



Heterogeneous Computing Based K-Means Clustering Using Hadoop-MapReduce Framework

Sandip A. Ganage, Dr. R. C. Thool
Department of IT
SGGSIE&T, Nanded, India

Heshsham Abdul Basit
HBeonLabs Technologies Pvt.Ltd
Greater Noida, India

Abstract— *K-means is a well-known clustering algorithm in the field of data mining. It is simple to implement and its speed allows it to run on large data sets. However, it also has a drawback. Advancement in many data collection techniques has been generating enormous amount of data, leaving scientists with the challenging task of processing them. Its performance will not be sufficient when it has to deal with large data sets. To solve this problem, a method is proposed in this paper by which k-means will be implemented using OpenCL heterogeneous computing platform with the help of Hadoop-MapReduce framework. MapReduce is a framework which is pioneered by Goggle for distributed programming. It includes user specified Map and Reduce functions which process inputs in the form of key/value pairs. Along with the MapReduce paradigm, Hadoop also implements HDFS which is known distributed file system. GPU Computing with many-core graphics processors is playing today an important role in the advancements of modern highly concurrent processors. Their ability to accelerate computation is being explored under several scientific fields. OpenCL is a heterogeneous computing platform and one of the widely used for GPU Computing. In the current paper we present the acceleration of a widely used data clustering algorithm, K-means, implemented using Hadoop & MapReduce framework, in the context of heterogeneous computing devices like CPUs and GPUs.*

Keywords— *Hadoop, MapReduce, GPU, OpenCL, HDFS*

I. INTRODUCTION

Cluster analysis is a study of algorithms and methods of classifying objects. Cluster analysis does not label or tag and assign an object into a pre-existent structure; instead, the objective is to find a valid organization of the existing data and thereby to identify a structure in the data. It is an important tool to explore pattern recognition and artificial learning. The k-means algorithm [1] is a well-known clustering method and widely used in various applications. It aims to partition n objects into 'K' clusters in which each object belongs to the cluster with the nearest mean. The k-means algorithm requires three user-specified parameters: number of clusters 'k', cluster initialization, and distance metric. The k-means algorithm assigns each data point to a closest cluster. For a data point, its distance to a cluster is defined as its distance to the centroid of the cluster, where the centroid of a cluster is defined as the mean position of all the data points contained in the cluster. K-means algorithm uses an iterative approach. Initially, the positions of k clusters' centroids are randomly chosen. In a standard k-means iteration, each data point is labelled to the nearest cluster. After all the data points labelled, the centroid of each cluster is then be updated according to its data points. The labelling step and updating step are iterated until the labels do not change any more. Parallelizing the K-means algorithm has also received a lot of research attention [2]. MapReduce is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real world tasks. Users specify the computation in terms of map and reduce function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of networks and disks. Google and Hadoop both provide MapReduce runtimes with fault tolerance and dynamic flexibility support [3]. Comparing with multicore CPUs, concurrent threads in GPUs is far more than CPUs. Hundreds of lightweight threads can be easily launched in concurrent. And this number, with the development of chip technologies, will continually increase without the limitation of Moore's law [4]. As a general-purpose and high performance parallel hardware, Graphics Processing Units (GPUs) develop continuously and provide another promising platform for parallelizing k-Means. GPUs are dedicated hardware for manipulating computer graphics. Due to the huge computing demand for real-time and high-definition 3D graphics, the GPUs have evolved into highly parallel many-core processors. The advances of computing power and memory bandwidth in GPUs have driven the development of general-purpose computing on GPUs (GPGPU) [5]. This paper focuses on the design, implementation, and performance evaluation of k-means clustering algorithm implemented using Hadoop - MapReduce framework on the multicore architecture of GPU using OpenCL Heterogeneous Computing Platform [6]. Rest of the paper is organized as follows. We studied the previous approaches for improving the performance or parallelizing the traditional algorithm in section II. Section III describes the preliminaries i.e. basics of the algorithm, Hadoop-MapReduce and OpenCL. Section IV describes the proposed framework along with the algorithms and pseudo codes and system flow diagram. Section V describes the experimental results and we conclude with section VI.

II. BACKGROUND AND RELATED WORK

Following are the early implementations of k-means along with the MapReduce framework or GPU Computing in order to implement the algorithm in parallel manner:

In [8] Reza Farivar, Daniel Rebolledo, Ellick Chan, Roy Campbell tried to implement the traditional k-means algorithm in parallel way using the GPU Computing platform NVIDIA G80 PCI express graphics board using the CUDA processing extensions. In [9] Mario Zechner and Michael Granitzer an optimized k-means implementation on Graphics Processing Unit (GPU) is presented. NVIDIA's Compute Unified Device Architecture (CUDA) available from the G80 family onwards is used as the programming environment. In [4] Miao Xin and Hao Li presented an approach of MapReduce with GPU acceleration on general platform, which is implemented by Hadoop framework [10] with Open Computing Language [11]. They attempted to extend the parallelism of MapReduce model, from multi-machine alone to multi-machine with multicore, so as to enlarge the use scope of Hadoop framework.

In [12] Ping ZHOU, Jingsheng LEI, Wenjun YE proposed a MapReduce and the Hadoop distributed clustering algorithm, the design and the implementation of the large-scale data processing model based on MapReduce and Hadoop. Summarizing the presented previous work the following can be observed:

- Some of the implementations are in homogenous CUDA platform which is platform independent. This system proposes the solution which uses OpenCL - Heterogeneous Computing platform.
- Previous implementations of K-Means on GPU uses traditional algorithm. This paper proposed a new method which divides the original algorithm into two different functions 'Map' and 'Reduce' which are executed separately one after another.
- Large amount of data can be given to the K-Means as input. However such input requires a framework to handle the input, like a File system. Previous implementations were based on file handling operations. This paper proposes an idea of using a distributed file system which handles the input and output data very efficiently.

Based on previous work the main contribution of this paper are as follows:

1. A parallel implementation of standard k-means using the non-graphics oriented programming model of OpenCL which is also known as a heterogeneous computing platform.
2. Removal of the limitations of classic k-means which was suitable for serial execution. Instead of using traditional approach, MapReduce technique used to implement the algorithm which simplifies the parallel implementation of the algorithm on multiple GPUs showing high speedups when compared to the speedup of previous GPU-based implementations.

K-Means may expect the input with a very large number of objects. Using a file system to manage the input – output. HDFS – a distributed file system is used to manage the input files, intermediate input files and output files which can be redirected as input. It gives flexibility to implement the parallel algorithm on multiple nodes.

III. PRELIMINARIES

A. K-Means algorithm

K-Means algorithm works in the following way: Given a set of observations ($P_1 \dots n$), where each observation is a d -dimensional vector, k-means clustering aims to partition these n observations into k clusters ($k \leq n$) $S = \{S_1, S_2, \dots, S_k\}$ so as to minimize the within-cluster sum of squares (WCSS):

$$\arg \min_s \sum_{i=1}^k \sum_{P_j \in S_i} \|P_j - C_i\|^2$$

This algorithm, starts off with the selection of the k initial random cluster centres from the n objects. Each remaining object is assigned to one of the initial chosen centres based on similarity measure. When all the n objects are assigned, the new mean is calculated for each cluster. These two steps of assigning objects and calculating new cluster centres are repeated iteratively until the convergence criterion is met. Comparing the similarity measure is the most intensive calculation in k-means clustering. For n objects to be assigned into k clusters, the algorithm will have to perform a total of nk distance computations. While the distance calculation between any object and a cluster centre can be performed in parallel, each iteration will have to be performed serially as the centre changes will have to be computed each time.

B. MapReduce model and Hadoop

The original MapReduce model is proposed by Google for the purpose of supporting its critical services such as web search, log analysis, data mining, etc. in petabyte-level datasets [4]. MapReduce is a programming model and an associated implementation for processing and generating large datasets. A typical MapReduce computation processes many terabytes of data on thousands of machines. As shown in Figure 1, MapReduce usually splits the input dataset into independent chunks. The number of splits depends on the size of the dataset and the number of nodes available. Users specify a map function that processes a (key, value) pair to generate a set of intermediate (key, value) pairs, and a reduce function that merges all intermediate values associated with the same intermediate key [7].

The MapReduce framework has a single master Job Tracker and multiple Task Trackers. Potentially, each node in the cluster can be a slave Task Tracker. The master manages the partitioning of input data, scheduling of tasks, machine failures, reassignment of failed tasks, inter-machine communications and monitoring the task status. The slaves execute

the tasks assigned by the master. Both input and output are stored in the file-system. The single Job Tracker can be a single point failure in this framework.

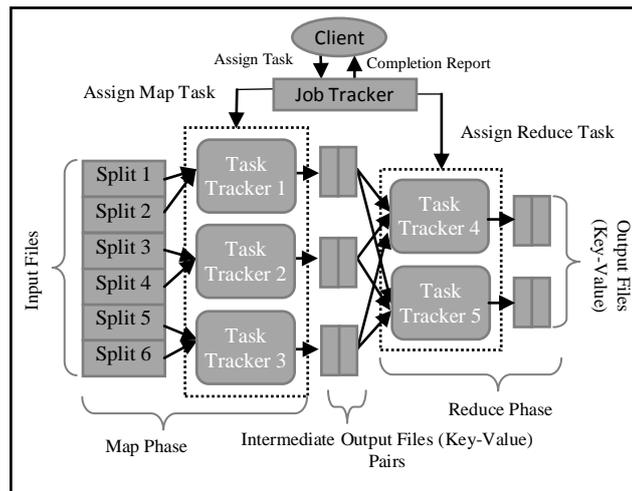


Figure 1 Architecture of MapReduce Framework

Hadoop is an open source software framework that can run large data-intensive, distributed applications and can be installed on commodity Linux clusters. Hadoop comes with its own file system called the Hadoop Distributed File System (HDFS) and a strong infrastructural support for managing and processing huge petabytes of data.

Each HDFS cluster consists of one unique server called the Namenode that manages the namespace of the file system, determines the mapping of blocks to Datanodes, and regulates file access. Each node in the HDFS cluster is a Datanode that manages the storage attached to it. The datanodes are responsible for serving read and write requests from the clients and performing block creation, deletion and replication instructions from the Namenode. HDFS is built to deal with large files by breaking them into blocks and replicating them across the network. The common policy is to replicate each file three times. HDFS will place two replicas on two different nodes on the same rack and place the third replica on a different node in a different rack. This ensures reliability even without a RAID backup. However, HDFS is based on the Google File System model that follows the write-once-read-many access model that are not meant for continuous updates of data.

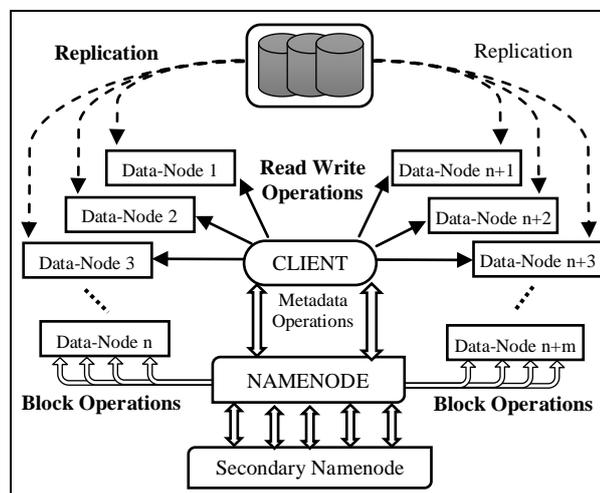


Figure 2 Architecture of HDFS

C. GPGPU and OpenCL

The GPU is a many-core device with multiple SIMD multiprocessors that can run hundreds of concurrent threads. Originally, it was a dedicated processor for graphics processing acceleration. But recently, being an extension of stream processor, the General Purpose Graphics Processing Unit (GPGPU) turns the massive floating-point computational power of a modern graphics accelerators' shader pipeline into general purpose computing power [4]. For the development, some GPGPU frameworks and platforms have been developed. NVIDIA has proposed own GPU Computing framework CUDA (Compute Unified Device Architecture), CTM is by AMD/ATI, and Brook+ by Stanford University. OpenCL is a programming framework and standard set from Khronos, for heterogeneous parallel computing on cross-vendor and cross-platform hardware. It provides a top level abstraction for low level hardware routines as well as consistent memory and execution models for dealing with massively-parallel code execution. The advantage of this abstraction layer is the ability to scale code from simple embedded microcontrollers to general purpose CPUs from Intel and AMD, up to massively-parallel GPGPU hardware pipelines, all without reworking code. While the OpenCL standard allows OpenCL code to

execute on CPU devices, this paper will focus specifically on using OpenCL with Nvidia and ATI graphics cards as this represents (in the authors opinion) the pinnacle of consumer-level high-performance computing in terms of raw FLOPS throughput, and has significant potential for accelerating “suitable” parallel algorithms.[8]

An OpenCL program usually consists of two parts: [4]

1) *Kernel code*: Kernel code is the programs that run on multicore devices (e.g. GPUs). It is written by the OpenCL language, which is similar to C language.

2) *Platform control program*: For the purpose of controlling the OpenCL runtime environment, a suit of API is necessary. It has several language bindings that can be chose by programmers, such as C, C++, Java, etc.

IV. PROPOSED FRAMEWORK

In [9] describes the comparison between implementation of sequential as well as parallel k - means. It explains the algorithm for sequential approach. In sequential approach initially K- Centroids are chosen randomly from the set of data points X. In the next stage each data object is compared with each centroid to find the closest centroid to that object. This is called as labelling stage. After finishing with labelling stage, each centroid is then recalculated by mean of all data points via (equation). The labelling and centroid update stage are executed repeatedly until centroids no longer change.

In [3] suggested the method of parallelizing K-Means clustering using Hadoop - MapReduce framework on clusters of CPU. The map function performs the procedure of assigning each sample to the closest centre while the reduce function performs the procedure of updating the new centres. In order to decrease the cost of network communication, a combiner function is developed to deal with partial combination of the intermediate values with the same key within the same map task. [3]

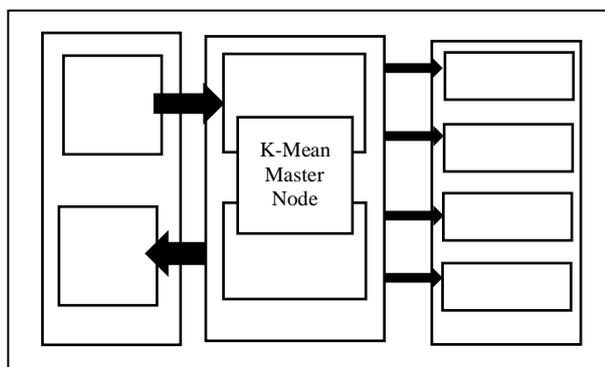


Figure 3 Proposed Framework

A. Mapper Function

Input dataset which contains the records is stored in HDFS in the form of <Key, Value> pairs. Multiple datanodes simultaneously execute this Map function, hence referred as Mappers. Input dataset is split and distributed to each Mapper as input. Mapper performs labelling operations, i.e. find out nearest centroid to the corresponding data object. This is done using Euclidean distance to measure proximity of points. The distance between each point and each centroid is measured and the shortest distance is calculated. Then, the object is assigned to its closest centroid. When all the objects are assigned to the centroids, the mapper sends all the input objects and the centroids they are assigned to the intermediate function which performs sorting and regrouping of data objects, which is further given as input the reducers function. Algorithm for the Mapper function is given in Algorithm 1.

B. Sorting And Grouping

Output of Mappers will be list of <Key, Value> pairs. In order to provide the input to the Reducers, this list should be sorted out using this intermediate function. This function also computes the number of data - objects assigned to each cluster, which will be used by the Reducers to compute the mean of the objects for each cluster. It works same as the intermediate function ‘Combiner’ implemented in [3]. This function needs serial implementation on CPU.

Algorithm 1 Algorithm for Mapper

Input: A set of objects $X = \{x_1, x_2, \dots, x_n\}$

A Set of initial Centroids $C = \{c_1, c_2, \dots, c_k\}$

Output: A output list which contains pairs of (C_i, x_j) where $1 \leq i \leq n$ and $1 \leq j \leq k$

Procedure

$M1 \leftarrow \{x_1, x_2, \dots, x_m\}$

current_centroids $\leftarrow C$

distance $(p, q) = \sqrt{\sum_{i=1}^d (p_i - q_i)^2}$ (where p_i (or q_i) is the coordinate of p (or q) in dimension i)

for all $x_i \in M1$ such that $1 \leq i \leq m$ **do**

 bestCentroid \leftarrow null

 minDist $\leftarrow \infty$

for all $c \in$ current_centroids **do**

 dist \leftarrow distance (x_i, c)

```

if bestCentroid = null || dist < minDist then
    minDist ← dist
    bestCentroid ← c
end if
end for
Outputlist << (bestCentroid, xi)
i += 1
end for
return outputlist
    
```

C. Reducer Function

The reducer accepts the <Key, Value> pairs output from the intermediate Sorting and Grouping function and calculates new centroid values. For each centroid, the reducer calculates a new value based on the objects assigned to it in that iteration. This is calculated by taking the mean of data objects within each clusters. This new centroid list is emitted as the reducer output. Algorithm for the reducer function is given in Algorithm 2.

Algorithm 2 Algorithm for Reducer

Input: (Key, Value)

where key = bestCentroid and Value = Objects assigned to the centroid by the mapper

Output: (Key, Value)

where key = oldCentroid and value = newBestCentroid which is the new centroid value calculated for that bestCentroid

Procedure

```

outputlist ← outputlist from mappers
 $\mathcal{D} \leftarrow \{ \}$ 
newCentroidList ← null
for all  $\beta \in$  outputlist do
    centroid ←  $\beta$ .key
    object ←  $\beta$ .value
     $\mathcal{D}$ [centroid] ← object
end for
for all centroid  $\in$   $\mathcal{D}$  do
    newCentroid, sumofObjects, numofObjects ← null
    for all object  $\in$   $\mathcal{D}$ [centroid] do
        sumofObjects += object
        numofObjects += 1
    end for
    newCentroid ← (sumofObjects ÷ numofObjects)
    newCentroidList << (newCentroid)
end for
return newCentroidList
    
```

D. OpenCL Implementation

This paper proposes the idea of implementing the major computational parts Mappers and Reducers on the GPU. Rest of the part including the intermediate function for sorting and portioning will be executed by the CPU. GPU achieves massive parallelism through simultaneously launching of a large number of lightweight threads. The GPU threads are scheduled onto processor cores in the unit of a warp rather than of a single thread. Different warps execute independently regardless of whether they are executing common or disjoint code paths. Threads within a warp execute the same instruction at a time but on different pieces of data, so maximum efficiency is achieved when all threads of a warp have

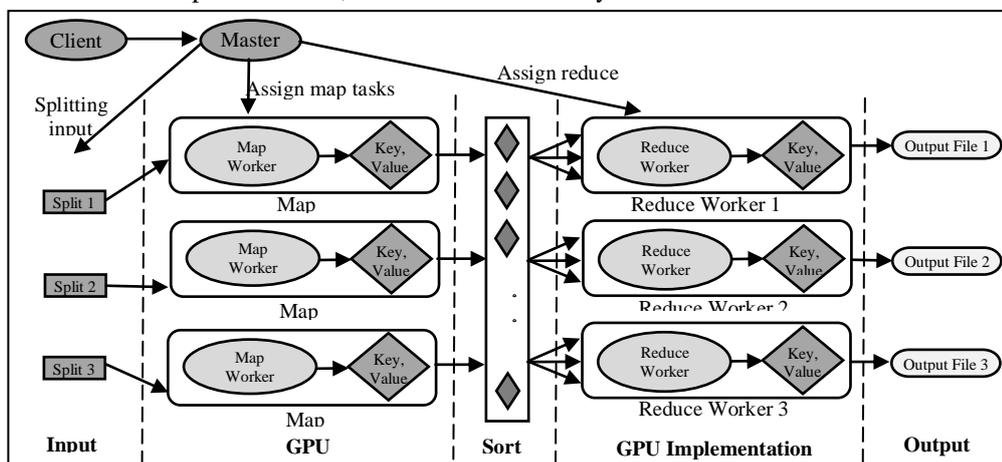


Figure 4 System Flow

identical execution path. If a warp of threads diverge on data-dependent conditional branches, the warp will sequentially execute all the branch paths, and subsequently nullify results from threads that are not on that path. The threads converge back to the same execution path after all paths are completed. Note that diverged execution paths often cause significant degradation to the efficiency of GPU programs. In summary, if an algorithm fits the Single Instruction Multiple Thread (SIMT) model (with few diverged execution paths), GPU can achieve substantial increase of computing performance. [2]

Initially all data objects are stored in the main memory of host machine. In order to execute the Mapper and Reducer, it is important to transfer data from the host's global memory to the device's global memory. As stated before, GPU achieves massive parallelism through simultaneously launching of a large number of lightweight threads. To execute this mapper kernel, number of threads equal to the number of data objects are called by `clEnqueueNDRRangeKernel`. Each thread has its own global id. Global id of each thread is accessed by `get_global_id(0)` function. Following is the pseudo code for mapper function.

Pseudo Code 1 Mapper on GPU

```
i ← get_global_id(0)
dmin = MAX_VALUE
for j=0 to no_of_centroids do
    dx = 0.0
    for k=0 to no_of_attributes do
        dx = (cent[j]->attr[k] - data[i]->attr[k])2
    end for
    if(dx < dmin)
        bestCent = j
        dmin = dx
    end if
    Key[i] = bestCent
    Value[i] = data[i]->id
end for
```

The output of Mapper is in the form of <Key, Value> pairs which is then transferred from device's main memory to host's main memory. This output is sorted and partitioned centroid wise. Also the number of data objects assigned to each centroid is computed in this stage.

Reducer does the remaining job of computing the mean of every cluster in order to get the new centroid values. Reducer kernel usually perform the reduction operation, i.e. perform addition of data objects. Following shows the pseudo code for reducer function.

Pseudo Code 2 Reduce on GPU

```
i ← get_local_id(0)
n ← get_local_size(0)
for j= 0 to no_of_attributes
    sum_reduction[j] += dataObjects[i] -> attribute[j]
end for
thread_barrier
if(i <= no_of_attributes)
    mean[j] = partial_sum[j] / n
```

In the above code sum reduction is used, as number of threads will be simultaneously try to write the single memory location which causes the Race Condition between these threads. These new centroids are given as input to the mapper function. Mapper again performs computation to find the nearest centroid to each data point. This cycles is repeated until the convergence criterion is reached.

V. EXPERIMENTS AND RESULTS

In this section we perform the analysis of the proposed system and comparison with the sequential implementations of K-Means.

A. Experimental Environment

The experimental environment is deployed on a node cluster of DELL commodity systems. The cluster configuration is:

- Intel (R) Core (TM) i7-3770K CPU @ 3.50 GHZ
- 8GB RAM per node
- One NVIDIA GeForce GTX 480 on 3 nodes
- One NVIDIA Tesla C2075 on one node
- 1 SATA disk per a node
- 1 Gigabit Ethernet on each node
- Ubuntu11.10 Linux
- Oracle Java JDK 1.6.0
- Hadoop 1.0.4

- OpenCL 1.2

The 4 nodes in this experiments are used to deploy Hadoop framework. HDFS is the native distributed file system here. Title of this paper suggests that the proposed system is for heterogeneous platform. OpenCL is basically known as heterogeneous computing platform which includes programs that execute across platforms consisting of central processing units (CPUs), graphics processing units (GPUs), digital signal processors (DSPs) and ARM Processors. In this experiment four NVIDIA's GPU are used to perform the computations.

B. Experimental Results

Experimental results were obtained on dataset which are categorized into different groups. Each group is having different number of data points and attributes. Datasets with 10000, 100000, and 1000000 data points. Each dataset is computed

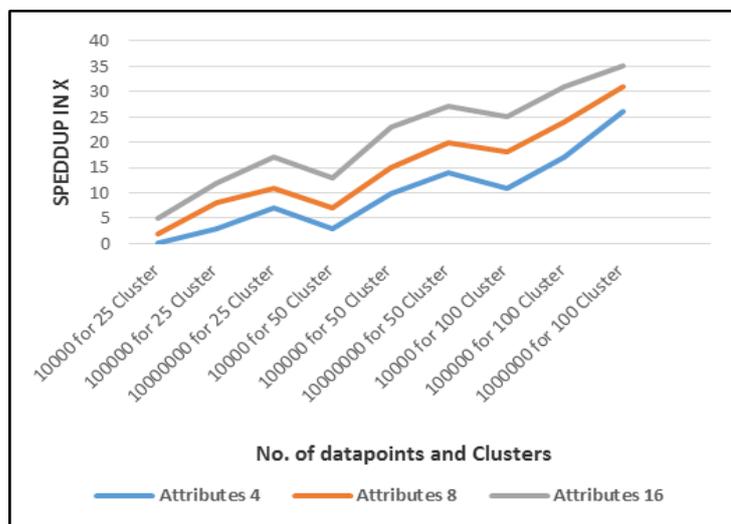


Figure 5 Performance

with 4, 8 and 16 dimensions and computed for different number of clusters (25, 50, and 100). Figure 5 presents speedups gained by using the proposed system at different levels. For comparison purpose the system was compared with sequential implementation of the same. It clearly demonstrates the speed gaps between serial implementation and parallel implementations on multiple nodes. The total execution time involves the time taken to copy and distribute the data, the time taken by Mapper and Reducer functions, time taken to perform sorting of data, and finally the time taken to write back data and result. Overall, our proposed system procures ample speedups over the serial implementations on a single node.

VI. CONCLUSIONS

In this paper, we presented a method by which we have implemented K-Means clustering algorithm on a parallel framework. This idea mainly focuses on distributing the core computational part of the algorithm on multiple nodes. The incorporation of the heterogeneous computing platform of OpenCL along with the Hadoop-MapReduce framework has demonstrated significant performance gain and speedup in our experiment. The algorithm has been tested with different number of dimensions, datapoints and clusters.

Future work will include experimenting with the other variations of k-means such as automatic centroid selection and also load balancing among nodes which performs the computation.

REFERENCES

- [1]. A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651-666, 2010
- [2]. Jiadong Wu; Bo Hong, "An Efficient k-Means Algorithm on CUDA," *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, vol., no., pp.1740,1749, 2011
- [3]. Weizhong Zhao, Hui Fang Ma, Qing He, "Parallel K-Means Clustering Based on MapReduce"
- [4]. Miao Xin; Hao Li, "An Implementation of GPU Accelerated MapReduce: Using Hadoop with OpenCL for Data- and Compute-Intensive Jobs," *Service Sciences (IJCSS), 2012 International Joint Conference on*, vol., no., pp.6,11, 2012
- [5]. You Li; Kaiyong Zhao; Xiaowen Chu; Jiming Liu, "Speeding up K-Means Algorithm by GPUs," *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, vol., no., pp.115,122, 2010
- [6]. J. Tompson and K. Schlachter, *An Introduction to the OpenCL Programming Model*, New York: NYU: Media Research Lab, 2012.
- [7]. R. Farivar, D. Rebolledo, E. Chan and R. Campbell, "A Parallel Implementation of K-Means Clustering on GPUs," in *WorldComp 2008*, Las Vegas, Nevada, 2008.
- [8]. M. Zechner and M. Granitzer, "Accelerating K-Means on the Graphics Processor via CUDA," in *First International Conference on Intensive Applications and Services*, Austria, 2009

- [9] . <http://hadoop.apache.org>
- [10] . <http://www.khronos.org/opencv>
- [11] . P. ZHOU, J. LEI and W. YE, "Large-Scale Data Sets Clustering Based on MapReduce and Hadoop," *Journal of Computational Information Systems*, vol. 7, no. 16, pp. 5956- 5963, 2011
- [12] . J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM - 50th anniversary issue: 1958 - 2008*, vol. 51, no. 1, pp. 107-113 , 2008