Big Data Reduction Technique using Parallel Hierarchical Agglomerative Clustering

Veronica S. Moertini, *Member, IAENG,* Gde W. Suarjana, Liptia Venica and Gede Karya

***Abstract*—Volume and velocity are two characteristics of big data. Big data "comes in" with high velocity that the volume increases quickly. Efforts are needed to resolve these issues. This paper presents a big data reduction technique that can be used to reduce incoming big data periodically. The results, patterns that represent the original data with smaller size can be kept for further analysis, while the voluminous big data can be discarded. Clustering is a technique that can be used for reducing data. Based on our study, we find that agglomerative clustering is suitable to be adopted for reducing big data having low to medium number of attributes. Our proposed technique is based on Hadoop MapReduce, a computing framework for distributed systems, where Map and Reduce functions run in parallel in machine nodes. The excerpt of our technique: Map preprocesses and randomly divides the big data into disjoint partitions, Reduce constructs cluster trees (dendrograms) from partitions and computes patterns from the clusters formed from the trees. The output is a collection of patterns having a lot smaller number of objects and attributes. To provide flexibilities, we design few input parameters set by users. The effect of those parameters are shown by our experiment results. By experimenting using big data in a Hadoop cluster with up to 15 commodity computers, we conclude that the Hadoop file system block size and number of nodes affect the execution time and the size of incoming big data that can be processed.**

***Index Terms*— Big data reduction, cluster pattern, MapReduce, parallel clustering**

I. Introduction

Big data is collected from heterogeneous data sources - such as social media systems, industrial sensor networks, scientific experimental systems, and several other application areas – and may reach giga to peta bytes in size [1, 2]. The needs to analyze big data to obtain knowledge or useful insights have been well known [3, 4]. However, managing and gaining insights from the big data remains a challenge. Among several characteristic of big data (volume, variety, value, velocity, veracity, and variability), the volume is the primary concern. Efforts are required to reduce the volume to effectively analyze big data [2].

Data reduction techniques can be applied to obtain a reduced representation of the data set that is much smaller in volume, yet closely maintains the integrity of the original data [5]. That is, mining on the reduced data set should be more efficient yet produce the same (or almost the same) analytical results. Data reduction strategies include dimensionality reduction, numerosity reduction, and data compression. Numerosity reduction techniques replace the original data volume by alternative, smaller forms of data representation, which can be generated by nonparametric methods such as histograms, clustering, sampling, and data cube.

Hadoop with its MapReduce framework, which works in distributed systems, has been developed to address the need for big data analysis. Small-medium organizations can adopt Hadoop as Hadoop clusters can be configured using commodity computers with low specification. MapReduce supports the processing of large datasets and has the advantage of easy scaling of data processing over multiple computing nodes [1, 7]. A MapReduce program takes input data in the form of key-value pairs, the Mapper and Reducer algorithms then manipulate those key-value pairs, and produce some other form of key-value pairs.

Recently, Rehman et.al. [2] reported their survey of big data reduction methods. They classified the methods into: Network theory, compression, data deduplication (redundancy elimination), data preprocessing, dimension reduction, and Data Mining and Machine Learning (DM and ML). The DM and ML methods can either be applied to reduce big data immediately after its acquisition or to customize big data to address some specific problems. These methods have the potential to be enhanced for Hadoop Map-Reduce distributed systems to handle big data. However, there still exists a huge research gap between these potencies and the works done. There are still very limited research results of DM and ML methods (that include supervised, unsupervised, semi-supervised, and hierarchical deep learning models) for big data reduction. For instance, one supervised technique that has been developed recently is MRPR [6]. MRPR is based on MapReduce. It selects instances from the original data set, or build new artificial prototypes, to form a set of prototypes that better adjusts the decision boundaries between classes in NN classification.

In this research, we intend to contribute in developing a big data reduction technique that can be used regularly. It is based on unsupervised (clustering) approach in the distributed system that will produce a collection of patterns

Manuscript received May 7, 2017; revised January 18, 2018. This work was supported by the Directorate General of Research Strengthening and Development, Ministry of Research, Technology and Higher Education, Indonesia, under Grant SP DIPA-042.06.1.401516/2017 and Informatics Department - Parahyangan Catholic University.

Veronica S. Moertini is with the Informatics Department Parahyangan Catholic University, Bandung, Indonesia (phone: 62-22-2041964; e-mail: moertini@unpar.ac.id).

Gde W. Suarjana was with the Informatics Department Parahyangan Catholic University, Bandung, Indonesia (e-mail: gdewira91@gmail.com).

Liptia Venica was with the Informatics Department Parahyangan Catholic University, Bandung, Indonesia (e-mail: liptiavenica@gmail.com)

Gede Karya is with the the Informatics Department Parahyangan Catholic University, Bandung, Indonesia (e-mail: gkarya@unpar.ac.id).

with smaller size (compared to the raw big data). These patterns can then be stored (permanently) while the raw data can be discarded to save disk space. Based on the organizations' need, the patterns can then be analyzed by suitable data mining techniques (outlier detection, summarization, clustering, etc.). To take advantage of Hadoop, our proposed techniques is based on MapReduce.

We find that the non-parallel hierarchical clustering algorithm [5] has some advantages to be enhanced into parallel algorithm for the purpose of reducing big data. One of the important advantages is that it does not take definite number of clusters. This algorithm builds a cluster tree (known as dendrogram) based on a distance metric, then clusters can be formed from the dendrogram by using a variable (such as distance cut-off) defined by users. Cluster patterns (such as discussed in [4]) can then be computed from the clusters as the reduced representations of the big data without loosing its semantic meaning.

A distributed single-linkage hierarchical clustering algorithm (DiSC) using MapReduce has been developed [8]. The Mapper constructs subgraphs (sub-dendrograms), where each is constructed from two data splits. The data splits may be used by more that one Mapper, hence producing subgraphs that have overlapping edges (this will be resolved in next stage for obtaining a complete graph of dendrogram). For the purpose of data reduction, we need to construct subgraphs (local dendrograms) that are non-overlapping. Thus, we can not adopt this part of algorithm.

To be more specific, our proposed big data reductions method is based on parallel hierarchical clustering algorithm that handles numeric attributes. To the best of our knowledge, there is no similar technique yet developed by other researchers. The problems that are resolved include:

(a) How to partition the raw big data into non-overlapping datasets to ensure that the resulted patterns are less biased towards objects order in the raw data?

(b) By considering that the data partition may still be large, how to construct dendrograms from this partition in every node and ensure that it is "fitted" in its memory (which may be small in commodity computers)?

(c) As the results of reduced data, what patterns that semantically represent the original data and how to compute these?

(d) There are several distance types that can be adopted in constructing dendrogram. How these influence the patterns generated and the execution speed?

(e) How is the time response of the proposed technique in Hadoop distributed systems affected by its configuration set-up?

In the rest of the paper, we discuss the related literature, proposed technique (the main idea, proposed algorithm, Map-Reduce functions design), four series of experiments (patterns evaluation, variables influencing the execution speed, reduction percentage, the performance related to Hadoop cluster configurations) and conclusion. In the appendix, we include additional experiment results.

II. Literature Review

A. Research Opportunities

As stated in Section I, Rehman et.al. [2] have conducted and reported their survey of big data reduction methods. Few of data reduction techniques based on DM and ML that are evaluated in [2] are:

(1) A MapReduce algorithm used to reduce the search space and mine frequent patterns from uncertain big data. It facilitates the users to confine their search space by setting some succinct anti-monotone (SAM) constraints for data analysis and subsequently mines the uncertain big data to uncover frequent patterns that satisfy the user-specified SAM constraints.

(2) Artificial neural networks (ANNs) self-organized Kohonen network-based that is proposed to reduce big hydrographic data acquired from the deep seas.

(3) Deep learning, which is based on deep neural network architectures, as an option for big data reduction methods. However, the models become computationally inefficient with the increase in big data complexity.

Based on their survey results, [2] formulates research opportunities related to big data reduction methods. The ones that are related to our interest are:

(1) Data preprocessing approach: The investigations of research problems relevant to preprocessing techniques of big data are still at the initial level. The forefront data preprocessing methods in the big data knowledge discovery process requires new, efficient, robust, scalable, and optimized preprocessing techniques for both historical and streaming big data.

(2) DM and ML approach: The DM and ML methods for big data reduction can be used at various levels of big data architectures. These methods find interesting patterns from big data streams as highly relevant and reduced data for further analysis. The DM and ML methods also have the potential to be implemented in the Hadoop MapReduce. There still exists a huge research gap for the implementation of other DM and ML methods for big data reduction that include supervised, unsupervised, semi-supervised, and hierarchical deep learning models.

One of the DM technique for reducing big data that have been developed recently is MRPR [6], which is based on supervised approach. It selects instances from the original data set, or build new artificial prototypes, to form a set of prototypes that better adjusts the decision boundaries between classes in NN classification. The core of Map and Reduce functions are as follows: Map reads a training set *TR*, which is stored as distributed blocks in the HDFS as a single file (*TR* contains records that have been randomized). By reading each split/block locally, each Map produced RS_j, which is a reduced set of records in a split. By taking RS_j and approach for reducing dataset (filtering or fusion) as its input, a single Reduce function computes the final reduced dataset *RS*. The experiment conducted with few datasets (PokerHand, KDDCup, Susy, RLCP) show that the accuracy depends on the approach selected (filtering or fusion) and the dataset used, where for KDDCup and RLCP reach more than 99%. MRPR is also proved scalable in the used distributed Hadoop cluster.

B. Hadoop, HDFS and Map-Reduce

Hadoop is a platform that has been developed for storing and analyzing big data in distributed systems [1, 7, 9]. It comes with master-slave architecture and consists of the Hadoop Distributed File System (HDFS) for storage and MapReduce for computational capabilities. Its storage and computational capabilities scale with the addition of slave hosts/nodes to a Hadoop cluster, and can reach volume sizes in the petabytes on clusters with thousands of hosts. The following is some brief overview of HDFS and MapReduce.

HDFS: HDFS is a distributed file system designed for large-scale distributed data processing under frameworks such as MapReduce and is optimized for high throughput. It automatically re-replicates data blocks on nodes (the default is 3 replications).

MapReduce: MapReduce is a data processing model that has the advantage of easy scaling of data processing over multiple computing nodes. Map and Reduce functions run in each slave node parallelly. Where as Map functions read local blocks, Reducer functions take input from Map. A MapReduce program processes data by manipulating key-value pairs in the general form:

map: ($k1$,$v1$) ➞ list($k2$,$v2$)

reduce: ($k2$,list($v2$)) ➞ list($k3$,$v3$).

Map reads (key, value) pairs, then based on the algorithm designed by developers, it generates one or more output pairs list ($k2$, $v2$). Through a complex shuffle and sort phase, the output pairs are partitioned and then transferred to Reducer: Pairs with the same key are grouped together as ($k2$, list($v2$)) and then each partition with unique value of $k2$ is sent to a Reducer. The Reduce function (with a specific algorithm assigned) generates the final output pairs list($k3$, $v3$) for each group. Fig. 1 illustrates Map Reduce processes for computing average attribute values (column 2 and 3) for every record with specific Id (column 1) stored in blocks.

The overall MapReduce processed is as follows [1]: A client submits a job to the master, which then assign and manage Map and Reduce job parts to slave nodes. Map reads and processes blocks of files stored locally in the slave node, sent list of key-value to Reduce via shuffle and sort, then Reduce may write its computation results to HDFS.

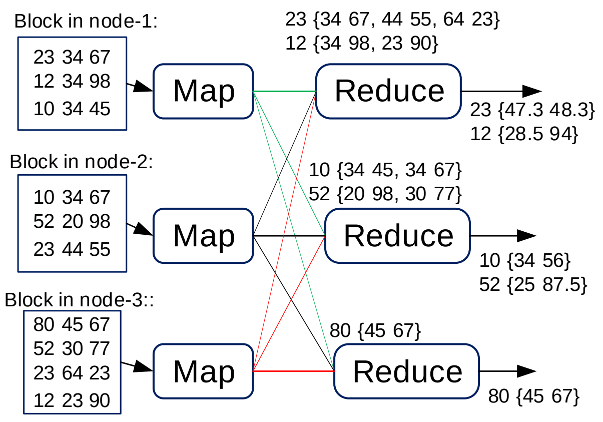


Fig. 1. Illustration of MapReduce processes.

C. Parallel Clustering based on MapReduce

Along with the popularity of Hadoop for analyzing big data, we found few research results for enhancing non-parallel into parallel (based on MapReduce) clustering techniques. The following are our findings:

(a) The most popular clustering algorithm, k-Means: Taking k (the number of cluster and initial cluster center), this algorithm iteratively assign objects into the closest center, then update the center based on the newly formed clusters. The technique in [11] is based on the core concept that the Map function assigns each sample to the closest center while the Reduce function performs the procedure of updating the new centers. Ma et al. in [12] add the capability of pre-computing the value of k and initial clusters (to reduce iterations). [4] and [10] enhance the technique in [11] to compute cluster patterns. Then, [13] enhances the fuzzy version of k-Means based on MapReduce.

(b) k-Medoids: The k-medoid that works similar to k-Means but by minimizing the absolute distance between the objects and the selected centroid, has been enhanced for MapReduce in [14].

(c) DBSCAN: The grid-based clustering algorithm that takes input of minimum points in a cluster and radius of a cluster (*eps*) form clusters by "connecting" objects with previously-formed clusters (from previous iteration) by using *eps*. Objects that are far from formed clusters will be regarded as outliers. Fu, Hu and Wang [15] have enhanced this algorithm based on MapReduce.

(d) Hierarchical: Aiming for clustering internet users by mining a huge volume of web access log, Vadivel and Raghunath [16] have developed MapReduce hierarchical clustering consisting three batches: First, selecting top N users with highest similarity values to form initial clusters. Second, adding other users to the initial clusters and merging clusters. Third, updating user-matrix file and modify the similarity values. As the clustering process are conducted in batches, the objects are queued (each queue contains m objects). By this approach, [16] intends to reduce the high IO and distributed communication overhead due to the read/write operations on distributed file system, caused by both matrix updating and similarity value modification. Another algorithm is DiSC [8] that is discussed below.

DiSC (distributed single-linkage hierarchical clustering algorithm) consists of two stages. The first stage (Prim map and Kruskal reducer) constructed subgraphs from data split and write the output as HDFS files that will be fed into the second stage (Kruskal map and Kruskal reducer) to "unite" those subgraphs.

The part that relates to our technique is the first stage, hence we only discuss this part in more detailed. The algorithm of the algorithm of the first stage is depicted in Algorithm 1 below.

Algorithm 1. Outline of DiSC, a distributed SHC algorithm
```
Input: a dataset $D$, $K$
Output: a MST for $D$
Steps:
1: Divide $D$ into $s$ roughly equal-sized splits: $D_1$, $D_2$, …., $D_s$
2: Form $C_s^2$ subgraphs containing the complete subgraph for every pair in {($D_i$, $D_j$) | | i < j and i, j ε [1; s]}
3: Use Prim's algorithm to compute the local minimum spanning tree (MST) for each subgraph in parallel, and output the MST's edge list in increasing order of edge weight
```

```
4: repeat
5:    Merge the intermediate MSTs for every $K$ subgraphs using the idea of Kruskal's algorithm
6: until all vertices belong to the same MST
7: return the final MST
```

In implementing the algorithm using MapReduce scheme, step 1, 2 and 3 are computed in the Mapper, while step 4 until 7 are performed in the Reducer. In the Mapper, the original dataset is divided into *s* splits ($D_1, D_2, \ldots, D_s$). Then subgraphs will be constructed from every two of these splits by allowing that some edges might be duplicated on multiple subgraphs. For example, a subgraph G_{ij} is constructed from a split pair (D_i, D_j) and a subgraph G_{ik} from (D_i, D_k), hence edges that are exclusively formed by the data points in D_i are duplicated for both G_{ij} and G_{ik}. Once a much smaller subgraph with the number of vertices roughly being 2*k* and the number of edges being C_k^2, where $k = \left\lceil \frac{n}{s} \right\rceil$, a serial MST algorithm is run locally for each subgraph using Prim's algorithm (starting with any random vertex, and growing the MST one edge at a time). With *K* is the number of subgraphs, Reducer is then merge those *K* subgraphs to form the final MST.

While the algorithms discussed above are applied to vectored-based objects, [17] has developed parallel clustering algorithms for non-vectored objects (such as bag of words).

It has been discussed previously that clustering can be adopted for reducing data. With regards to this objective, our opinions towards those parallel clustering techniques are as follows: Adopting clustering techniques that take *k* (number of clusters) and other variables (minimum points, radius in a cluster, etc.) that limit the clusters formed are not suitable for data reductions. These variables need to be defined in advance (by knowing the properties of the dataset), which require some computation (before clustering process). In this research, we aim to reduce the vectored-based objects. Thus, we can not adopt parallel k-Means, k-Medoids, DBSCAN and the technique proposed in [17].

Our better option is hierarchical clustering. However, we would not adopt the algorithm that employ three batches [16], as it needs to supply *N* (number of objects having the highest similarity) as it will require prior computation (to define *N*). As for DiSC [8], our comments: The Mapper constructs subgraphs, where each is constructed from two data splits. The data splits may be used by more that one Mapper, hence producing subgraphs that have overlapping edges (this will be resolved in Stage 2 for obtaining a complete graph of dendrogram). For reducing data, we need to construct subgraphs (local dendrogram) from data splits that are non-overlapping. Thus, we can not adopt this algorithm.

D. Agglomerative Hierarchical Clustering

Clustering techniques consider data tuples as objects. They partition the objects into groups, or *clusters*, so that objects within a cluster are “similar” to one another and “dissimilar” to objects in other clusters. In data reduction, the cluster representations of the data are used to replace the actual data [5]. The effectiveness of this technique depends on the data’s nature. It is much more effective for data that can be organized into distinct clusters than for smeared data.

A hierarchical clustering method can be either *agglomerative* or *divisive*, depending on whether the hierarchical decomposition is formed in a bottom-up (merging) or top down (splitting) fashion.

An agglomerative hierarchical clustering method uses a bottom-up strategy. It starts by letting each object form its own sub-cluster and iteratively merges sub-clusters into larger and larger sub-clusters, until all the objects are in a single cluster or certain termination conditions are satisfied (Fig. 2). The single cluster becomes the hierarchy’s root. For the merging step, it finds the two clusters that are closest to each other (according to some similarity or distance measure), and combines the two to form one cluster. Because two clusters are merged per iteration, where each cluster contains at least one object, an agglomerative method requires at most *n* iterations. A tree structure called a dendrogram is commonly used to represent the process of hierarchical clustering. In an agglomerative method, it shows how objects are grouped together step-by-step.

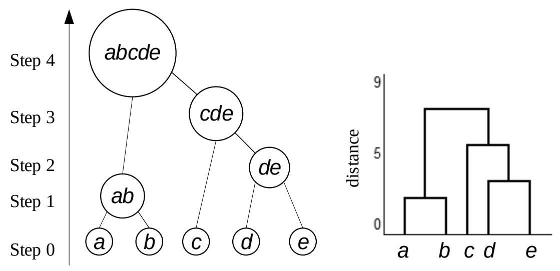


Fig. 2. Steps of the agglomerative (left) and the dendrogram tree (right).

The following are three distance measures that can be used:

(1) Minimum/single distance: A sub-tree/sub-cluster (or object) c_p is grouped with another sub-cluster c_q using the minimum distance between object members in c_p and c_q (the closest edge):

d_{min} = min(d_{ij}), where $1 \leq i \leq m$; $1 \leq j \leq n$; d_{ij} = distance between the object *i* in c_p and j in c_q; *m* = number of objects in sub-cluster c_p and *n* = number of objects in sub-cluster c_q.

(2) Maximum distance: A sub-tree/sub-cluster (or object) c_i is grouped into another sub-cluster c_j using the maximum distance between object members in c_i and c_i (the furthest edge):

d_{max} = max(d_{ij}).

(3) Means distance: A sub-tree/sub-cluster (or object) c_i is grouped into another sub-cluster c_j using the distance between centroids of c_i and c_i.

d_{means} = distance(centroid$_{cp}$, centroid$_{cq}$), where centroid$_{cp}$= the attributes means of all objects in sub-cluster c_p and centroid$_{cq}$ = the attributes means of all objects in sub-cluster c_q.

E. Data Stream Mining

A data stream, *DS*, can be defined as a sequence of data objects or samples. $DS = \{x_1, x_2, \ldots, x_t, \ldots\}$, where x_i is the *i*-th arrived data object [18]. In the data stream mining system, when a data stream comes, a temporary storage is used to store the most recent data. Then, the system may apply different time window approaches to create data synopsis that would be computed by analytical algorithms to produce useful patterns.

The time window can be selected from the following approaches:
(a) Landmark window: The data selected is the data stream from starting time instant 1 to the current time instant t_c, hence, the window is $W[1, t_c]$.
(b) Sliding window: The data selected is in the w most recent transactions (and the others are eliminated).
(c) Fading window: Each data object is assigned a different weight according to its arrival time so that the new transactions receive higher weights than the old ones.
(d) Tilted time window: It is somewhere between the fading window and sliding window variants.
More discussion of these can be found in [18].

Computational approaches applied to the synopsis can be:
(a) Incremental learning: The mining model incrementally evolves to adapt to changes in incoming data (an instance or a window).
(b) Two-phase or online–offline learning: In the first (online) phase, the data synopsis is updated in a real-time manner; in the second (offline) phase, the mining process is performed (based on a user request) on the stored synopsis.

Data stream mining may produce approximate results and has to satisfy the following constraints [18]:
(a) Single-pass: Each sample in a data stream is examined at most once and cannot be backtracked. The reason is: I/O operations are quite expensive than memory operations.
(b) Real-time response: The data processing must be fast.
(c) Bounded memory: The amount of arriving data is extremely large or potentially infinite. Ones may compute and store a small summary of the data streams and possibly throw away the rest of the data. Approximate results are acceptable.
(d) Concept-drift detection: The discovered patterns (or the underlying data distribution) change over time.

III. Proposed Technique

The proposed technique, namely BDRT-ParAgglo, can be used as part of big data stream mining (see Subsection II.E) using clustering technique. BDRT-ParAgglo is used to reduce the synopsis dataset stored as HDFS files. It handles dataset having numeric attribut

es (or ones that can can be transformed into numeric) in the two-phase learning system: Once the synopsis of raw big data is available, BDRT-ParAgglo is executed to generate patterns (see Fig. 4 3). The function to collect raw big data may adopt landmark or sliding window. The outputs of BDRT-ParAgglo, which are patterns with a lot smaller size (compared to the raw dataset), can be stored permanently while the raw big data itself can then be erased.

The resulted patterns can be analyzed by parallel data mining techniques in the distributed systems or exported to non-distributed and be analyzed by "traditional" (non-parallel) techniques provided by many existing tools or applications.

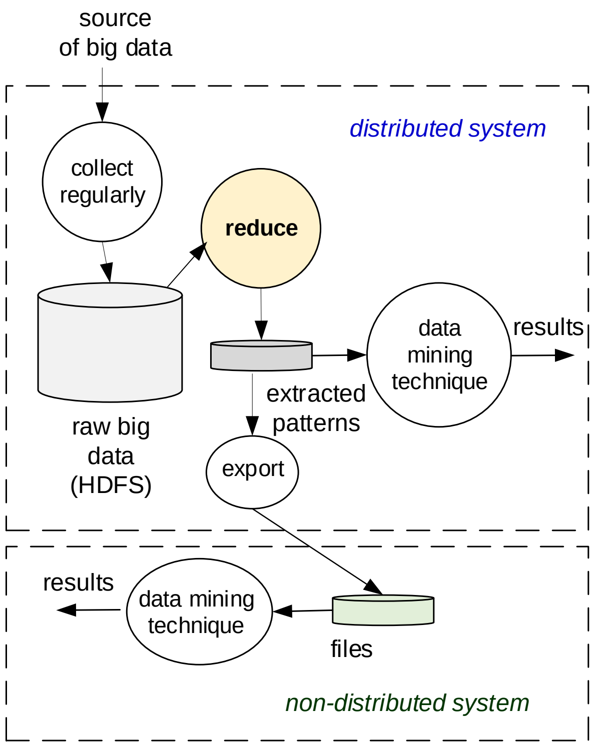


Fig. 3. Scheme of batch big data reducer.

BDRT-ParAgglo is designed based on agglomerative clustering algorithm. By employing this algorithm, BDRT-ParAgglo handles big dataset of matrices (all objects possess the same number of numerical attributes) only. Missing values and/or wrong value of attributes should be addressed in the data preprocessing step.

The concept our proposed technique is as follows:
(a) To ensure that large volume of data can be handled, we randomly divided the objects into n disjoint partitions, then a dendrogram tree is built from each partition (in every slave node of Hadoop cluster) using agglomerative hierarchical clustering method. The maximum number of objects in a dendrogram is defined as a parameter (*maxObject*) such that its value can be defined by considering the slave node memory capacity. Hence, if a partition contains greater than *maxObject*, the Reducer processing this partition will create more than one dendrogram.
(b) To allow flexibility, we provide three choices of distance type for constructing dendrograms, which are single (minimum), complete and means (average).
(c) Once a complete dendrogram is constructed from a partition, it is "cut" using certain distance value (defined by users) to form clusters.
(d) Cluster patterns (such as centroids, number of objects in each cluster, etc.) are computed from each cluster formed, then written to HDFS. As the objects in each partition/dendrogram are obtained by random selection from the raw big data, it is expected that the patterns are less biased (by the incoming sequence of objects).

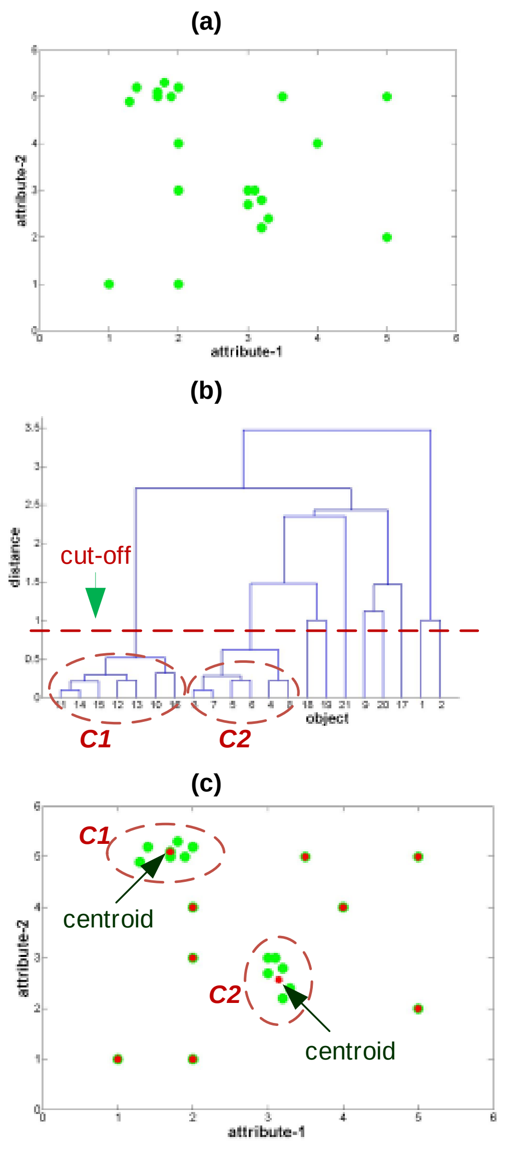


Fig. 4. (a) Data distribution; (b) Dendroram with its cut-off; (c) The patterns or reduced data (red points).

The illustration of "cutting" dendrogram to obtain patterns is depicted on Fig. 4: (a) The distribution of a dataset having 21 objects; (b) cutting the dendrogram constructed from dataset using certain distance that forms *C1*, *C2* clusters with few members and other clusters each having one object member; (c) if cluster centroids are regarded as the patterns, from those 21 objects can be obtained 10 patterns (red dots), hence the dataset is reduced by approximately 50% (10/21). As shown on the figure, the dense objects are replaced by its centroids while the sparse objects are preserved as is.

By selecting the agglomerative method and MapReduce, the objectives and advantages of our proposed technique include:

(a) Reducing the volume of "dense objects" with their cluster patterns having a lot smaller size;

(b) Allowing flexibility in defining cut off distance to adjust the granularity (fine-grain) of the the reduced dataset (resulted cluster patterns);

(c) Permitting flexibility in using single, complete or mean distance in constructing dendrogram such that users can select the most suitable one for a certain dataset;

(d) Outliers are preserved as they may be needed by data analysis technique;

(e) Numbers of data partitions (*n*) can be adjusted for optimal computation in the distributed network (by considering the slave/data nodes number and their specification, i.e. available memory).

A. The Proposed Algorithm

Our proposed overall algorithm for reducing dataset is depicted in Algorithm 2 and its visualization is presented in Fig. 5.

```
Algorithm 2: Reduce dataset
Input: raw dataset (TO); number of partitions (n), distance type for constructing dendrogram (distType), cut-off distance (co)
Output: cluster patterns CP = {CP_j} as the reduced dataset
Steps:
(1) for each object (o_i) in TO
(2)     check and preprocessed each attribute value then put the object in a bin/partition (TO_j), send TO_j to one of the parallel processes
(3)  for each TO_j: construct dendrogram, cluster the objects in dendrogram using cut-off co, compute local patterns CP_j
(4) collect and store all CP_j in CP then write CP into files
```

As discussed in [19] and [4], patterns generated from clusters may include the number of objects in each cluster, the average (means), minimum, maximum, standard deviation of each attribute values and percentage of objects having each of the attribute values. Hence, the attribute of the pattern (CP_j) can be selected from these components. In doing so, the characteristic of the dataset and purpose of reducing data should be considered.

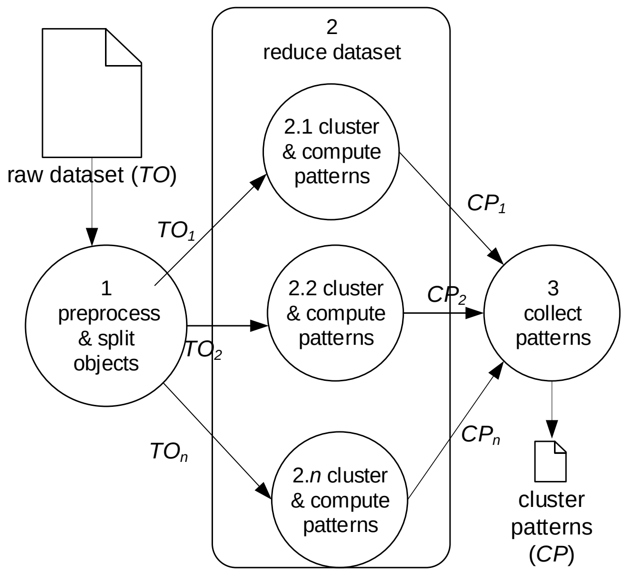

2
reduce dataset
raw dataset (TO)
2.1 cluster & compute patterns
TO_1
CP_1
1 preprocess & split objects
2.2 cluster & compute patterns
CP_2
3 collect patterns
TO_2
TO_n
CP_n
2.n cluster & compute patterns
cluster patterns (CP)

Fig. 5. The proposed parallel algorithm.

B. Map-Reduce Functions Design

Algorithm 2 is implemented as Map and Reduce functions as illustrated in Fig. 6 and described as follows. As discussed in Subsection II.B, each Reducer receives and processes a collection of <key, value> pairs having the same key value. We view this strict rule as an advantage and use it in our Mapper to partition the dataset by simply assigning a random number as the output key (for every value of record emitted). As stated in [1, 7], each Reducer that runs in every slave node is allowed to write its output directly into HDFS files. We embrace this advantage for eliminating the process of merging of all Reducer results (as employed in [8]) to speed up the computation. Hence, our Reducer write the output (reduced dataset) into HDFS files directly.

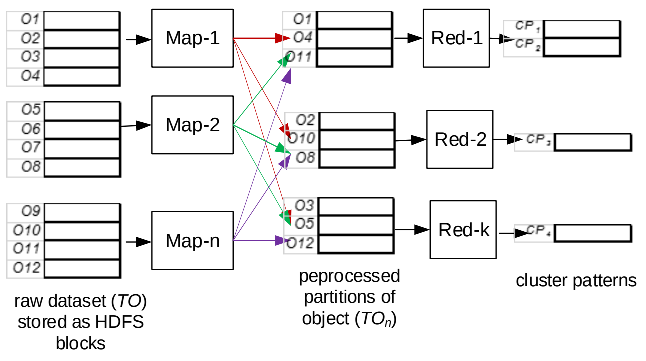


Fig. 6. Illustration of the Map-Reduce function.

As sown in Fig. 6, each Mapper (Map-*i*) read local blocks of HDFS containing consecutive of records/objects (O_i) then distributes every records randomly to every Reducer. The Reducer then perform data reduction computations and write the results (cluster patterns, CP_i) to files. Here, we do not use local HDFS blocks (read by local Mapper) as data partition because: Hadoop creates HDFS blocks consecutively by "chunking" the input file. Hence, each block contains ordered (not random) that are not proper for constructing representative dendrograms.

The detailed steps of Mapper and Reducer functions are depicted in Algorithm 3 and 4.

Algorithm 3: Mapper

Input: raw dataset (*TO*); number of partitions (*nPar*)

Output: *key* = an integer number ε {1… *nPar* }, *value* = text of preprocessed set of attribute values

Description: Split *TO* object disjointly by assigning a random number for each object

Steps:

```
(1) value ← read a line (an object) and preprocessed its attributes accordingly
(2) key ← a random integer k, where 1 ≤ k ≤ nPar // By assigning key with a random integer k, where 1 ≤ k ≤ nPar, we intend to split the records (values) evenly (approximated) to every Reducer.
(3) emit pair of <key, value>
```

Depending on the *k* value, the number of randomized objects (*values*) received by Reducer may still be very large such that forming a dendrogram from these large objects may not fit into the node memory. Cluster patterns created from very large large of objects may not highly accurate in representing the semantic of the objects as well. Hence, in Reducer, we introduce *maxObject* variable that is used to limit the maximum number of objects in a dendrogram.

Algorithm 4: Reducer

Input: list of records <*key*, *value*> emitted by Mapper where *key* has the same value in every record, *maxObject*, *distType* ε {single, complete, means}, cut-off distance (*co*)

Output: cluster patterns, ***CP*$_j$**

Description: Construct dendrograms from list of *value* by applying constraint that a dendrogram has at most *maxObject* objects, form clusters from dendrograms using *co,* compute and write the patterns from each cluster formed into files

Steps:

```
(1) listTrees ← [] // array of single_tree
(2) for each pair of <key, value>
(3)     form an independent tree, single_tree, from value
(4)     add single_tree to listTrees
(5)     isProcessed = false
(6)     if the number of single_tree in listTrees = maxObject
(7)         form dendrogram_tree from listTrees using distance measure of distType
(8)         form object clusters from dendrogram_tree using co
(9)         compute patterns CP_j from every cluster formed in step (8) then write to files
(10)        clear object trees in listTrees // remove objects from node memory
(11)        isProcessed = true
(12) if isProcessed = false // there are objects in listTrees that have not been reduced
(13)        do step (7, 8, 9, 10) // compute the cluster patterns from the remaining objects
```

Data structure used in Algorithm 4 is as follows:

(a) *single_tree* = {*atrVal[]*, *clsLabel*} where *atrVal[]* = array of the object attribute values, *clsLabel* = an integer denoting the number of cluster that this tree belongs (filled at the clustering process using cut-off distance).

(b) *dendrogram_tree* = {*leftCls*, *distType*, *distance*, *clsLabel*, *rightCls*} where *leftCls* = reference/pointer to left-side of *single_tree*, *rightCls* = pointer to right-side of *single_tree*, *distType* ε {0, 1, 2} where 0 is single, 1 is complete, 2 is mean distance type used to form dendrogram, and *distance* = a real number denoting the distance between its pair of left and right tree/cluster (Fig. 7).

(c) ***CP*$_j$** = {*nObjects*, *minAtrVal[]*, *maxAtrVal[]*, *meansAtrVal[]*}, where *nObjects* = number of objects, *minAtrVal[]* = array of minimum attribute values, *maxAtrVal[]*= array of maximum attribute values, *meansAtrVal[]*= array of average (means) attribute values. As for computing attribute deviations and the value percentages require one more iteration for visiting objects in every cluster (see Subsection III.A), we exclude these 2 components as elements in the cluster patterns to simplify Reducer time complexity.

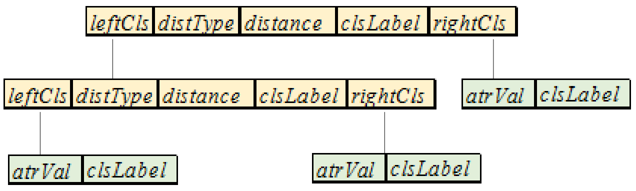


Fig. 7. The dendrogram tree structure.

The visual illustration of the Reducer algorithm in processing a single dendrogram is depicted on Fig. 8. Each Reducer constructs a dendrogram tree from the data partition containing a collection of objects fed into this function. Then all Reducer functions "cut" the dendrogram using the same value of distance cut-off (*co*) where each branch below the cut-off will form one cluster having the member objects included in the branch. Hence, each

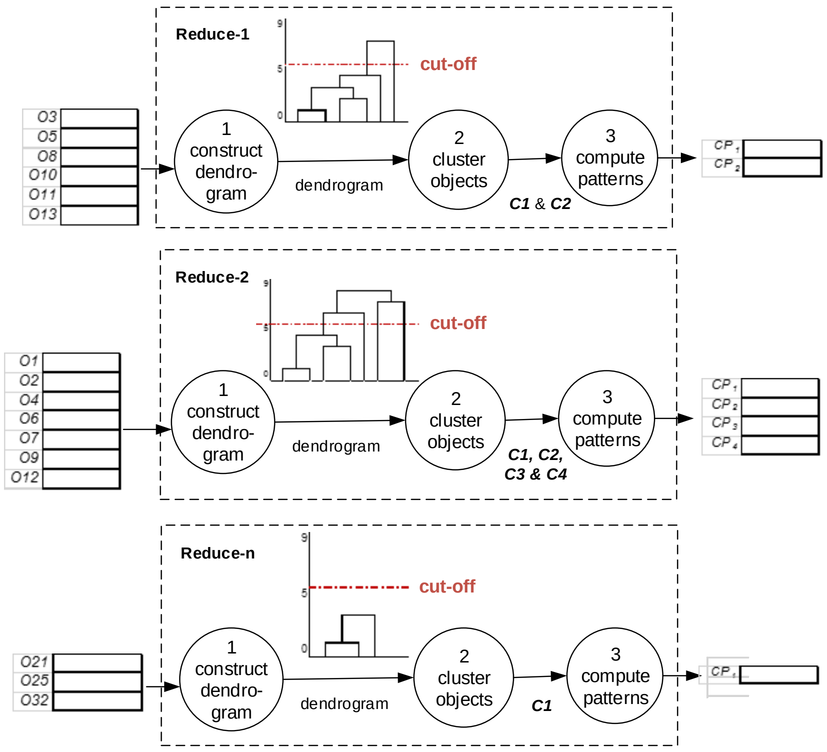


Fig. 8. Illustration of three Reducer functions that produce 2, 4 and 1 cluster patterns from records emitted by Map functions.

Reducer may form different number of clusters. Generally, the smaller the value of cut-off, the more clusters will be resulted (but this depends on the dendrogram structure). As a pattern is computed from a cluster, the number of cluster pattern dispatched by every Reducer is equal to the clusters number.

The detailed of three steps in Algorithm 4 is discussed below.

Step-7: form *dendrogram_tree* from *listTrees* based on *distType*
(7.1.) if no *dendrogram_tree* existed, get two *single_tree* from *listTrees* having the shortest *distType* distance and form a *dendrogram_tree* from these two trees, store the distance in the tree, then remove these two from *listTrees*
(7.2.) else: get the next element of *single_tree* in *listTrees* then
(a) if *distType* = *single* or *complete*, find the shortest (or longest) distance of this *single_tree* to every *single_tree* in *dendrogram_trees*,
(b) if *distType* = *means*, compute the average distance between the sub-tree/branches of the *dendrogram_tree* using the stored distance and by adding this *single-tree* in it, select the shortest mean distance, construct left branch as the existing *dendrogram_tree* having the shortest/longest distance, store the distance in the tree, and the right branch as *single_tree* (or the opposite), remove this *single_tree* from *listTrees*
(7.3) repeat (7.2) until *listTrees* is empty.

The method for computing the distance between 2 objects can be selected from Euclidean, Manhattan, etc., which are applicable for numeric attributes. As every object attribute must be read, the complexity is *O(n)*, where n is the number of attributes. The complexity for finding shortest distance of *single* and *complete* type between a *single_tree* object and *single_tree* objects in previously formed *dendrogram_tree* is *O(n)*, where *n* is the count of single_tree. For finding shortest distance of *average* or *means*, the computation is more efficient, as the centroid of every sub-dendrogram-tree is stored in the tree then each time an object is added the tis centroid is recomputed. The complexity is also *O(n)* but with *n* is the number of branches, which is smaller than the count of single_tree. Hence, the overall complexity of Step-7 is between *O(n log(n))* to $O(n^2)$ where *n* = count of nodes *listTrees.*

Step-8: form object clusters from *dendrogram_tree* using cut-off distance *co*
(8.1.) create list of empty tree, *clsResults*
(8.2) if *dendrogram_tree* has only a single tree, add (the reference of) this to *clsResults* as its element
(8.3) else
(8.4) if the *dendrogram_tree* distance <= *distance* add (the reference of) this *dendrogram_tree* to *clsResults* as its element
(8.5) else repeat (6.2) for *dendrogram_tree* left and right branch of the tree

The complexity of Step-8 is O(*m*), where $1 \leq m \leq$ depth of the *dendrogram_tree.*

Step-9: compute patterns $\boldsymbol{CP_j}$ from every cluster formed for each cluster element in *clsResults* (the result of Step 6): visit each object member of this cluster for computing each element of its pattern, $\boldsymbol{CP_j}$ (the elements are *nObjects*, *minAtrVal[]*, *maxAtrVal[]* and *meansAtrVal[]*).

The complexity of Step-9 is *O*(*stn*), where *s* = count of elements in *clsResults*, *t* = count of elements in a cluster in the particular element of *clsResults* (here, $1 \leq t \leq$ count of objects in this data partition received by Reducer), *n* = count of object attributes.

In our proposed algorithm, Mapper only partitions the dataset. The complex computation is performed by Reducer. Hence, we present the complexity of the Reducer algorithms only.

IV. Experiments, Analysis and Discussion

To evaluate BDRT-ParAgglo, three series of experiment, where each has specific objective, were performed. All of the experiments were conducted in Hadoop clusters with *N* slave nodes (Fig. 9) where all are commodity computers with low specification.

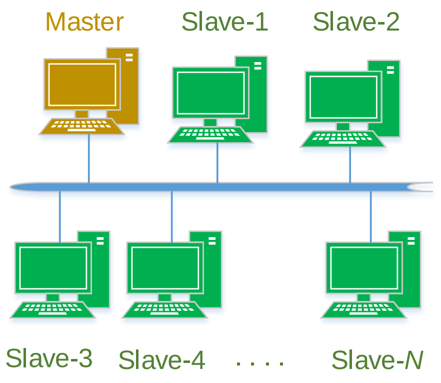


Fig. 9. Hadoop cluster with *N* slave nodes.

A. Patterns Evaluation

The intention of the experiments are to evaluate the proposed techniques and to answer the following questions:

Q-1: Will the proposed technique generate reduced dataset (patterns) that semantically represent the original data?

Q-2: How are the comparison among the three distance types computation (in generating the reduced dataset)?

Q-3: What variables that affect the percentage of the reduced data? Will the number of partition and maximum object in a dendrogram correlate to that percentage?

Experiments using Synthetic Datasets

For these experiments, we used synthetic and real datasets. The experiments were performed on a Hadoop Cluster having one master with four slave nodes (Fig. 10 9) where each node has the following specifications: The processor is Quad-Core running at 3.2 GHz with 8 Gbyte of memory.

In the first experiments, which is intended to answer Q-1 and Q-2, we use two synthetic datasets with 2 numeric attributes, where objects are created randomly such that they have particular distributions. The first and second dataset contains 400 and 1200 objects. The patterns as the reduced dataset consist of two elements (proposed in [4]), which are the attributes means/centroids and the number of objects in the sub-clusters.

The dataset with 400 objects has three dense-objects and outliers. We reduce this dataset using the parameters:

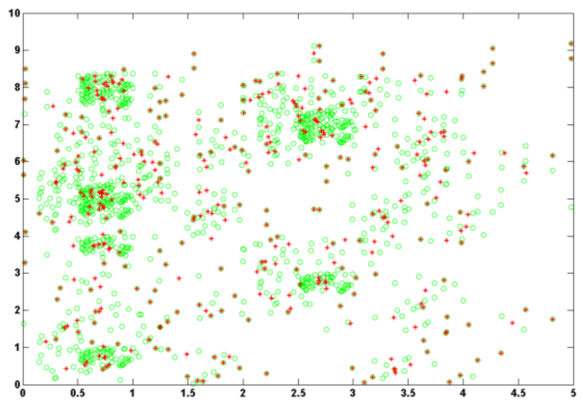
(a) *nPar* = 1, *maxObject* = 20, *distType* ε {single, complete, means}, *co* = 0.5

(b) *nPar* = 3, *maxObject* = 20, *distType* ε {single, complete, means}, *co* = 0.5

By observing the results, we conclude that the results of (a) and (b) are similar, which can be interpreted that partitioning the dataset (*nPar* = 3) will not greatly change the patterns computed from the dataset without being partitioned (*nPar* = 1). Also, by visualizing the original dataset vs the resulted output of patterns, we found that the distance type of single, complete and average linkage produce similar patterns. However, patterns of complete and average distance type are closer.

The dataset with 1200 objects has six dense-objects and outliers. We reduce this dataset using combination parameters of *nPar* = 3, *maxObject* ε {30, 50, 70}, *distType* ε {single, complete, means}, *co* ε {0.2, 0.4, 0.6, 0.8, 1.0}. Some sample of the experiment results are presented in Fig. 10. By comparing the distribution of original dataset and the cluster centroids, it can be seen that the centroids (red) represents the original dataset (green). On Fig. 11 we present the count of cluster centroid members using circles with the size of the diameter represents the count of cluster centroid members. It shows that among the original dataset with green dense objects, the circles have larger diameter. Hence, similar/close objects are greatly reduced, while sparse objects are less reduced.

(a)



(b)

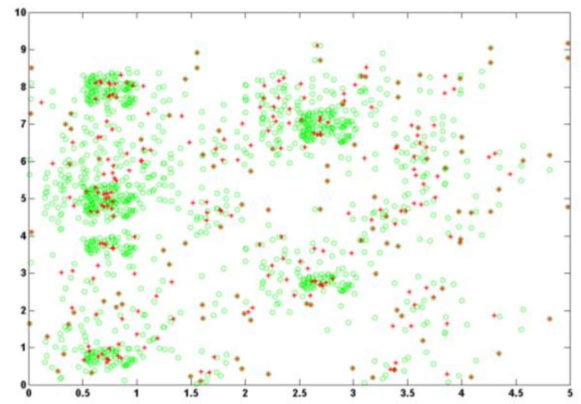


Fig. 10. The distribution of dendrogram cluster centroids (red) vs original data (green) for 2-d 1200 objects with *nPar* = 3, *maxObject* = 50, *distType* = means with *co* = (a) 0.6 and (b) 0.8.

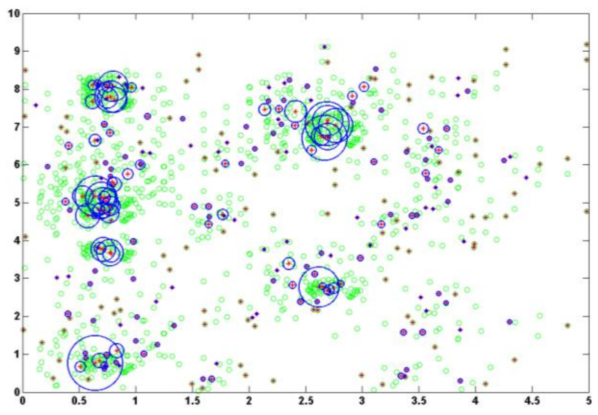


Fig. 11. The object member counts of sub-clusters in Fig. 10(b), where the more members represented by larger size of blue circle.

By observing the plots of the experiments using those two datasets (400 and 1200 objects), we conclude that: The generated reduced dataset (patterns) semantically represents the original dataset, where:

(a) Dense original objects are represented with fewer patterns but with larger member counts;

(b) Outliers are preserved (with small member counts).

Thus, Q-1 is resolved.

By observing the experiment results using *distType* of single, complete and means, we find that while the results of complete and means/average are similar, where as the results of single is slightly different. The cluster centroids generated using complete and average distance represents the original data better. Thus, Q-2 is partially answered.

For finding more answer of Q-2 and Q-3, we use the synthetic dataset of 1200 objects and reduce this dataset using combination variables of *nPar* ε {2, 3, 4, 5}, *maxObject* ε {50, 70}, *distType* ε {single, complete, mean}, *co* ε {0.2, 0.4, 0.6, 0.8, 1.0}. The sample of the results are presented in Table I and II.

By analyzing the results, the following are findings of the variables influences:

(a) *co* (cut-off distance): As discussed in Subsection III.B and illustrated in Fig. 8, the larger value of *co*, the less clusters or patterns will be created, which means less percentage of the reduced dataset (compared to the original one). Our experiment results confirm this (as shown in Table I and II).

(b) *nPar*: Frequently, the larger the partitions, the larger the number of patterns resulted (but sometime it is smaller). This fact is justified, as the larger the partitions, the more dendrograms will be produced (by more slave nodes) and the more clusters/ patterns will be resulted (using certain *co*). Hence, the experiment results prove this.

(c) *maxObject*: The larger value of *maxObject*, the smaller number of patterns created. Using larger number of *maxObject* will result in fewer dendrogram being created. Hence, the clusters/patterns will also be fewer.

(d) *distType*: Using single *distType* will reduced the original dataset the most, follow by means, then complete (the least). Creating dendrogram using single distance type (the closest distance between object in two sub-cluster) will produce smaller distance among clusters and when be cut using certain value of *co* will produce fewer clusters/patterns. Our experiments results conform to this.

TABLE I. COMPARISON OF REDUCED DATA USING SEVERAL VARIABLES

mO	*co*	*nPar* = 2					
		d.t. = single		*d.t.* = complete		*d.t.*=mean	
		#ptrn	%	#ptrn	%	#ptrn	%
50	0.2	673	56.1	734	61.2	737	61.4
	0.6	234	19.5	398	33.2	328	27.3
	1.0	101	8.4	243	20.3	207	17.3
70	0.2	574	47.8	683	56.9	679	56.6
	0.6	199	16.6	334	27.8	311	25.9
	1.0	68	5.7	198	16.5	179	14.9
		nPar = 3					
mO	*co*	*d.t.* = single		*d.t.* = complete		*d.t.*=mean	
		#ptrn	%	#ptrn	%	#ptrn	%
50	0.2	703	58.6	761	63.4	779	64.9
	0.6	265	22.1	426	35.5	383	31.9
	1.0	116	9.7	288	24.0	241	20.1
70	0.2	646	53.8	699	58.3	687	57.3
	0.6	197	16.4	367	30.6	306	25.5
	1.0	77	6.4	207	17.3	171	14.3

Note: *mO = maxObject, co= cut-off distance, d.t. = distType,* #ptrn = patterns count, % = (#ptrn /original object counts) x 100%

TABLE II. COMPARISON OF REDUCED DATA USING *DISTTYPE* = MEAN

nPar	*co*	*maxObj*=50		*maxObj*=70	
		#ptrn	%	#ptrn	%
2	0.2	737	61.4	679	56.6
	0.6	328	27.3	311	25.9
	1.0	207	17.3	179	14.9
3	0.2	779	64.9	687	57.3
	0.6	383	31.9	306	25.5
	1.0	241	20.1	171	14.3
4	0.2	804	67.0	764	63.7
	0.6	381	31.8	339	28.3
	1.0	256	21.3	197	16.4
5	0.2	767	63.9	733	61.1
	0.6	413	34.4	350	29.2
	1.0	237	19.8	203	16.9

Note: *co= cut-off distance*, #ptrn = patterns count, % = (#ptrn/original object counts) x 100%

Thus, from those experiments, it can be concluded that:

(1) The larger the partition (*nPar*), the more patterns (reduced data) resulted;

(2) The smaller the *co* (distance cut-off) applied to dendrograms in forming clusters, the more patterns resulted;

(3) The larger the maximum objects in a dendrogram, the smaller patterns resulted;

(4) The selection of distance type influence the resulted number of patterns or percentage of reduction, with the following order: single (reduced the dataset the most), mean, complete (the least.

Experiments using Real Large Dataset

The experiments are conducted with larger size of Hadoop cluster, which has 1 master and 14 slave nodes (Fig. 9), where each node machine has processor of Quad-Core running at 3.2 GHz, the memory of 7 machines is 4 Gbyte and of 8 machines is 6 Gbyte, the HDFS block (for input and output) size is configured as 16 Mbyte.

As a sample of reducing real dataset, we use: The dataset of household energy consumption, which is obtained from https://archive.ics.uci.edu/ml/datasets/ with the size of approximately 132 Mb. This archive contains 2075259 measurements (records/objects) gathered between December 2006 and November 2010. The sample of the dataset are as follows:

```
9/6/2007; 17:31:00 ; 0.486 ; 0.066; 241.810; 2.000; 0.000 ; 0.000 ; 0.000
9/6/2007; 17:32:00 ; 0.484 ; 0.066; 241.220; 2.000 ; 0.000 ; 0.000; 0.000
9/6/2007; 17:33:00 ; 0.484 ; 0.066 ; 241.510; 2.000; 0.000 ; 0.000 ; 0.000
```

Each line presents a record with 9 attributes, the excerpts are: (1) Date; (2) Time; (3, 4, 5, 6) some results of metrics; (7) sub_metering_1: energy sub-metering (watt-hour) that corresponds to the kitchen, (8) sub_metering_2: energy sub-metering that corresponds to the laundry room; (9) sub_metering_3: energy sub-metering that corresponds to a water-heater and an air-conditioner.

The reason that we select that dataset is: We can preprocess the dataset and obtain numeric attributes that are meaningful and can be mined to obtain interesting patterns (see [10] for more discussion). The data preprocessing performed in Map function (see Algorithm 3) is as follows:
(a) Number of day (1, 2, …7) is extracted from Date and stored as attribute-1;
(b) Hour (1, 2,…24) is extracted from Time and stored as attribute-2;
(c) The value of sub_metering_1, _2 and _3 are taken as is and stored as attribute-3, -4, -5.
Thus, the preprocessed dataset has 5 attributes, which are day number, hour and 3 sub-metering measures.

The results of reducing the dataset using *distType* = single and means, for several value of *nPar*, *maxObj* and *co* are presented in Table III. As shown in the table, we find that the experiment results are very much consistent with the results with synthetic dataset (see Table I and II), which are:
a) Reducing the dataset using single *distType* will produce fewer patterns compared to using mean *distType*.
b) The larger the partition (*nPar*), the larger the count of patterns resulted;
c) The larger the maximum objects (*maxObj*) used in a dendrogram, the fewer the patterns resulted;
d) The smaller the *co* (distance cut-off), the more patterns resulted (the smaller reduced percentage).

B. *Variables Influencing Execution Speed*

The objective of this experiment is to find BDRT-ParAgglo variables that influence the execution time. The data used is the dataset of household energy consumption as discussed in Subsection IV.A. In conducting experiments, each run was repeated five times and the execution times were averaged.

TABLE III. THE RESULTS OF REDUCING REAL DATASET

distType = single, *nPar* = 10				
maxObj	*co*	#patterns	%	exec time (min:sec)
50	1	798,980	38.50	1:35
	1.5	674,265	32.49	1:21
	2	590,422	28.45	1:35
100	1	636,668	30.68	6:12
	1.5	508,425	24.50	6:03
	2	412,906	19.90	5:58
200	1	494,078	23.81	23:08
	1.5	359,842	17.34	21:52
	2	281,691	13.57	22:18
distType = single, *nPar* = 20				
maxObj	*co*	#patterns	%	exec time (min:sec)
50	1	1,053,130	50.75	1:41
	1.5	641,642	30.92	2:50
	2	723,728	34.87	2:03
100	1	865,759	41.72	4:11
	1.5	641,642	30.92	4:50
	2	494,815	23.84	4:00
200	1	676,871	32.62	19:27
	1.5	444,253	21.41	22:31
	2	315,646	15.21	21:59
distType = mean, *nPar* = 10				
maxObj	*co*	#patterns	%	exec time (min:sec)
50	1	996,653	48.0	3:55
	1.5	638,426	30.8	4:24
	2	521,195	25.1	4:18
100	1	896,315	43.2	4:33
	1.5	638,426	30.8	4:24
	2	521,195	25.1	4:18
200	1	801,405	38.6	14:53
	1.5	523,971	25.2	14:53
	2	408,027	19.7	15:37
distType = mean, *nPar* = 20				
maxObj	*co*	#patterns	%	exec time (min:sec)
50	1	1,241,876	59.8	2:19
	1.5	992,973	47.8	2:02
	2	847,262	40.8	1:53
100	1	1,121,013	54.0	4:33
	1.5	824,520	39.7	4:31
	2	669,022	32.2	4:26
200	1	1,018,257	49.1	15:06
	1.5	678,300	32.7	14:51
	2	513,655	24.8	14:54

Table III shows that *maxObj* is the most important variable that influences time executions. The variable of *distType*, *nPar* and *co* do not influence the time significantly (using the same *maxObj* value, the use of different *distType*, *nPar* and *co* only slightly affect the execution time). This is in accordance with the time complexity analysis (see Subsection III.B), where the most complex computations is the dendrogram creation, which is $O(n^2)$ with n is the object number.

It also shows in Table III that the execution time using *single* distance type is longer than *mean* distance type. This complies to the Step 7 of the Reducer algorithm (Algorithm 4), where *mean* distance computation is more efficient.

Another finding, the use of larger *nPar*, which means that through Hadoop shuffle process more data partitions are created and each is sent to a Reducer running in a slave node, slightly increase the time execution. Our analysis: As discussed in [20], the Hadoop shuffle process is a complicated phase between Map and Reduce functions. It involves sorting, grouping (among pairs of (*k*,*v*), pairs having the same value of *k* is placed in one group), and HTTP transferring. During this shuffle phase, a large mount of time is also consumed disk I/O (storing the grouping results and reading the grouped records) with a low speed of data throughput. In the case of sorting, depending on the algorithm used (Quicksort, Mergesort, Heapsort, etc.), the best complexity is $O(n)$ where as the worst is $O(n \log n)$ or $O(n^2)$. The complex computation, disk I/O and data transfer through HTTP are the caused why the larger *nPar* used the more time needed.

C. Reduction Percentage

The objective of this experiment is to observe the percentage of data reduction when the input is big dataset. Since we can not find real big dataset with numerical attributes, we create the big data by "multiplying" the preprocessed household energy consumption dataset (used in the previous experiments) up to 5 gigabyte.

Some example of the results, the reduction statistics with *nPar* = 9, *distType* = mean and *co* = 1 is depicted in Table IV. It shows in the table that using this value of *nPar*, although the input data size vary, the percentage of reduction (number of output records/input records), more or less is constant. Other results are depicted on Table V that shows the average of reduction percentage for *nPar* = 5, 9, 14 and 18. It can be observed that the larger *nPar* value, the more the number of output records or patterns. The analysis is: The more number partitions (*nPar*) the more dendrograms created. Hence, cutting these dendrograms using the same value of *co* will produce more number of subclusters and their patterns (computed for each sub-cluster).

TABLE IV. DATA REDUCTION WITH *nPar* = 9 AND *distType* = means

Input Size (Gb)	#Input Records	#Bk	Time (sec)	# Output Records	%Re-duced
0.51	8,301,036	32	302	3,389,523	40.83%
1.01	16,602,072	64	307	6,794,618	40.93%
1.49	24,903,108	96	319	10,236,593	41.11%
1.98	33,204,144	127	469	13,582,097	40.90%
2.48	41,505,180	159	527	17,001,881	40.96%
2.97	49,806,216	191	612	20,377,226	40.91%
3.47	58,107,252	222	647	23,778,819	40.92%
3.96	66,408,288	254	823	27,177,419	40.92%
4.46	74,709,324	286	892	30,578,403	40.93%
4.95	83,010,360	318	955	33,956,222	40.91%

#Bk = #Blocks, %Reduced = #InputRecords/#OutputRecords

TABLE V. AVERAGE PERCENTAGE OF RECORDS

nPar	%Reduction
5	30.9107
9	40.9329
14	48.3333
18	52.2957

D. Evaluating Hadoop Configuration towards BDRT-ParAgglo

As BDRT-ParAgglo is based on Hadoop MapReduce, in these experiments, we intend to evaluate the influence of Hadoop cluster configuration towards BDRT-ParAgglo. The data used is from 1 Gb up to 20 Gb, obtained by multiplying the household energy consumption dataset. The Hadoop cluster (Fig. 9) is configured with (maximum of) 14 slave nodes (each node has 4Gb of memory and Quad-Core processor running at 3.2 GHz). The Hadoop configuration that are evaluated are number of slave nodes and HDFS block size. Here, *maxObj* is set to 100 as this is the largest that can be handled by the slave nodes with that limited memory (we experimented with larger size of *maxObj*, but the executions were failed due to lack of slave nodes memory capacity). As one important finding of the previous experiments (see Subsection IV.B) suggest that *nPar* influence the execution speed, to obtain more facts, we use 3 values of *nPar* that are related to the number of slave nodes (5, 9 and 14).

The first series of experiments are performed by executing the algorithm using *distType* = means and varying the values of data input size, number of slave nodes, partition (*nPar*) and HDFS block size. The results are presented using plots.

The plots of the experiments results using block size of 32 and 64 Mb are presented on Fig. 12 and 13, while using 16 Mb is depicted on Fig. A.1 in Appendix A. (Note: Block size = 16 Mb is actually too small and not recommended. The experiments using 16 Mb block size are performed for comparison intention only.)

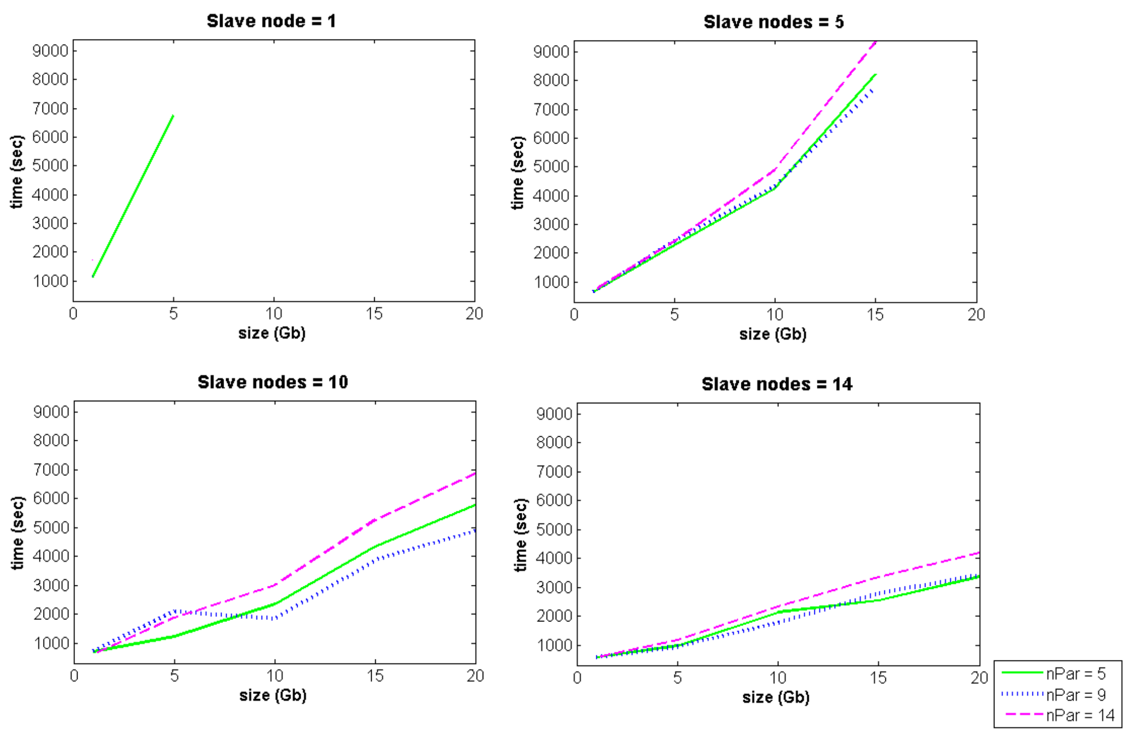


Fig. 12. Plot of execution time for distance type = means, HDFS block size = 32 Mb.

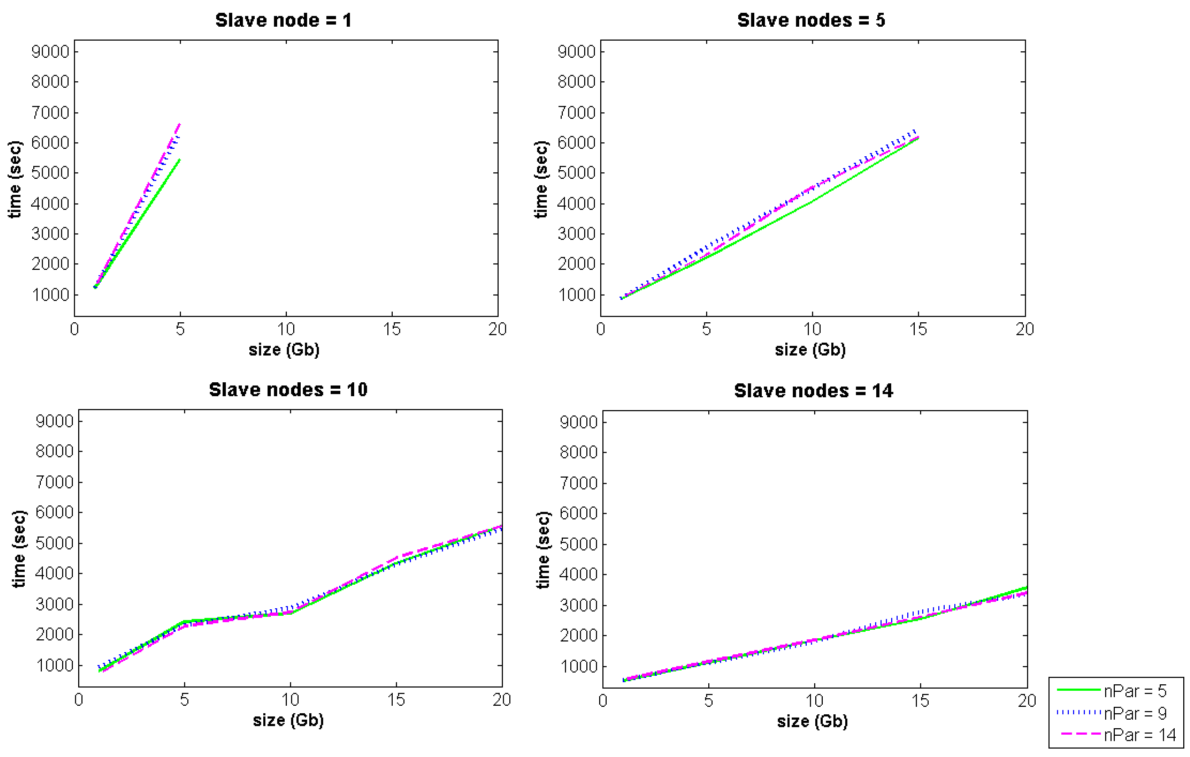


Fig. 13. Plot of execution time for distance type = means and HDFS block size = 64 Mb

The experiment findings using block size of 16Mb (Fig. A.1) and 32 Mb (Fig. 12) are similar, which are: (a) By adding slave nodes, bigger size of dataset can be processed (using node = 1, the maximum of data input that can be processed is up to 5Gb only, using node = 5 is up to 15 Gb); (b) The more slave nodes, the faster the execution time. (c) Execution using *nPar* = 14 is the slowest; (d) Using larger block size (32Mb) on 14 nodes, the speed of *nPar* = 9 is almost equal to *nPar* = 5.

The experiment findings using block size of 64 Mb (Fig .14) are similar to the 16 Mb and 32 Mb findings, with this additional fact: Using block size of 64 Mb, the execution speed on 10 and 14 nodes with *nPar* = 5, 9 and 14 are approximately equal.

Thus, the summary of the above findings are:

(a) Using more slave nodes will increase the size of big data that can be processed as well as the speed of the execution time;
(b) In general, using larger block size reduces the execution time (the input dataset itself can be partitioned into up to the count of slave nodes);
(c) Using larger value of *nPar* will slow down the execution.

Analysis of the findings:
(a) The findings of (a), using more nodes will increase the input data size and faster execution, is due to the fact that the computation are performed parallely in more nodes with more processor and memory. Ideally, by parallelizing a computation, the time complexity is divided by the number of processor, *np*. Hence, the execution should be *np* faster. On Fig. 12 and 13, and data size = 5 Gb, however, it can be seen that the execution on nodes = 14 is approximately only 5 times compared to node = 1. In the Hadoop cluster environment, this is due to network cost and the complexity of shuffle process as discussed in Subsection IV.B.
(b) By using larger HDFS block size, less data will be sent to the network in the Hadoop shuffling stage (as more objects are computed locally), hence it reduces the overhead of data communication and increases overall execution speed. (In the shuffling process, the smaller the block size, the more grouping and sorting computation are performed for distributing the outputs of Mapper to the corresponding destination Reducers.)
(c) The cause has been discussed in Subsection IV.B. But, here, we find that the influence of *nPar* also depends on the block size. To obtain more facts, additional plots for observing the influence of block size towards *nPar* have been created. The sample plot is selected for node = 14 and is depicted on Fig. 14. It can be observed that using *nPar* = 9 and 14, the fastest is execution is achieved with block size of 64 Mb. The reason is discussed in item (b) as stated above.

The second series of experiments are performed by repeating all of the experiments discussed above but using *distType* = single (the complexity of computation using single and complete distance is the same, so here only single distance is evaluated). The results are presented on Appendix B. As discussed on the appendix, the execution is slower and BDRT-ParAgglo can handle smaller input dataset. However, it shows similar patterns compared to the results using *distType* = means, which are: adding more nodes and using larger block size will speed up the computation.

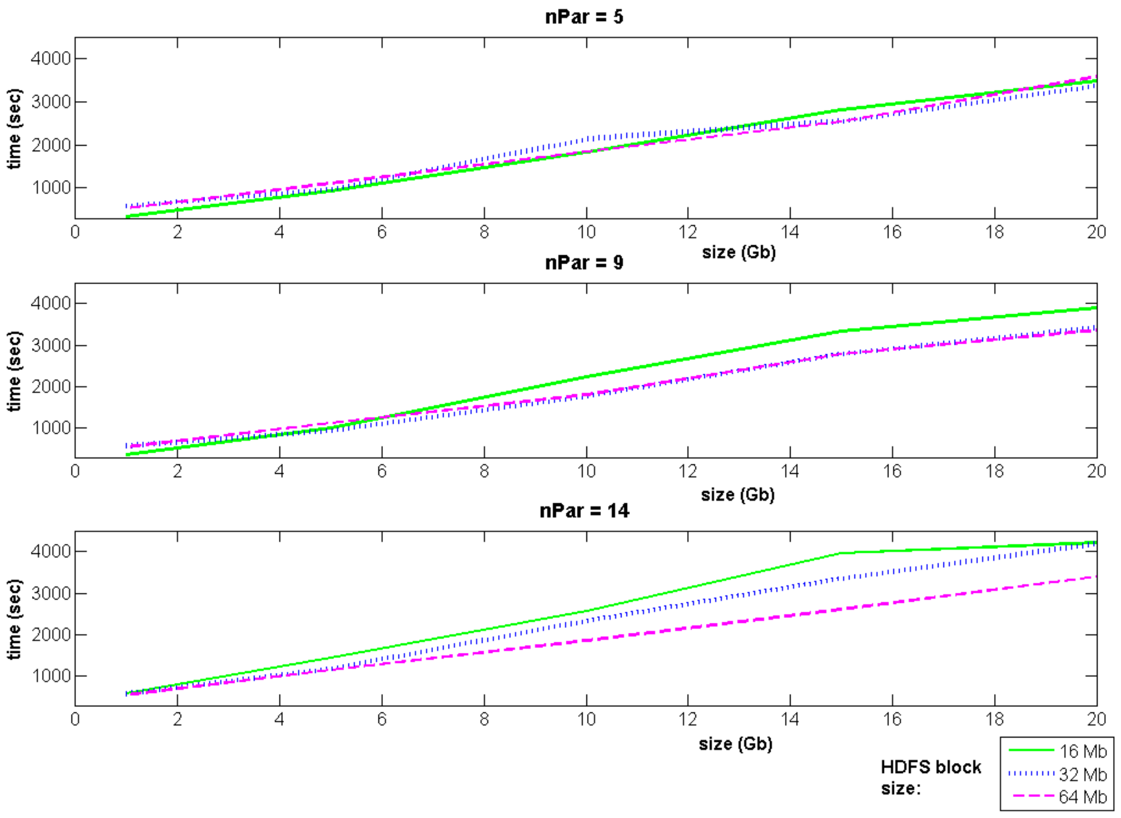


Fig. 14. Time execution on Hadoop cluster with 14 slave nodes.

E. Experiment Conclusions

The overall conclusion of experiments discussed in Subsection IV.A, B. C and D are excerpted as follows:

(1) The influence of BDRT-ParAgglo variables: (a) The counts of patterns created are affected by the value of *maxObj* (larger *maxObj* – fewer patterns), *nPar* (larger *nPar* –more patterns), *distType* (descending order: means-single/complete) and *co* (smaller *co* – more patterns); (b) On a specific configured Hadoop cluster, the larger value of *maxObj* or *nPar*, the slower the execution; (c) The size of big data that can be processed on a specific configured Hadoop cluster is influenced by the selection of *maxObj* and *distType* values.

(2) The influence of Hadoop cluster configuration: (a) Execution times are affected by number of slave nodes and HDFS block size; (b) Adding slave nodes will increase the size of the data that can be processed.

(3) The main objective of partitioning the input dataset (using *nPar* value in Mapper) is to randomize the objects fed into Reducer. Hence, the larger the *nPar* value, the more randomized the data partitions received by Reducer that will produce patterns that are less biased towards the object order. However, the cost is the execution time, which can be reduced by using larger HDFS block size.

F. Evaluating Towards Constraints

As discussed in Subsection II.E, data stream mining technique has to satisfy four constraints. Based on the experiment results and their analysis, Table VI shows the results of the proposed techniques evaluation towards the four constraints.

TABLE VI. THE RESULTS OF CONSTRAINTS EVALUATION

Constraint	Comply?	Rationale
Single-pass	Yes	BDRT-ParAgglo reads and processes (to generate patterns) a dataset once only.
Real-time response	Somewhat	The variables of *maxObj* and *nPar*, HDFS block size and number of slave nodes affect the BDRT-ParAgglo execution speed. Thus, real-time response can be approached by selecting the proper variables' values and Hadoop configurations.
Bounded memory	Yes	Achieved by adopting *landmark* or *sliding window* and assigning a value of *maxObj* that fit into slave node memory.
Concept-drift detection	Yes	The dataset being processed or discovered patterns change over time.

G. Open Discussion: Measuring Clusters Quality

As discussed in [5], there are two approach for measuring clustering quality, which are extrinsic and intrinsic methods. Extrinsic methods compare the cluster results against the group truth and measure. If the ground truth is unavailable, ones can use intrinsic methods, which evaluate the goodness of a clustering by considering how well the clusters are separated. Generally, group truth for big data is unavailable, so only intrinsic methods can be adopted.

The silhouette coefficient is a measure of intrinsic methods, which is described as follows: For a data set, *D*, of *n* objects, *D* is partitioned into *k* clusters, C_1, ….,C_k. For each object ***o*** ϵ *D*, *a(**o**)* is the average distance between ***o*** and all other objects in the cluster to which ***o*** belongs. Similarly, *b(**o**)* is the minimum average distance from ***o*** to all clusters to which ***o*** does not belong. The silhouette coefficient of ***o*** is then defined as:

$$s(o) = \frac{b(o) - a(o)}{\max\{a(o), b(o)\}}$$

If *s(o)* < 0, it indicates that ***o*** is wrongly clustered. The closer the value of *s(o)* to 1, the better ***o*** is clustered. To measure the quality of a clustering, ones may use the average silhouette coefficient values of all objects in the data set. The larger the value, the better the quality of clustering.

The complexity of computing all of *s(o)* values is approximately $O(kn^2m)$, where *k* = count of clusters, *n* = count of objects and *m* = count of object attributes. Then, the complexity to compute the average of all *s(o)* values would be *O(n)*. Thus, including functions in the MapReduce program to compute every *s(o)* and the average will slow down the computation and require lots of memory space in the slave nodes (for storing the objects, distances, objects' cluster membership and all of *s(o)* values). For these reasons, we have not implemented this function.

Ideally, a function for measuring the quality of clusters is provided such that before patterns are computed, ones can measure the quality of the formed clusters (in Algorithm 4, it should be called after line 8, form clusters using *co*). Thus, in this research, this remains unresolved yet.

As discussed in [5] and [18] agglomerative clustering algorithm has the advantage that it derives more meaningful cluster structures (compared to other approaches) as illustrated on Fig. 8. But its limitations are high complexity (or not suitable for high dimensional data) and sensitive to the order of the data records. In our proposed techniques, the second limitation has been resolved by randomly partitioning the objects (see Subsection IV.E). As a consequence of the first limitation, BDRT-ParAgglo is suitable for reducing big data having small to medium dimension (number of attributes).

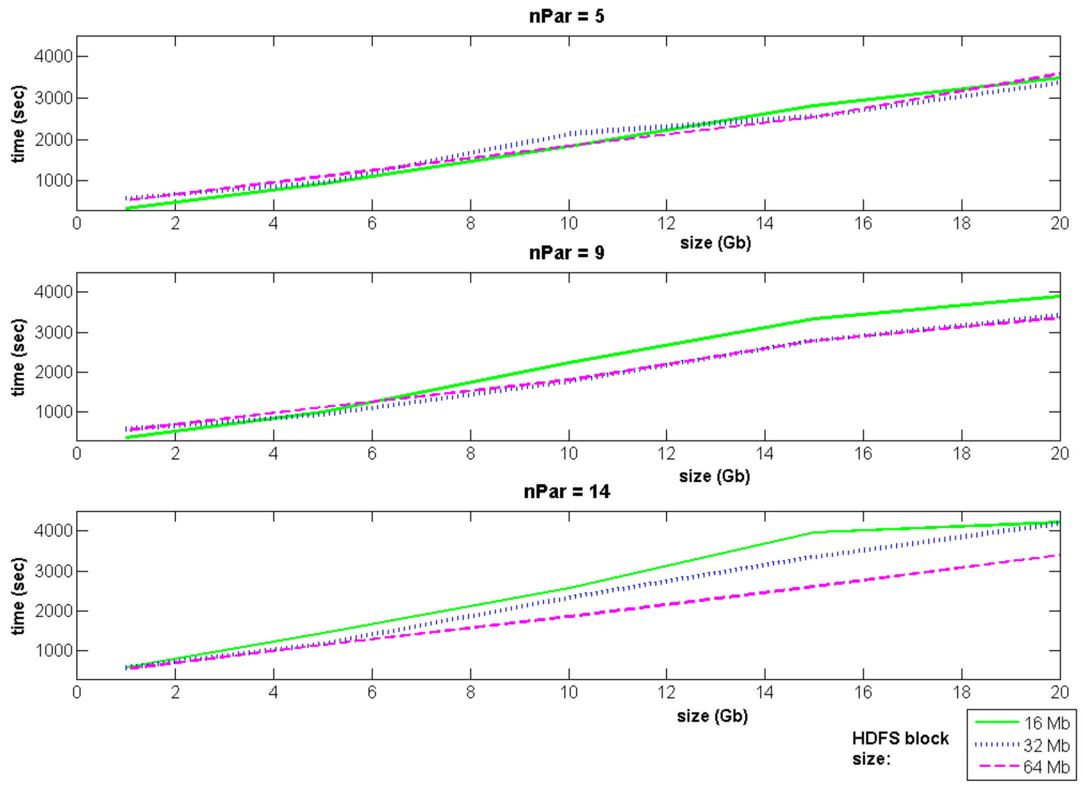


Fig. 14. Time execution on Hadoop cluster with 14 slave nodes.

V. Conclusion

A parallel method for reducing big data based on hierarchical clustering algorithm for Hadoop environment has been developed. In essence, Map preprocesses and randomly partitions the data, whereas Reduce computes the cluster patterns from each data partition. The components of the pattern can be selected from object counts, the attribute average, minimum, maximum and deviation value of each attribute in every cluster.

The variables of BDRT-ParAgglo as well as Hadoop cluster configuration may influence the execution time and the size of big data that can be processed. Hence, the variable values and configuration should be selected such that the performance is the best.

Through the experiments using low specification of computers as master and slave nodes, it has been shown that up to 20 Gb of big data can be processed with certain speed. Those experiments have proved that BDRT-ParAgglo can be implemented on Hadoop clusters configured using just commodity computers. By improving the hardware specifications (increasing memory and speed of processors), the size of data input that can be handled will be larger and the speed will also be increased.

Further research is suggested as follows:

(a) To increase the execution speed and addressing the complex shuffle process problem, the technique proposed in [20] for improving the Hadoop configuration, may be studied and, if found as the solution, can be adopted. Another option is: Creating HDFS blocks from preprocessed data that contains random objects, then Reducer is designed to form dendrogram without involving shuffling process.

(b) Designing and implementing functions to measure clustering quality suitable to handle big data in the distributed systems.

(c) By adopting the cloud clustering technique discussed in [21], this technique can further be enhanced such that it works in cloud platform.

Appendix

A. Experiment Results using Distance Type = Means and HDFS block size = 16 Mb

Fig. A.1 shows that: (a) By adding slave nodes, bigger size of dataset can be processed: Using single node, only 5 Gb of dataset can be processed (using *nPar* = 5 only), while using 5 nodes, it is up to 15 Gb and using 10 and 14 nodes, it is up to 20 Gb; (b) The more slave nodes, the faster the execution time; (c) The larger value of data partition (*nPar*), the more execution time needed.

B. Experiment Results using Distance Type = Single

Compared to using distance type = *means*, when using distance type = *single*, the proposed algorithm can only handle smaller size of dataset. An example of the experiment results are depicted on Fig. B.1. The broken charts mean that the execution were failed due to lack of memory (in the slave nodes). This is because we implement the function for finding the shortest distance using recursive functions (in a class method) that occupy large space of memory during its computation (involving large size of objects). By comparing Fig. B.1 with Fig. 12 and 13, it can also be noted that the execution with single distance is slower. This is inline with its complexity that is discussed in Subsection III.B.

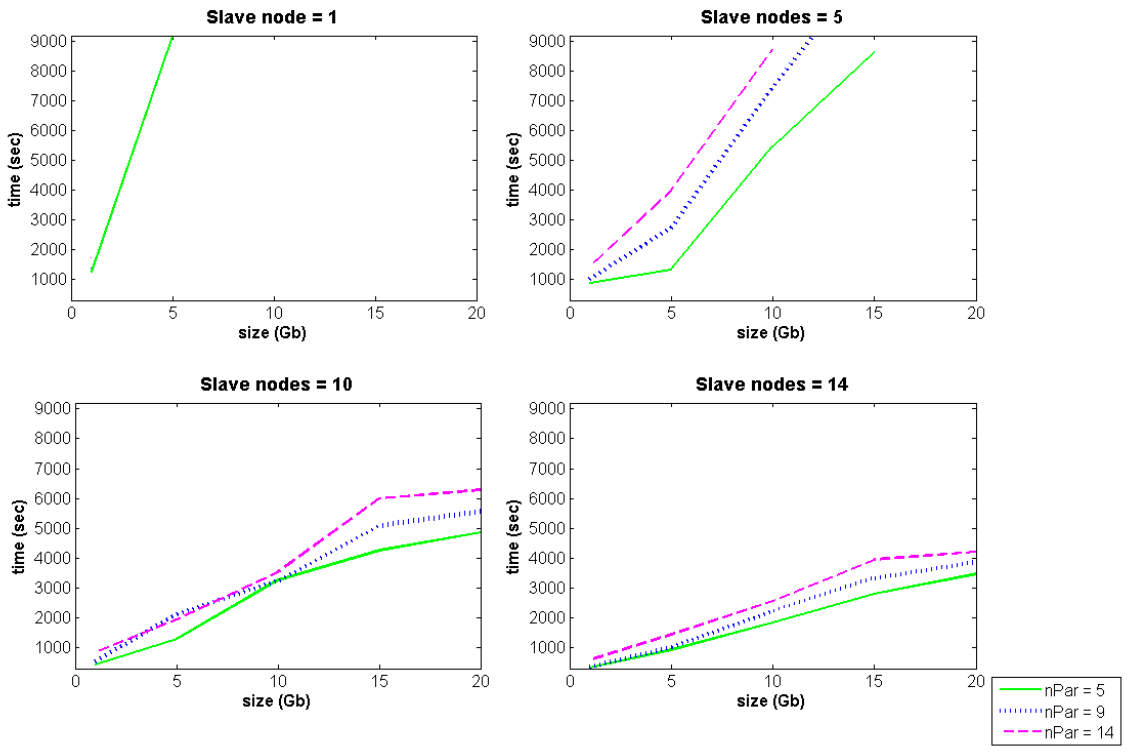


Fig. A.1. Plot of execution time for distance type = means, HDFS block size = 16 Mb

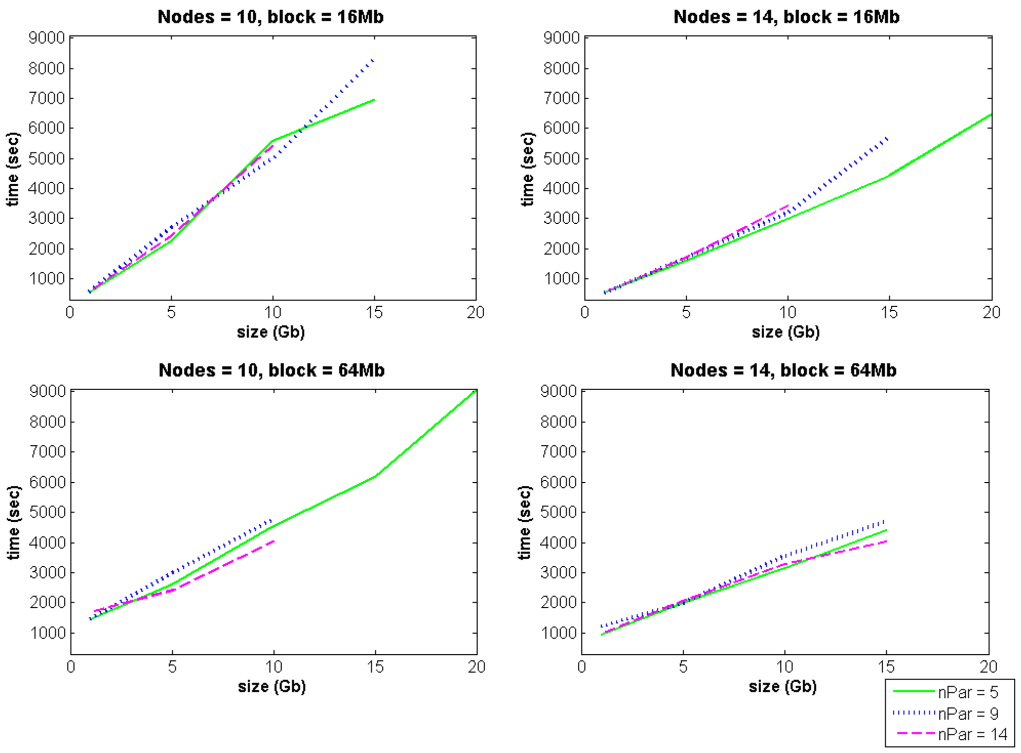


Fig. B.1. Plot of execution time for distance type = single.

C. Example of Proposed Technique Applications

The results of data reduction can be used for: (a) Direct applications: Histogram analysis and outlier detection; (b) Indirect applications: As the input of data mining techniques that can process the sub-cluster patterns. In this experiment, we intend to show the example of the direct application using histogram analysis. The data used is the one discussed in Subsection IV.A.

Table C.1 depicts some example of the histogram of the reduced dataset of household energy consumption from experiments in Subsection IV.A using *Npar* = 20, *dType* = single, *maxObj* = 50 and *co* = 1.5. It can be seen that there are five patterns having object members 46-50 (patterns with the most object members). To find the insights of those five patterns, Table C.II shows the values of each pattern (presented here for example). The interpretation patterns depicted in Table C.II is: The household mostly use electricity power in sub-meter 2 and 3 in the morning or late afternoon on Tuesday, Wednesday (the most) or Thursday.

Other histogram component having large object members can also be examined to find the information or insights. In these experiments, we find that the information (that can be dig from the data reductions) is more detailed compared to our previous results published in [10], where patterns were obtained from parallel k-Means clustering algorithm.

TABLE C.I. EXAMPLE OF PATTERNS HISTOGRAM

No	#Objects in a Pattern	Count
1	1-5	590,611
2	6-10	34,301
3	11-15	10,285
4	16-20	3,915
5	21-25	1,541
6	26-30	656
7	31-35	224
8	36-40	75
9	41-45	29
10	46-50 (47-48)	5
	Total	641,642

TABLE C.II. FIVE PATTERNS WITH THE LARGEST OBJECT MEMBER

CP	#Object	Minimum attribute value					Maximum attribute value					Cluster centroids					
		D	H	M-1	M-2	M-3	D	H	M-1	M-2	M-3	D	H	M-1	M-2	M-3
1	47	4	0	0	0	0	4	1	0	2	0	4	6.45	0	0.13	0.00
2	47	3	8	0	0	0	3	2	0	2	1	3	15.2	0	0.19	0.55
3	47	2	2	0	0	0	2	1	0	1	1	2	7.79	0	0.17	0.60
4	48	3	0	0	0	0	3	1	0	2	1	3	7.71	0	0.27	0.77
5	48	3	6	0	0	0	3	1	0	2	1	3	13.5	0	0.21	0.75

Note: Attribute of each pattern: D = day number, H = hour, M-1 = meter-1, M-2 = meter-2, M-3 = meter-3

REFERENCES

[1] A. Holmes, *Hadoop in Practice*, Manning Publications Co., USA, 2012.

[2] M. H. U. Rehman, C. S. Liew, A. Abbas, P. P. Jayaraman, T. Y. Wah, S. U. Khan, "Big data reduction methods: a survey", *Data Science and Engineering*, pp. 1-20, December 2016.

[3] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A.S. Netto, R. Buyya, "Big data computing and clouds: trends and future directions", *J. Parallel Distrib. Comput*., vol. 79, no. 80, pp. 3–15, 2015.

[4] V. S. Moertini, L. Venica, "Enhancing parallel k-means using map reduce for discovering knowledge from big data", *Proc. of. 2016 IEEE Intl. Conf. on Cloud Computing and Big Data Analysis (ICCCBDA 2016)*, Chengdu China, 4-7 July 2016, pp. 81-87.

[5] J. Han, M. Kamber and J. Pei, *Data Mining Concepts and Techniques* 3rd Ed., The Morgan Kaufmann Publ., USA, 2012.

[6] I. Triguero, D. Peralta, J. Bacardit, S. García, F. Herrera, "MRPR: A MapReduce solution for prototype reduction in big data classification", *Neurocomputing*, vol. 150, 2015, pp. 331–345

[7] C. Lam, *Hadoop in Action*, Manning Publ., USA, 2010

[8] C. Jin, M. M. A. Patwary, A. Agrawal, W. Hendrix, W. Liao, A. Choudhary, "Disc: a distributed single-linkage hierarchical clustering algorithm using mapreduce", *Proc. of the International SC Workshop on Data Intensive Computing in the Clouds (DataCloud)*, 2013.

[9] E. Sammer, *Hadoop Operations*, O'Reilly Media, Inc., USA, 2012.

[10] V. S. Moertini and L. Venica, "Parallel k-means for big data: on enhancing its cluster metrics and patterns", *Journal of Theoretical and Applied Information Technology*, vol. 95, no. 8, 2017. Pp. 1844-1857.

[11] W. Zhao, H. Ma and Q. He, "Parallel k-means clustering based on mapreduce", *CloudCom 2009*, LNCS 5931, pp. 674–679, Berlin Heidelberg: Springer-Verlag, 2009.

[12] L. Ma, L. Gu, B. Li, Y. Ma and J. Wang, "An improved k-means algorithm based on mapreduce and grid", *International Journal of Grid Distribution Computing*, vol. 8, no. 1, pp. 189-200, 2015.

[13] J. M. Mathew and J. Joseph, "Parallel implementation of fuzzy k-means algorithm using hadoop", *International Journal of Advanced Research in Computer Science & Technology*, vol. 4, issue 2, Apr. - Jun. 2016.

[14] L. Srinivasulu, A. V. Reddy, V. S. G. Akula, "Improving the scalability and efficiency of k-medoids by map reduce", *International Journal of Engineering and Applied Sciences*, vol. 2, issue 4, April 2015.

[15] X. Fu, S. Hu and Y. Wang, "Research of parallel DBSCAN clustering algorithm based on mapreduce", *International Journal of Database Theory and Application*, vol. 7, no. 3, pp. 41-48, 2014.

[16] Vadivel and Raghunath, "Enhancing map-reduce framework for bigdata with hierarchical clustering", *International Journal of Innovative Research in Computer and Communication Engineering*, vol. 2, special issue 1, March 2014.

[17] Y. Zhang, J. Z. Wang and J. Li, "Parallel Massive Clustering of Discrete Distributions", *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 11, no. 4, article 40, pp. 49:1- 49:24, 2015.

[18] Nguyen H. L., Woon Y. K., Wee K. N., "A Survey on Data Stream Clustering and Classification", *Knowledge Information System Journal Springer – Verlag*, Vol. 45, pp. 535–569, 2015

[19] K. Tsiptsis and A. Chorianopoulos, *Data Mining Techniques in CRM: Inside Customer Segmentation*, John Wiley and Sons, L., UK, 2009.

[20] J. Wang, M. Qiu, B. Guo, Z. Zong, "Phase–Reconfigurable Shuffle Optimization for Hadoop MapReduce", *IEEE Transactions on Cloud Computing*, vol. PP, Issue 99, 2015.

[21] X. Zhong, G. Yang, L. Li and L. Zhong, "Clustering and correlation based collaborative filtering algorithm for cloud platform", *IAENG International Journal of Computer Science*, vol. 43, no. 1, pp. 108-114, 2016.