TECHNICAL REPORT

No. CCLS-11-04

Title: PARABLE: A PArallel RAndom-partition Based HierarchicaL ClustEring Algorithm for the MapReduce Framework

Authors: Shen Wang and Haimonti Dutta

PARABLE: A PArallel RAndom-partition Based HierarchicaL ClustEring Algorithm for the MapReduce Framework

Shen Wang and Haimonti Dutta
The Center for Computational Learning Systems (CCLS),
Columbia University, New York, NY 10115.
{FelixSWang, haimonti}@ccls.columbia.edu

Abstract. Large datasets, of the order of peta- and tera- bytes, are becoming prevalent in many scientific domains including astronomy, physical sciences, bioinformatics and medicine. To effectively store, query and analyze these gigantic repositories, parallel and distributed architectures have become popular. Apache Hadoop is one such framework for supporting data-intensive applications. It provides an open source implementation of the MapReduce programming paradigm which can be used to build scalable algorithms for pattern analysis and data mining. In this paper, we present a PArallel, RAndom-partition Based hierarchicaL clustEring algorithm (PARABLE) for the MapReduce framework. It proceeds in two main steps – local hierarchical clustering on nodes using mappers and reducers and integration of results by a novel dendrogram alignment technique. Empirical results on two large data sets (High Energy Particle Physics and Intrusion Detection) from the KDD-Cup competition on a large cluster indicates that significant scalability benefits can be obtained by using the parallel hierarchical clustering algorithm in addition to maintaining good cluster quality.

1 Introduction

Large datasets are ubiquitous and efficient knowledge discovery from them usually involves usage of significant compute power, storage capacity and high speed communication. Several astronomy and physical science projects such as CERN's[1] Large Hadron Collider (LHC) [1], Sloan Digital Sky Survey (SDSS[2], bioinformatics projects, gene and protein archives[3]), meteorological and environmental surveys[4] are already producing peta- and tera-bytes of data which requires to be stored, queried and analyzed.

To meet the challenges posed by such large datasets, scalable and efficient data mining algorithms need to be designed. MapReduce ([2]) is a programming

[1] Conseil Europen pour la Recherche Nuclaire - European Organization for Nuclear Research
[2] `http://www.sdss.org`
[3] `http://www.rcsb.org/pdb/Welcome.do` and `http://www.expasy.org/sprot/`
[4] `http://www.ncdc.noaa.gov/oa/wmo/wdcamet.html`

paradigm which allows large datasets to be broken down into small chunks and processed in parallel on multiple cluster nodes. In this paper, we present a parallel, random-partition based hierarchical clustering algorithm for the MapReduce framework. The algorithm contains two main components – a divide-and-conquer phase and a global integration phase. In the divide-and-conquer phase, the data is randomly split into several smaller partitions by the mapper. Each partition is forwarded to a reducer on which a sequential hierarchical clustering algorithm is run. The clustering results from the reducer are stored on local machines. In the global integration phase, the local clustering of all the data points are used to suggest a final solution. A key contribution of our work is *dendrogram alignment* which provides a mechanism to merge local dendrograms consistently to form a global one. This alignment also enables comparison of the performance of the parallel hierarchical algorithm against centralized counterparts. To the best of our knowledge, this is the first work that describes a random-partition based parallel hierarchical clustering algorithm for the MapReduce framework that also provides a mechanism to obtain a global dendrogram from the parallel implementation.

This paper is organized as follows: Section 2 presents related literature; Section 3 provides an overview of Apache Hadoop and the MapReduce framework; Section 4 reviews agglomerative hierarchical clustering; Section 5 introduces the **PA**rallel **RA**ndom-partition **B**ased Hierarchica**L** Clust**E**ring (PARABLE) Algorithm and analyzes its complexity; Section 6 presents metrics for evaluation; Section 7 presents empirical results on clustering two datasets on a large cluster and Section 8 concludes the paper.

2 Related Work

2.1 Clustering using the MapReduce framework

Several MapReduce-based clustering algorithms have been proposed – Zhao et. al. [3] present a parallel K-means algorithm where the data is partitioned among multiple processors which can read the previous iteration's cluster centers and assign instances to respective clusters. Ngazimbi [4] provides a case-study of clustering Netflix movie data using K-Means, Greedy Agglomerative and Expectation Maximization algorithms in the MapReduce framework. Sun et. al. [5] use hierarchical clustering for grouping internet users by mining a huge volume of web-access logs. Their method uses two optimization techniques – batch updating to reduce computational time and communication costs among cluster nodes and co-occurence based feature selection to decrease the dimension of the feature vectors and eliminate noisy features. Papadimitriou et. al. [6] implemented a distributed Co-Clustering algorithm called DisCo for applications in text mining, collaborative filtering, bio-informatics and graph mining. MapReduce is used both for distributed data pre-processing and clustering.

2.2 Parallel and Distributed Hierarchical Clustering Algorithms

Hierarchical clustering has been studied extensively in distributed and parallel data mining communities. Rasmussen et. al. [7] provide a parallel implementation of the single link and minimum variance hierarchical clustering algorithms on SIMD processors. Their algorithms do not decrease the $O(n^2)$ time requirement of the serial implementation, but a constant speed-up factor is obtained. Olson [8] describes an $O(n)$ time algorithm for hierarchical clustering using single, average and complete link and centroid, median and minimum variance metrics on an n node Concurrent Read, Concurrent Write Parallel Random Access Machine (CRCW PRAM) and an $O(nlogn)$ algorithm of a $\frac{n}{logn}$ butterfly network or trees. Zhaopeng, Kenli, Degui and Lei [9] propose a parallel Parallel Random Access Machine (EREW) deterministic algorithm for hierarchical clustering, which is based on complete graph and Euclidean minimum spanning tree algorithms. It can cluster n objects with $O(p)$ processors in $O(n^2/p)$ time. Sanguthevar [10] considered hierarchical clustering algorithm on both the PRAM and the Arrays with Reconfigurable Optical Buses (AROB) models and give algorithms with worst-case guarantees as well as algorithms with expected performance better than existing algorithm. performance. Manoranjan, Simona and Peter [11] presentd pPOP, the parallel version of POP, that is implemented on a shared memory multiprocessor architecture. Extensive theoretical analysis and experimental results are presented and show that pPOP gives close to linear speedup and outperforms the existing parallel algorithms signicantly both in CPU time and memory requirements. Tian, Raghu and Miron [12] develop the BIRCH clustering method for very large databases. To ensure accuracy and efficiency, BIRCH defines clustering feature (CF) which summarizes the information in a cluster. The dataset can then be represented by a B+ tree, called CF tree, which supports insertion of new data points. Sudipto, Rajeev and Kyuseok [13] developed the ROCK clustering algorithm where a similarity function measures closeness between data points and *neighbors* are defined as a pair of points that have similarity greater than a threshold. If two points share many neighbors (relative to the expected number of neighbors), they are put into one single cluster. George, Eui-Hong and Vipin [14] develop the CHAMELEON clustering algorithm that measures the similarity of two clusters based on a dynamic model that facilitates discovery of natural and homogeneous clusters.

Johnson and Kargupta [15] develop a *distributed* hierarchical clustering algorithm on heterogeneously distributed data. Samatova et. al. [16] develop a method for merging hierarchical clusterings from homogeneously distributed, real-valued data. Each site produces a dendogram based on local data, then transmits it to a central site. To reduce communication costs,they do not send a complete description of each cluster in a dendogram. Instead an approximation of each cluster is sent consisting of various descriptive statistics *e.g.* number of points in the cluster, average square Euclidean distance from each point in the cluster to the centroid. Hui, Jun, Li and Yan [17] proved an algorithm for combining Map Reduce framework and the neuron initialization method of VPSOM (vector pressing Self Organizing Model) algorithm. This algorithm is used for text clustering. However, in their paper, no detail about how the local results on each computing site are merged together to help make a global decision, which is not a trivial task at all.

3 MapReduce and Hadoop Framework

Apache Hadoop ([18]) inspired by Google Map-Reduce ([2]) and Google File System ([19]), is a framework for supporting data intensive applications on a cluster. It has an open source MapReduce implementation that has been used both by industry and academia to develop tools and architectures for supporting Data Intensive Scalable Computing ([20]) using Hadoop.

MapReduce is a distributed computing framework for large datasets and has two computation phases – `map` and `reduce`. In the `map` phase, a dataset is partitioned into disjoint parts and distributed to workers called `mappers`. The mappers implement compute-intensive tasks (such as clustering) on local data. The power of MapReduce stems from the fact that many map tasks can run in parallel. The output of the `map` phase is of the form $< key, value >$ pairs which are passed to the second phase of MapReduce called the `reduce` phase. The workers in reduce phase (called reducers) then partition, process and sort the $< key, value >$ pairs received from the Map phase according to the *key* value and make final output. For a complex computation task, several MapReduce phase pairs may be involved.
The architecture that we use for developing the Parallel Hierarchical Clustering Algorithm comprises two main parts: (1) **Data Storage**, using the Hadoop Distributed File System (HDFS). (2) **Computation**, utilizing MapReduce programming paradigm to meet the computation needs of the clustering algorithm. Figure 1 shows the above components in our cluster. Specifically, these include: **(1) The Hadoop Distributed File System (HDFS) Namespace**: The Namenode maintains the file system namespace and records any changes made to it. It also keeps track of the number of replicas of a file that should be maintained in the HDFS typically called the replication factor. **(2) The Master / Slaves of HDFS**: A master server manages the file system namespace and regulates access to files by clients. In addition, there are a number of HDFS Slaves, usually one per node, which manage the data associated with that node. They serve read and write requests from the users and are also responsible for block creation, deletion and replication upon instruction from the NameNode. **(3) Data Access to MapReduce Master**: The HDFS file system will be accessed by a MapReduce (MR) master. The input files to the MR Master can be processed in parallel by different machines in cluster. **(4) The Map Assigner on MapReduce Master**: It stores data structures such as the current state (idle, in-progress or completed) of each map task in the cluster. It is also responsible for pinging the map-workers occasionally. If no response is received from the worker, it assumes that the process has failed and re-schedules the job. **(5) The Intermediate processor**: The intermediate $< key, value >$ pairs produced by map function are buffered in the local memory of machines. This information

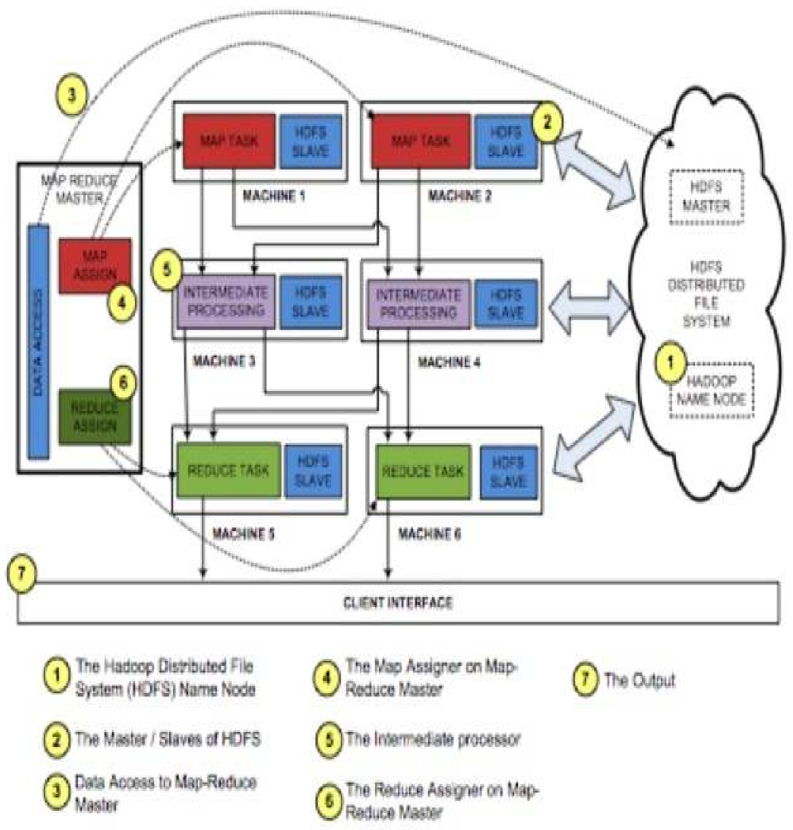


Fig. 1. The architecture of a Cluster illustrating the Map, Reduce and Intermediate operations along with the Hadoop Distributed File System (HDFS).

is sent to the MR Master which then informs the Reduce Assigner. **(6) The Reduce Assigner on MapReduce Master**: This takes in the location of the intermediate files produced from a Map operation and assigns reduce jobs to the respective machines. **(7) The Output**: The output of a Reduce function is appended to a final output file. When all the map and reduce tasks are over, the MR Master wakes up the user program.

Having described the MapReduce and Hadoop frameworks, we proceed in the next sections to provide a brief review of agglomerative hierarchical clustering and our PARABLE algorithm.

4 A Brief Review Of Agglomerative Hierarchical Clustering

The agglomerative hierarchical clustering algorithm proceeds in four steps: (1) At the start of the algorithm, each data point is taken as an individual cluster and the distance between every pair of clusters is calculated. (2) The two clusters with the shortest distance are merged into one single cluster. (3) The distance between the newly merged cluster and the other clusters are calculated and the distance matrix is updated accordingly. (4) If more than one cluster still exists, goto step 2.

There are a number of methods for determining the distance between clusters and the metrics can be classified as follows [21]: **1. Graph based methods:** These methods use a graph of points in each cluster to determine inter-cluster distance. (a) **Single link:** The distance between any two clusters is the minimum distance between any two points such that one of the points is in each cluster. (b) **Average Link:** The distance between any two clusters is the average distance between each pair of points such that each pair has a point in each cluster. (c) **Complete Link:** The distance between any two clusters is the maximum distance between two points such that one of the points is in each of the cluster. **2. Geometric Methods:** These methods define a cluster center for each cluster and use them to determine the distance between clusters. (a) **Centroid:** The cluster center is the centroid of the points in the cluster and euclidean distance is used as distance between cluster centroids. (b) **Median:** The cluster centroid is the unweighted average of all the centers of the two clusters agglomerated to form it. (c) **Maximum Variance:** The distance between two clusters is the increase in sum of squared distances from each point to the center of the cluster caused by agglomerating the clusters.

In step 3, the time to compute the distance between each pair of points is $O(n^2)$ where n is the number of data points. In step 4, only the distances between the newly merged cluster and other clusters need to be calculated, so the time is $O(n)$ (for single link metric, the time cost is constant because the shortest distance between the new cluster and other clusters is just the shorter distance of the distances between two merged clusters and their nearest neighbors). Since each time two clusters merge, the number of remaining clusters decreases by 1, and there are at most n iterations. Therefore, the total time required by the algorithm to converge is $O(n^2)$.

During the merging step, only the nearest neighbor of each cluster needs to be examined, hence only the distance to the nearest cluster needs to be stored. The overall space complexity of agglomerative hierarchical clustering is $O(n)$.

5 The PARABLE Algorithm

The **PA**rallel **RA**ndom-partition **B**ased Hierarchica**L** clust**E**ring algorithm (PARABLE) randomly splits the large data set into several smaller partitions on the mapper. Each partition is distributed to a reducer on which the sequential hierarchical clustering algorithm is executed. After this *local clustering* step, the dendrograms obtained at each reducer are aligned together. The pseudo code for PARABLE is presented in Algorithm 1 and the dendrogram alignment is presented in Algorithm 2.

Algorithm 1 PArallel RAndom Partion-Based HierarchicaL ClustEring Algorithm (PARABLE)

```
1: INPUT: Number of data points: $N$, Number of partitions: $M$, Number of cycles: $T$
2: {On each mapper}
3: Calculate the number of data points in each partition. Let $S_m$ be the initial number of data points each $m_{th}$ partition can still accept. If $N mod M \neq 0$, the last partition takes the remainder.
4: Read the data from data file.
5: Each data is put in the value part of the $< key, value >$ pair.
6: A random integer partition index $i$ is picked from $1, 2, 3, ..., M$ and assigned to the key part of the $< key, value >$ pair. The probability of the partition index $m$ is picked proportional to $S_m$.
7: Flush the $< key, value >$ pair to reducer.
8: {On each reducer}
9: Collect the $< key, value >$ pair records fed by the mapper.
10: Perform Local clustering on the data. Due to the MapReduce design, each individual reduce task works only on all the records that share the same key. Therefore, the local clustering algorithm is indeed applied within each partition.
11: Write the dendrogram of the local clusters to Local Dendrogram File (LDF).
12: {Controller}
13: Read the local dendrograms from LDF.
14: Make a copy of the first dendrogram as the dendrogram template for alignment.
15: for each dendrogram except the first do
16:     align the dendrogram with the dendrogram template using the recursive algorithm described in algorithm 3.
17: end for
18: According to the cutting criterion, cut the template dendrogram into clusters and label them.
19: for each subroot C of the clusters in template dendrogram do
20:     for each dendrogram D do
21:         locate the branch in D that is aligned with C
22:         label all the data points on the leaves of D with label of C
23:     end for
24: end for
```

Algorithm 2 Recursive dendrogram aligning algorithm

```
1: function $align$(Dendrogram $D1$, Dendrogram $D2$)
2: if $similarity$($D1$.leftChild, $D2$.leftChild)+$similarity$($D1$.rightChild,$D2$.rightChild)
   $<$ $similarity$($D1$.rightChild,$D2$.leftChild)+$similarity$($D1$.leftChild,$D2$.rightChild)
   then
3:    Switch $D2$'s two children
4: end if
5: align($D1$.leftChild, $D2$.leftChild)
6: align($D1$.rightChild, $D2$.rightChild)
```

5.1 Technical Details

In this section, we discuss the MapReduce implementation of the local and global clustering steps. Assume that there are N data points divided into M partitions in each MapReduce phase.

Mapper task: Since the total number of data points is N and there are M partitions, the number of data points assigned in each partition is calculated as $n_p = N/M$ and passed as a parameter to the mapper. If N mod $M \neq 0$, then the last partition takes all the remaining points. Let $S_m, m = 1, 2, 3, \cdots, M$ be the number of data points the m^{th} partition can accept. The initial value of S_m is set to n_p. Each mapper takes a data point one at a time and provides it with a $< key, value >$ pair. The data is put as the *value* while a random integer i, picked from $1, 2, 3, ..., M$ is assigned as *key*. This is also the index of the partition the data point will be placed. The probability of the partition index m is proportional to S_m, which ensures that at the end, the number of data points in each partition is consistent with n and the work load of all the local clustering tasks is even. Finally, all the $< key, value >$ pair records are collected and sent to the reducers. The records are sorted and grouped according to *key* before being sent to the reducer. The design of MapReduce makes sure that all the records with the same *key* are sent to the same reducer.

Reducer task: In each reduce task, the reducer only deals with the $< key, value >$ pairs sharing the same *key*. It simply applies the sequential hierarchical clustering algorithm on the data it has. The locally generated dendrograms are stored for future use.

Dendrogram Alignment: In the MapReduce phase, each reducer generates a dendrogram. The dendrogram, a binary tree organized by linkage length, is built on a local subset of data. To obtain a global cluster assignment across all mappers and reducers, dendrogram integration needs to be implemented. Such an integration is non-trivial because insertion and deletion of a single data point changes the structure of the dendrogram. Our approach is to *align* them by "stacking" them one on top of another by using a recursive algorithm described in Algorithm 2. Example 1 provides an illustration of the technique.

Example 1: Consider two dendrograms a (Fig. 2 (a)) and b (Fig. 2 (b)) that need to be aligned. Assume a is the *template* dendrogram – this means dendrogram b is aligned to a and all structure changes will happen on b only. First, the

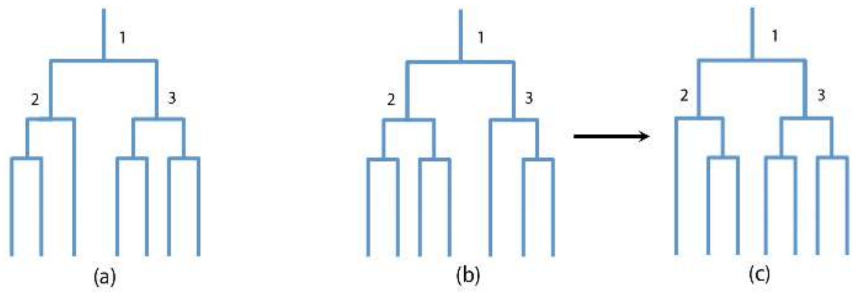


Fig. 2. An illustrative example for dendrogram alignment.

roots of these two dendrograms (nodes of depth 1) are aligned to each other. Then, nodes at depth 2 need to be aligned. There are two choices. The first choice is to align a_2 with b_2 and a_3 with b_3 while the other choice is opposite, which aligns a_2 with b_3 and a_3 with b_2. The decision is made by comparing $similarity(a_2, b_2) + similarity(a_3, b_3)$ with $similarity(a_2, b_3) + similarity(a_3, b_2)$ and taking the one with higher *similarity* value. In this case, we will find it more reasonable to align a_2 with b_3 and a_3 with b_2. Therefore, b_2 and b_3 are switched and dendrogram b is transformed to dendrogram c as shown in Fig. 2(c). Then, for each pair of nodes that have been aligned in two dendrograms, say a_2 and c_2, we repeat the same procedure to align c_2's two children with a_2's two children. This procedure is repeated recursively until it reaches a depth deep enough for labeling.

The description of a subtree in the dendrogram as well as the definition of the *similarity* function are very important in this procedure. We adapt the CF vector introduced by *BIRCH* [12] for this work. It is defined as $CF = (N, LS, SS)$ where N is the number of data points in the subtree, LS is the linear sum of the N data points, and SS is the square sum of the N data points.

Definition: Similarity$(\mathbf{a}, \mathbf{b})$ between two dendrograms in PARABLE is defined as

$$Similarity(a, b) = (LS_a \cdot LS_b + SS_a \cdot SS_b)/(N_a \cdot N_b). \tag{1}$$

The alignment procedure is reasonable when the dendrograms to be aligned have similar structure. This is the case in PARABLE since dendrograms collected from local clustering are generated from randomly sampled subsets of the whole dataset. If each subset contains many samples, it can be safely assumed that these dendrograms have similar structure, especially for the top levels. The structures of these dendrograms may differ more and more as the depth increases. However, assuming that the number of clusters is usually much smaller than the number data points, it suffices to ensure that the top levels of the dendrograms are aligned well.

Cluster Labeling: The cluster labeling comprises of two steps – the first step involves cutting the *template* dendrogram into subtrees, similar to what the sequential hierarchical clustering algorithm does during labeling. Each subtree is given a cluster label. Then for the root a_i of each subtree i, the algorithm finds out all the nodes in other dendrograms that were aligned with it in the alignment step. Each of these nodes is also a root of a subtree in its own dendrogram and the algorithm will label all the data points belonging to this subtree with the same cluster label as a_i. Intuitively, this is like "stacking" all the aligned dendrograms together with the template dendrogram being put on top. Then a knife is used to cut the template dendrogram into pieces. After the template dendrogram is cut, the knife does not stop but cuts all the way to the bottom of the stack. By doing this, the entire stack is cut into several smaller stacks, and data points in each small stack are given the same cluster label.

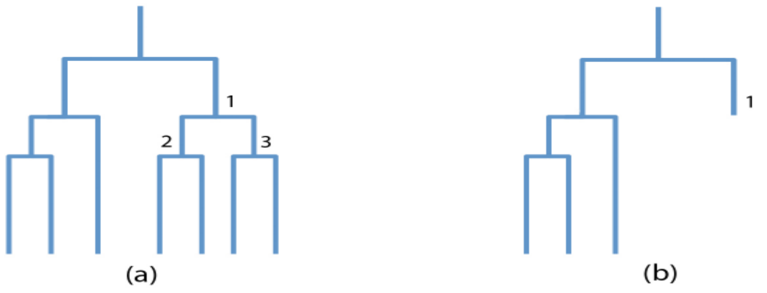


Fig. 3. An illustrative example for shallow leaf nodes boundary case of dendrogram alignment.

There is one boundary condition that needs to be addressed. As shown in Fig. 3, some of the dendrograms may have leaf nodes with very small depth. This is usually caused by outliers in the dataset. If the roots of subtrees obtained from dendrogram cutting are below these leaf nodes, then these leaf nodes are not going to be labeled in the cluster labeling step since the nodes on the template dendrogram, with which these leaf nodes were aligned, are not given any cluster label. For example, in Fig. 3, b_1 is a leaf node which is aligned with a_1. If later on, it is decided that a_2 and a_3 are labeled with different cluster indices, then the label of b_1 is undecided. In this case, we calculate the centroids of each cluster according to all the labeled data points. Then, all these shallow leaf nodes are revisited. Each of them will be assigned the label of the cluster which has the closest centroid to it.

5.2 Analysis

The dataset has M partitions with approximately $\frac{N}{M}$ data points in each partition. Assume there are n mappers/reducers involved in the computation, then it takes the mappers $O(\frac{N}{n})$ time to separate the dataset into partitions and the reducers $O((\frac{N}{M})^2 \cdot \frac{M}{n})$ time to implement the local clustering. For the dendrogram alignment, the total number of the nodes that need to be aligned is at most $2(N-1)$ – each node in all the dendrograms is visited at most once during alignment. The time required for dendrogram alignment is $O(N)$. Thus, the total time complexity of PARABLE is $O(\frac{N^2}{Mn}) + O(N)$. For the message complexity, each data point is passed from mappers to reducers once, which yields $O(N)$ message complexity. After the reducing stage finishes, the dendrograms are collected by the central site for dendrogram alignments. There are $2(N-1)$ nodes of dendrograms passed, which contains the CF vector as well as the node wrappers. The CF has dimension a little larger than 2 times the dimension of the data while the wrappers takes constant space for each node. Therefore, the total message complexity of PARABLE is $O(N)$.

6 Metrics for Evaluation

The performance of PARABLE is evaluated with respect to a centralized hierarchical clustering algorithm. The implementation of the centralized algorithm assumes that the data sets partitioned among mappers can be collected and stored in one single repository[5] and a hierarchical clustering algorithm can be run on the entire data set at once. To compare performance of the parallel and centralized algorithms, one could "inspect" the clusters found in either case and locate interesting patterns – the composition of clusters from one dendrogram can be compared with those of another. However, manual inspection is labor and time intensive. Fowlkes and Mallows [22] propose a method of comparing two hierarchical clusters by giving a numerical measure to the degree of similarity between then. We adapt this metric for our work. Assume that there are two dendrograms D_1 and D_2 built on n data points by running PARABLE and the centralized algorithm. The dendrograms can be cut into $k = 2, \cdots n-1$ clusters which corresponds to setting a value of the cluster strength and determining the cluster assignments of the dendrogram at that strength. For each value of k, we may label clusters for D_1 and D_2 arbitrarily from 1 to k and form the matrix $M = [m_{ij}], i = 1, 2, \cdots k, j = 1, 2, \cdots k$ where m_{ij} is the number of elements in common between the i^{th} cluster of D_1 and j^{th} cluster of D_2. The similarity of the dendrograms can then be represented by $B_k = \frac{T_k}{\sqrt{P_k Q_k}}$, where $T_k = \sum_{i=1}^{k} \sum_{j=1}^{k} m_{ij}^2 - n$; let $m_{i*} = \sum_{j=1}^{k} m_{ij}$, $m_{*j} = \sum_{i=1}^{k} m_{ij}$ and $m_{**} = n = \sum_{i=1}^{k} \sum_{j=1}^{k} m_{ij}$. Then, we define $P_k = \sum i = 1^k m_{i*}^2 - n$ and $Q_k = \sum j = 1^k m_{*j}^2 - n$. B_k is calculated for every value of k and an intuition of the similarity between the clusters is obtained by examining the plot of B_k versus k; usually $0 \leq B_k \leq 1$ for each k – when $B_k = 1$ the two dendrograms are exactly identical and when $B_k = 0$ every pair of objects that are assigned to the same cluster in D_1 are assigned to a different one in D_2.

[5] This is a strong assumption since in most cases one may end up with memory restrictions. The hypothetical scenario is created for evaluation purposes only.

7 Empirical Evaluation

In this section, we demonstrate the performance of the PARABLE algorithm on real world datasets. The first data set is the high energy particle dataset as presented in the ACM KDD Cup competition[6] in 2004. It has 50,000 data points, each with 78 dimensions. The second dataset is the network intrusion detection dataset provided in the ACM KDD Cup competition in 1999. The full dataset contains over 4 million data points with 41 dimensions. We only use part of it since it is already large enough to show the performance improvement of PARABLE. The in-house cluster available to us for experimentation had 25 nodes, 8-core processors each with 24 GB RAM, 1 TB RAID connected via a fiber channel. Apache Hadoop (version 0.21.0) was set-up on this cluster.

7.1 Clustering quality

PARABLE is an approximate hierarchical clustering algorithm since MapReduce does not allow communication between nodes during map and reduce cycle. In this section, we compare the PARABLE hierarchical clustering result with the result of sequential hierarchical clustering algorithm with the evaluation metrics mentioned in section 6.

Fig. 4 show the plots of B_k vs. k, in which k is the number of clusters that the dendrogram of sequential hierarchical algorithm and the aligned dendrograms of PARABLE are cut into. The plots illustrate that the alignment is quite effective since the value of B_k is very close to 1 (it is actually precisely 1 when k is small) even when the number of clusters is increased up to 100. It also implies that the structure of the dendrograms obtained from the centralized algorithm and PARABLE are very similar to one another.

The plots in Fig. 4 are stair shaped, drop suddenly and then increase gradually afterwards. The potential reason for that is in the PARABLE aligned dendrograms, two sibling subtrees may have some erroneously clustered data points. As long as the two sibling subtrees are labeled as the same cluster, the error in clustering these two subtrees are not accounted for in the B_k value. However, when the linkage between these two siblings is cut such that they fall into different clusters, these erroneously clustered data points reduce the value of B_k. However, as the sibling subtrees are cut into more fine-grained clusters, some of the erroneously clustered data points form their own clusters, which may not be wrongly clustered any more. This explains why the B_k bounces back to a higher value gradually after the sudden drop.

In another experiment, we deliberately aligned the dendrograms badly by either randomly making alignment decisions or make reverse alignments (align nodes with lower *similarity* function value together). These “control” cases helped validate our experiments and show what would happen if no dendrogram alignment was used. Fig. 5(a) shows the result of the high energy particle dataset. For the random alignment, B_k is about 0.3 while reverse alignment achieves only

[6] http://www.sigkdd.org/kddcup/

about 0.2. Fig. 5(b) shows the result for network intrusion dataset. This result is not that different from good alignment result. The potential reason is that the network intrusion dataset is an extremely biased dataset and very few of the network activities are actually labeled as intrusions.

7.2 Efficiency and Scalability

In this section, we evaluate the efficiency and scalability of PARABLE. We evaluate how the number of partitions, mappers/reducers and data points influence the time complexity of the algorithm.

First, we vary the number of partitions that the dataset is split into, keeping the number of mappers/reducers (hence the computing power) and the total number of data points constant. For the high energy particle data set, 20 mappers/reducers are used to cluster 5×10^4 data points. For the network intrusion data set, 25 mappers/reducers are used to cluster 1×10^5 data points. The result is shown in Fig. 6. The speed up is obvious when the number of partitions is relatively small. However, when the number of partitions gets large, the total time is dominated by non-computing tasks such as the time spent on messaging, I/O and MapReduce job setup overhead – this reduces the speed-up somewhat for large number of partitions.

Next, we adjust the number of mappers and reducers while keeping the number of data points and partitions constant. In this experiment, 5×10^4 data points are sampled from the high energy particle dataset and 1×10^5 data points are sampled from the network intrusion dataset. These data points are separated into 40 and 100 partitions, respectively. The result is shown in Fig. 7. The time required by the algorithm to terminate decreases when more mappers and reducers are added. It is worth to notice that the sequential hierarchical clustering algorithm takes more than 10 hours to finish the same work load.

Finally, we present PARABLE's scalability with respect to dataset size. In Fig. 8(a), 10^4 to 5×10^4 data points are sampled from high energy particle dataset for clustering. These data points are split into 25 partitions and 25 mappers/reducers are involved in the computation. In Fig. 8(b), 6×10^4 to 1.4×10^5 data points are sampled from network intrusion dataset for clustering. These data points are split into 100 partitions and 25 mappers/reducers are involved in the computation. It is observed that the time complexity increases as the number of data points increases.

8 Conclusion and Future Work

Tera- and peta- byte sized data sets are becoming increasingly popular in recent years. The phenomenal growth of data and the need to extract useful information from it motivates the development of scalable algorithms for data mining and knowledge discovery. In this paper, we present a novel parallel random-partition based hierarchical clustering algorithm that solves the problem using a divide-and-conquer approach – first, the algorithm randomly splits a large data set into smaller partitions and distributes it amongst available nodes; then a sequential hierarchical clustering algorithm is applied to each partition. A novel dendrogram alignment scheme allows the local clusters to be merged into a global model. The algorithm is implemented on an Apache Hadoop framework using the MapReduce programming paradigm. Empirical results on two large data sets from the ACM KDD Cup competition suggests that the PArallel RAndom-partition Based hierarchicaL clustEring algorithm (PARABLE) has significantly better scalability than centralized solutions in addition to maintaining good quality of clustering. Future work involves implementation of multiple levels of local clustering and making the dendrogram alignment algorithm have sub-linear time complexity.

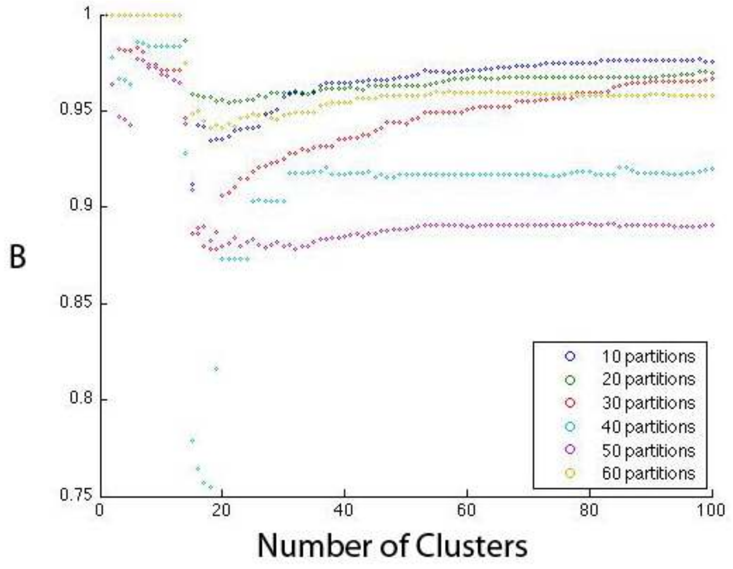
Acknowledgment

Funding for this work is provided by National Science Foundation award, IIS–0916186.

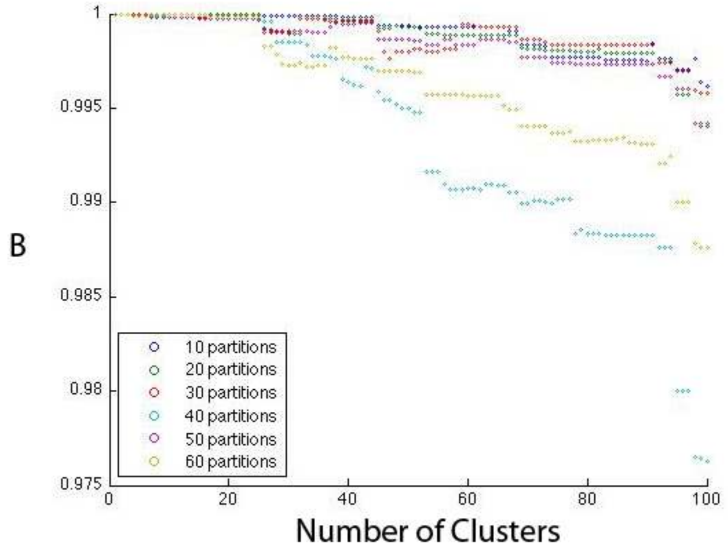
References

1. Large Hadron Collider (LHC). http://lcg.web.cern.ch/LCG/.
2. J. Dean and S. Ghemawat "Mapreduce: Simplified data processing on large clusters. " In Proceedings of the OSDI Conference, 2004.
3. W. Zhao, H. Ma and Q. He "Parallel K-means Clustering Based on MapReduce" In Proceedings on Cloud Computing, LNCS, Vol 5931, Pgs 674 – 679, 2009.
4. M. Ngazimbi "Data Clustering Using Mapreduce," In Master's Thesis, Boise State University, 2009.
5. T. Sun, C. Shu, F. Li, H. Yu, L. Ma and Y Fang "An Efficient Hierarchical Clustering Method for Large Datasets with Map-Reduce" In Proceedings of the International Conference on Parallel and Distributed Computing, Applications and Technologies, 12:2, Pgs 494 – 499, 2009.
6. S. Papadimitriou and J. Sun "DisCo: Distributed Co-Clustering with MapReduce" In Proceedings of ICDM, pages 512 – 521, 2008
7. E. M. Rasmussen, P. Willett "Efficiency of hierarchical agglomerative clustering using the ICL Distributed Array Processor" In Journal of Documentation, 45 (1), 1 –24, March 1989.
8. C. F. Olson "Parallel Algorithms for Hierarchical Clustering" In Parallel Computing, 8, pgs 1313–1325, 1993.
9. Z. Li, K. Li, D. Xiao, and L. Yang "An Adaptive Parallel Hierarchical Clustering Algorithm". In HPCC 2007, LNCS 4782, pp. 97107, 2007.
10. S. Rajasekara "Efficient Parallel Hierarchical Clustering Algorithms" In IEEE Transactions On Parallel And Distributed Systems, Vol. 16, No. 6, 2005
11. M. Dash, S. Petrutiu, and P. Scheuermann "E?cient Parallel Hierarchical Clustering". In Euro-Par 2004, LNCS 3149, pp. 363371, 2004.
12. T. Zhang, R. Ramakrishnan and M. Livny "BIRCH: An Efficient Data Clustering Method for Very Large Databases". In SIGMOD, pages 103–114, 1996.

13. S. Guha, R. Rastogi and K. Shim "ROCK: A Robust Clustering Algorithm for Categorical Attributes". In Information System, Vol. 25, No. 5, pp 345-366, 2000.
14. G. Karypis, E. Han and V. Kumar "CHAMELEON: A Hierarchical Clustering Algorithm Using Dynamic Modeling". In IEEE Computer, Vol. 32, No. 8, pp 68-75, 1999.
15. E. L. Johnson and H. Kargupta "Collective, Hierarchical Clustering from Distributed, Heterogeneous Data" In Large-Scale Parallel Data Mining, Pgs 221 – 244, 2000.
16. N. Samatova, G. Ostrouchov, A. Geist, and A. Melechko. "RACHET: An Efficient Cover-Based Merging of Clustering Hierarchies from Distributed Datasets". In *Distributed and Parallel Databases*, 11(2):157–180, 2002.
17. H. Gao, J. Jiang, L. She, Y. Fu "A New Agglomerative Hierarchical Clustering Algorithm Implementation based on the Map Reduce Framework". In International Journal of Digital Content Technology and its Applications Volume 4, Number 3, June 2010
18. Website. http://hadoop.apache.org/core/.
19. S. Ghemawat, H. Gobioff, S. T. Leung "The google file system. " In Proceedings of the 19th ACM Symposium on Operating Systems Principles, Lake George, NY.
20. DISC, Website. http://www.cs.cmu.edu/ bryant/.
21. F. Murtagh "Multidimensional Clustering Algorithms" Physica-Verlag, 1985
22. E. B. Fowlkes and C. L. Mallows A Method for Comparing Two Hierarchical Clusterings. In Journal of American Statistical Association, Vol 78, No 383, pp 553 – 569.

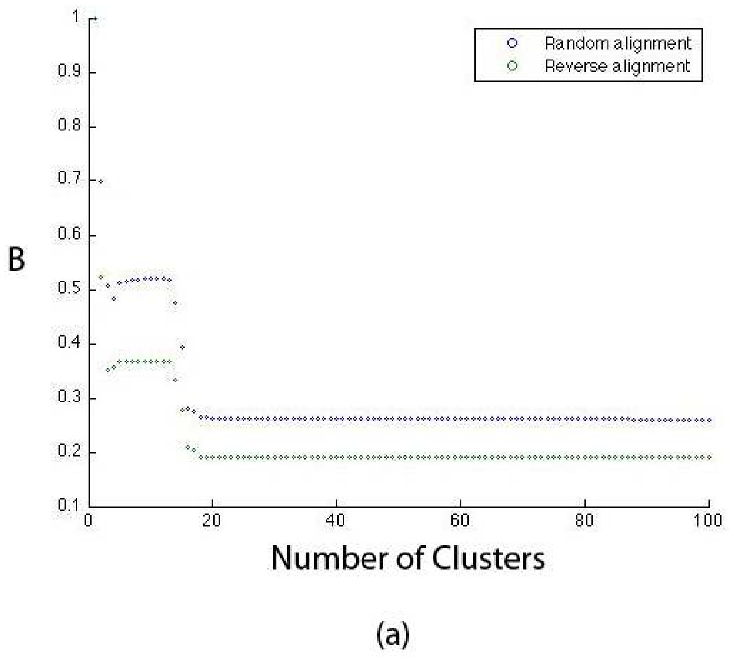


(a)

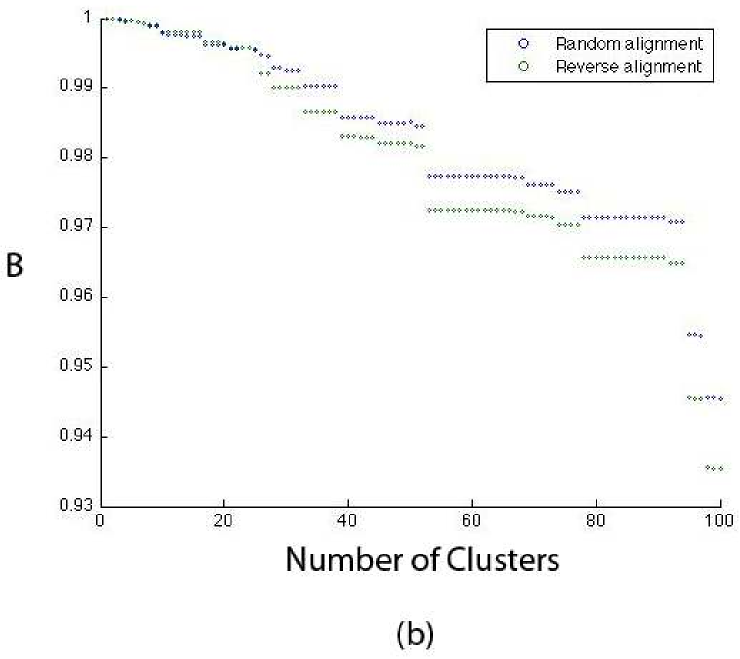


(b)

Fig. 4. Measurement of similarity between the dendrogram generated by sequential hierarchical clustering algorithm and the aligned dendrograms generated by PARABLE. (a) shows the relation between B_k and the number of clusters for high energy particles experiment. (b) shows the same relation for network intrusion experiment.



(a) Caption of subfigure 1



(b) Caption of subfigure 2

Fig. 5. B_k versus k for the control experiments. (a) shows the result of high energy particles experiment. (b) shows the result of network intrusion experiment.

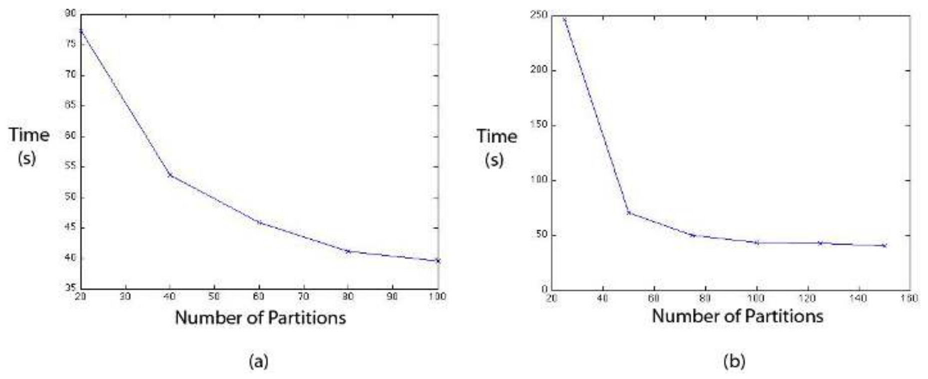


Fig. 6. The relation between the number of data partitions and the time complexity for clustering a fixed data set size. (a) shows the result of high energy particles experiment. $N = 5 \times 10^4$ in this experiment. (b) shows the result of network intrusion experiment. $N = 1 \times 10^5$ in this experiment.

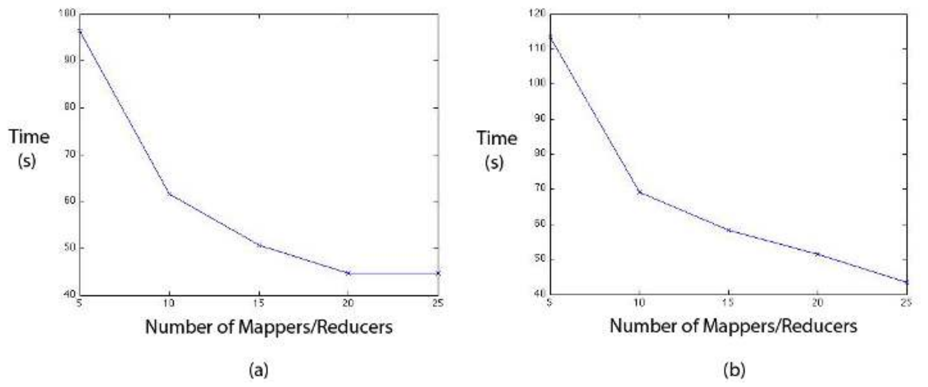
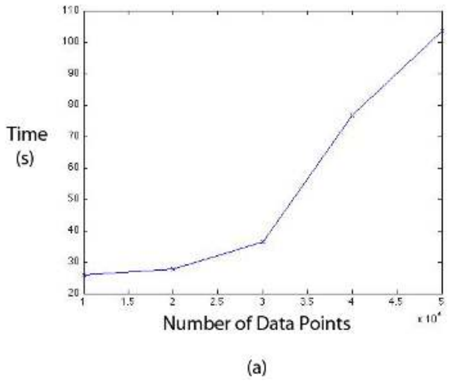


Fig. 7. The relation between the number of mappers/reducers and the time cost for clustering a given number of data points. (a) shows the result of high energy particles experiment. N=5×10^4 in this experiment. (b) shows the result of network intrusion experiment. N=1×10^5 in this experiment.



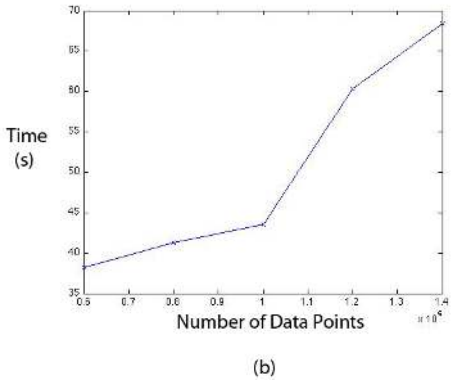


Fig. 8. Time cost of clustering different number of data points. (a) shows the result of high energy particles experiment. (b) shows the result of network intrusion experiment.