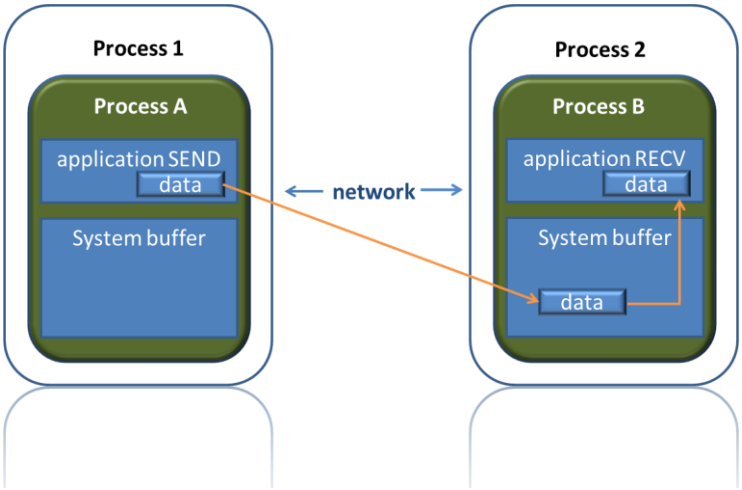

Figure 3. Message transmissions between processes.

The system buffer space has the features like it is invisible to the programmer and is the MPI library can use it freely. It can exist in the sender side, the receiver side or on both sides as well. By permitting asynchronous send/receive operations the performance of parallel nodes increases because they don't have to wait for each other to become available to accept the sent message.

3.1.1 Blocking and Non-Blocking Communication

In case of blocking mode of Point-to-Point communication routine the send routine will return only when it confirms that the application buffer is safe to reuse. The send operation can be synchronous (i.e. the partner must respond) meaning that the receiver task must be ready to accept the sent message and confirm that it was safely received. It can also be done in asynchronous mode in which the send data is buffered in the system buffer for eventual delivery to the recipient.

In case of non-blocking mode of Point-to-Point communication routine, both send and receive routines need not wait for these communication operations to be completed. The send/receive operations require primitive operations such as copying message data from user program's memory space to the communication system's buffer. A similar operation is performed when a message arrives at the destination node and the addressed/waiting process is identified. There are no real-time message delivery guarantees and the sender and receiver may go on other businesses causing messages to be delayed in message buffers while awaiting to be processed. In order to eliminate the negative effects of such message processing delays and responses, it is possible for senders to send "asynchronously", and then going about doing its other tasks.

3.2 Collective Communication Routines

These messaging functions can provide an easy method for all tasks/processes working on the same problem to receive the same message (i.e. a broadcast operation). All of these routines are blocking operations.

3.3 Fundamental MPI Concepts

3.3.1 MPI Communicator

MPI communicators are the communication channels that MPI uses to communicate with the nodes for message passing. Communicator can have group of processes, the default communicator is MPI_COMM_WORLD whose group contains initial processes. MPI communicators can be of two types:

1) Intra-Communicator, it's a collection of processes that use collective communications for message sending.
2) Inter-Communicators are for sending messages between processes of disjoint intra-communicators.

3.3.2 MPI Process Rank

An MPI communicator can have group of processes, each process in a group has its own identity which is named as rank of that process. Ranks are contiguous and begin at zero.

3.4 Basic MPI Functions

Below are the basic MPI functions that can be used for communication between the nodes.

a) MPI_Send: This routine sends messages using blocking phenomenon and for each send routine and there should be a receive routine in order to release the blockings.
int MPI_Send(void ***buf***, int ***count***, MPI_Datatype ***datatype***, int ***dest***, int ***tag***, MPI_Commcomm)

Table 2. MPI_Send ( ) Parameters Description.

Parameter Name	Description
Buffer (buf)	[in] Address of the buffer for data to be send.
Count	[in] Number of elements in the send data.
Data Type (datatype)	[in] Data type of the send data.
Destination (dest)	[in] Destination rank.
Tag	[in] Key value to tag the send message.
Communicator (comm)	[in] Communicator group.

b) MPI_Recv: This routine is used to receive the data sent by MPI_Send routine.
Int MPI_Recv (void ****buf***, int ***count*** MPI_Datatype datatype,
int ***source***, int ***tag***, MPI_Comm ***comm***,
MPI_Status ****status***)

Table 3. MPI_Recv ( ) Parameters Description.

Parameter Name	Description
Buffer (buf)	[out] Address of the buffer for data storage.
Count	[in] Number of elements in the received data.
Data Type (datatype)	[in] Data type of the send data.
Destination (dest)	[in] Destination rank.
Tag	[in] Key value to tag the send message.
Communicator (comm)	[in] Communicator (handle)
Status	[out] Communicator group.

To store the count number of succeeding *datatype* elements with constant size, receive buffer is used. The starting memory address is *buf*.

The received message's length should not be more than the length of this buffer. Overflow error is reported if the incoming data does not fit into the receive buffer causing truncation.
Since the incoming message's length is known in advance, when it arrives at receive buffer, only the memory locations that correspond to the message are modified.
Maximum length of the message is indicated by the count argument and the message's actual length is determined with *MPI_Get_count*.

c) MPI_Init: This routine Initializes execution environment for MPI programs.
int MPI_Init(int ***argc***, char ****argv***)

Table 4. MPI_Init ( ) Parameters Description.

Parameter Name	Description
argc	[in] Total number of sent arguments
argv	[in] An argument vector pointer.

The purpose of this routine is to initialize MPI execution environment and it must be called once in a program.

d) **MPI_Comm_size:** This routine gives the communicator group.
int MPI_Comm_size(MPI_Commcomm, int ***size***)

Table 5. MPI_Comm_size ( ) Parameters Description.

Parameter Name	Description
Communicator (comm)	[in] Associated communicator
Size	[out] Total number of processes in the communictor.

In order to know the number of process that are in the communicator group we use MPI_Comm_size routine.

e) **MPI_Comm_rank:** The rank information about the calling process can be determined using this routine
int MPI_Comm_rank (MPI_Comm comm, int ***rank***)

Table 6. MPI_Comm_rank ( ) Parameters Description.

Parameter Name	Description
Communicator (comm)	[in] Associated communicator
Rank	[out] provides rank information for the calling process.

f) **MPI_Finalize:** int MPI_Finalize (void)

This routine completely closes the MPI execution environment and after this routine no MPI functions can be can execute even MPI_INIT method cannot be run. It is therefore recommended that the user should make sure that all the communications are finished before the call to MPI_Finalize routine.

A sample Program ("Hello World")

```
#include <stdio.h>
#include <mpi.h>
int main (int argc, char *argv[ ])
```

```
{
        int rank;
        int size;
        MPI_Init(&argc, &argv);
        MPI_Comm_size( MPI_COMM_WORLD, &size);
        MPI_Comm_rank( MPI_COMM_WORLD, &rank);
        printf("Hello world from process %d of %d\n", rank, size);
        MPI_Finalize();
        return 0;
}
```

3.5 MPI Data Types

Table 7. MPI Data Types.

Data Type Name	Description
MPI_CHAR	8-bit character
MPI_SHORT	32-bit floating point
MPI_INT	32-bit integer
MPI_LONG	32-bit integer
MPI_UNSIGNED_CHAR	8-bit unsigned character
MPI_UNSIGNED_SHORT	16-bit unsigned integer
MPI_UNSIGNED	32-bit unsigned integer
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	32-bit floating point
MPI_DOUBLE	64-bit floating point
MPI_LONG_DOUBLE	64-bit floating point
MPI_BYTE	Untyped byte data
MPI_PACKED	Packed data (byte)

Besides these, MPI also supports "*struct"*s and derived data types.

3.6 MPI Job Management

In this part the MPI job management is discussed and some commands are defined for job submission and job tracking. MPI uses a script file for managing the jobs. All the mandatory commands are written into this script file and some of the script commands are described below:

1) PBS -N used to give the name to the output file.
   e.g. PBS -N PGDBScan
2) PBS –q used to mention the queue used by the job.
   e.g. PBS –q cenga
3) PBS –M used to mention the email in order to notify any runtime errors via mentioned email address.

4) #PBS -m used to mention the condition when to send the email i.e. (a) for abort, (b) begin and (e) for end of the program.
e.g. #PBS –m a

5) mpiexec command to run the program executable.
A sample PBS file is given below.

```
#!/bin/sh
# name of the job -can be anything-
#PBS -N PGDBScan
# name of the queue that the job will be sent
#PBS -q cenga
# to use the environment variables of the shell that the job sent
#PBS -V
# e-mailadres
#PBS -M hunain.durrani@ceng.metu.edu.tr
# mailing options:
# (a)bort
# (b)egin
# (e)nd
#PBS -m a
# change directory ! YOU SHOULD CHANGE !
cd /home1/e1692433/ParallelDbScan
# run the executable over infiniband
mpiexec -hostfile $PBS_NODEFILE GDBScan 611170 networkDataset16MB.csv /tmp/cellDatasetFile /tmp/dbscanOutputFile normalOutputFile.csv anomalyOutputFile.csv 1
```

3.7 MPI Job Submission

After writing the script file we can submit the MPI job using the script file as parameter:

$ qsub –l nodes=n [script name]

After we submit the job, the job is automatically given a jobid such as:

9012.kavun-ib

We can use the above provided jobid to track the status of the job using the *qstat* command:

$ qstat 9012

We can also delete the job by using the *qdel* command:

$ qdel 9012

3.8 High Performance Computing

High-Performance Computing (HPC) is hardware which is a combination of multi processors. HPCs are used for the computation of huge amount of data which are not possible to compute on normal desktop computers which is generally comprised of a single processing chip named as CPU. An HPC system in general is a network node that contains multiple processing chips as well as its own RAM and hard disk storage.

3.9 Parallel Computing

HPC systems split up the programs into many smaller processes. These small processes run separately on different nodes and/or cores. In order to combine the results of these partial processes each process should be able to communicate with the others.

The purpose of HPC systems is to execute the programs that compute huge amount of data and require large amount of space in the memory. Standard file systems cannot work for large data because of their limited max data size. HPC systems can be efficiently process huge data and can perform data comparison in an efficient way.

For solving a problem we have to write a computer program which is a set of instructions given to computer, the computer will process these instructions serially, i.e. one instruction is executed at one time and the proceeding instruction must always wait for the preceding instruction to be executed. In this way suppose if our program has a large number of instructions and also has large number of data items to process, it may take too much time to finish the processing. Parallel computing overcomes this problem by dividing the big problem into fragments and let these fragments run independently on separate nodes.

Figures 4 and 5 **[13]** depict a clear picture for serial and parallel computing.

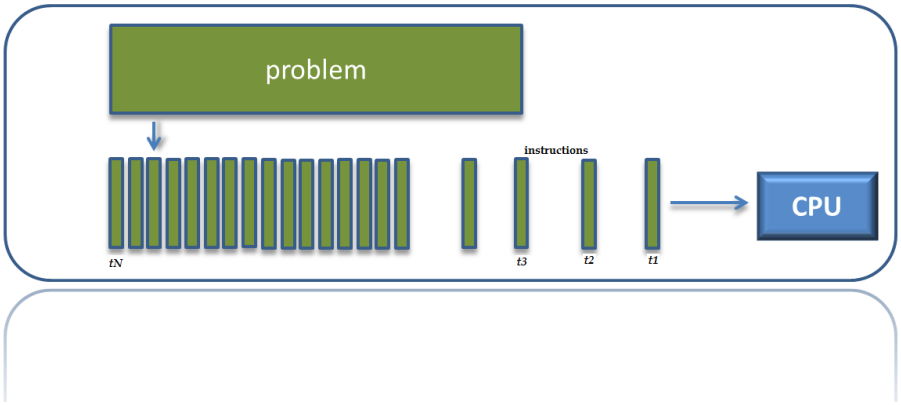

Figure 4. Serial computing [13].

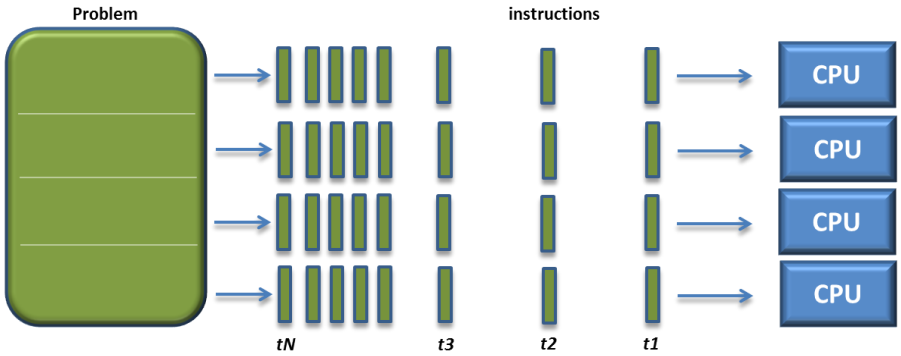

Figure 5. Parallel computing [13].

Doubling the number of processing nodes will reduce the processing time to half as compared to the previous execution time this process is known as speed-up parallelization. For some other reasons such as communication cost, the speed-up processing may not be linear.

3.10 An Example Parallel Program: Quick Merge Sort

The “Parallel Quick Merge Sort” shortly named as “Parallel QM Sort” is the combination of famous sorting algorithms “Quick Sort” and “Merge Sort”. An array of randomly generated integers has been used as dataset for experimental purposes.

3.10.1 Methodology

The methodology used in “Parallel QM Sort” is explained in the following steps:

1. The algorithm follows the master-slave mechanism.
2. The “Master Node” prepares the dataset by creating random integer numbers. The total number of elements to be produced is provided by the user as a command line parameter.
3. During the preparation of data the “Master Node” has to check whether the data scattered to each “Slave Node” is even or not. Suppose the total number of elements in the data set is 8 and the total number of nodes is 2 then, the data set will be evenly divided with a slot of 4 between each node but what if the total number of elements is 7, in this case the “Master Node” will add an extra element in order to make the distribution even. This process is explained in the Figures 8 and 9 and the process explained in this text is also presented.

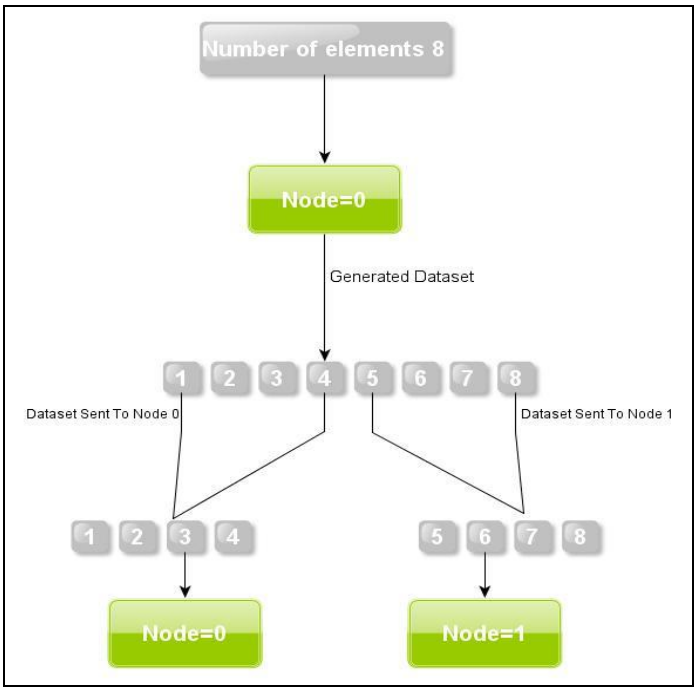

Figure 6. Even number of data elements.

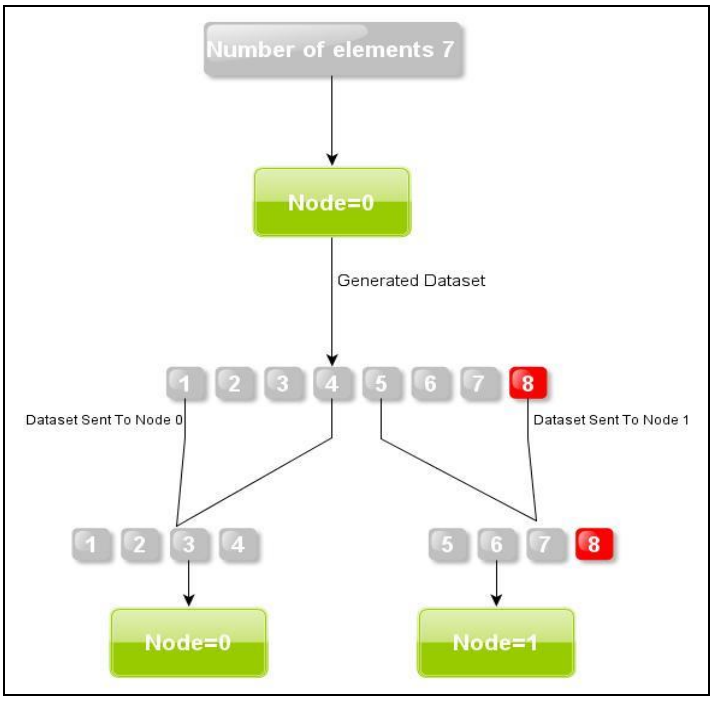

Figure 7. Extra elements inserted to balance the transmission.

4. After each "Slave Node" receives their part of the dataset, they perform the "Quick Sort" on their received dataset.
5. The merging phenomenon is the critical part of QM Sort since it merges all the data set parts into a single sorted data set. Suppose that S1 and S2 are two sorted parts of a bigger data set **S,** with elements n1 and n2 respectively. The merge function then compares each single value of S1 with S2 and saves the lower one in the resultant array. The efficiency of the merge function will be high if all the values in one part of data set is lower than the other part because the function will just have to append the part with higher values to the part with lower values and the merge function will have a complexity of **O**(n1 + 1) or **O**(n2 + 1). Where *n1* and *n2* are the number of elements in the subparts of the data set.

   The merge function will perform worst if the elements in both the parts are in shuffled state. In that case the merge function will show a complexity of **O**(n1 + n2).
   One way of implementing the merge function could be that the "Master Node" gathers the partitioned data set from each "Slave Node" and merge them respectively but this could be also very costly since the "Master Node" will have to call merge function for N-1 times where N is the number of nodes used to perform the sorting. Therefore the complexity rate will be **O**((N-1)(n1 + n2))

   In order to overcome this problem we used the tree based merge mechanism suggested byPuneetKataria**[7]**.In tree based merge sort each node sends its sorted data set to its neighbor and the respective node upon receiving the data set performs merge process and in this way the merge operation is performed at each step. This reduces the overhead burden of merging on

"Master Node" and the complexity to **O** ((log N)(n1 + n2)). The merging mechanism for 8 nodes is shown in Figure 10.

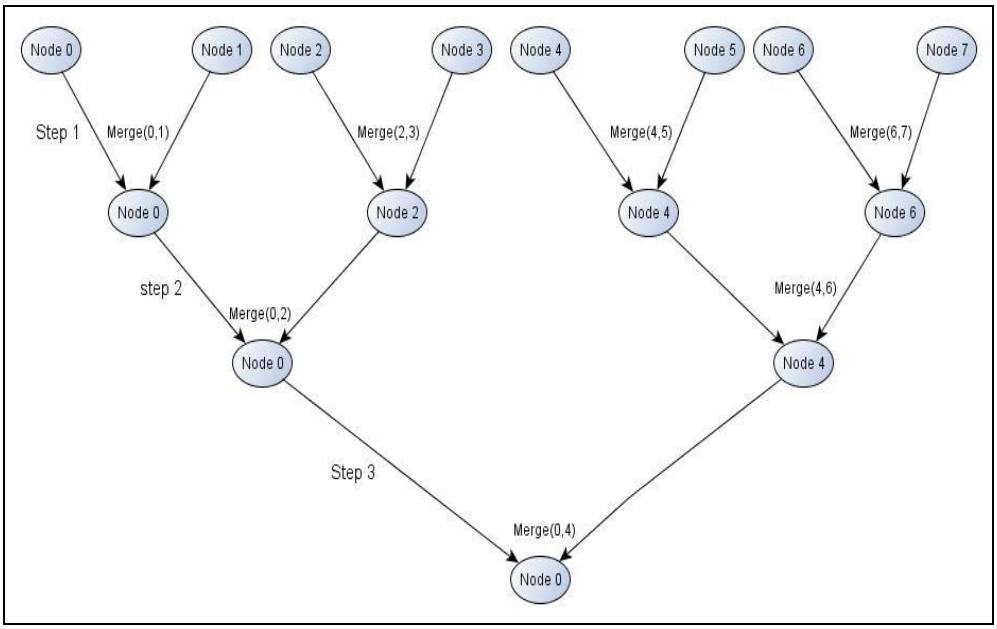

Figure 8. Tree merge phenomenon.

3.11 Experimental Measurements for QM Sort

We perform the QM Sort on different size of data set using 2, 4, 8, 16 and 32 processors. The table 8 provides the information about the data set we used.

Table 8. Dataset size used for QM Sort.

File Size	Number of Elements
2 MB	300,000
4 MB	600,000
8 MB	1180000
16 MB	2200000

We ran our algorithm on the above mentioned data set by keeping the data set size constant and increasing the number of nodes and evaluated the performance of the algorithm. Table 4.3-2 shows the time taken by each node for 2, 4, 8 and 16 MB of dataset size.

Table 9. Time taken by nodes to perform QM Sort in seconds.

Nodes	2 MB	4 MB	8 MB	16 MB
	Time Taken in Seconds			
2	0.05835720	0.12183700	0.19592400	0.40170900
4	0.03640380	0.07690410	0.145364	0.26718100
8	0.08321720	0.09822840	0.183156	0.30866300
16	0.18145300	0.18638300	0.25014600	0.33414700
32	0.38420600	0.397043	0.50261500	0.51052700

The graphical representation of the above table is presented in as line graph below. In the graphs above the Y-axis represents the time in seconds and X-axis represents the number of nodes.

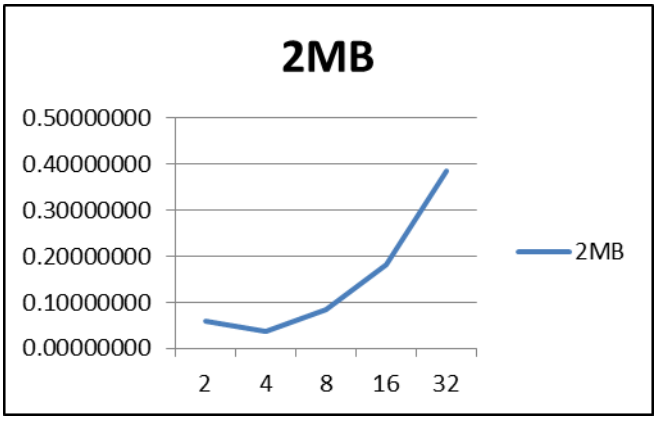

Figure 9. Relationships between time(seconds) and nodes for 2MB dataset.

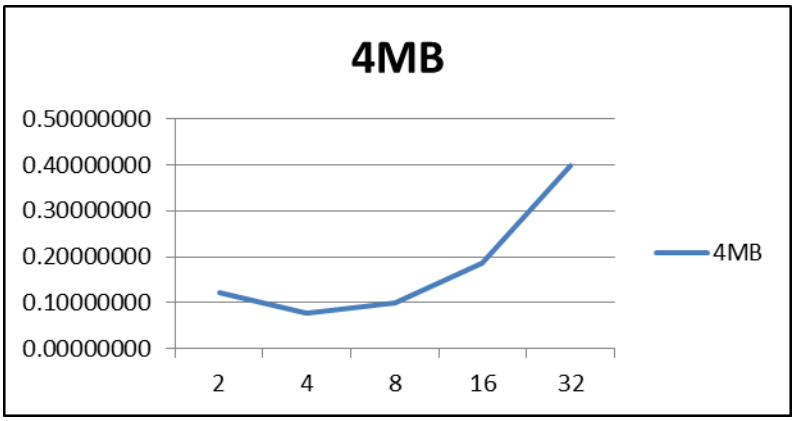

Figure 10. Relationships between Time in sec and nodes for 4MB dataset.

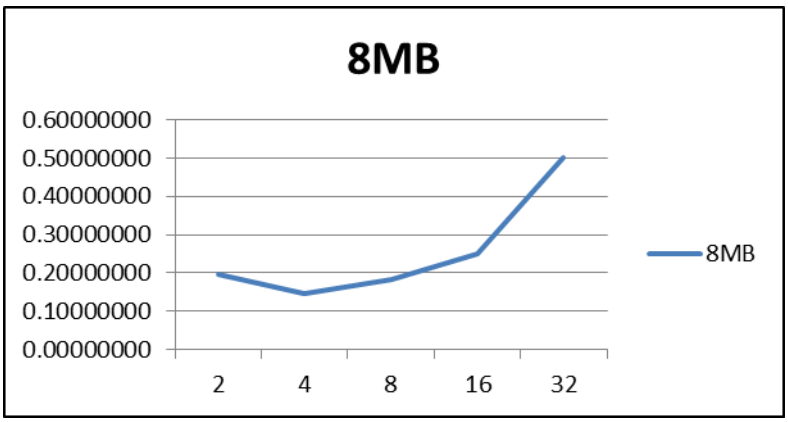

Figure 11. Relationship between time in sec and nodes for 8MB dataset.

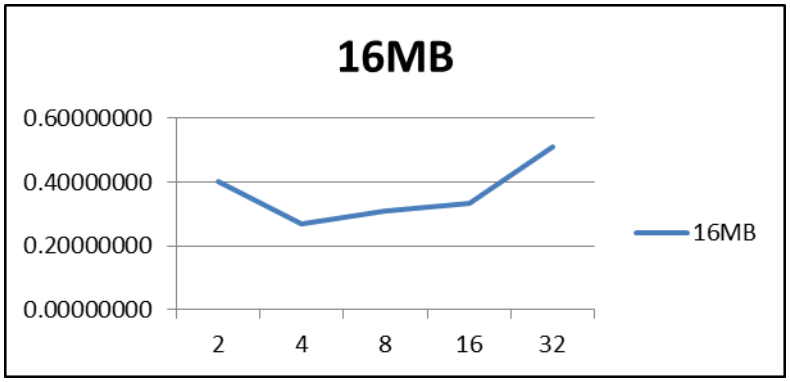

Figure 12. Relationship between time in sec and nodes for 16MB dataset.

In our experiments for "QM Sort" we kept the dataset constant but increased the number of nodes in order to check the behavior of the proposed method for "QM Sort". During the experiment we took 20 readings for each of the nodes and then calculated the average of those readings.

We can observe that there is a linear decrease and increase in the graph. This is due to the reason that as the number of nodes increases so as the communication time increases between the nodes for message passing. We observed that the behavior of the algorithm for 4 nodes is consistent which means that it keeps consuming the small time than the other nodes. Therefore we can say that the algorithm is suitable for 4 nodes. The time taken by 32 nodes is maximum due to the increase rate of the message communication between nodes.

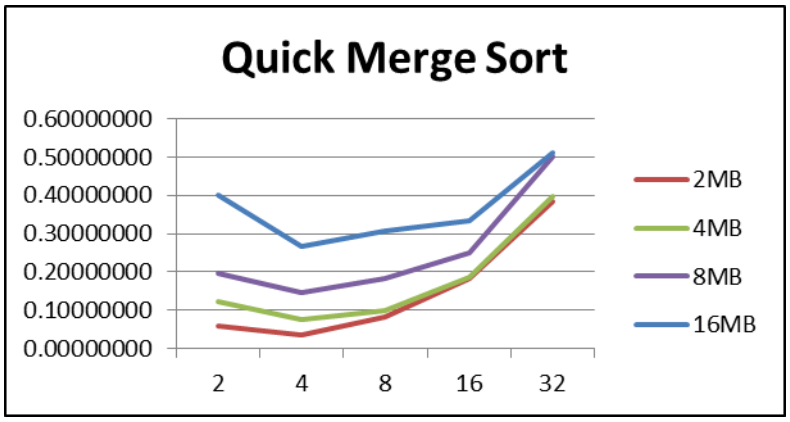

Figure 13. Combined graph for all of the dataset size.

Figure 15 is the combined version of all the four graphs mentioned above, here we can notice that each node shows a linear increase in time taken as the dataset size increases we can say there is a proportional relationship between time and the dataset size.

CHAPTER 4

IMPLEMENTATION OF K-MEANS AND DBSCAN

Since in large business scale organizations the quantity of data store is bigger enough to extract useful information (*cluster*) from that data. This is where data mining clustering algorithms play their role for human beings to extract the useful information from the irrelevant data store. But these algorithms show higher efficiency if the dataset is smaller which does not meet today's requirements. This is where the need of parallelizing the data mining clustering algorithms arises. In this chapter we will describing the methodology we used to parallelized the DBSCAN algorithm over the HPC hardware using C++ as programing too and MPI library for communication between the nodes.

Before we explain P-DBSCAN, since we are using Weka clustering libraries in next section we will briefly discuss about Weka.

4.1 Weka Machine Learning Tool

Weka (Waikato Environment for Knowledge Analysis) developed at university of University of Waikato, New Zealand in one of the popular tool of machine learning software written in Java. Weka together with it strong visualization tools and algorithms for data analysis and predictive modeling provides a user friendly interface for easy access to these functionalities. We can perform several standard data mining tasks using weka, such as data processing, clustering, classification, regression, visualization, and feature selection. Weka supports input data in the form of single flat file or relation. These input files contain data points that are described by a fixed number of attributes. By using weka we can also connect to SQL databases or oracle databases and can process the data derived from the database queries.

Weka's user interface contains an *explorer* which has several panels for different data mining operations. We can call weka library functions in our java or C++ codes such as, below where we are calling Weka's DBSCAN function in C++.

"*java -Xmx1024m –classpath Path/weka-3-6-1/weka.jar weka.clusterers.DBScan*

4.2 Parallel K-Means Implementation

We parallelize the K-Means between N numbers of nodes. The master slave methodology is being used. All the nodes read data from CSV files, master nodes make the initial selection of centroids and broadcasts them to other nodes then each slave node perform the K-Means independently, below is the state diagram for parallel K-Means.

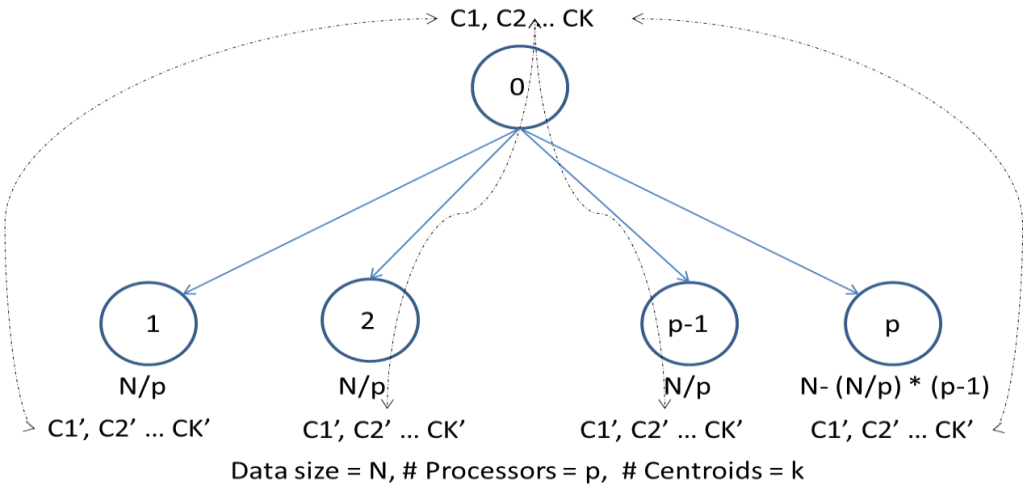

Figure 14. Parallel K-Means algorithm work distribution.

4.3 Parallel DBSCAN Implementation

Parallel DBScan is the parallel implementation of the classical sequential DBScan explained in section 2.5. It follows the Master and Slave format of parallelization. A CSV file is provided as an input to the master node which prepares the dataset. Then, the master node send point wise data to the slave node in order to calculate the neighbors of the point. The master node after receiving the neighbors for all the points from the slave nodes starts the DBScan algorithm. This is how we achieve the parallel DBScan. Below is the state diagram for parallel DBScan.

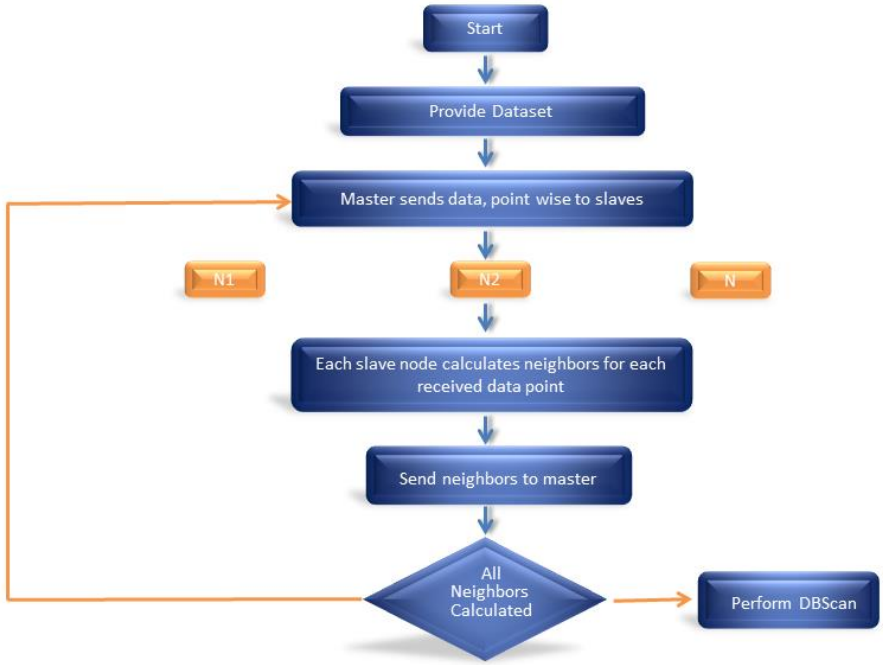

Figure 15. State diagram for parallel DBScan.

1 EXPERIMENTAL RESULTS

This chapter explains the performance test for “Parallel K-Means” and “Parallel DBScan” algorithms. The results of the experiments are explained with graphics created using Microsoft Excel.

This chapter starts with the discussion of the environment used for performing the performance tests followed by the discussion about the dataset used for each of the above algorithms. The last section of the chapter presents results taken from the performance tests experiments.

5.1 Experimental Environment

The entire performance test discussed in the following sections is performed on the cluster environment of the High Performance Computing laboratory at the Department of Computer Engineering at Middle East Technical University. The hardware properties of the High Performance Clustering machine consists of 2 x Intel Xeon E5430 Quad Core CPU 2.66 GHz, 12 MB L2 Cache,1333 MHz FSB processor a 16 GB of memory and 2 x 3Com 4200G 24 Port Gigabit Ethernet Switch & Voltaire 9240D 24 Port Infiniband Switch for connectivity.

Open MPI and mpich / mvapich libraries are installed on the cluster along with multiple compilers such as GNU Compiler Collection (gcc) and Intel C++ Compiler (icc). We can install additional software on the users root directory according to needs. Weka is one of the best data mining tools available these days; therefore we are using the DBScan libraries of weka in our algorithm. We installed weka on HPC on our root directory for utilization.
We used special MPI methods such as **MPI_Aint**, **MPI_Datatype** and **MPI_Address** for creating the custom data types other than the primitive data types of MPI.

5.2 Experimental Measurements

In this section we will be discussing about the measurements taken from the performed experiments. The section starts with describing the data set structure we used and next we discuss about the graphical representation of the measurements.

5.2.1 Structure of the Dataset used

We used two different datasets for our P-DBScan algorithm

- Network Dataset (KDD Dataset 1999)
- Wine Dataset

Table 10 and 11 show the fields for Network and Wine datasets respectively.

Table 10. Network Dataset Structure.

Feature Name	Description	Type
ProtocolType	Type of the protocol, e.g. tcp, udp, etc.	Discrete
Service	Network service on the destination, e.g., http, telnet, etc.	Discrete
Source Bytes	Number of data bytes from source to destination.	Continuous
Destination Bytes	Number of data bytes from destination to source.	Continuous
Flag	Normal or error status of the connection.	Discrete

Table 11. Wine Dataset Structure.

Field Name	Data Type
CultivarId	Integer
Alcohol	Float
MalicAcid	Float
Ash	Float
AshAlkalinity	Float
Magnesium	Float
TotalPhenols	Float
Flavonoids	Float
NonflavanoidPhenols	Float
Proanthocyanins	Float
ColorIntensity	Float
Hue	Float
OD280_OD315	Float
Proline	Float

Before performing the experiment directly on network dataset we used wine dataset for testing purpose. We ran our algorithm on the wine dataset and compared its result with the result generated by weka itself. We found no difference between the two generated outputs.

5.3 Comparison of Weka K-Means and our Sequential Approach

We made a comparison between weka and our proposed approach for sequential K-Means. The results taken from our experiments are shown in Table 12 and 13.

Table 12. Time analysis for weka.

	#Rows		
# Clusters	**40K**	**60K**	**80K**
10	4.64	7.42	3.33
15	11.1	7.35	5.99
20	17.58	9.55	7.33

Table 13. Time analysis for proposed sequential method.

	#Rows		
# Clusters	**40K**	**60K**	**80K**
10	1.8225	2.5445	1.3158
15	3.2179	6.2109	3.2195
20	6.9889	13.7919	3.8798

It is quite clear from the results that our proposed approach is faster than the Weka's K-Means. Our algorithm is almost 3 times faster than the weka's K-Means. It can be noted that as the number of centroids are increased the time take to perform K-Means is also increased and the increase rate is almost 2 times the previous time take for lesser number of centroids.

5.4 Graphical Representation of Sequential K-Means

Figures 18 and 19 present the graphical form of the above readings. The x axis represents the number of centroids whereas the y axis represents the time taken in seconds to run the algorithm. The graph also represents a linear increase in the time taken as the number of centroids increases.

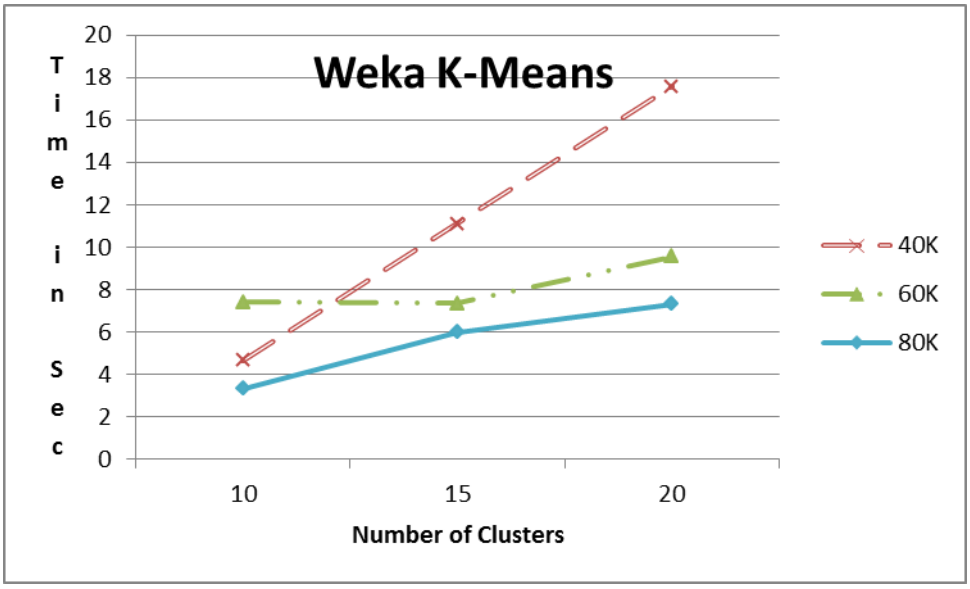

Figure 16. Weka K-Means.

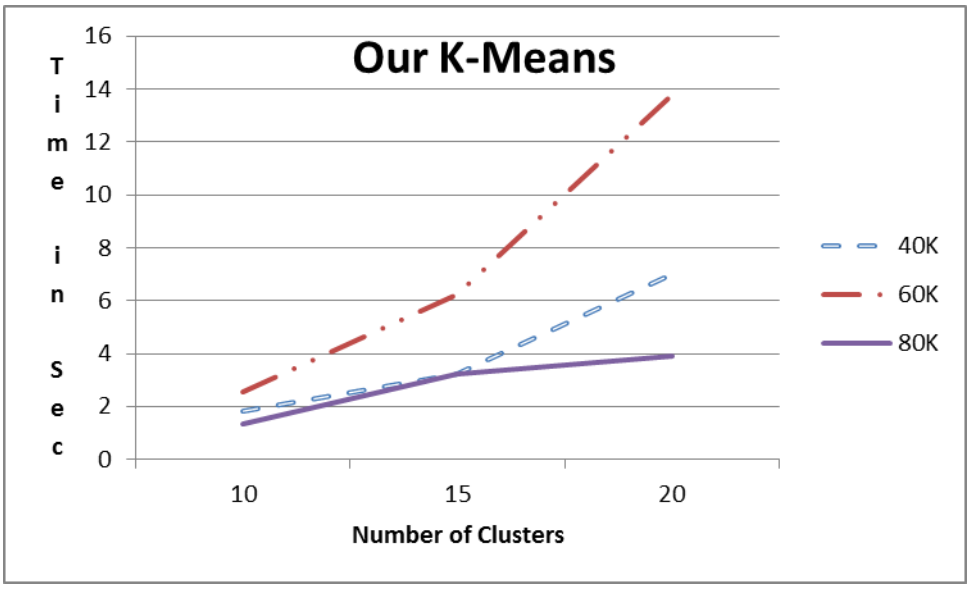

Figure 17. Proposed sequential K-Means.

5.5 Experimental Measurements for Parallel K-Means

In order to measure the performance of parallel K-Means we ran our algorithm for 40K, 60K and 80K respectively on 10, 20, 30, 40 and 50 nodes. We also kept the number of centroids as 10, 15 and 20. The tables from 14 to 18 show the measurements taken during the experiments.

Table 14. Computation time of Parallel K-Means in sec with K=10

Nodes	**10**	**20**	**30**	**40**	**50**
Data Set Size	**Time in Seconds**				
40K	0.520	0.375	0.273	0.221	0.198

60K	0.700	0.413	0.321	0.216	0.211
80K	0.937	0.714	0.621	0.442	0.342

Table 15. Computation time of Parallel K-Means in sec with K=15

Nodes	**10**	**20**	**30**	**40**	**50**
Data Set Size	**Time in Seconds**				
40K	2.763	1.887	1.576	1.321	1.112
60K	3.892	2.768	2.334	2.211	1.967
80K	4.454	3.879	2.678	2.567	2.311

Table 16. Computation time of Parallel K-Means in sec with K=20

Nodes	**10**	**20**	**30**	**40**	**50**
# Clusters	**Time in Seconds**				
40K	3.211	2.673	2.344	2.123	1.993
60K	4.231	4.007	3.784	3.238	2.778
80K	6.294	5.523	5.042	4.834	4.233

5.6 Graphical Representation of Parallel K-Means

The graphical representation of the results is shown below.

Figure 18. Parallel K-Means performance for 10 clusters.

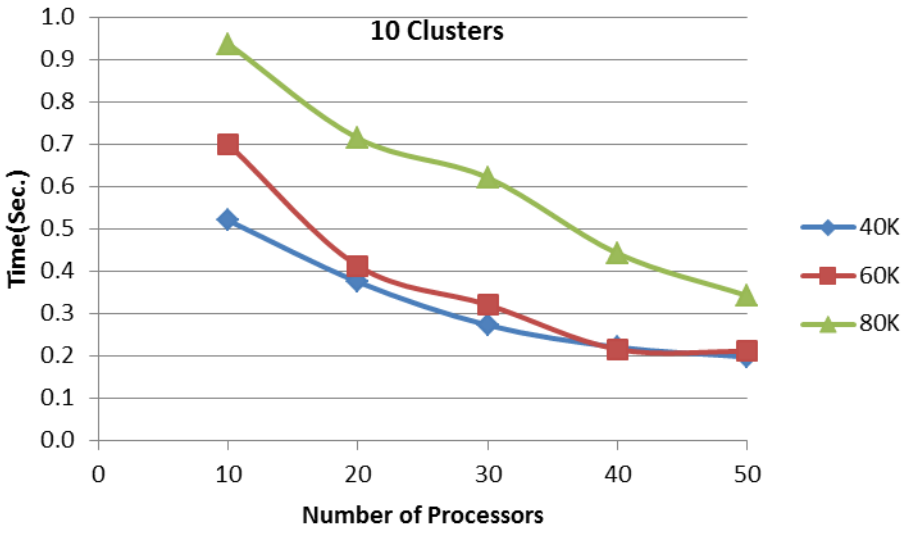

Figure 19. Parallel K-Means performance for 15 clusters.

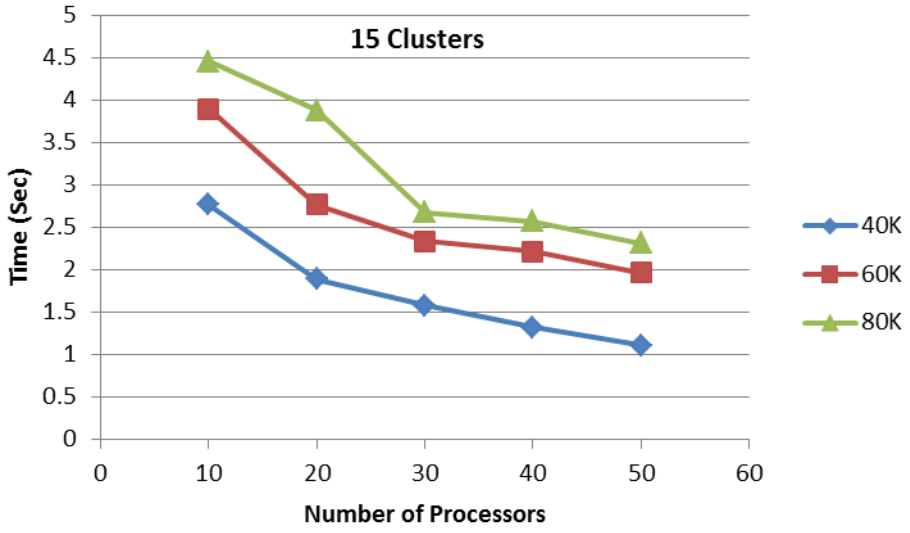

Figure 20. Parallel K-Means performance for 15 clusters.

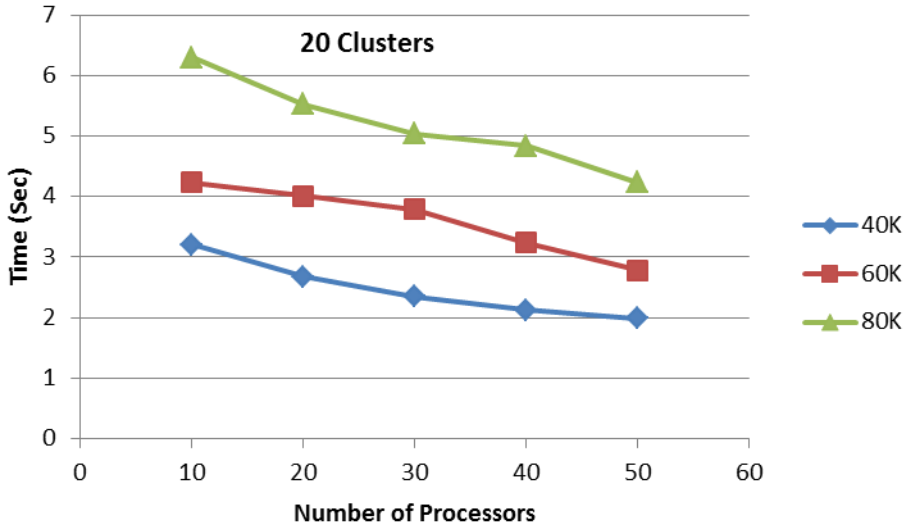

Figure 21. Parallel K-Means performance for 20 clusters.

5.7 Experimental Measurement for Sequential & Parallel DBScan

In this section we will be discus the experimental results taken for the sequential DBScan and parallel DBScan. We made a test run for our test dataset which wine in order to make the result confirmation. We matched our result with weka's output which was same as ours. We performed the experiments of network dataset.

5.8 Graphical Representation of Sequential DBScan

Tables 24 and 25 present the tabular and graphical form for sequential DBScan. The x-axis represents the dataset size in *K* whereas the y axis represents the time taken in seconds to run the algorithm. The graph also represents a linear increase in the time taken as the size of dataset increases.

Table 17. Sequential DBScan.

Dataset	Time (Sec)
2000	17
3000	43
4000	67
5000	97

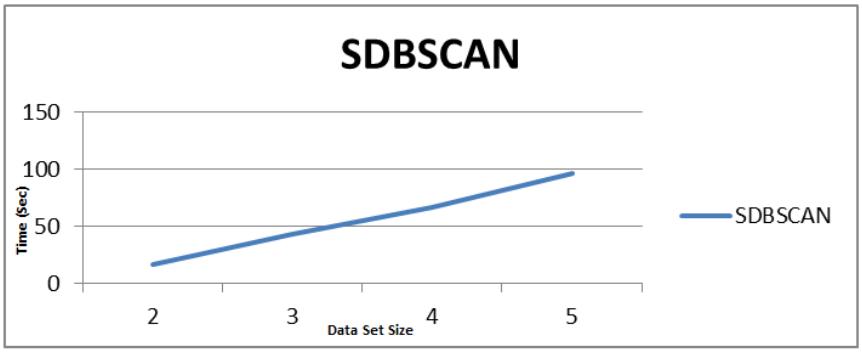

Figure 22. Sequential DBScan on network data.

5.9 Experimental Measurements for Parallel DBScan

In order to measure the performance of parallel DBScan we ran our algorithm for 20K, 40K, 60K and 80K respectively on 5, 10, 15, 20, 25, 30 and 40 nodes. The tables 20 show the measurements taken during the experiments.

Table 18. Parallel DBScan execution times with increasing number of nodes.

	Nodes						
Dataset size	**5**	**10**	**15**	**20**	**25**	**30**	**40**
Time (Sec)							
20K	136.465	74.5752	56.9424	48.5689	43.5946	40.4853	36.4378
40K	552.299	304.206	233.062	199.291	179.546	166.67	151.009
60K	3704.39	3168.17	2422.96	2139.73	2004.74	1985.02	1740.67
80K	12301.5	11858.3	11198.7	10253.8	9633.11	8882.79	8705.4

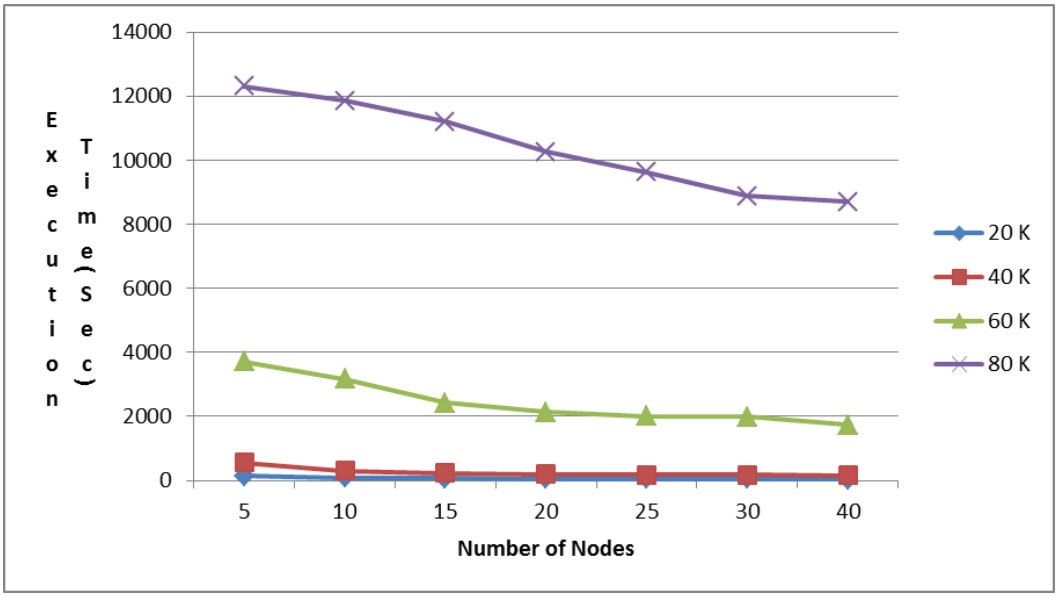

Figure 23. Parallel DBScan graph.

We can observe that there is almost a straight line for dataset size 2K since the size is very less and also there is also negligible amount of communication time between the nodes. On the other hand as the dataset size increases so as the clustering time increases. We can notice that as the number of nodes increases the clustering time decreases.

CHAPTER 6

CONCLUSIONS

In this last chapter of the thesis, an interpretation of the results presented in the previous chapter is given, providing conclusions and discussing possible future work which would be beneficial.

According to our previous discussions the rapid increase of data in large scale or medium scale business forced to develop the data mining techniques in parallel. Just developing these techniques in parallel would not be a solution but also achieving high performance would also be the basic need of this era data mining.

As our experiments show that the speed-up of the algorithm doubles as the number of nodes increase but in some cases in cannot be linear as of the data set size distributed to the nodes. Even then we can achieve efficiency from our approach. This is one of the achievements of our approach.

REFERENCES

[1] J. M.Jolion, P. Meer, and S. Bataouche. Robust clustering with applications in computer vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(8):791-802, Aug. 1991.

[2] R. Krishnapuram and C.-P. Freg. Fitting an unknown number of lines and planes to image data through compatible cluster merging. *Pattern Recognition*, 25(4), 1992.

[3] H. Frigui and R. Krishnapuram. Clustering by competitive agglomeration. *Pattern Recognition*, 30(7):1109-1119, 1997.

[4] K. G. Coffman and A. M. Odlyzko, Internet growth: Is there a "Moore's Law" for data traffic? , AT&T Labs, 2001.

[5] Eskin, E. , Arnold, A., Prerau, M., Portnoy, L.&Stolfo, S.A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data. (2002)

[6] Jiawei Han & Micheline Kamber, Data Mining: Concepts and Techniques, *3ed.*, Morgan-Kaufmann, 2012.

[7] Parallel Quick Sort Implementation Using MPI and PThreads. [PDBCCOB] W.K. Ng et al. (eds.): PAKDD 2006, LNAI 3918, pp. 179—188, 2006.

[8] Martin Ester, Hans-Peter Kriegel, Jörg Sander and XiaoweiXu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", The Second Int. Conf. on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA, 1996

[16] A Density Based Algorithm for Discovering Clusters in Large Spatial Databases With Noise, Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu

Online References:

[9]Url: http://publib.boulder.ibm.com/infocenter/clresctr/vxrx/topic/com.ibm.cluster.pe431.mpiprog.doc/am106_pmd.html, last accessed …

[10]Url: http://www.internetworldstats.com/stats.htm (2010)

[11] Geodetic Datum Overview, Last Date Accessed: 10/01/2010, Owner: Dana, Peter H. (2010), Url: http://www.colorado.edu/geography/gcraft/notes/datum/ datum.html

[12] GDI+, Last Date Accessed: 13/01/2010, Owner: Microsoft Corporation,(2010) Url:http://msdn.microsoft.com/en-us/library/ms533798(VS.85).aspx

[13] *Blaise Barney, Lawrence Livermore National Laboratory,* Introduction to Parallel Computing, Url: https://computing.llnl.gov/tutorials/parallel_comp/

[14] Amdahl's Law, Url: http://en.wikipedia.org/wiki/Amdahl%27s_law, http://en.wikipedia.org/wiki/Parallel_computing#cite_note-12

[15] Parallel and Distributed Data Mining by Dr (Mrs). Sujni Paul.*(http://www.intechopen.com/source/pdfs/13261/InTech-Parallel_and_distributed_data_mining.pdf)*