Parallel *k*-modes Algorithm based on MapReduce

Guo Tao
College of Computer Science and Technology
Donghua University
Shanghai, China
j2cms.org@gmail.com

Ding Xiangwu, Li Yefeng
College of Computer Science and Technology
Donghua University
Shanghai, China
dingxw@dhu.edu.cn

***Abstract*—*K*-modes is a typical categorical clustering algorithm. Firstly, we improve the process of *k*-modes: when allocating categorical objects to clusters, the number of each attribute item in clusters is updated, so that the new modes of clusters can be computed after reading the whole dataset once. In order to make *k*-modes capable for large-scale categorical data, we then implement *k*-modes on Hadoop using MapReduce parallel computing model. Experiments show that, parallel *k*-modes archives good speedup ratio when dealing with large-scale categorical data.**

Keywords—categorical data; k-modes; parallel clustering; MapReduce

I. Introduction

Clustering [1, 2], which belongs to the field of unsupervised learning, is an organizaing process of grouping data objects into clusters based on similarity. In other words, data objects in different clusters are as dissimilar as possible, and those in the same cluster are as similar as possible. Categorical data[3]contains non-numeric attributes , and is a simplified version of the symbolic objects. Along with the development of Internet, the categorical data in real life exists in continuous explosive growth. Clustering such large-scale and high dimension categorical data has been widely concerned in recent years.

As a typical categorical clustering algorithm, *k*-modes[4, 5] is widely used in various situations. Traditional *k*-modes implemented on a single computer can't execute effectively especially when dealing with large-scale categorical data. In order to make *k*-modes capable for parallelism, we improve and implement *k*-modes based on MapReduce[6, 7] computing model on Hadoop[8], which is a popular distributed system architecture with high scalability and widely applied. At the end of this paper, we compare the clustering rate between parallel *k*-modes and serial *k*-modes through experiments.

II. *K*-modes Algorithm

K-modes is a typical categorical clustering algorithm which was proposed by Huang[5]as an extension the *k*-means[9]. *K*-modes introduces a simple dissimilarity metric to deal with categorical objects by replacing the distance metric in *k*-means with modes, and uses a frequency based method to update modes to minimize the clustering cost function during the clustering process.

A. Dissimilarity Measures

Let X, Y be two categorical objects with m categorical attributes. The dissimilarity metric between X and Y is calculated by the total mismatches of the corresponding attribute categories of the two objects. The definition of dissimilarity is as follows:

$$d(X,Y)=\sum_{j=1}^{m}\delta(\mathrm{x_j},\mathrm{y_j}) \tag{1}$$

where

$$\delta(\mathrm{x_j},\mathrm{y_j})=\begin{cases}0 & (\mathrm{x_j}=\mathrm{y_j})\\ 1 & (\mathrm{x_j}\neq\mathrm{y_j})\end{cases} \tag{2}$$

B. Mode of a Set

Let $\boldsymbol{X}=\{X_1, X_2, \dots, X_n\}$ be a set of categorical objects with categorical attributes $A_1, A_2, \dots A_m$.

Definition: A mode of $\boldsymbol{X}$ is a vector $Q=[q_1, q_2, \dots, q_m]\in\Omega$ that minimizes the sum of dissimilarity of $\boldsymbol{X}$, defined as follows:

$$D(Q,\boldsymbol{X})=\sum_{i=1}^{n}d(X_i,Q) \tag{3}$$

C. Find the mode

Let $c_{(k,j)}$ be the number of objects having item $c(k,j)$ in attribute A_j, then the frequency of item $c(k,j)$in $\boldsymbol{X}$ ***is*** defined as follows:

$$f_r(\mathrm{A_j}=c(k,j)\,|\,\boldsymbol{X})=\frac{n_{c(k,j)}}{n} \tag{4}$$

Theorem[5] : The function $D(Q,\boldsymbol{X})$ is minimized if $f_r(\mathrm{A_j}=c(k,j)\,|\,\boldsymbol{X})\geqslant f_r(\mathrm{A_j}=c(k,j)\,|\,\boldsymbol{X})$ for $q_j\neq c(k,j)$ for all $j=1\dots m$.

D. The clustering process of k-modes

Let $\{C_1, C_2, \dots, C_k\}$ be a partition of $\boldsymbol{X}$, where $C_l\neq\emptyset$for $1\leq l\leq k$, and $\{Q_1,Q_2,\dots,Q_k\}$the modes of $\{C_1, C_2, \dots, C_k\}$. The total cost is defined as follows:

$$E=\sum_{l=1}^{k}\sum_{i=1}^{n}w_{i,l}\,d(X_i,Q_l) \tag{5}$$

where $w_{il} \in \{0,1\}$, $\sum_{l=1}^{k} w_{i,l} = 1$, and $0 < \sum_{i=1}^{n} w_{i,l} < n$ for $1 \le l \le k, 1 \le i \le n$.

$\boldsymbol{W}$ is an $n \times k$ matrix of {0, 1} where *n,k* are used respectively for the number of categorical objects and the number of clusters. Variable w_{il}=1 means the object X_i *is* partitioned to the cluster C_l.

The goal of clustering $\boldsymbol{X}$ is to find a set $\{Q_1, Q_2, \ldots, Q_k\}$ that can minimize *E*. The basic steps are as follows[5]:

①.Select *k* initial modes, one for each cluster.

②.Allocate an object to the cluster whose mode is the nearest to it.

③.Use a frequency based method to update modes of the cluster after each allocation,

④. Repeat ② and③ until no object has changed clusters after a full cycle test of the whole data set

III. MapReduce Description

A. Programming model of MapReduce

MapReduce[6] is a parallel programming model proposed by Google in 2004. Hadoop[8] is An Apache's open source project that implements the MapReduce framework, as well as the distributed file system named HDFS[10, 11].

MapReduce contains five external components (InputFormat, Mapper, Partitioner, Reducer and OutputFormat) which belong to the callback interface. The user-defined Mapper class should extend the system's Mapper class, which has four methods: setup, map, cleanup and run. Among them, the setup method and the cleanup method is used to manage the Mapper's resources in life cycle, where the setup method is called before the map method, and the cleanup method is called after the completion of the map method. Map is used to process the input key/value pairs. The run method executes the described the whole process described above, including call for the setup, then iterating all the key/value pairs, finally call for the cleanup function. The run method doesn't need to be overridden generally. Map's output is also in the form of key/value pairs, which are assigned to the appropriate Reduce according to the partition rules. The user-defined Reducer class should extend the system's Reducer class, which has four methods: setup, reduce, cleanup and run. Among them, the opportunity for calling setup, cleanup and run method is similar to that in Mapper class. The reduce method processes output from the Map method as the input which is a collection of values which have the same key.

B. The advantages of MapReduce

MapReduce is a data-parallel model, and it can share variables and communicate conveniently. Compared with other parallel computing model(such as MPI[12], OpenMP[13]), MapReduce has great advantages: Firstly, it can not only process large-scale data, but also hide a lot of tedious details, such as automatic parallelization, load balancing and disaster management. It simplifies the development process greatly. Secondly, MapReduce has a very good scalability. Every time we add a server, the computing power of this machine will be able to added to the Hadoop cluster. The scalability of most distributed processing framework in the post are far less than MapReduce.

IV. Parallel k-modes algorithm

Without changing the principle of *k*-modes algorithm, firstly we improve the implementation process of *k*-modes clustering algorithm: when allocating categorical objects to clusters, the number of each attribute item in clusters is update so that we can compute the new modes of clusters after reading the whole dataset once. Then we implement k-modes on Hadoop using MapReduce parallel computing model so that it can deal with large-scale categorical data. The specific implementation is divided into 4 steps.

STEP1 Preparation

We randomly select *k* initial categorical objects from $\boldsymbol{X}$, one for each cluster as a mode, and numbered from 0 to *k*-1. The FileInputFormat function is redefined according to the computing capability of Hadoop to divide the set $\boldsymbol{X}$ into *p* blocks, thereby generating *p* parallel Map tasks. Each categorical object is represented by a line of text with key-value pair, whose key is the cluster id and value contains information of the object. All data is stored in the HDFS.

STEP2 Allocating objects

Firstly, the setup function initializes *k* clustering modes from HDFS. Secondly, the map function reads the objects one by one, with the input key-value pair of map function as <old_cluster_id, object>. Then we compute the dissimilarity between the each two modes in k and allocate the object to the cluster whose dissimilarity is minimized. At the same time, we update the number of each attribute item in the cluster. The output key-value pair of map function is <new_cluster_id, object>. Finally, the cleanup function writes the *k* clusters' information back to the HDFS distinguished by Map's id.

STEP3 Update modes

Firstly, we read the clusters' information written by *p* Map from the HDFS, then combine the clusters according to the corresponding cluster's id. We can choose the item with max frequency as the one of new modes and write the new modes back to HDFS.

Then we calculate the dissimilarity between the new mode and the old mode of all clusters. If the dissimilarity value is less than a threshold, go to the step 4. Otherwise, return to the step 2.

STEP4 Output the clustering result

We define *p* Map tasks and *k* Reduce tasks to deal with the *p* output blocks of step 2, with each Map reading a block, and allocating the object to the corresponding Reduce according to the key value (cluster's id), the Reduce process will generate *k* files in the HDFS, each of which contains a set of categorical objects of a cluster.

The execution flow chart of k-modes is as follows:

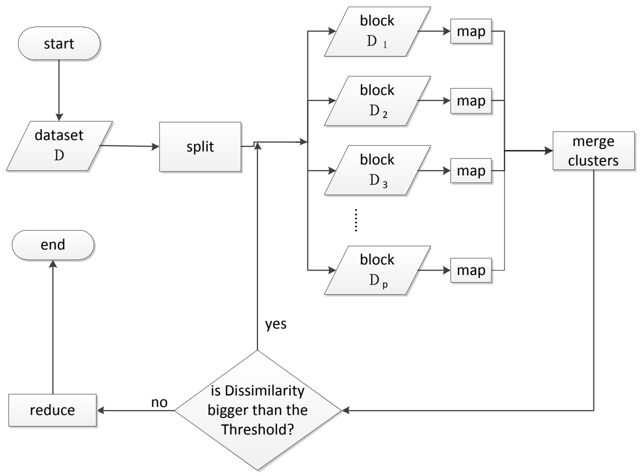

Fig. 1. The execution flow of *k*-modes

V. Speedup Ratio

The speedup[14] ratio is the time of serial executing to parallel executing. Hadoop (version: 1.xx) is a Master/Slave architecture, includeing one Master server and several Slave servers. The processes running on Master are NameNode, SecondaryNameNode and JobTracker, and the processes running on Slave server are DataNode and TaskTracker. TaskTracker needs to set the maximum number of runnable tasks. The process running on two types of servers also represents their functional role. JobTracker emerges TaskTracker to start the task, and the TaskTracker will create a single Java virtual machine for each task to prevent interference between tasks, and have special thread to monitor the usage of its resources.

At present, the computer's CPU is generally multicore and multithreaded (for example, Intel i7 CPU is four cores and eight threads), memory is also bigger enough. We assume that when setting the number of tasks on each TaskTracker to TN, all the tasks have enough CPU and memory to parallel execution in parallel. If the cluster has NT TaskTrackers, then total number of tasks of whole cluster is TT.

$$TT = TN \times NT \tag{6}$$

Parallel *k*-modes will assign data to the TT tasks to execute in parellel, and then to do merging processing, so the ideal speedup is TT, also the upper limit. When processing a large amount of data, parallel k-Modes has significant advantages, when the computing resources of cluster such as CPU and memory is relatively insufficient, or the TaskTracker number is set too big, or the influence of network and I/O overhead, and so on and so forth, it can't achieve the ideal speedup.

VI. Experiments

Parallel *k*-modes algorithm was tested on Hadoop clusters, compared with serial *k*-modes tested on single PC. The Hadoop cluster has 9 PCs, one as Master, and the rest eight as Slave. The hardware and software configuration of PCs as shown in Table 1 and Table 2.

TABLE 1. The Hardware environment

Hardware	Parameters
CPU	Intel i7-3370 3.43GHz 4 cores
CACHE	8192KB
RAM	2*4=8GB 1333MHz
DISK	500GB 1GB/s 8192KB(buffer memory)

TABLE 2. The Software environment

Software	Parameters
OS	CentOS 6.5
Hadoop	1.02
JDK	1.6

Experiments use US Census Data (1990) Data Set (http://dataferrett.census.gov), which contains a one-percent sample of the Public Use Microdata Samples (PUMS) person records drawn from the full 1990 census sample. It contains 2458285 categorical objects with 68 categorical attributes, such as Age, Ancestry, Citizen, School, and so on. It's345MB in size, and we generate 10GB random data according to its data format. By increase the amount of data, the processing time of parallel *k*-modes and serial *k*-modes is shown in Figure 1.

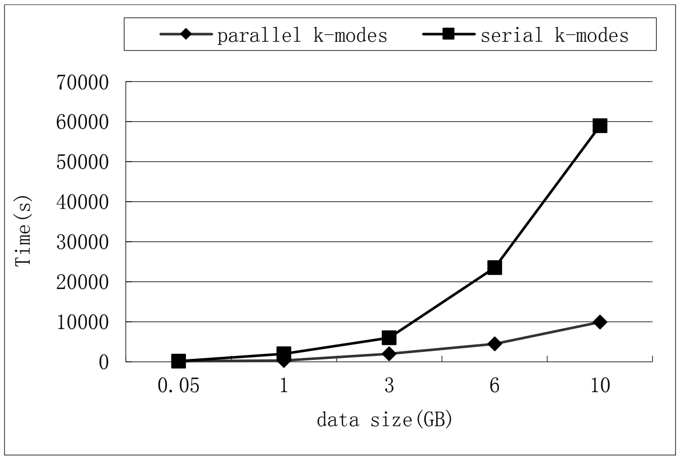

Fig. 2. Comparison of parallel *k*-modes and serial *k*-modes

Experiments show that, compared with serial *k*-modes, parallel *k*-modes has no obvious advantage when dealing with small data, but it greatly shorten the computing time and archive good speedup ratio when dealing with large-scale data.

VII. Summary

We improve the process of k-modes: when allocating categorical objects to clusters, the number of each attribute item in clusters is updated so that we can compute the new modes of clusters after reading the whole dataset once. In order to make k-modes capable for large-scale categorical data, we implement k-modes on Hadoop using MapReduce parallel computing model. Experiments show that, parallel k-modes archive good speedup ratio when dealing with large-scale categorical data.

The implementation of parallel *k*-modes has been released to the open source community (https://github.com/j2cms/dog).

ACKNOWLEDGMENT

I would like to express my gratitude to all those who have helped me during the writing of this paper. I gratefully acknowledge the help of my supervisor. I do appreciate his patience, encouragement, and professional instructions during the paper's writing.

REFERENCES

[1] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques, Third Edition*: Morgan kaufmann, 2011.

[2] A. Rodriguez and A. Laio, "Clustering by fast search and find of density peaks," *Science,* vol. 344, pp. 1492-1496, 2014.

[3] A. Agresti, *Categorical data analysis*: John Wiley & Sons, 2014.

[4] A. Chaturvedi, P. E. Green, and J. D. Caroll, "K-modes clustering," *Journal of Classification,* vol. 18, pp. 35-55, 2001.

[5] Z. Huang, "A Fast Clustering Algorithm to Cluster Very Large Categorical Data Sets in Data Mining," in *DMKD*, 1997, pp. 281-297.

[6] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM,* vol. 51, pp. 107-113, 2008.

[7] C. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A. Y. Ng, K. Olukotun, "Map-reduce for machine learning on multicore," *Advances in neural information processing systems,* vol. 19, p. 281, 2007.

[8] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, 2010, pp. 1-10.

[9] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, 1967, pp. 281-297.

[10] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM SIGOPS Operating Systems Review*, 2003, pp. 29-43.

[11] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS),* vol. 26, p. 4, 2008.

[12] P. S. Pacheco, *Parallel programming with MPI*: Morgan Kaufmann, 1997.

[13] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *Computational Science & Engineering, IEEE,* vol. 5, pp. 46-55, 1998.

[14] D. L. Eager, J. Zahorjan, and E. D. Lazowska, "Speedup versus efficiency in parallel systems," *Computers, IEEE Transactions on,* vol. 38, pp. 408-423, 1989.