A Parallel Clustering Algorithm with MPI – MKmeans

Jing Zhang
School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, China
hfzjwjl@gmail.com

Gongqing Wu, Xuegang Hu, Shiying Li and Shuilong Hao
School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, China
Email: {wugq, jsjxhuxg}@hfut.edu.cn, {hfutlsy, virtualsky607}@gamil.com

***Abstract*—Clustering is one of the most popular methods for exploratory data analysis, which is prevalent in many disciplines such as image segmentation, bioinformatics, pattern recognition and statistics etc. The most famous clustering algorithm is K-means because of its easy implementation, simplicity, efficiency and empirical success. However, the real-world applications produce huge volumes of data, thus, how to efficiently handle of these data in an important mining task has been a challenging and significant issue. In addition, MPI (Message Passing Interface) as a programming model of message passing presents high performances, scalability and portability. Motivated by this, a parallel K-means clustering algorithm with MPI, called MKmeans, is proposed in this paper. The algorithm enables applying the clustering algorithm effectively in the parallel environment. Experimental study demonstrates that MKmeans is relatively stable and portable, and it performs with low overhead of time on large volumes of data sets.**

***Index Terms*—clustering, K-means algorithm, MPI, parallel computing**

I. INTRODUCTION

Clustering is a method of unsupervised learning and a common technique for data analysis used in many disciplines, including image segmentation, bioinformatics, pattern recognition and statistics etc [1].

Clustering is the process of grouping objects into subsets that have meaning in the context of a specific problem. Unlike classification, clustering does not rely on predefined classes, and no information is provided about the "right answer". K-means is a well-known clustering algorithm for its simplicity and easy implementation. K-means was ranked second of top 10 algorithms in data mining by the ICDM Conference in October 2006, while C4.5 was ranked first [2]. Compared with other clustering algorithms, K-means algorithm has three major advantages covering simple implementation, efficient when handling a large data sets and a solid theoretical foundation based on the greedy optimization of Voronoi partition [3].

With the development of information technology, data volumes are becoming increasingly mass, which makes clustering large-scale and complicated data a challenging problem. Efficient parallel clustering algorithms and implementation techniques are the key to meet the scalability and performance requirements entailed in such scientific data analysis. Cloud computing is a form of technology that uses the Internet to maintain data and application. It is based on the development of distributed computing, parallel computing and grid computing [4]. Meanwhile, "Top 10 obstacles and opportunities for Cloud Computing" are presented in [5], which predict that cloud computing will grow and believe that computing, storage and networking must all focus on the horizontal scalability of virtualized resources rather than the performance of single node.

MPI is a library specification for message passing, which is proposed as a standard programming model. MPI is an application programmer interface of message passing, together with protocol and semantic specifications for how its features must behave in any implementation [6]. The MPI interface is used to provide essential virtual topology, synchronization, and communication functionality between a set of processes in a language-independent way. MPI is an API library consisting of hundreds of function interfaces called by computers such as computer clusters communicate with one another in the parallel development environment. MPI aims to maintain the high performance, scalability and portability, and it remains the dominate model for high-performance computing today.

Many scholars focus on the studies in the fields of parallel and distributed clustering. More specifically, (1) Rasmussen et al. [7] suggested that the parallel processing using an array processor like the DAP (Distributed Array Processor) can provide significant speedups over serial processing for the hierarchic agglomerative cluster analysis of large data sets. (2) Olson et al. [8] considered parallel algorithms for hierarchical clustering using several inter-cluster distance metrics and parallel computer architectures. (3) Zhao et al. [9] proposed a fast parallel K-means clustering algorithm based on MapReduce. The proposed algorithm can scale well and efficiently to process large data sets on commodity hardware.

However, our contributions in this paper are as follows:

doi:10.4304/jcp.8.1.10-17

- MKmeans, a parallel K-means clustering algorithm with MPI, is proposed.
- A simple merging algorithm in MKmeans, used to merge the generated centroid sets into a final centroid set with a greedy style, is proposed.
- The configuration of MPI parallel development platform based on open resource project Eclipse and MPICH2 in Windows is implemented. Our method can be ported to Linux or other platforms.
- Experimental results illustrate that our MKmeans algorithm is relatively stable and portable, it improves the time performance of clustering on large data sets. Meanwhile, MPI is quite appropriate for the parallel and distributed computing environment.

The rest of this paper is organized as follows. Section II gives the related work. Section III mainly presents our MKmeans algorithm in detail. Section IV shows our experimental results and performance evaluation. Finally, Section V concludes the paper and outlines our future work.

II. Related Work

Related work including K-means clustering algorithm, MPI and Weka are involved in this section. More precisely, an outline on K-means is summarized in Section *A*, the concept of MPI is given in Section *B*, brief introduction of MPI functions is given in Section *C*, the messing passing process of MPI is described in Section *D* and a classical experimental tool Weka is described in Section *E*.

A. K-means Algorithm

The most popular and the simplest partitional algorithm is K-means [10]. K-means has a rich and diverse history as it was independently discovered in different scientific fields by Steinhaus (1956), Lloyd (proposed in 1957, published in 1982), Ball and Hall (1965), and MacQueen (1967) [1]. K-means is a method of data analysis, which is easy to implement and apply even on large data sets. As such, it has been successfully used in various fields, ranging from computer vision, statistics to market segmentation.

K-means clustering algorithm is simple and fairly fast. It is initialized from some random and approximate solution. The K-means clustering is to find cluster centers that minimize the sum of squared distances from each data point being clustered to its closest cluster center. The objective is to partition N data objects into K clusters ($K<N$) such that the objects in the same cluster are as similar as possible and as dissimilar as possible in different clusters. K-Means algorithm inputs parameter K, and then divides data objects N into K clusters to make the obtained clustering to meet the conditions, which the similarity degree of the same clustering objects is higher, and that of the different clustering objects is lower. Clustering similarity degree is calculated by "central object" (gravity center) to be obtained by the mean of objects in each cluster. The resulting clusters have high explanatory and precision.

Given a set of N objects, then partition N objects into K clusters, the time complexity of K-means algorithm is $\boldsymbol{O}(NKt)$, where t is the number of iterations, generally, $K<<N$, $t<<N$.

Main steps of the K-means algorithm are described by Jain and Dubes [11] as follows:

1) Select an initial partition with K clusters; repeat steps 2 and 3 until cluster memberships stabilize.
2) Generate a new partition by assigning each pattern to its closest cluster center.
3) Compute new cluster centers.

In recent years, with the development of computer technology, some methods of cluster analysis are developed quickly and applied broadly. There are several variations of K-means algorithm:

- The fuzzy C-means clustering is a soft version of K-means, where each point has a degree of belonging to cluster, as in fuzzy logic.
- The EM (expectation-maximization) algorithm is a generalization of K-means, which is an iterative method to maintain probabilistic assignments to clusters, instead of deterministic assignments in K-means.
- Some seeding methods are proposed for choosing the initial values (seeds) for the K-means clustering algorithm. K-means++ [12] is one of the methods to successfully overcome the problems associated with the sometimes poor clustering results by the standard K-means clustering algorithm.
- The kd-tree (k-dimensional tree) is a space-partitioning data structure for organizing points in a k-dimensional space. Kanungo et al. [13] present a simple and efficient implementation of K-means clustering algorithm, called the filtering algorithm. This algorithm is easy to implement, requiring a kd-tree as the only major data structure, which can speed up each step of K-means.
- The coreset is a small weighted set of points that approximates the original point set with respect to the considered problem. Frahling et al. [14] develop an efficient implementation for a K-means clustering algorithm. It uses coresets to speed up the algorithm. The main strength of the algorithm is that it can quickly determine clusters of the same point set for many values of K. This is necessary in many applications, since, typically, one does not know a good value for K in advance.
- The Minkowski metric deals with the problem of noise features by assigning weights to each feature for each cluster. Amorim et al. [15] represent another step in overcoming a drawback of K-means, its lack of defense against noisy features, using feature weights in the criterion. The Weighted K-means method is extended to the corresponding Minkowski metric for measuring distances. Under Minkowski metric the feature weights become intuitively appealing feature rescaling factors in a conventional K-means criterion.

Meanwhile, there are still several drawbacks of those variation algorithms. Despite its simplicity and its linear time, the time complexity of a serial K-means algorithm remains expensive when it is applied to a problem of large size of multi-dimensional vector. Therefore, it is of vital importance and urgent necessity to think about how to apply K-means clustering algorithm to the distributed environment for large scale of data sets.

B. The Concept of MPI

Most popular high-performance parallel architectures used in the parallel programming environment are divided into two classes: message passing and shared storage. The cost of message passing parallel processing is larger, suitable for large-grain process-level parallel computing. Compared with other parallel programming environment, message passing has good portability, supported by almost all parallel environments. Meanwhile, it has good scalability and complete asynchronous communication function, which can well decompose tasks according to the requirements of users, organize data exchange between different processes and is applied to scalable parallel algorithms.

MPI is an interface mode widely used in various parallel clusters and network environments based on a variety of reliable message passing libraries.

MPI is a message passing parallel programming standard used to build highly reliable, scalable and flexible distributed applications, such as workflow, network management, communication services.

MPI is a language-independent communications protocol. FORTRAN, C and C++ can directly call the API library. The goals of MPI are high performance, scalability and portability.

MPI is a library specification for message passing, not a language. Message Passing Interface is a standard developed by the Message Passing Interface Forum (MPIF). MPI is a standard library specification designed to support parallel computing in a distributed memory environment. The first version (MPI-1) was published in 1994 and the second version (MPI-2) was published in 1997 [16]. Both point-to-point and collective communication are supported.

MPICH is an available and portable implementation of MPI, a standard for message passing used in parallel computing. MPI has become the most popular message passing standard for parallel programming. There are several MPI implementations among which MPICH is the most popular one. MPICH1 is the original implementation of MPICH that implements the MPI-1standard. MPICH2 is a high-performance and widely portable implementation of the MPI standard (both MPI-1 and MPI-2). The goals of MPICH2 are to provide an implementation of MPI that efficiently supports different computation and communication platforms including commodity clusters, high-speed networks and proprietary high-end computing systems.

The standards of MPI are as follows [17]:

- Point-to-point communication
- Collective operations
- Process groups.
- Communication contexts
- Process topologies
- Bindings for FORTRAN 77 and C
- Environmental management and inquiry
- Profiling interface

C. MPI Functions

MPI is a library with hundreds of function-calling interfaces, and FORTRAN, C language and C++ can directly call these functions. Many parallel programs can be written with just six basic functions, almost complete all of the communication functions.

Table I illustrates the basic functions. MPI_Init() initializes the MPI environment and assigns all spawned processes; MPI_Finalize() terminates the MPI environment; MPI_Comm_size() finds the number of processes in a communication group; MPI_Comm_rank() gives the identification number of a process in a communication group; MPI_Send() sends message to the destination process of rank *dest* and MPI_Recv() receives message from the specified process of rank *source*.

TABLE I
MPI BASIC FUNCTIONS

Function	***Functionalities***
MPI_Init	Initialization
MPI_Finalize	Termination
MPI_Comm_size	Access to the number of processes
MPI_Send	Send
MPI_Recv	Receive
MPI_Comm_rank	Access to the identification number of a process

D. The Messing Passing Process of MPI

MPI is a parallel programming standard based on message passing, whose function is to exchange information, coordinate and control the implementation steps with the definition of program grammar and semantics in the core library by sending messages between the concurrent execution parts.

First all of the MPI programs contain “mpi.h” header file, and then complete the initialization of the program by MPI_Init(), after that, establish process topology structure and new communicator and call the functions and applications to be used for each process, finally use MPI_Finalize() to terminate each process.

The parallel program design flow of message passing process is shown in Fig. 1.

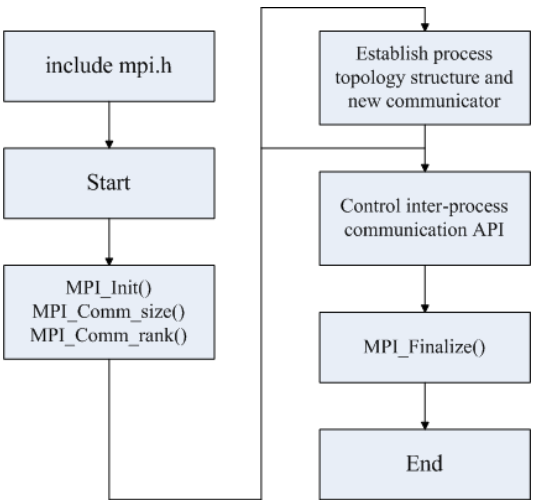

Figure 1. Flow of message passing process

E. Weka

Weka (Waikato Environment for Knowledge Analysis) is a comprehensive suite of Java class libraries that implement many state-of-the-art machine learning and data mining algorithms [18]. Weka contains implementations of algorithms for classification, clustering, association rule mining, along with graphical user interfaces and visualization utilities for the data exploration and algorithm evaluation. Weka is a significant tool to bring machine learning technology into the workplace.

Methods of clustering do not seek rules that predict a particular class, but rather try to divide the data into natural groups of "clusters". K-means algorithm is the most widely used clustering algorithm. SimpleKMeans from Weka is a simplified version of K-means [19]. Clustering data using the K-means algorithm can use either the Euclidean distance (default) or the Manhattan distance. If the Manhattan distance is used, centroids are calculated as the component-wise median rather than the mean.

III. MKMEANS – A PARALLEL K-MEANS ALGORITHM WITH MPI

In this section, we will propose an implementation of a parallel K-means algorithm based on MPI, called MKmeans. Before introducing our MKmeans algorithm, we first give a short description of the SKmeans (Sequential K-means) algorithm in Section *A*, because MKmeans is composed of SKmeans and MPI, and then we give the detailed description of MKmeans in Section *B*, Merge function is shown in Section *C*.

A. SKmeans Algorithm

The clustering algorithm K-means is one of the most popular partitioning clustering algorithms. The main idea in the algorithm is to define *K* centroids (one for each cluster). These centroids should be placed in a cunning way because different locations cause different results [3]. In the SKmeans algorithm, the objective function *J* is defined as in (1). SKmeans aims to minimize the objective function, i.e., a squared error function. In (1), *J* refers to the distance index of *N* data objects from the corresponding cluster centroids, x_n ($1 \leqslant n \leqslant N$) indicates a data point, and c_k ($1 \leqslant k \leqslant K$) specifies the cluster centroid. $\|x_n - c_k\|^2$ is the distance measure between x_n and c_k. In (2), μ_k ($1 \leqslant k \leqslant K$) refers to the mean of data points that belong to cluster *k*, and N_k indicates the number of objects belong to cluster *k*.

In our SKmeans algorithm, the number of clusters *K* is a user-specified parameter. First, read *N* objects from the input file. The initial *K*-centroids are randomly selected, defined as μ_k ($1 \leqslant k \leqslant K$). Second, the SKmeans algorithm iteratively assigns each object into the corresponding cluster by the minimum distance. When all objects are assigned, update the *K* centroids. This process will be repeated until the user-specified threshold is met. The SKmeans algorithm is shown in Fig. 2.

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} \|x_n - c_k\|^2 . \quad (1)$$

$$\mu_k = \frac{1}{N_k} \sum_{n \in c_k} x_n . \quad (2)$$

SKmeans algorithm

Input: number of clusters *K*, number of data objects *N*

Output: *K* centroids

1: Read *N* objects from the file;
2: Randomly select *K* points as the initial cluster centroids, denoted as μ_k ($1 \leqslant k \leqslant K$);
3: Calculate *J* in Formula (1), denoted by *J'*;
4: Assign each object *n* ($1 \leqslant n \leqslant N$) to the closest cluster;
5: Calculate new centroid of each cluster μ_k in Formula (2) ;
6: Recalculate J in Formula (1);
7: Repeat steps 3-6 until *J'*- *J* < threshold;
8: Output the clustering results: *K* centroids;

Figure 2. SKmeans clustering algorithm

B. MKmeans Algorithm

Fig. 3 shows the processing flow of our MKmeans algorithm, which is the key algorithm in our work. This algorithm utilizes K-means and the MPI parallel framework, it is hence simple and portable.

All initialization is implemented by the MPI_Init function, which is the first call of MPI program. It is the first executable statement of the MPI program. To start the MPI environment, it means the beginning of the parallel

codes. The MPI_Finalize function is the last call of MPI program, and it ends the running of MPI program. It is the last executable statement of MPI program, otherwise, results of the procedure is unpredictable. MPI_Finalize symbolizes the end of the parallel codes.

In the first step, read *N* objects from the input file, and partition *N* data objects evenly among all processes, randomly select *K* points as the initial cluster centroids, and then iteratively assigns each object into the corresponding cluster by the minimum distance according to (1). This process will be repeated until the user-specified threshold is met. When all objects are assigned, generate a final centroid set *Centroid* by Function *Merge* and output the clustering results.

In our MKmeans algorithm, the parallelism is implemented by the data parallelism. The parallel processing in MKmeans is consistent with that of SKmeans. Data objects are evenly partitioned in all processes and cluster centroids are replicated. The global operation for all cluster centroids is performed at the end of each iteration in order to generate new cluster centroids. Finally, output the clustering results: *K* centroids, I/O time and clustering time.

MKmeans algorithm

Input: number of clusters *K*, number of data objects *N*

Output: *K* centroids

1: MPI_Init// start the procedure;
2: Read *N* objects from the file;
3: Partition *N* data objects evenly among all processes, and assume that each process has *N'* data objects;
4: For each process, install steps 5-11;
5: Randomly select *K* points as the initial cluster centroids, denoted as μ_k ($1 \leqslant k \leqslant K$);
6: Calculate *J* in (1), denoted as *J'*;
7: Assign each object *n* ($1 \leqslant n \leqslant N$) to the closest cluster;
8: Calculate the new centroid of each cluster μ_k in (2) ;
9: Recalculate *J* in (1);
10: Repeat steps 6-9 until *J'*- *J* < threshold;
11: Generate the cluster id for each data object;
12: Generate new cluster centroids according to the clustering results of all processes at the end of each iteration;
13: Generate a final centroid set *Centroid* by Function *Merge* and output the clustering results: *K* centroids;
14: MPI_Finalize// finish the procedure;

Figure 3. MKmeans clustering algorithm

C. Merge Function

We assume that processes are *n*, which MKmeans generates *n* new data sets from the original data set. Since each data set uses K-means algorithm to generate *K* centroids, we can get *n* centroid sets from $Centroid_1$ to $Centroid_n$. Therefore, our goal is to merge the *n* centroid sets into a final centroid set. In our MKmeans algorithm, a simple merging algorithm is applied to ensemble with a greedy style, shown in Fig. 4.

Function Merge

Input: *n* centroid sets from $Centroid_1$ to $Centroid_n$ (*n*K* centroids)

Output: a centroid set *Centroid* (*K* centroids)

1: Initialize a centroid set *Centroid* as empty;
2: If any centroid set $Centroid_i$ ($i=1,\ldots,n$) is empty, exit and the final centroid set is *Centroid*, otherwise, go to step 3;
3: Find a vector of centroids ($c_1,\ldots, c_K$) with the minimum inner distance, which is defined as in (3), and then delete c_i from $Centroid_i$, in addition, add $\bar{c}$ into *Centroid*. Go to step 2;
4: Output *K* centroids;

Figure 4. Merge function

The Function *Merge* aims to find a vector of centroids ($c_1,\ldots, c_K$) with the minimum inner distance. In addition, the Euclidean distance is used to calculate the inner distance, denoted as *Distance*. In (3), c_i comes from centroid set $Centroid_i$ ($i=1,\ldots,K$) and $\bar{c}$ is the centroid.

$$D(c_1,\ldots,c_K) = \sum_{i=1}^{K} Distance\ (c_i, \bar{c}) . \qquad (3)$$

The merging process is shown in Fig. 5. At first, initialize the finally centroid set *Centroid* as empty, that is to record the results of *K* centroids, if $Centroid_i$ ($i=1,\ldots,n$) is empty, the algorithm exits and the final centroid set is *Centroid*, otherwise, find a vector of centroids ($c_1,\ldots, c_K$) with the minimum inner distance according to (3), and then delete c_i from $Centroid_i$, in addition, add $\bar{c}$ into *Centroid*, finally, output the centroid set *Centroid*.

It is obvious that the calculation of each inner distance is independent. Therefore, all calculations can be performed in parallel. The Function *Merge* merges *n*K* centroids into new *K* centroids, which are the final centroids.

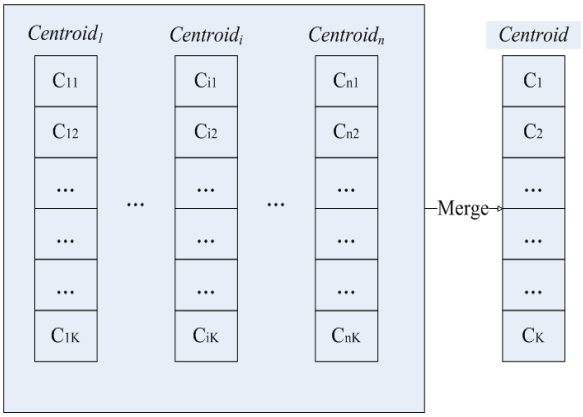

Figure 5. Merging process

TABLE II
DESCRIPTION OF DATA SETS

Code Number of Data Sets	Name of Data Sets(.arff)	Size(KB)	Number of Instances
1	zoo	14.0	101
2	breast-cancer	28.7	286
3	credit-a	33.5	690
4	vowel	90	990
5	segment	298	2310
6	hypothyroid	303	3772
7	letter	703	20000

IV. EXPERIMENTS

In this section, we first give the experimental environment in Section *A*. Second, we give the description of experimental data sets in Section *B*. Last, we give the experimental results and analysis in Section *C*.

A. Experimental Environment

The hardware platform in this paper uses a PC with the configuration: Intel Xeon 5110 dual-core processor, 2GB RAM, 250GB hard driver; the software environment uses the following configuration: the operation system is Windows XP Professional Service Pack 3, the parallel and distributed environment is the Windows version of MPICH2, Java development platform is the JDK 1.6; Network environment is 100M- LAN.

In terms of aforementioned platform, eclipse SDK 3.2.1 is used to develop procedures. Considering the fairness of comparison, the configuration of MPI parallel development platform is based on open resource project Eclipse in Windows, and the experimental platform has a C/C++ complier based on MinGW (Minimalist GNU for Windows).

B. Data Sets

All experimental data sets are selected from the UCI Machine Learning Dataset Repository [20]. The information of all data sets is illustrated as shown in Table II.

In this table, seven testing data sets are listed corresponding to the number of instances. As the number of instances increases, the space consumption of data sets also increases, denoted as size.

C. Experimental Results and Analysis

In our experiments, the time cost is the key performance. The I/O time and clustering time are calculated respectively in MKmeans and SKmeans. To

TABLE III
COMPARISON RESULTS BETWEEN MKMEANS AND SKMEANS

Code Number of Data Sets	MKmeans(ms)			SKmeans(ms)		
	I/O	Clustering	Total	I/O	Clustering	Total
1	12.5	0.2	12.7	0.0	0.0	0.0
2	22.2	0.2	22.4	0.0	0.0	0.0
3	48.6	7.6	56.2	15.6	0.0	15.6
4	72.0	8.8	80.8	15.6	15.6	31.2
5	179.3	45.4	224.7	46.9	46.9	93.8
6	251.3	57.9	309.2	31.3	62.5	93.8
7	1268.2	268.8	1537.0	171.9	281.3	453.2

TABLE IV
SPEEDUP IN SKMEANS AND MKMEANS

Code Number of Data Sets	MKmeans(ms)	SKmeans(ms)	Speedup
1	0.2	0.0	0
2	0.2	0.0	0
3	7.6	0.0	0
4	8.8	15.6	1.773
5	45.4	46.9	1.033
			1.0

reflect the fairness and authenticity of the proposed algorithms, the number of processes is 1 in MKmeans.

Table III reports the results of the aforementioned two algorithms, including the I/O time, the clustering time and the total running time. Fig. 6 compares the clustering running time of seven data sets in our algorithm with other different algorithms directly. From this figure, we can see that with the increasing of the size of data sets, the clustering time of MKmeans is slightly lower than that of SKmeans. However, the total time cost of MKmeans is higher than that of SKmeans on each testing data set, a preliminary analysis is that the I/O time occupies a large proportion. If our algorithm MKmeans is run in a large cluster, or as the number of processes increases, the experimental results will be more significant. In sum, we conclude that our MKmeans algorithm enables improving the time performance of clustering on large-scale data sets and MPI is quite appropriate for the parallel and distributed computing environment.

Table IV illustrates speedup according to MKmeans and SKmeans for comparing the time performance. Speedup can be calculated in (4), which MCT_i indicates the clustering cost time of running MKMeans in data set i ($1 \leqslant i \leqslant 7$), SCT_i indicates the clustering cost time of running SKMeans in data set i. From this table we can see that with the size of data set increasing, the time performance of MKmeans is better than that of SKmeans. In addition, if the scale of data set is small, it is not a good choice to use MKmeans because the time of dividing the data set and assigning tasks into each process occupies a certain proportion.

$$speedup = \frac{SCT_i}{MCT_i}. \quad (4)$$

Table V illustrates the total time overheads of SKmeans and WKmeans. To make a convenient comparison in the experiments, WKmeans is based on the SimpleKMeans algorithm from Weka. In Fig. 7, we can see that the time cost of WKmeans is significantly higher than that of SKmeans, and SKmeans performs much more stable on the overhead of time in the large data sets compared to WKmeans. MKmeans and SKmeans share the similar idea, it is hence to conclude that MKmeans is also a stable clustering algorithm.

TABLE V
OVERHEADS OF TIME IN SKMEANS AND WKMEANS

Code Number of Data Sets	*SKmeans(ms)*	*WKmeans(ms)*
1	0.0	78
2	0.0	78
3	15.6	156
4	31.2	219
5	93.8	562
6	93.8	625
7	453.2	6297

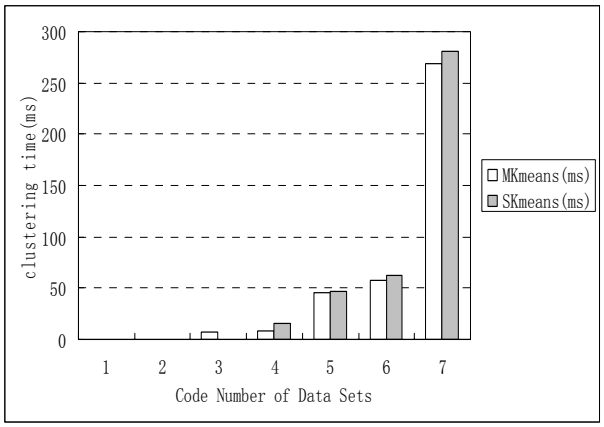

Figure 6. Clustering time of MKmeans and SKmeans

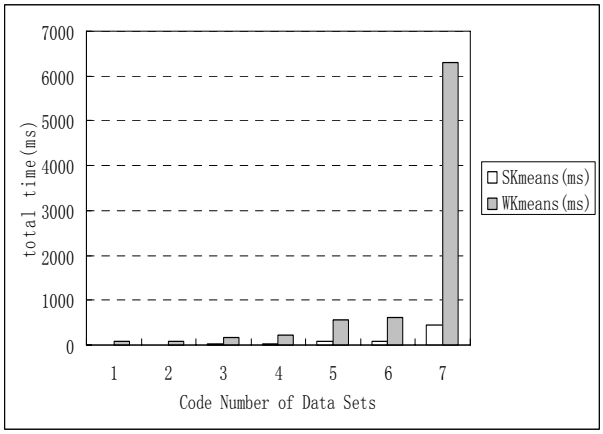

Figure 7. The total time overheads of SKmeans and WKmeans

V. CONCLUSIONS

We proposed a parallel K-means algorithm based on MPI (called MKmeans) in this paper. Meanwhile, the configuration of MPI parallel development platform based on open resource project Eclipse and MPICH in Windows is implemented, whose ideas and methods can be ported to Linux or other platforms. Experimental results show that MKmeans is relatively stable and portable, and it is efficient in the clustering on large data sets.

However, how to study the cluster validity of MKmeans with theoretical analysis, how to study the effect of clustering performance varying with the number of processes, how to compare the clustering performance in MPI and Map-Reduce are our future work [21].

ACKNOWLEDGMENT

We wish to thank Peipei Li for her valuable comments on an earlier draft. This work was supported in part by the National Basic Research Program of China (973 Program) under grant 2009CB326203 and the Natural Science Foundation of Anhui Province of China under grant 090412044.

REFERENCES

[1] A. Jain, "Data clustering: 50 years beyond K-means," *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, June 2010.

[2] X. Wu, V. Kumar, Ross, J. Ghosh, Q. Yang, H. Motoda, G. Mclachlan, A. Ng, B. Liu, P. Yu, Z.-H. Zhou, M. Steinbach, D. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1-37, January 2008.

[3] J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations," *Berkeley: University of California Press*, pp. 281-297, 1967.

[4] B. Hayes, "Cloud computing," *Communications of the ACM*, vol. 51, no. 7, pp. 9-11, July 2008.

[5] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinsli, G. Lee, D. Patterson, A. Rabkin, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50-58, April 2010.

[6] E. Lusk, N. Doss, A. Skjellum, "A High-Performance, Portable Implementation of the MPI Message Passing Interface," *Parallel Computing*, vol. 22, pp. 789-828, 1996.

[7] E. Rasmussen, P. Willett, "Efficiency of hierarchic agglomerative clustering using the ICL distributed array processor," *Journal of Documentation*, vol. 45, March 1989.

[8] C. Olson, "Parallel algorithms for hierarchical clustering," *Parallel Computing*, vol. 21, no. 8, pp. 1313-1325, August 1995.

[9] W. Zhao, H. Ma, Q. He, "Parallel K-Means Clustering Based on MapReduce," *Cloud Computing*, vol. 5931, pp. 674-679, 2009.

[10] S. Kantabutra, A. Couch, "Parallel K-means Clustering Algorithm on NOWs," *Technical Journal*, vol. 1, no. 6, 2000.

[11] A. Jain, R. Dubes, "Algorithms for Clustering Data," *Prentice Hall*, 1988.

[12] D. Arthur, S. Vassilvitskii, "k-means++: the advantages of careful seeding," *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027-1035, 2007.

[13] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu, "An Efficient K-means Clustering Algorithm: Analysis and Implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 7, pp. 881-892, 2002.

[14] G. Frahling, C. Sohler, "A Fast K-means Implementation Using Coresets," *Proceedings of the Twenty-second Annual Symposium on Computational Geometry*, pp. 135-143, 2006.

[15] R. Amorim, B. Mirkin, "Minkowski metric, feature weighting and anomalous cluster initializing in K-Means clustering," *Pattern Recognition*, vol. 45, no. 3, pp. 1061-1075, 2012.

[16] W. Gropp, E. Lusk, "Implementing MPI: the 1994 MPI Implementors' Workshop," *Proceedings of Scalable Parallel Libraries Conference*, pp. 55-59, 2002.

[17] Y. Aoyama, J. Nakano, "RS/6000 SP: Practical MPI Programming," *International Technical Support Organization*, August 1999.

[18] I. Witten, E. Frank, L. Triqq, M. Hall, G. Holmes, S. Cunninqham, "Weka: Practical Machine Learning Tools and Techniques with Java Implementations," *Proceedings of ICONIP/ ANZIIS/ ANNES99 Future Directions for Intelligent Systems and Information Sciences*, 1999.

[19] E. Frank, M. Hall, G. Holmes, R. Kirkby, B. Pfahringer, I. Witten, L. Trigg, "Weka-A Machine Learning Workbench for Data Mining," *Data Mining and Knowledge Discovery Handbook*, pp. 1269-1277, 2010.

[20] C. Blake, E. Keogh, C. Merz, "UCI Repository of machine learning databases," *Department of Information and Computer Science*, *University of California*, 1998.

[21] J. Zhang, G. Wu, H. Li, X. Hu, X. Wu, "A 2-Tier Clustering Algorithm with Map-Reduce," *Proceedings of the 5th ChinaGrid Annual Conference (ChinaGrid'10)*, pp. 160-166, July 2010.

Jing Zhang, born in 1987. She is currently a Ph.D. Student led by Professor X. Hu. She received her B.S. degree in computer science and technology from School of Computer Science and Information, Heifei University of Technology, China in 2009. Her main research interests are high-performance data mining and artificial intelligence.

Gongqing Wu, born in 1975. He is currently an associate professor of School of Computer Science and Information Engineering at Hefei University of Technology. He received his M.S. degree from Department of Computer Science and Technology, University of Science and Technology of China (USTC) in 2003. His main research interests are data mining and Web intelligence.

Xuegang Hu, born in 1961. He is currently a professor of School of Computer Science and Information Engineering at Hefei University of Technology. He received his M.S. and Ph.D. degrees in computer application technology from School of Computer Science and Information, Heifei University of Technology, China in 1988 and 2000, respectively. His main research interests include data mining, machine learning, knowledge engineering and artificial intelligence.

Shiying Li, born in 1986. He is currently a graduate student led by Professor X. Hu. He received his B.S. degree in information and computing science from School of Mathematics, Heifei University of Technology, China in 2009. His main research interests are Web data mining and user interest modeling.

Shuilong Hao, born in 1986. He is currently a graduate student led by Professor X. Hu. He received his B.S. degree in information and computing science from School of Mathematics, Heifei University of Technology, China in 2009. His main research interests are data mining and user interest modeling.