IJCSE International Journal of Computer Sciences and Engineering Open Access
Review Paper Volume-5, Issue-10 E-ISSN: 2347-2693

A Brief Account of Iterative Big Data Clustering Algorithms

M. Shankar Lingam[1], A. M. Sudhakara[2*]

[1]University of Mysore, Manasa Gangotri, Mysore, India
[2*]Director, CIST, University of Mysore, India

**Corresponding Author: sudhakara.mysore@gmail.com*

Available online at: www.ijcseonline.org

Received: 02/Oct/2017, Revised: 10/Sep/2017, Accepted: 20/Oct/2017, Published: 30/Oct/2017

Abstract- Today, maximum of the organizations have to deal with big quantities of records, that is hastily growing. In order to address these explosively growing amounts of information, one has so that it will extract, examine, and process information time to time. Clustering has for this reason been identified keeping in view this example and it is considered as an essential device used to analyze huge statistics. Technological progress, specifically inside the regions of finance and enterprise informatics, poses a big task for big scale records clustering. To deal with this issue, researchers have provided you with parallel clustering algorithms that are primarily based on parallel programming fashions. MapReduce is one of the most typically used frameworks used for this motive and it has received high consciousness thanks to its flexibility, fault tolerance and programming ease. However, the overall performance has trouble for iterative packages. This paper gives an in depth evaluation of iterative frameworks which could help MapReduce for overcoming boundaries for iterative algorithms.

Keywords: clustering, framework and Map reduces.

I. INTRODUCTION

In the big data era, huge amounts of data have to be dealt carefully. In the year 2002, Yahoo! Web graph has reported that about a billion nodes and 7 billion edges have been reached.[1] Social networking sites such as Twitter, Facebook etc., use several terabytes of data. Websites such as Wikipedia and YouTube produce about few hundreds of gigabytes per minute[2]. In order to handle such enormous data, clustering is being used. It allows analyzing and discovering distribution patterns for large spectrum of data. Data scientists have studied and proposed various algorithms in data mining for handling large scale datasets. But these data mining algorithms are unable to deal with the exponentially increasing data. Hence, the research community has shifted its focus from these techniques to parallel clustering algorithms that would be a better alternative to traditional cluttering methods operated on a single system[3]. Thus, parallel processing techniques that can easily handle and process huge data has become the prime focus of many. Of the many parallel processing techniques available, MapReduce is the simplest and popular choice of many huge companies, example Facebook4. In spite of its popularity, MapReduce has certain critical issues and challenges such as bandwidth wastage, I/O and CPU cycles for iterative algorithms like PageRank[5,6], social network analysis, clustering and neural network analysis[7].

II. BIG DATA MANAGEMENT

The term 'Big Data' is generally used to refer to large amount of data that companies and people use. The public and private data are obtained from sources that are unorganized and so it is difficult to cluster data. In other words, it is difficult for most of the corporations to store, process and analyze big data. Big data has many definitions and ways of interpretation. However, the widely accepted definition is characterized by the five Vs, that is, Volume, Variety, Velocity, Value and Veracity[8,9].

Volume–this dimension is associated with the enormous data handling capability or ability of a clustering algorithm. Using parallel algorithms, one can divide big data into smaller units such that all units are handled simultaneously by machines at the rate of one machine per one unit. Thus one can solve the problem of the exponentially growing big data needs.

Variety – this dimension is associated with the ability of a clustering algorithm to handle a variety of data.

Velocity–this dimension is associated with the speed of a clustering algorithm handling big data. Most of the iterative clustering algorithms have iterative computations and this would probably utilize more time to process. Use of parallel algorithms in this process could help increase speed.

Value – this dimension is associated with the process of determining hidden values in the data and document those values.

Veracity – this dimension is associated with reliability, origin and the accuracy of the data.

2.1. MapReduce Overview

Many programmers distribute the data among several machines to speed up the processing of massive amounts of data and compute them as the volume of data has been multiplying. Data processing models such as Samza[10], Tez[11], and MapReduce[12] divide the data between machines and enables each processor to work separately with the similar algorithm on various parts of the data. There are two main classifications of parallel processing applications, traditional parallel applications[13,14] and data-intensive applications[15,17]. The assumption of data being fit into the memory of the distributed machines is predominantly seen in the traditional parallel applications. Due to the rapid increase in the data collection size, traditional parallel computation mechanisms fail to solve the big data issue. Whereas, data-intensive applications require additional computational and data resources that are run inside a conditional loop until terminated. The selection of moving the data to a computer or computer to a data is totally relied on the requirements and load. A major advantage in data-intensive applications is that data locality has significantly become more important than ever before as it reduces the network cost.

Among the several parallel processing models that compute data-intensive applications, like the Open MP, MPI[18], MapReduce is the most widely used framework as it very simple (has only two functions, a map and a reduce) allowing the programmers to monitor data distribution, replications, load balancing etc., easily[19] . There are two phases in MapReduce where the data points are processed over a set of commodity machines in a distributed environment.

The first phase is the Mapping phase that is defined by the Map function where the input data is divided into Map functions, and after computation the intermediate results are produced in the form of key/value pairs. The second phase is the Reduce phase which is defined by the Reduce function taking a single key and processing the given function on its associated values at a time, collects and is considered as output of the job. The Reducer task is performed after the intermediate results are generated and the data is shuffled into the corresponding Reducer. Majority of the job execution time is taken to move the intermediate results over the network. The figure 2 depicts the process involved in data processing.

Map function:

Data points are read as inputs by the Map task which then produces the intermediate key/value pairs that primarily consists of a key, a group number of values, and a value, associated with the key. A Map task then enables a process to generate new key/value that are sorted by their keys locally.

Combiner function:

This is an optional function that when applied to the intermediate results that are considered to have significant repetition in their respective Map task, will perform a partial reduction and passes on the intermediate key/value pairs to the reducer.

***Partition function**:*

It is mainly used for partitioning the intermediate keys that are hashed to determine which reduce partition will process the keys. To provide good balancing, Hadoop uses has partitioner by default.

Reduce function:

The output of Map tasks is considered collectively as the input for the reduce phase. With a user defined reduce function that is primarily called by each reducer for each key only once, the reduce task for each key is applied and its corresponding values are then processed. The processing of inputs to each reducer is done in the increasing key order.

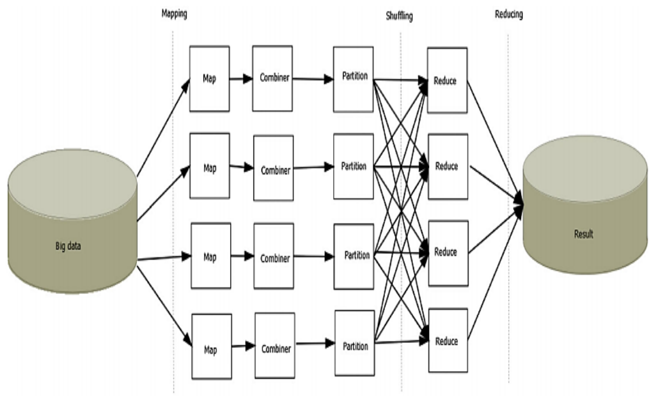

Figure 1: Architecture of MapReduce

2.2. Hadoop Overview

Analyzing and querying massive amounts of data is difficult through traditional mechanisms. Data scientist can use Hadoop [5], an open source software, to distribute huge scale data sets into several blocks and take advantage of the distributed environment. Rather than scaling up, that implies to add more resources such as RAM or CPU which is expensive, Hadoop is designed to scale out by addition of more machines in a cluster in distributed network with commodity hardware. Rather than a primary storage mechanism, the Hadoop Distributed File System (HDFS) is designed as a workflow. Basically, the data is dumped into the HDFS once and can be accessed any number of times, bringing the code to data, that are split into blocks of typically 64MB and can be accessed by any machine of that cluster for parallel processing. Failures can be handled by Hadoop by data replication where data is stored and replicated over machines making sure that at least one replica is stored in a different rack to ensure further data availability and reliability. A set of commodity hardware connected to one network in one location form the Hadoop cluster. For parallel processing purposes, Hadoop consist of HDFS and MapReduce engine whereas the cluster consists of several daemons/servers.

A Name-node stores the meta data that is mapped from file to block and location of block. A Data-node in HDFS stores data, provides the exact physical location of data and connects to Name-node. There is a Secondary Name-node that is used to monitor the state of the cluster in case the Name-node fails. An integration between HDFS and MapReduce will occur where the performance is through the master and slave nodes. Figure 3 explains the master node (consisting of the Task Tracker and Job Tracker) from the MapReduce layer and slave nodes that only have the Task Tracker. Scheduling of all MapReduce jobs is done by the Job Tracker which then assigns these tasks to the Task Trackers that are used for execution in each slave node.

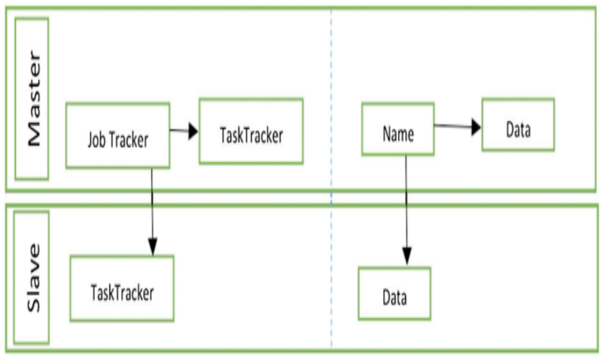

Figure 2: Hadoop Distributed File System and MapReduce Layer in Hadoop

Depending upon the availability of resources, Job Tracker assigns several tasks to Task Tracker. As and when the Task Tracker completes the tasks new tasks are assigned by the Job Tracker. Task Tracker sends a heartbeat signal periodically to the Job Tracker in case of machine failure. The master node will check for the heartbeat signal from Task Tracker and if it doesn't receive any such signal for some time from a machine then it will consider it to be a failed machine. Rather than fixing the failed machine, Hadoop assigns the failed task to another machine in the cluster as a backup. Since the probability of failure of one particular machine is very low, the failure of Job Tracker itself is not considered very seriously as it cannot be solved automatically [7].

2.3. Hadoop/MapReduce Limitations for Iterative Applications

There has been a lot of research carried out in terms of performance limitations on MapReduce[19,20]. The generic weaknesses and limitations discussed in[7] of MapReduce is shown in figure 4. The focus is mainly on iterative programs and the limitations related to MapReduce for the programs discussed previously have been identified and are discussed below:

Lack of loop aware task scheduling:

Multi-staging of tasks in a single run is not supported in MapReduce. Therefore, operations that are repeated during iterations like task initialization and cleanup (reload and dump data) usually start new MapReduce jobs. So, when dealing with large scale data sets scheduling overhead is included.

Lack of handling static data:

There are primarily two kinds of data involved in a MapReduce process, static data and state data. Static data is unchanged during iterations and hence, a key problem in MapReduce. In some of the iterative algorithms like the k-means, by design, the Reducer needs the static data for performing operations such as averaging etc., after state data is generated. Therefore, there needs to be reloading and shuffling of the static data during iterations that is practically difficult when large data sets are involved and resulting in overhead of communication.

Lack of Asynchronous execution:

Before completing a reduce task in the previous iteration a new map task cannot be allowed in standard MapReduce making each iteration wait for the previous iteration to complete resulting in synchronization overhead.

Lack of built-in termination condition:

An additional MapReduce job is needed at each iteration for termination as there is no provision of a built-in feature for termination leading to overhead because of scheduling extra costs.

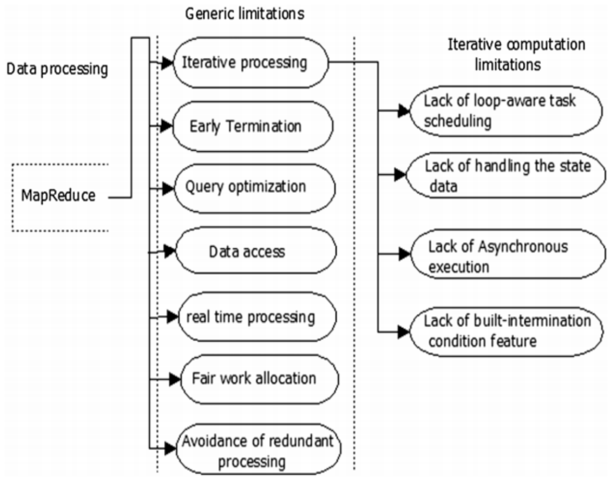
Generic limitations
Data processing
MapReduce
Iterative processing
Early Termination
Query optimization
Data access
real time processing
Fair work allocation
Avoidance of redundant processing
Iterative computation limitations
Lack of loop-aware task scheduling
Lack of handling the state data
Lack of Asynchronous execution
Lack of built-intermination condition feature

Figure 3: Limitations of MapReduce

III. REVIEW OF THE AVAILABLE WORKS

A review of the frameworks that enhanced the performance of MapReduce based on iterative processing techniques is presented in this section first. This is followed by a review of parallel clustering algorithms on MapReduce.

3.1. Review of iterative MapReduce Frameworks

Consider the following summarized figure of the current popular iterative frameworks. The following sub sections these significant factors are discussed.

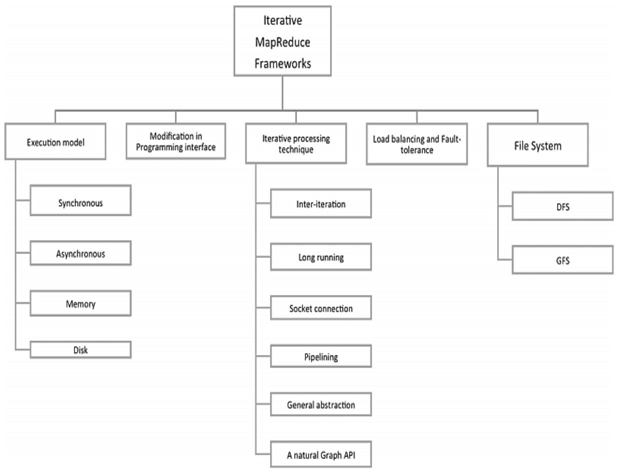

Figure 4: Iterative MapReduce Frameworks

3.1.1. Execution Model

Iterative MapReduce programs make use of a chain of map/reduce process and hence is slow. It splits the task into multiple tasks and these are not processed in a single run and hence this slows down the entire processing. For this purpose, many iterative frameworks have been proposed.

HaLoop framework is similar to MapReduce and the tasks are assigned to slave nodes using a parallel master node with TaskTracker daemon. This communicates with the master node. Each task can either Map or Reduce tasks. There are major changes in the HaLoop framework. Primarily, there is a new feature in the master node called the loop control. It initializes new map-reduce tasks recurrently until the end terminate condition is encountered. It eliminates the extra MapReduce jobs used to check stop condition and hence the overall communication overhead. The next change is that the algorithm has a new task scheduler specifically for iterative applications that supports data locality to co-locate the tasks for same operations. It enables reuse of data within the iterative process. Thirdly, the performance is improved by caching the intermediate results obtained to the disk. The issue, however, is that it uses synchronous execution like MapReduce. Hence there would be delay as the reducer has to wait till the map tasks are finished.

There is yet another framework that is similar to the execution model in MapReduce called Twister. For the purpose of communication, it utilizes publish/subscribe messaging mechanism. However, the architecture of Twister is different to MapReduce architecture. The input data in Twister doesn't have a distributed system. It is stored in a local disk as native files. The issue relating to this is that the collective memory of worker nodes has to accommodate this static data and this would require them to handle huge amounts of data sets. The performance of Twister is enhanced because it stored the intermediate results. Also, a driver is used for computation and a broker network is established. The main driver program sends the input data to the Mapper and Reducer. The results are collected and sent to the main program[21].

Another model, Pregel[22] is an in-memory system which utilizes the bulk synchronous parallel model for the purpose of computations [23]. The challenges in this model is that issues because of synchronous scheduling. MapReduce online[24] also has synchronous execution method. The intermediate results are pipelined between tasks and thus the data is transferred between Reduce and Map tasks and hence a connection is created between them and the output from Reduce task is sent directly to Map tasks. Since the framework doesn't support overlapping of tasks, the Map tasks are left pending until the Reduce tasks are finished.

There have been many frameworks that have been proposed for asynchronous execution but this cannot be used in MapReduce. One such is Spark[25] which is an in-memory framework and the computation model is different. It is similar to DryadLINQ[26] and FlumeJava[27] as every data set is considered as objects which can be converted using transformations such as Map and Filter to Resilient Distributed Dataset (RDD). There are options called as actions and these collect and return the results to the driver. Spark utilizes an advanced directed acyclic graph that can reduce the number of excess stages in MapReduce jobs[28] and thus faster execution of jobs. Spark has the capacity to run many complex tasks in a single run in multi stages and splitting of jobs into multiple ones is not done.

According to[29], Continuous bulk processing (CBP) can maintain the iteration state in memory.it has a cyclic data flow structure which speeds up iterative tasks unlike in MapReduce. The data abstraction in CBP can be used to control iterative dataflow. PrIteris another typical framework that makes use of priority execution and every data point is assigned a priority value. The subset of data that has the highest priority is chosen and executed in the iteration [30]. In iMap Reduce algorithm, the authors proposed a persistent socket connection that connects a Mapper to its corresponding Reducer. Thus asynchronous method of execution is used for data that is static and state data when the state results arrive[31].

3.1.2. Iterative Processing Technique

In the case of iterative processing techniques, HaLoop has a better performance when compared to the standard MapReduce method as it uses the technique of co-locating tasks for a particular data point over various iterations. The same machine deals with Map and Reduce tasks using the same data with the concept of 'inter-iteration locality'. This approach helps in creating a cache for data and reuse of that data during the iterations, thus reducing the I/O costs involved. The technique used in Twister is the 'long-running map/reduce tasks' technique which doesn't change the state data and reloading data for different iterations is not needed. This method improves performance overhead for iterative applications. iMapReduce makes use of persistent mechanism for iterations and the Map or Reduce tasks are kept active during the iteration process. This framework could reduce the necessity of creating new repeated MapReduce jobs for the same tasks. Also, the socket connection in this framework doesn't require a new MapReduce job to combine static and state data. Similar to iMapReduce technique, PrIter also provides a constant connection between Map and Reduce tasks. The data is divided into initial static and state and this is preloaded before the map tasks begin in the iterative processing. The static data is not altered and the state data is updated and merged with the static data for iterations. When the iterative end condition is met, the computation is terminated[7].

In MapReduce online, a pipeline architecture is used to deal with the iterative process which links Map and Reduce tasks. Map tasks are paused until the previous iteration Reduce tasks are completed. However, the introduction of online aggregation[31] can deal with the issues in computation amid the MapReduce jobs.

Spark uses data abstraction to cache static and state data in the device memory for reuse. Each data set is represented as an object and the users have the option to choose the data set that has to be cached for further iterations. CBP also uses data abstraction method for iterative processing. Persistent states are maintained for the reuse of previous jobs for incremental processing and group wise processing and thus the data movement within the system is reduced[7].

3.1.3. Load balancing and fault tolerance.

Fault tolerance in HaLoop is similar to that of Hadoop. It makes use of intra-iteration technique and then the cache is reconstructed and sent to the assigned worker with failed partition. In Twister, the mechanism for fault tolerance is similar to that of the implementations in Google and Hadoop wherein the application state is saved through the iterations. If there is a failure encountered, the complete iteration can be rolled back. iMapReduce has the fault tolerance similar to the standard MapReduce. But the results from the previous iterations are retained by storing the state data in a distributed file system. In case of a failure, the last iteration would be recomputed using the stored state data. The tasks in iMapReduce are persistent and they are in contrast to the task scheduling mechanism used inMapReduce and hence it cannot utilize the mechanism used for balancing load in MapReduce. Comparison of the completion time of the received notification of tasks completed after every iteration is done by the master node and this is used to differentiate the leaders and stragglers. In the MapReduce online, a bookkeeping mechanism has been added to the current Hadoop implementation. Also, the map task output is kept tentative till the reducer receives a notification about the successful execution of the tasks from Job Tracker. A new copy of the tasks is started when a failure is encountered. The output of the reduce tasks can be reproduced in case of failure using the stored map task outputs.

For load balancing, PrIter makes use of 'MRPair migration' mechanism. In this the average processing time that is received from map/reduce pair threshold is compared by the master node and if the processing time is less than the threshold, the assigned worker is considered slow and the pairs are assigned to a faster worker. The State Table and graph partition are loaded from the DFS. Thus PrIter is resistant from failuresin task completion. If there is a worker failure, a healthy worker is assigned the failure task and the other MRPairs roll back to the same fix point and the last failed task is re-computed. Spark relies on RDDs for fault tolerance. When failure occurs, roll back of the entire program is not required. The lost partitions or the RDDs are reloaded for fault recovery. Fault tolerance in Pregelis done with the help of check points which is set using ping message. When the master node doesn't receive a ping message from the workers, then that worker is considered as a failure and the graph partition is assigned to healthy workers. These healthy workers use the last check point to reproduce partition state. But for failure recovery, all the vertices in the iteration have to be re-executed [7].

3.1.4. File system.

HDFS is supported in Haloop, PrIter, MapReduce online, Spark, and CBP; whereas Pregel supports Google File System (GFS). Twister does not have this feature and large data sets have to be split by the user only.

3.2. Review of Parallel Clustering Algorithms Based on MapReduce

Similar to the iterative algorithms discussed in the previous section, this section deals with the categorized summary and review of various parallel clustering algorithms that are based on MapReduce. The categories are - centroid-based clustering; density-based clustering; connectivity-based clustering; high-dimensional clustering; similarity-based clustering and co-clustering. This is summarized in the following figure.

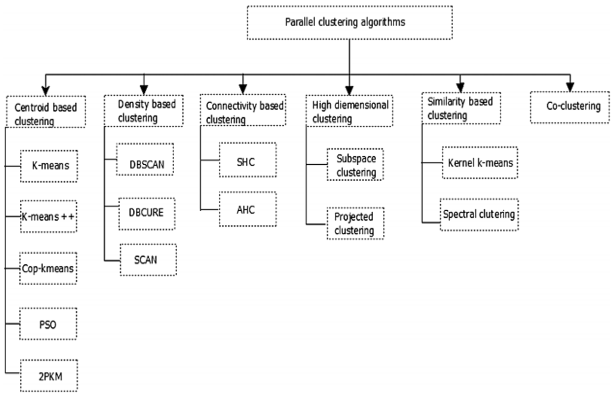

Figure 5: Parallel Clustering Algorithms

IV. ANALYSIS

A detailed analysis of parallel clustering algorithms has been provided that is based on several vital parameters. One should be aware of the fact that comparing these algorithms is not an easy task given the wide variety of mechanisms used. Main objective is to find the aspects of what has been used in the algorithm that helps identify the disadvantages of several existing parallel algorithms and those which can be used for future research.

4.1. Algorithm class

As explained in[44], the current clustering algorithms will be classified based on the total number of MR jobs needed to perform the necessary MapReduce model. This classification helps in selecting the suitable framework for executing the algorithms. For illustration purposes, ParC is the only algorithm that can be classified into the first class among all the compared algorithms. This class is known as embarrassing algorithms as the entire algorithm can be performed in a single MR round. Second class algorithms like SnI and CLARA perform MapReduce model in a constant number of MR rounds. As there are very few MR jobs needed to implement the first and second class algorithms the cost of running them is negligible. The other compared algorithms need synchronization steps between iterations like checking the termination conditions and hence, classified into third class. The work carried out in[33] belongs to the third class as it requires multiple iterations until the termination condition is met which is not suitable for any MapReduce style. Using the Twister approach to support 'PAM like' in the work done in [44], reduces the whole of an algorithm providing a long running strategy to keep tasks alive until termination is met in a single MR round.

4.2. Data shuffling

By reducing the shuffling of data over iterations can achieve improvement in performance. Techniques like directly passing the data to Mapper from Reducer or joining the state data with static data will result in improved performance whilst removing extra MR jobs also there would not be any necessity to write the output to the DFS. Data shuffling cost in the work done in[44] that uses CLARA is medium as only two MR jobs are need to be executed. The k-means based in MapReduce in[39] will generate and send intermediate results to the Reducer and the static data is gathered for averaging the operations. Due to this operation, the static data will be shuffled between the Mapper and Reducer at each iteration increasing the network communication cost. In[66], iMapReduce is used which will let the reduce tasks broadcast to all Map tasks the updated state data that will minimize the communication cost. The shuffling in case of Single-linkage Hierarchical Clustering (SHC) algorithm is dramatically reduced as the Prim-MR job is performed that cuts the edges in an early stage. Though the MSTs are shuffled at the Reducers once stored on DFS at Map side. Processing of Kruskal Reducer needs to wait for input shuffling. To minimize the number of iterations along with reducing the shuffling cost, the work in[52] has used batch updating.

Grid file has been used in the case of DBSCAN by[67] that performs in a single MR round minimizing the shuffling cost at the partitioning stage. The main reason why this happens is the data from within the space Si and its halo replication from bordering spaces are shuffled to the target Reducer easily. Spatial indices such as breath-first-search are used in the next stage of running local DBSCAN that is considered an iterative algorithm increasing the shuffling cost for the computations of the neighbors in almost each node, which is very expensive considering big data. Structural similarity for each edge of the graph cutting off the edges within the threshold is used to reduce the cost of shuffling in[50].

To exert their advantages on big data and also aiming to reduce the data shuffling, algorithms for subspace clustering such SnI and ParC are used. Because of the sampling method in the first phase and filtering in the second phase, SnI provides better performance. The work in [57] is termed as an exact algorithm as it does not use sampling approach. It also increases the shuffling cost as there are extra MR jobs to handle. The approach in [68] where the synchronization of the only updated cluster assignments and cluster information happens to reduce shuffling cost along with avoiding the shuffling of row and columns between iterations. iMapReduce in [66] implements co-clustering that supports the iterative processes of co-clustering algorithms by joining the state and static data.

4.3. Mechanism, problem, and the required modification

Simple MR jobs have been applied to achieve a parallel clustering algorithm in the works of[33-35,44,39,40]. Good results in terms of clustering time have been produced by using these approaches. Re-reading the input data at each iteration and capturing of large data sets due to job lag have been shown in these results. If following modifications are done, then these approaches would work better:

a. To help keep the static data in memory, caching of Mapper input can be done, so reloading will not happen at each iteration
b. To check the termination condition, caching of Reducer input can be done, along with terminating extra MapReduce jobs
c. A joining mechanism can be used to build a connection between the Reduce task and the Map Task enabling algorithms such as k-means to use the static for averaging operations as Reducer needs it

Some studies that use iterative graph algorithms such as breath-first-search[69,50] are not correct for MapReduce style. This will cause the entire graph structure to reload and shuffle from Mapper to the Reducer at each iteration which needs to remain unchanged during iterations. There are alternatives for iterative graph algorithms such as the Bulk synchronous parallel model [29], that will let graph algorithms to run on large vertices and edges.

Kruskal's algorithm in[53] is used to reduce the single-linkage problem in many of the hierarchical clustering algorithms. Re-reading the MSTs would be considered the main issue in this approach as the cost of doing this operation for large data sets is huge. To avoid materialization of the MSTs on DFS, an alternative approach can be used known as the lazy operations. In addition to that, to help the KruskalReducer to process without holding on for input shuffling, a location aware schedule can be proposed.

Hybrid mechanism is proposed in the approach described in [63] that uses MapReduce and MPI. Iterative part of the spectral algorithm can be now handled by the MPI, that seems to work better than MapReduce in terms of iterative applications. Based on Twister, the work in[42,44] implemented iterative algorithms like k-means, PAM and IB. Overhead within the iterations is now reduced as expected by the proposed long-running mechanism.

Implementation of iterative MapReduce (HaLoop) in the work of [5] takes advantage of caching mechanisms. Surprisingly, the main purpose of the proposed approach is to cache the Mapper input and Reducer output in order to reduce MR rounds but, [5] claims that HaLoop works better when compared to Twister in terms of fault tolerance due to the long-running MapReduce tasks. All these works lack the ability for executing asynchronously making the Map tasks not to start executing as they need to wait for other Map tasks to complete. To support asynchronous execution, iMapReduce and Spark have been used in[70,66], that minimize the clustering time for k-means and co-clustering algorithms.

V. CONCLUSIONS AND OUTLOOK

When dealing with large scale data sets, efficient parallel clustering algorithms and frameworks ensure scalability and good performance are achieved. A new data processing approach is introduced by MapReduce in this era of big data. Because of its ease of programming, flexibility and fault tolerance, MapReduce has garnered considerable attention in the world of big data. One has note that there are still certain limitations when pertaining to MapReduce like dealing with iterative applications. An analysis has been made in this study on several other parallel clustering algorithms that have attracted significant attention in the big data era. Most of these algorithms can perform better than various traditional sequential algorithms on scale up and speed up metrics. It is evident from the detailed discussion of several parallel algorithms that the field of parallel big data clustering is very young and is open for further research. The rate of improvement in parallel processing models and the amount of data growth is indirectly proportional as the volumes of data is increasing but the improvement in the parallel processing models is still very slow. An observation has been made that MapReduce has limitations in the case of iterative applications. Also, consideration of several state-of-the-art techniques like the Spark, Twister, HaLoop and MapReduce that are vital big data parallel processing models too cannot figure out the issues with iterative algorithms. In summary, the principle findings of this study are:

a. A combination of all the merits of the existing parallel processing models is needed to design a new parallel processing model from scratch that is more fault tolerant whilst supporting iterative algorithms
b. Because MapReduce/Hadoop implementation does not fully support iterative techniques, it is not the end of MapReduce. One should understand the maturity of MapReduce over iterative algorithms and also many vendors support it. Therefore, a good alternative to handle iterative big data algorithms could be a hybrid system
c. Improvement of the clustering time of large data sets can be achieved by parallel data processing but it also degrades the performance and quality. Hence, the main worry is to achieve a reasonable trade-off between quality and speed in the case of big data
d. A study on the unavailability of clustering algorithms that can be processed in parallel has been carried out. Major challenge is to figure out a method that will eliminate the inherent dependence of algorithms
e. Many studies have implemented MapReduce directing the research to be applied more towards the parallel clustering algorithms using other iterative frameworks such as Pregel, CBP and PrIter.

REFERENCES

[1]. Kang U, Tong H, Sun J, Lin C-Y, Faloutsos C. GBASE, in Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining – KDD’11, San Diego,CA,USA,2011 Aug, pp. 1091.

[2]. “YouTube statistic.” [Online]. Available:http://www.youtube.com/yt/press/statistics.html. [last accessed June 22, 2014],Dare Accessed:22/6/2014.

[3]. Kim W. Parallel clustering algorithms: survey, CSC 8530 parallel algorithms, 2009, pp.1-32.

[4]. Aiyer A, Bautin M, Chen GJ, Damania P, Khemani P, Muthukkaruppan K, Vaidya M. Storage infrastructure behind facebook messages using HBase at scale. IEEE Data Engineering 2012, 35(2), pp.4–13.

[5]. Bu Y, Howe B, Balazinska M, Ernst MD. HaLoop: efficient iterative data processing on large clusters, in 36thInternational Conference on Very Large Data Bases, 2010 Sep, 3(1-2),pp.285-96.

[6]. Page L, Sergey Brin RM, Winograd T. The PageRank citation ranking: bringing order to the web, 1998 Jan, pp.1-17..

[7]. Mohebi A, Aghabozorgi S, Ying Wah T, Herawan T, Yahyapour R. Iterative big data clustering algorithms: a review, Software Practicle Experience, 2016,46,pp. 107–29.

[8]. Zikopoulos P, Parasuraman K, Deutsch T, Giles DC. Harness the Power of Big Data the IBM Big Data Platform [Kindle Edition], 1st edn. McGraw-Hill Osborne Media: New York, 2012 Sep.

[9]. Assunção MD, Calheiros RN, Bianchi S, Netto MAS, Buyya R. Big Data computing and clouds: Trends and future directions, Journal of Parallel and Distributed Computing 2014,75(13),pp.156–75.

[10]. Riccomini C. Samza: Real-time stream processing at LinkedIn, 2013. Retrieved July 5, 2014, from http://www. infoq.com/presentations/samza-linkedin, Date ACCESSED: 5/7/2014.

[11]. Murthy A. Tez: accelerating processing of data stored in HDFS, 2013. [Online]. Available: http://hortonworks.com/blog/introducing-tez-faster-hadoop-processing/,Date Accessed: 20/2/2013.

[12]. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters, Communications of the ACM 2008, 51(1),pp.1–13.

[13]. Dhillon IS, Modha DS. A data-clustering algorithm on distributed memory multiprocessors, in Proceeding Revised Papers from Large-Scale Parallel Data Mining, Workshop on Large-Scale Parallel KDD Systems, SIGKDD, 1999,pp.245–60.

[14]. Forman G, Zhang B. Distributed data clustering can be efficient and exact, ACM SIGKDD Explorations Newsletter 2000, 2(2),pp.34–38.

[15]. Kang U, Papalexakis E, Harpale A, Faloutsos C. GigaTensor, in Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining – KDD’12, 2012,pp.1-316.

[16]. Kang U, Tsourakakis CE, Faloutsos C. PEGASUS: mining peta-scale graphs, Knowledge and Information Systems 2010,27(2),pp.303–25.

[17]. White T. Hadoop—The Definitive Guide: Storage and Analysis at Internet Scale, 3rd edn. O’Reilly R (ed.). Ireland, 2012 Jan, pp.1-647.

[18]. Snir M, Otto S, Huss-Lederman S, Walker D, Dongarra J. MPI–the Complete Reference, Second edn, the MPI Core, MIT Press: Cambridge, MA, USA, 1998 Sep,pp.1-350.

[19]. Stonebraker M, Abadi D, DeWitt DJ, Madden S, Paulson E, Pavlo A, Rasin A. MapReduce and parallel DBMSs,Communications of the ACM 2010, 53(1),pp.1-64.

[20]. Pavlo A, Paulson E, Rasin A, Abadi DJ, DeWitt DJ, Madden S, Stonebraker M. A comparison of approaches to largescale data analysis, In Proceedings of theACMSIGMODInternational Conference on Management of Data (SIGMOD), 2009, pp.165–78.

[21]. Ekanayake J, Li H, Zhang B, Gunarathne T, Bae S-H, Qiu J, Fox G. Twister, in Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC’10,Chicago, 2010 Jun, pp. 810-18.

[22]. Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G. Pregel, in Proceedings of the 2010 international conference on Management of data - SIGMOD’10, 2010,pp.1- 135.

[23]. Valiant LG. A bridging model for parallel computation. Communications of the ACM 1990, 33(8),pp.103–11.

[24]. Condie T, Conway N, Alvaro P, Hellerstein JM, Elmeleegy K, Sears R. MapReduce online, in NSDI’10 Proceedings of the 7th USENIX conference on Networked systems design and implementation Berkley,,2010,21,pp.1-15.

[25]. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: cluster computing with working sets, in HotCloud’10 Proceedings of the 2nd USENIX conference on Hot topics in cloud computing, 2010, pp.1-10.

[26]. Yu Y, Isard M, Fetterly D, Budiu M, Erlingsson Ú, Gunda PK, Currey J. DryadLINQ: *a system for general-purpose*

distributed data-parallel computing using a high-level language, in OSDI'08 Proceedings of the 8th USENIX conference on Operating systems design and implementation, Berkeley,USA,CA,2008, pp. 1–14.

[27]. Chambers C, Raniwala A, Perry F, Adams S, Henry RR, Bradshaw R, Weizenbaum N. FlumeJava, ACM SIGPLAN Notices 2010, 45(6),pp.1-363.

[28]. Thulasiraman K, Swamy MNS. Graphs: Theory and Algorithms. John Wiley & Sons, Inc.: New York, 1992.

[29]. Logothetis D, Olston C, Reed B, Webb KC, Yocum K. Stateful bulk processing for incremental analytics, in Proceedings of the 1st ACM symposium on Cloud computing - SoCC'10, 2010 Jun, pp.1-12 ..

[30]. Zhang Y, Gao Q, Gao L, Wang C. PrIter: a distributed framework for prioritizing iterative computations. IEEE Transactions on Parallel and Distributed Systems 2013, 24(9),pp.1884–93.

[31]. Hellerstein JM, Haas PJ, Wang HJ. Online aggregation, Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data – SIGMOD'97, 1997, 26(2),pp.171–82.

[32]. Wu X, Kumar V, Ross Quinlan J, Ghosh J, Yang Q, Motoda H, McLachlan GJ, Ng A, Liu B, Yu PS, Zhou Z-H, Steinbach M, Hand DJ, Steinberg D, Top 10 algorithms in data mining, Knowledge and Information Systems 2007, 14(1),pp.1–37.

[33]. Zhao W, He Q, Ma H. Parallel K-Means Clustering Based on, 2009, pp.674–79.

[34]. Zhou P, Ye W, Lei J. Large-scale data sets clustering based on MapReduce and Hadoop, The Journal of ComputerInformation Systems 2011, 16(7),pp.5956–63.

[35]. Nguyen CD, Nguyen DT, Pham V. LNCS 7975 - Parallel two-phase K-means, 2013,7975, pp.224–31.

[36]. Pham DT, Dimov SS, Nguyen CD. An incremental K-means algorithm, in Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science, 2004, pp.783–95.

[37]. Bahmani B, Moseley B, Vattani A, Kumar R, Vassilvitskii S. Scalable k-means++. Proceedings of the VLDB Endowment 2012, 5(7),pp.622–33.

[38]. Arthur D, Vassilvitskii S. k-means++: the advantages of careful seeding, in SODA'07 Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, 2007,pp. 1027–35.

[39]. Li C, Zhang Y, Jiao M, Yu G. Mux-Kmeans : Multiplex Kmeans for clustering large-scale data set categories and subject descriptors, in Proceedings of the 5th ACM workshop on Scientific cloud computing - ScienceCloud'14, 2014, pp. 25–32.

[40]. Aljarah I, Ludwig SA. *Parallel particle swarm optimization clustering algorithm based on MapReduce methodology*, in 2012 Fourth World Congress on Nature and Biologically Inspired Computing (NaBIC), Mexico City,2012 Nov,pp. 104–11.

[41]. Kennedy J, Eberhart R. *Particle swarm optimization*. Proceedings of ICNN'95 - International Conference on Neural Networks, 1995,4,pp.1942–48.

[42]. Sun Z, Fox G, Gu W, Li Z. *A parallel clustering method combined information bottleneck theory and centroid-based clustering*, Journal of Supercomputing 2014, 69(1),pp.452–67.

[43]. Tishby N, Pereira FC, Bialek W. *The information bottleneck method*, Apr. 2000,pp.1-16.

[44]. Satish Narayana Srirama PJ, Vainikko E. *Adapting scientific computing problems to clouds using MapReduce*, Future Generation Computer Systems 2012, 8(1),pp.184–92.

[45]. Kaufman L, Rousseeuw P. *Finding groups in data: an introduction to cluster analysis*, in Wiley Interscience, 1990, pp.1-5.

[46]. Martin Ester XX, Kriegel H-P, Jörg S. *A density-based algorithm for discovering clusters in large spatial databases with noise*, in 2nd International Conference on Knowledge Discovery and Data Mining, Portland, 1996,pp. 226–31.

[47]. Li L, Xi Y. *Research on clustering algorithm and its parallelization strategy,* International Conference on Computational and Information Sciences Chengudu,China, 2011, pp.325–28.

[48]. Kim Y, Shim K, Kim M-S, Sup Lee J. *DBCURE-MR: an efficient density-based clustering algorithm for large data using MapReduce.*,Information Systems 2014, 42,pp.15–35.

[49]. Ankerst M, Breunig MM, Kriegel H-P, Sander Journal of OPTICS, ACM SIGMOD Record, 1999, 28(2),pp.49–60.

[50]. Zhao W, Martha V, Xu X. PSCAN: *A Parallel Structural Clustering Algorithm for Big Networks in MapReduce*, in 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA), Barcelona,2013, pp.862–69.

[51]. NandiniRaghavan U, Albert R, Kumara S. *Near linear time algorithm to detect community structures in large-scale networks*, Physical Review 2007 Sep, 76(3),pp.1.

[52]. Sun T, Shu C, Li F, Yu H, Ma L, Fang Y. *An efficient hierarchical clustering method for large datasets with MapReduce*, in 2009 International Conference on Parallel and Distributed Computing, Applications and Technologies,Higashi Hiroshima, 2009 Dec,pp. 494–99.

[53]. Jin WLC, Patwary MMA, Agrawal A, Hendrix W. *DiSC: A distributed single-linkage hierarchical clustering algorithm using MapReduce*, in Proceedings of the 4thInternational SC Workshop on Data Intensive Computing in the Clouds, 2013,pp.1-10.

[54]. Cormen TH, Leiserson CE, Rivest RL, Stein C. *Introduction to Algorithms*, 2nd edn, McGraw-Hill Higher Education: New York, USA, 2001,pp.1-640.

[55]. Parsons L, Haque E, Liu H. *Subspace clustering for high dimensional data*, ACM SIGKDD Explorations Newsletter 2004, 6(1),pp. 90–105.

[56]. R. Murugesh, I. Meenatchi, "*A Study Using PI on: Sorting Structured Big Data In Distributed Environment Using Apache Hadoop MapReduce*", International Journal of Computer Sciences and Engineering, Vol.2, Issue.8, pp.35-38, 2014.

[57]. Fries ST, Wels S. *Projected clustering for huge data sets in MapReduce* | Chair of Computer Science 9, in International Conference on Extending Database Technology (EDBT 2014), Athens, Greece, 2014,pp. 49–60.

[58]. Moise G, Sander J, Ester M. *P3C: a robust projected clustering algorithm*, in Sixth International Conference on Data Mining (ICDM'06), Hong Kong,2006 Dec, pp.414–25.

[59]. Hyndman RJ. The problem with Sturges' rule for constructing histograms, no. 1995 Jul,pp. 1–2.

[60]. Elgohary A, Farahat AK, Kamel MS, Karray F. *Embed and conquer: scalable embeddings for kernel k-means on MapReduce*, in Appears in Proceedings of the SIAM International Conference on Data Mining (SDM), 2014, 2013,pp. 1–18.

[61]. Über die praktische Auflösung von linearen. "*Integralgleichungen mit Anwendungen auf Randwertaufgaben",* Acta Mathematica, *1930, 54(1),pp.185–204.*

[62]. Rahul R. Ghuleand Sachin N. Deshmukh, "*Comparative Study on Speculative Execution Strategy to Improve MapReduce Performance*", International Journal of

Computer Sciences and Engineering, Vol.3, Issue.3, pp.197-200, 2015.

[63]. Chen W-Y, Sunnyvale Y, Song H, Bai C-JL, Chang EY. *Parallel spectral clustering in distributed systems*, IEEE Transactions on Pattern Analysis and Machine Intelligence 2011, 33(3),pp.568–86.

[64]. Maschho K, Sorensen D. *A portable implementation of ARPACK for distributed memory parallel architectures*, In Proceeding of Copper Mountain Conference on Iterative Methods, 1996,pp.1-8.

[65]. Papadimitriou S, Sun J. *DisCo: distributed co-clustering with Map-Reduce: A Case Study Towards Petabyte-Scale End-to-End Mining*, in 2008 Eighth IEEE International Conference on Data Mining, Pisa,2008 Dec,pp.512–21.

[66]. Su S, Cheng X, Gao L, Yin J. Co-Cluster *D: a distributed framework for data co-clustering with sequential updates*, in International Conference on Data Mining (ICDM), 2013 IEEE 13th,Dallas TX, 2013, pp.1193–98.

[67]. M. Shankar Lingam, A. M. Sudhakara, "*A Brief Account of Iterative Big Data Clustering Algorithms*", International Journal of Computer Sciences and Engineering, Vol.5, Issue.10, pp.300-309, 2017.

[68]. Dhillon IS. *Co-clustering documents and words using bipartite spectral graph partitioning*, in Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD'01, 2001,pp. 269–74.

[69]. P. Dadheech, D. Goyal, S. Srivastava, "*Performance Improvement of Heterogeneous Hadoop Clusters Using MapReduce For Big Data*", International Journal of Computer Sciences and Engineering, Vol.5, Issue.8, pp.211-214, 2017.

[70]. Zaharia M, Chowdhury M, Das T, Dave A, Ma J, McCauley M, Franklin MJ, Shenker S, Stoica I. *Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing*, in Proceeding NSDI'12 Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation, 2012,2,pp.1-14.