ISSN 1330-3651 (Print), ISSN 1848-6339 (Online)
DOI: 10.17559/TV-20150126121041

A COMMON FRAMEWORK OF PARTITION-BASED CLUSTERING FOR LARGE SCALE DATASET USING SAMPLING AND ITS MapReduce IMPLEMENTATION

Ran Jin, Chunhai Kou, Ruijuan Liu, Tao Guo

Original scientific paper

Clustering is one of the significant tasks in data mining, and partition-based clustering algorithms such as k-means are one of the popular solutions. However, with the increasing development of cloud computing and big data, large scale dataset has been a big challenge for clustering. For example, the execution of clustering algorithm is too time-consuming, the optimization of parameters is difficult, and the quality of clusters is not good. To this end, in this paper, we proposed a common framework of partition-based clustering algorithms such as *k*-means, and designed its MapReduce implementation. Specifically, in order to deal with the representation of large scale dataset, we propose to employ sampling technique. Then, inspired by *k*-means algorithm, we propose a common procedure of clustering, and provide a *k*-means based implementation. Furthermore, we implement proposed framework using MapReduce programming model. Experiments show that our method is efficient for large scale dataset.

Keywords: *large scale dataset; MapReduce; partition-based clustering; sampling*

Uobičajeni okvir grupiranja utemeljenog na raspodjeli za veliki sustav uzorkovanja podataka i njegova MapReduce implementacija

Izvorni znanstveni članak

Grupiranje (clustering) je jedan od važnih zadataka u rudarenu podataka (data mining), a algoritmi grupiranja utemeljenog na raspodjeli kao što su *k*-način jedno su od popularnih rješenja. Ipak, sve većim razvojem računarstva u oblaku i ogromne količine podataka, prijenos velikog broja podataka postao je veliki izazov za grupiranje. Na primjer, izvođenje algoritma grupiranja oduzima previše vremena, optimizacija parametara je teška, a kvaliteta grupa (klastera) nije dobra. U tu smo svrhu u ovom radu predložili uobičajeni okvir za algoritme grupiranja utemeljenog na raspodjeli kao što su *k*-način i dizajnirali njegovu MapReduce implementaciju. Posebice smo, u svrhu predstavljanja prijenosa velikog broja podataka, predložili primjenu tehnike uzorkovanja. Zatim, koristeći *k*-način algoritam, predlažemo uobičajeni postupak grupiranja i opisujemo primjenu na temelju *k*-način algoritma. Nadalje, implementiramo predloženi okvir primjenom MapReduce modela programiranja. Eksperimenti pokazuju da je naša metoda učinkovita za prijenos velikog broja podataka.

Ključne riječi: *MapReduce; prijenos velikog broja podataka; rupiranja utemeljenog na raspodjeli; uzorkovanje*

1 Introduction

Clustering is one of the significant tasks in data mining, also called unsupervised learning, defined as the process of grouping a set of objects into multiple groups such that the objects within the same group are similar and the objects across different groups are different [1]. The most challenges in clustering tasks are: (1) how to represent the whole dataset with enough information as little as possible; (2) how to measure the similarity between objects as well as the cost function.

With the increasing development of cloud computing [2] and big data [3÷5], large scale dataset has become a common source of clustering. In face of large scale dataset, clustering analysis has the following issues: (1) the dataset is complicated, such as large-scaled, high-dimensional, non-linear, and skewed; (2) the execution of clustering algorithm is too time-consuming, and the optimization of parameters is difficult; (3) the quality of clusters is not good. To solve the above challenges, many researchers have proposed parallel and distributed clustering methods. For example, Feng et al. [6] proposed parallel *k*-means algorithm based on MPI and applied to resume dataset. Kantabutra et al. [7] proposed a distributed version of *k*-means but dramatically increased the communication cost between nodes. Yang et al. [8] designed a cloud implementation of SPRINT algorithm based on Hadoop.

In this paper, inspired by existing efforts, we propose a common framework of partition-based clustering algorithms such as *k*-means, and design its MapReduce [9] implementation. Specifically, out contributions are as follows:

(1) We propose a common framework of partition-based clustering algorithms using sampling, and validate the effectiveness of proposed framework by implementing *k*-means and *k*-medoids algorithms;
(2) We modify the basic random sampling method with a partition-based method, to reduce the time cost of sampling on large scale dataset;
(3) We provide its implementation with MapReduce programming diagram, and design Map and Reduce procedure for each step;
(4) We evaluate the efficiency of proposed framework with *k*-means and *k*-medoids implementation. Besides, we compare the performance to the MPI based implementation, with different sizes of dataset and different numbers of nodes.

The remains of this paper are organized as follows. In Section 2 we provide some related work. Section 3 presents the common framework of sampling based clustering algorithm, and Section 4 describes the MapReduce implementation. Then experiments are conducted in Section 5. Finally, the paper is concluded in Section 6.

2 Related work

The common clustering algorithms include partition-based clustering, hierarchical clustering, density-based clustering, and others.

Partition-based clustering algorithms typically include *k*-means [10, 11] and *k*-medoid [12, 13]. K-means uses the average of objects within clusters as reference, while *k*-medoid uses the object in the centre of clusters as reference. There are three requirements of partition-based clustering: (1) the distance between data objects as the similarity measurement; (2) a cost function to evaluate the quality of clustering results; and (3) the initial centroids and clusters.

Hierarchical clustering algorithms [14, 15] repeatedly split or aggregate data through a hierarchical structure, in order to form a hierarchical sequence of solutions. The complexity is $O(n^2)$, and applicable to small scale dataset. For example, CURE [16] uses a novel hierarchical strategy by choosing a fixed number of representative points and multiplying a shrinking factor to approach the centre of the cluster. Chameleon [17] is a dynamic hierarchical clustering algorithm. It first splits the data objects into relatively smaller groups based on a graph partitioning method, and then uses an agglomerative hierarchical clustering method to repeatedly find out the real clusters.

Density-based clustering algorithms explore clusters with any shape based on the data density. For example, DBSCAN [18] can find clusters with any shape and also deal with noises. OPTICS [19] solves the problem of wide range of local density across different clusters. Instead of directly generating clustering results, it gives a hierarchical sequence of density-based clustering structure. Fraley and Hinneburg et al. [20, 21] proposed a kernel density estimation method, which studies the data distribution using statistical methods without any prior knowledge.

Besides, there also exist other clustering algorithms. For example, Pileva et al. [22] proposed a grid based clustering algorithm GCHL on large scale and high dimensional spatial database. Tsai et al. [23] designed a novel data clustering approach for data mining in large databases using ant systems, called ACODF. Andrew et al. [24] provided analysis of spectral clustering. Kawaji et al. [25] proposed a graph-based clustering method to cluster protein sequences into families, which automatically improves clusters of the conventional single linkage clustering method. Some researchers proposed clustering analysis based on intelligent algorithms such as genetic algorithm [26] and particle swarm optimization [27]. Fuzzy clustering was also proposed in [28, 29].

In this paper, we focus on one of the most popular clustering algorithms, partition-based algorithms, and explore the solutions of applying partition-based clustering onto large scale dataset. Indeed, there exist some efforts on tailoring partition-based clustering algorithms using parallel and distributed solutions. For example, Tsoumakas and Dhillon et al. [30, 31] developed the parallel version of k-means on distributed memory multiprocessors using data parallelization method. Manasi [32] proposed another parallel *k*-means by passing the centres of clusters between processors. Forman et al. [33] proposed to pass only statistical variables to improve the efficiency of *k*-means. Kantabutra et al. [34] designed a distributed *k*-means called *k*-Dmeans. Zheng et al. [35] proposed DK-means, which modified *k*-Dmeans by solving the problem of massive communication. Later, Li et al. [36] proposed a P2P based grid distributed clustering, called *k*-DmeansVM by solving the single point of failure issue of *k*-Dmeans. Mao [37] introduced Minimum-Maximum principle to modify Canopy-*k*-means algorithm, and implemented it using MapReduce framework.

3 Common framework of partition-based clustering using sampling

In this study, we leverage sampling technique to deal with large scale dataset. As proved in existing works [16], [38÷41], sampling can be used to accelerate the clustering analysis in large scale dataset scenarios. However, random sampling would lead to awful clustering results.

3.1 Overview

As one of the most popular partition-based clustering algorithms, k-means uses centroid to represent the whole cluster. Suppose the number of clusters is K, the number of data objects is N, and the number of dimensions is d. Given the set of data $D=\{x_1,x_2,...,x_N\}$, and the clusters $\{C_1,C_2,...,C_K\}$, where C_j is the set of data objects that belong to cluster j, and μ_j is the centre of cluster. Suppose Euclidean distance is used to measure the distance between objects, and denoted as $\|\cdot\|$. *k*-means updates the centroids of clusters and moves the members until the ideal clusters are found. The centroid is defined upon the average of data objects:

$$\mu_j=\sum_{x\in C_j}\frac{x}{|C_j|},\ j=1,2,...,K, \tag{1}$$

and the centroid is updated as:

$$f(x,C)=\sum_{j=1}^{K}\sum_{x\in C_j}\left\|x-\mu_j\right\|^2. \tag{2}$$

The objective is to minimize the cost function in Eq. (2), and the centroid is updated in each iteration.

However, the result quality of *k*-means clustering is unstable, especially in large scale dataset scenarios. To this end, we employ sampling method to adapt partition-based clustering to large scale dataset applications. The intuitive method is to randomly sample several partitions from the original large scale dataset, so that the clustering algorithm can be applied on each partition, and the result is reliable and can represent that of the whole dataset. For example, suppose the original dataset has K clusters, in ideal situation each partition should also has K clusters. However, in some partitions, the number of clusters can be less than K. Therefore, how to deal with clustering in each partition independently with unknown number of clusters is one of the big challenges in this paper.

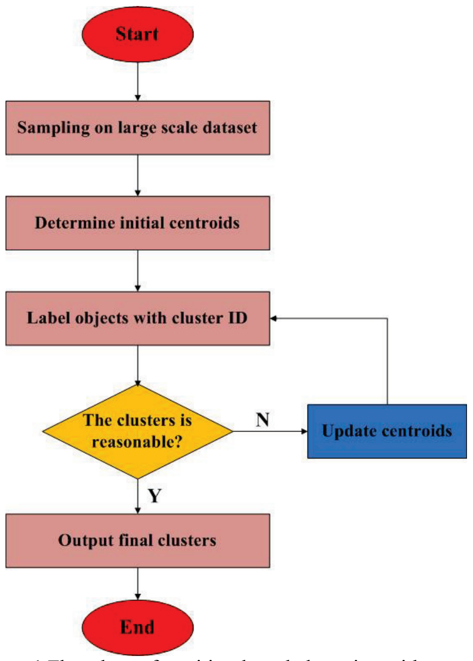
Start
Sampling on large scale dataset
Determine initial centroids
Label objects with cluster ID
The clusters is reasonable?
N
Update centroids
Y
Output final clusters
End

Figure 1 Flowchart of partition-based clustering with sampling

The basic idea of partition-based clustering is: given some initial centroids and clusters, enable the data objects approach to the centre of clusters based on some predefined rules, and then adjust until the clustering results remain stable and reasonable. Inspired by k-means, we design a common framework of partition-based clustering algorithm with sampling, as shown in Fig. 1. There are mainly four steps: (1) sampling upon large scale dataset; (2) determine initial centroids using sampled data; (3) update centroids; and (4) label all data objects with cluster IDs.

3.2 Sampling

As mentioned earlier, we want to sample a smaller size of partitions such that all K clusters are included in each partition. Suppose the original dataset D has K $K<N$ clusters $\{C_1,C_2,...,C_K\}$ with centroids $\{\mu_1,\mu_2,...,\mu_K\}$. The number of data objects in C_i is m_i.

Apply M times sampling on D, and the number of data objects in each sample D_i is N_i. The sampling satisfies the following requirements:

$$D_i \cap D_j = \Phi, N_i = N_j, N_i << N \tag{3}$$

where $i=1,2,...,M, i\neq j$, and there exists no overlap between samples.

However, the basic sampling method needs to traverse the whole dataset N times for each single record, where N is the size of the original dataset. Therefore the complexity of random sampling is $O(MN)$. To reduce the cost of sampling, we use a partition-based random sampling method. Specifically, split D into N_i partitions equally, where N_i is the size of each sample. Then randomly select one record from each partition. Therefore, the time cost of modified random sampling is $O(MN_i)$. Since $N_i << N$, we have $MN_i << MN$. Therefore, the sampling cost is dramatically reduced.

Inspired by [16], the sample size N_i can be estimated as:

$$N_i = \alpha N + \frac{N}{m_i}\log\frac{1}{\delta} + \frac{N}{m_i}\sqrt{log(\frac{1}{\delta})^2 + 2\alpha m_i \log\frac{1}{\delta}}, \tag{4}$$

where $\alpha(0\leq\alpha\leq 1)$ is the sampling proportion from D.

Suppose $C_{ij}(1\leq j\leq m_i)$ is one cluster in sample D_i, and the number of members in C_{ij} is m_{ij}. Let $d_{ij}(1\leq i\leq K, 1\leq j\leq N_i)$ be a data object in C_{ij}. The probability of d_{ij} also belonging to C_i is calculated as:

$$p_{ij} = \frac{m_{ij}}{\sum_{j=1}^{m_i} N_j}. \tag{5}$$

3.3 Calculating initial centroids

Initial centroids are determined by clustering on sample dataset. However, the real centroids are typically deviated from the initial. It could be adjusted by updating the average, which will be discussed in Section 3.4.

There are two steps in determining initial centroids: (1) apply clustering in each sample; and (2) combine results from all samples, as shown in Fig. 2. Note that if the number of clusters in samples is actually less than K, some cluster would be forced to split into several clusters so that there are always K centroids in each sample.

For simplicity, we use k-means to describe the centroids calculation in the first step. Note that any simple clustering algorithms can be applied here, since the data scale is dramatically reduced. Then, we get $K\times M$ small clusters.

Next we need to combine the clustering results of all samples. Use local centroid μ_{ij} to represent each cluster of samples, and the global centroid μ_j is calculated as:

$$\mu_j = \frac{m_{1j}\mu_{1j} + m_{2j}\mu_{2j} + ... + m_{Mj}\mu_{Mj}}{m_{1j} + m_{2j} + ... + m_{Mj}}, \tag{6}$$

where m_{ij} is the number of objects in cluster j of sample i, and M is the number of samples.

Theorem: The combined clustering results of each partition are equivalent to the results of single clustering on all the partitions.

Proof: The centroid of each cluster samples is calculated as:

$$\mu_{ij} = \frac{1}{m_{ij}}\sum_{d\in C_{ij}} d, \tag{7}$$

where C_{ij} is the j^{th} cluster of i^{th} sample, and m_{ij} is the size of cluster.

Since there is no overlapping between D_i, no data object would be labeled twice with different cluster ID. Substituting Eq. (7) into Eq. (6), we get:

$$\mu_j = \frac{m_{1j}\frac{1}{m_{1j}}\sum_{d\in C_{1j}} d + m_{2j}\frac{1}{m_{2j}}\sum_{d\in C_{2j}} d + \ldots + m_{Mj}\frac{1}{m_{Mj}}\sum_{d\in C_{Mj}} d}{m_{1j}+m_{2j}+\ldots+m_{Mj}} = \frac{\sum_{d\in C_{1j}} d + \sum_{d\in C_{2j}} d + \ldots + \sum_{d\in C_{Mj}} d}{m_{1j}+m_{2j}+\ldots+m_{Mj}}, \tag{8}$$

where $\sum_{d\in C_{1j}} d + \sum_{d\in C_{2j}} d + \ldots + \sum_{d\in C_{Mj}} d$ is the total data objects in the j^{th} cluster, and $m_{1j}+m_{2j}+\ldots+m_{Mj}$ is the size of global j^{th} cluster.

The left side of Eq. (8) is the combining local clustering results of each sample, and the right side is the single clustering results on the whole dataset. Therefore, the partition based method is equivalent to the single clustering algorithm on the whole dataset.

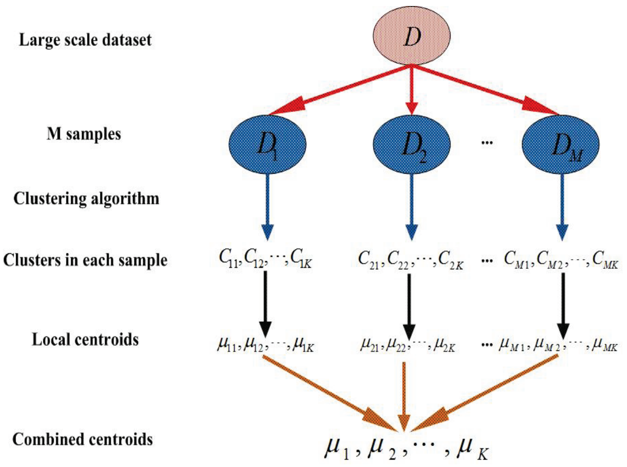
Large scale dataset
M samples
Clustering algorithm
Clusters in each sample
Local centroids
Combined centroids
D
D_1
D_2
D_M
$C_{11}, C_{12}, \cdots, C_{1K}$
$C_{21}, C_{22}, \cdots, C_{2K}$
$C_{M1}, C_{M2}, \cdots, C_{MK}$
$\mu_{11}, \mu_{12}, \cdots, \mu_{1K}$
$\mu_{21}, \mu_{22}, \cdots, \mu_{2K}$
$\mu_{M1}, \mu_{M2}, \cdots, \mu_{MK}$
$\mu_1, \mu_2, \cdots, \mu_K$

Figure 2 Determining initial centroids

3.4 Updating centroids

In previous step, we determined the initial centroids based on sampled dataset. However, only the sampled data is used, the results cannot represent the whole dataset D. Therefore, in this step, we add the remaining data objects into each cluster, and further update the centroids of clusters.

Assign data x in the remaining dataset to current clusters based on the minimum distance principle. That is,

$$c = \arg\min(|x-\mu_j|^2) \tag{9}$$

where μ_j is the centroid of cluster C_j, and c is the assigned cluster.

Once a new data is assigned with cluster label, the centroid of clusters should be updated in an iterative way, until all data objects are processed:

$$\mu_j = \frac{\mu_j m_j + x}{m_j + 1}, m_j = m_j + 1. \tag{10}$$

3.5 Labeling data objects

Now the new centroids are computed for all clusters. Re-label data object with cluster ID based on Eq. (7).

Now we consider the satisfaction of clustering results. Similar to k-means algorithms, we define a cost function, and try to minimize it:

$$c = \arg\min_j \left\| x - \mu_j \right\|^2. \tag{11}$$

4 MapReduce implementation

In previous sections, we discussed the common process of partition-based clustering using sampling. Although the proposed sampling based framework can handle large scale dataset in some way, the computation is still sequential. In order to parallelize and distribute the whole clustering process, we employ MapReduce for implementation.

MapReduce is a programming model for large scale parallel and distributed processing on clusters. Basically, there are two procedures in MapReduce: Map() and Reduce(). Typically, all the data is processed in the form of key/value pairs. As shown in Figure 3, first the input component reads data from splits. Then, the Map() procedure takes a series of key/value pairs, and generates processed key/value pairs, which are allocated to a particular reducer by partition function. Later, after data shuffling, the Reduce() procedure iterates through the values that are associated with specific key and produces zero or more outputs. MapReduce model provides convenience to programmers so that only Map and Reduce procedures need to be implemented, while other details are handled by mature platforms such as Hadoop.

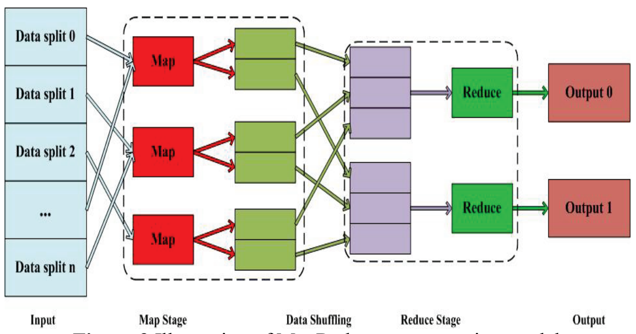

Figure 3 Illustration of MapReduce programming model

The MapReduce implementation of sampling based clustering is composed of four steps:

Step 1: perform sampling on large scale dataset D on M Map nodes, and on each Map node perform k-means clustering.

Step 2: using Reduce procedure to combine the results from M nodes and compute the initial centroids.

Step 3: distribute D equally onto n nodes, and on each node perform: (1) labelling data objects with cluster ID, and (2) update centroids incrementally;

Step 4: combine intermediate results from n nodes, and compute the new centroids.

If the termination condition is not satisfied, repeat steps 3 and 4. The overall MapReduce implementation is illustrated in Fig. 4. Details will be presented as follows.

4.1 Sampling

In this step, original large scale dataset D is sampled and processed on each node independently. Since we have M samples, M nodes are used. Specifically, the sampling process is implemented as REDUCE_SAMPLING(), which randomly select one row_id of each partition to decide the sample_id . Note that here *data* is partitioned equally by size N_i , as discussed in Section 3.2. The clustering on each sample is implemented as MAP_CLUSTERING(), as shown in Algorithm 1.

4.2 Computing initial centroids

After clustering on each node, we have the centroids of local clusters on each node. In this step, we need to combine the results to generate the initial centroids for original D. To achieve this, we split D equally into n partitions, and distribute each partition to one node without overlapping. Therefore, on each node, only D/n data objects are processed, as implemented in MAP_DISTRIBUTE() procedure. After computing centroids on each local node, REDUCE_CENTROIDS() procedure is called to combine the results.

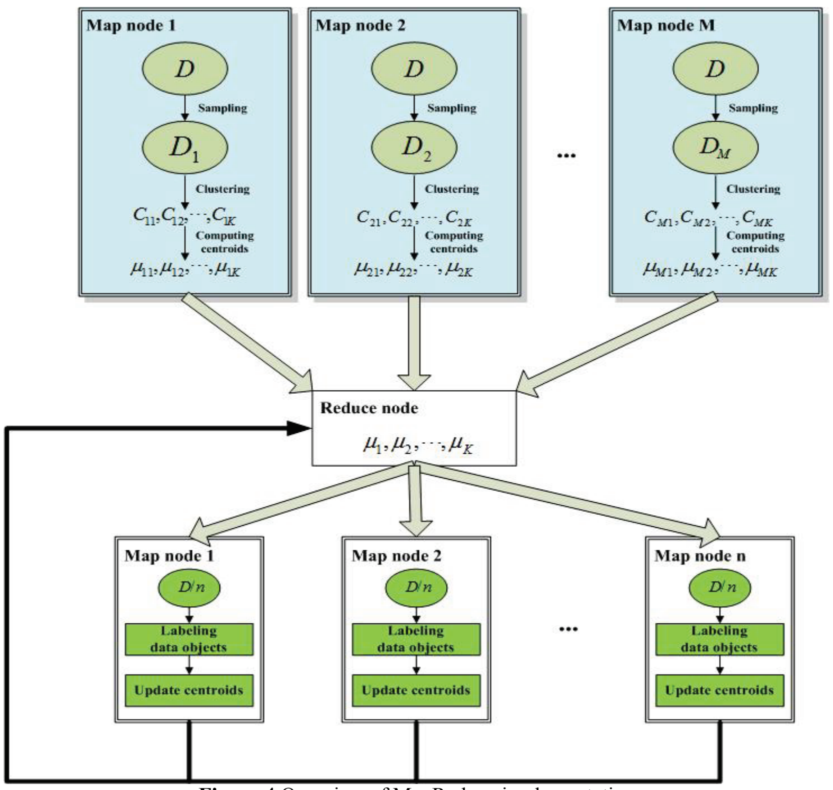

Figure 4 Overview of MapReduce implementation

Algorithm 1 Sampling

```
1: procedure REDUCE_SAMPLING((row_id, data))
2:     emit(sample_id, (row_id, data))
3: end procedure
4: procedure MAP_CLUSTERING((sample_id, (row_id, data)))
5:     perform k-means clustering for each sample_id
6:     compute centroid $\mu$ for each sample_id
7:     emit(cluster_id, $\mu$ )
8: end procedure
```

Algorithm 2 Computing centroids

```
1: procedure MAP_DISTRIBUTE((row_id, (data, $\mu$ _list)))
2:     compute $d=|data-\mu|^2$ for each $\mu$ in $\mu$ _list
3:     assign row_id to the cluster with minimum d, denoted as c
4:     update $\mu$ increasingly based on Eq. (10)
5:     emit(row_id, (c, $\mu$ ))
```

```
6: end procedure
7: procedure REDUCE_CENTROIDS((row_id, (c, $\mu$)))
8:     update $\mu$ based on Equation(6)
9:     assign row_id to the cluster with minimum d, denoted as c
10:    emit(cluster_id, (c, $\mu$))
11: end procedure
```

Algorithm 3 Labeling data objects

```
1: procedure MAP_LABELING((row_id, (data, $\mu$ _list)))
2:     compute $d=|data-\mu|^2$ for each $\mu$ in $\mu$ _list
3:     assign row_id to the cluster with minimum d, denoted as c
4:     emit(row_id, (data, c))
5: end procedure
```

Algorithm 4 Updating centroids

```
1: procedure MAP_CENTROIDS((row_id, (data, c)))
2:     compute centroid $\mu$ for each cluster c
3:     update $\mu$ increasingly based on Eq. (8)
4:     emit(row_id, $\mu$)
5: end procedure
6: procedure REDUCE_CENTROIDS((row_id, $\mu$))
7:     $\mu = 1/n * \sum \mu$
8:     compute $d=|data-\mu|^2$ for each $\mu$ in $\mu$ _list
9:     assign row_id to the cluster with minimum d, denoted as c
10:    emit(row_id, (c, $\mu$))
11: end procedure
```

4.3 Labelling data objects

Similar to the previous step, we distribute D to n nodes, and each node labels D/n data objects. Indeed, this step is similar to MAP_DISTRIBUTE() procedure, but without computing local centroids. The reason is that if the termination condition is satisfied, this step would be the last one to output clusters as well as data objects associated with cluster ID.

4.4 Update centroids

If the results are not satisfactory, centroids would be updated in this step. Firstly, MAP_CENTROIDS() procedure computes centroid in each local node, and then REDUCE_CENTROIDS() procedure combines results from all Map nodes, and generates the final cluster ID and centroids.

5 Experiment

In this study, we have 4 PCs with 3,00 GHz Intel dual-core processors, 2 GB RAM and 160G disk storage for our MapReduce cluster. We assign one as NameNode and JobTracker, and the rest three as computing nodes. Each PC can be used as 2 nodes, and therefore we have 8 nodes maximum. We employ two common clustering algorithms, *k*-means and *k*-medoids, to implement our sampling based clustering framework using MapReduce. The dataset is collected from online application. After pre-processing, we have 10 dimensions here, and the dataset size is represented as the number of records. For comparison, we also provide the MPI implementation of both algorithms.

Table 1 Execution time (ms) of different methods

Dataset size	1M	10M	50M	100M	200M
k-means	89	173	2813	18544	890459
MPI based k-means	60	91	234	5625	95487
k-means implementation of proposed method	1124	2895	3252	4122	10258
k-medoids	81	152	2508	16774	755323
MPI-based *k*-medoids	52	87	228	5443	94056
k-medoids implementation of proposed method	1013	2757	3045	3998	9887

Tab. 1 lists the results of different methods with different sizes of dataset, when 4 nodes are used. From Tab. 1 we have the following observations: (1) The basic *k*-means or *k*-medoids performs worst, because it is more suitable for small dataset on single node. (2) For relatively small dataset, MPI based clustering is faster than proposed method, because the processing logic behind Hadoop is complicated and therefore increases the overhead. (3) When the size of dataset grows, the proposed method has the best performance. Therefore, we can see that our MapReduce based solution can efficiently deal with the large scale challenge.

As shown in Figs. 5 and 6, we evaluate the efficiency of *k*-means and *k*-medoids implementation of proposed method with different numbers of nodes. We can observe that: (1) basically the execution time decreases a lot when more nodes are deployed; and (2) when the size of dataset is relatively small, the improvement of multi-node execution is unstable as shown in Fig. 5, while for large scale dataset, the performance is almost linearly promoted as shown in Fig. 6.

Besides, we also evaluate the accuracy of our clustering method using SSE (Sum of Squared Errors) measure, calculated as:

$$SSE = \sum_{j=1}^{K} \sum_{d \in C_j} \left\| \mu_j - d \right\|^2, \tag{12}$$

where $\|\cdot\|$ denotes the distance.

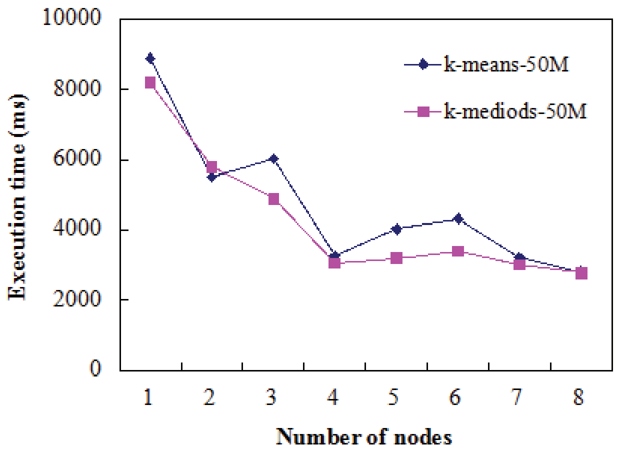

Figure 5 Execution time of proposed method with 50M dataset

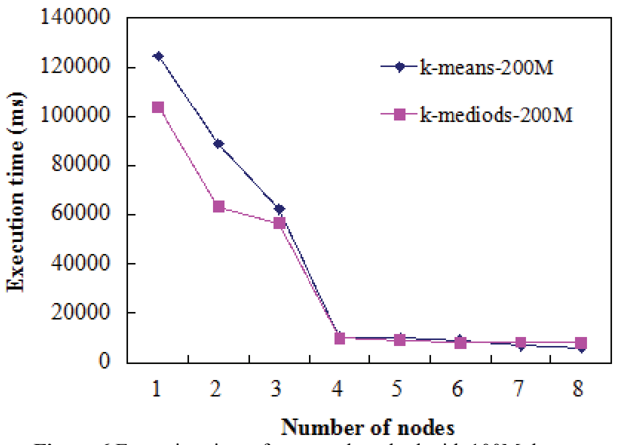

Figure 6 Execution time of proposed method with 100M dataset

As illustrated in Fig. 7, we have four lines for 50M and 200M dataset with *k*-means and *k*-medoids implementation of proposed framework respectively. We can see that when the number of nodes is less than 4, the SSE reduces dramatically when more nodes are used. However, SSE remains relatively stable when adding more nodes. Therefore, we conclude that due to the overhead of parallel and distributed processing, it is not always the fact that the more nodes are deployed, the more efficient the algorithm is. For example, the suggested number of nodes in these experiment settings is 4.

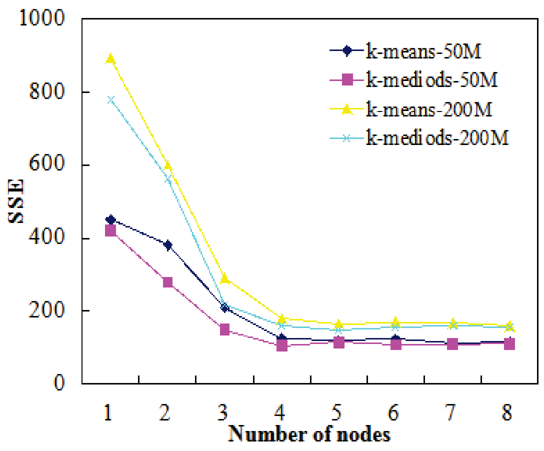

Figure 7 SSE measure of proposed clustering method

Moreover, we investigate the sampling cost. Fig. 8 gives the ratio of sampling time cost to the total execution time. We have the following observations. (1) *k*-means implementation has less sampling cost than *k*-mediods implementation. Because the total time cost of *k*-means is bigger than *k*-mediods while the sampling cost with specific data size is fixed. (2) The larger the dataset size is, the less percentage of sampling cost is. The reason is that the total time cost increases with the dataset size. (3) The more nodes are involved, the less ratio of sampling cost is. Since the sampling cost of specific data size remains stable, the more nodes are deployed, the more extra cost is introduced, which leads to a decrease in the ratio of sampling cost to the total cost.

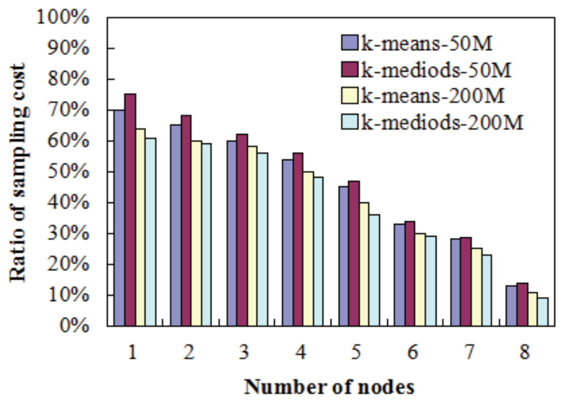

Figure 8 Ratio of sampling cost to the total execution time

6 Conclusion

In this work, we proposed a sampling based clustering for partition-based clustering algorithms for *k*-means, and provided a MapReduce implementation. However, there are some simplifications in this paper. First, we simply use multiple times of independently random sampling on the original large scale dataset.

Second, we only implement *k*-means clustering algorithms as an example of partition-based clustering. In future works, we would dive deeper to extend our solution.

Acknowledgements

This work was supported by the Science and Technology Research Program of Zhejiang Province, under grant No. 2011C21036, and by the Ningbo Natural Science Foundation under grant No. 2015A610141, and by the Zhejiang Science and Technology Program under grant No. 2016C33195.

7 References

[1] Han, J.; Kamber, M. Data Mining, Southeast Asia Edition: Concepts and Techniques. Morgan Kaufmann, California, 2006.

[2] Peter, M.; Grance, T. The NIST Definition of Cloud Computing (draft). // NIST Special Publication. (2011), pp. 1-7.

[3] LaValle, S. et al. Big Data, Analytics and the Path from Insights to Value. // MIT Sloan Management Review. 52, 2(2011), pp. 21-31.

[4] Zikopoulos, Paul; Chris, Eaton. Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data. McGraw-Hill Osborne Media, USA, 2012.

[5] Yang K. W.; Zhang P. L.; Ge B. F.; Dou Y. J. A Variables Clustering Based Differential Evolution Algorithm to Solve Production Planning Problem. // International Journal of Simulation Modelling. 14, 3(2015), pp. 525-538. DOI: 10.2507/IJSIMM14(3)CO13

[6] Feng Lina. K-Means Clustering Method and Its Application in Resume Data Parallelism. Thesis of Yunnan University, 2010.

[7] Kantabutra; Sanpawat; Alva L. Couch. Parallel K-means Clustering Algorithm on NOWs. // NECTEC Technical Journal. 1, 6(2000), pp. 243-247.

[8] Yang Chenshou. Data Mining Research Based on HADOOP. Chongqing University, Chongqing, 2010.

[9] Dean, J.; Ghemawat, S. MapReduce: Simplified Data Processing on Large Clusters. // Communications of the ACM. 51, 1(2008), pp. 107-113. DOI: 10.1145/1327452.1327492

[10] MacQueen, J. Some Methods for Classification and Analysis of Multivariate Observations. // Proc. of the 5th Berkeley Symposium on Mathematical Statistics and Probability. (1967), pp. 281-297.

[11] Li, Yang. K-means Clustering Algorithm in Intrusion Detection. // Computer Engineering. 33, 14(2007), pp. 154-156.

[12] Chen, L.; Özsu, M. T.; Oria, V. Robust and Fast Similarity Search for Moving Object Trajectories. // Proc. of the 2005 ACM SIGMOD International Conference on Management of data. (2005), pp. 491-502. DOI: 10.1145/1066157.1066213

[13] Park, H. S.; Jun, C. H. A Simple and Fast Algorithm for K-medoids Clustering. // Expert Systems with Applications. 36, 2(2009), pp. 3336-3341. DOI: 10.1016/j.eswa.2008.01.039

[14] Marques, J. P. Pattern Recognition Concepts, Methods and Applications. // Tsinghua University Press, Beijing, 2002.

[15] Fred, A. L. N.; Leitão, J. M. N. Partitional vs Hierarchical Clustering Using a Minimum Grammar Complexity Approach. // LNCS. 1876(2000), pp. 193-202. DOI: 10.1007/3-540-44522-6_20

[16] Guha, S.; Rastogi, R.; Shim, K. CURE: An Efficient Clustering Algorithm for Large Databases. // ACM SIGMOD Record. 27, 2(1998), pp. 73-84. DOI: 10.1145/276305.276312

[17] Karypis, G.; Han, E. H.; Kumar, V. Chameleon: Hierarchical Clustering Using Dynamic Modeling. // Computer. 32, 8(1999), pp. 68-75. DOI: 10.1109/2.781637

[18] Ester, M.; Kriegel, H. P.; Sander, J. et al. A Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. // Proc. of 2nd International Conference on Knowledge Discovery and Data Mining. (1996), pp. 226-231.

[19] Ankerst, M.; Breunig, M. M.; Kriegel, H. P. et al. Optics: Ordering Points to Identify the Clustering Structure. // ACM SIGMOD Record. 28, 2(1999), pp. 49-60. DOI: 10.1145/304181.304187

[20] Fraley, C.; Raftery, A. E. Model-based Clustering, Discriminant Analysis, and Density Estimation. // Journal of the American Statistical Association. 97, 458(2002), pp. 611-631. DOI: 10.1198/016214502760047131

[21] Hinneburg, A.; Gabriel, H. H. Denclue 2.0: Fast Clustering Based on Kernel Density Estimation. Advances in Intelligent Data Analysis VII. Springer Berlin Heidelberg, 2007.

[22] Pilevar, A. H.; Sukumar, M. GCHL: A Grid-clustering Algorithm for High-dimensional Very Large Spatial Databases. // Pattern Recognition Letters. 26, 7(2005), pp. 999-1010. DOI: 10.1016/j.patrec.2004.09.052

[23] Tsai, C. F.; Tsai, C. W.; Wu, H. C.; Yang, T. ACODF: A Novel Data Clustering Approach for Data Mining in Large Databases. // Journal of Systems and Software. 73, 1(2004), pp. 133-145. DOI: 10.1016/S0164-1212(03)00216-4

[24] Ng, A. Y.; Jordan, M. I.; Weiss, Y. On Spectral Clustering: Analysis and An Algorithm. // Advances in Neural Information Processing Systems, Proc. of the 15th Annual Conference on Neural Information Processing Systems. (2002), pp. 849-856.

[25] Kawaji, H.; Yamaguchi, Y.; Matsuda, H. et al. A Graph-based Clustering Method for a Large Set of Sequences Using a Graph Partitioning Algorithm. // Genome Informatics, Proc. of the International Conference on Genome Informatics. (2001), pp. 93-102.

[26] Su, Shoubao; Yu, Shuhao; Chen, Minghua. Progress on Clustering Mining Based on Intelligent Computing. // Computer Measurement & Control. 14, 5(2006), pp. 561-563.

[27] Liu, Xiangdong; Sha, Qiufu; Liu, Yongkui. Clustering Analysis Based on PSO. // Computer Engineering. 32, 6(2006), pp. 201-202.

[28] Corsini, P.; Lazzerini, B.; Marcelloni, F. A Fuzzy Relational Clustering Algorithm Based on A Dissimilarity Measure Extracted from Data. // IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics. 34, 1(2004), pp. 775-781. DOI: 10.1109/TSMCB.2003.817041

[29] Chiang, J. H.; Hao, P. Y. A New Kernel-based Fuzzy Clustering Approach: Support Vector Clustering with Cell Growing. // IEEE Transactions on Fuzzy Systems. 11, 4(2003), pp. 518-527. DOI: 10.1109/TFUZZ.2003.814839

[30] Tsoumakas, G.; Vlahavas, I. P. Distributed Data Mining of Large Classifier Ensembles. // Proc. of the 2nd Hellenic Conference on Artificial Intelligence. (2002), pp. 249-256.

[31] Dhillon, I. S.; Modha, D. S. A Data-clustering Algorithm on Distributed Memory Multiprocessors. // Proc. of the Large-Scale Parallel KDD Systems Workshop. (2000), pp. 245-260.

[32] Joshi, M. N. Parallel K-Means Algorithm on Distributed Memory Multiprocessors. // Twin City. The Web Version (2003), http://www.cs.umn.edu/~mnjosh i/PKMeans.pdf.

[33] Forman, G.; Zhang, B. Distributed data clustering can be efficient and exact. // ACM SIGKDD Explorations Newsletter. 2, 2(2000), pp. 34-38. DOI: 10.1145/380995.381010

[34] Kantabutra, S.; Couch, A. L. Parallel K-means Clustering Algorithm on NOWs. // NECTEC Technical journal. 1, 6(2000), pp. 243-247.
[35] Zheng, Miaomiao; Ji, Genlin. DK-Means-Improved Distributed Clustering Algorithm of K-Dmeans. // Computer Research and Development. z2(2007), pp. 84-88.
[36] Li, Liu; Tang, Jiuyang; Ge, Bing et al. k-DmeansWM: Distributed Clustering Algorithm Based on P2P Network. // Computer Science. 37, 1(2010), pp. 39-41.
[37] Mao, Dianhui. Improved algorithm of Canopy-Kmeans Based on MapReduce. // Computer Engineering and Applications. 48, 27(2012), pp. 22-26.
[38] Choromanska, A.; Jebara, T.; Kim, H. et al. Fast Spectral Clustering via the Nyström Method. // Proc. of the 24th International Conference on Algorithmic Learning Theory. (2013), pp. 367-381. DOI: 10.1007/978-3-642-40935-6_26
[39] Achlioptas, D.; Mcsherry, F. Fast Computation of Low-rank Matrix Approximations. // Journal of the ACM. 54, 2(2007), pp. 1-19. DOI: 10.1145/1219092.1219097
[40] Hearn, T. A.; Reichel, L. Fast Computation of Convolution Operations via Low-rank Approximation. // Applied Numerical Mathematics. 75, (2014), pp. 136-153. DOI: 10.1016/j.apnum.2013.06.002
[41] Zhang, J.; Wu, G.; Hu, X. et al. A Parallel K-means Clustering Algorithm with MPI. // Proc. of the 4th International Symposium on Parallel Architectures, Algorithms and Programming. (2011), pp. 60-64. DOI: 10.1109/paap.2011.17

Authors' addresses

Ran Jin, Associate Professor
1) School of Computer Science and Information Technology, Zhejiang Wanli University,
No. 8 South QianHu Road, Ningbo, Zhejiang, 315100, China

2) College of Computer Science and Technology, Zhejiang University,
No. 38 Zheda Road, Hangzhou, Zhejiang, 310027, China
E-mail: ran.jin@163.com

Chunhai Kou, Professor
School of Science, Donghua University,
No. 2999 North Renmin Road, Songjiang district, Shanghai, 201620, China
E-mail: kouchunhai@dhu.edu.cn

Ruijuan Liu, Lecturer
School of Information Science and Technology, Donghua University,
No. 2999 North Renmin Road, Songjiang district, Shanghai, 201620, China
E-mail: ruirui0516@163.com

Tao Guo
School of Computer Science and Technology, Donghua University
No. 2999 North Renmin Road, Songjiang district, Shanghai, 201620, China
E-mail: gt@j2cms.org