1 JOURNAL OF COMPUTERS, VOL. 8, NO. 1, JANUARY 2013

Simplex Method Implementation with MPI

Tulshi Chandra Das-BSSE0811, Maloy Kanti Sarkar-BSSE0834

Institute of Information Technology, University of Dhaka

Email: bsse0811@iit.du.ac.bd, bsse0834@iit.du.ac.bd

**Abstract- The simplex method is frequently the most efficient method of solving linear programming (LP) problems. We review previous attempts to parallelize the simplex method in relation to efficient serial simplex techniques and the nature of practical LP problems. For the major challenge of solving general large sparse LP problems, there has been no parallelization of the simplex method that offers significantly improved performance over a good serial implementation. However, there has been some success in developing parallel solvers for LPs that are dense or have particular structural properties**.

**Keywords: linear programming, simplex method, parallel computing**

1. 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

Simplex method is an algorithm which is used to solve the linear programming problems effeciently.

We consider the computation of econometric estimators given by the optimization problem: minx z = 8x1 +9x2, where x1,x2 are the variables.

2x1 + 3x2 ≤ 50

2x1 + 6x2 ≤ 80

3x1 + 3x2 ≤ 70

Simplex method works on standard form of the equations. So, we have to trans-form the equations to standard form: z = = 8x1 +9x2 + s1 + s2 + s3 2x1 + 3x2 + s1 = 50

2x1 + 6x2 + s2 = 80

3x1 + 3x2 + s3 = 70

Here s1,s2 and s3 are slack variable.

The Starting Tableau

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Basic | Z | X1 | X2 | S1 | S2 | S3 | Solution |
| Z | 1 | -8 | -9 | 0 | 0 | 0 | 0 |
| S1 | 0 | 2 | 3 | 1 | 0 | 0 | 50 |
| S2 | 0 | 2 | 6 | 0 | 1 | 0 | 80 |
| S3 | 0 | 3 | 3 | 0 | 0 | 1 | 70 |

For optimization problems without closed form solutions, an iterative procedure is typically used. These algorithms successively evaluate the objective function z at different trial parameter vectors until a parameter vector achieves a convergence criterion. Let bi be the RHS of the ith row. Let aij be the coefficient of the entering variable xj in the ith row. The following “minimum ratio test” decides the leaving variable.

The optimization steps are:

1. Determining the pivot row and pivot element using aij and xj
2. Make a non-basic variable to basic variable
3. change the pivot row
4. Then change all other element in the table

## The step 1 and 2 and 3 are serialized process and not costly. The step 4 is more costly and time consuming. We will parallelize the step 4.

## There are two main sources of computation time in these numerical procedures. First, for each trial vector of parameters, there is the computational cost of evaluating the objective function at this vector of parameters. Where the objective function involves simulation or the solution of a complex behavioral model, such as a dynamic programming model, the objective function level computation costs can be large. A second source of computational costs is that for a high dimensional and continuous parameter space, a large number of vectors of parameters may need to be tried before the convergence criterion is achieved. This parameter level computation cost is increasing in the number of parameters, which is generally related to the complexity of the underlying behavioral model.

1. Background

*A. The Concept of MPI*

Most popular high-performance parallel architectures used in the parallel pro-gramming can be 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 en-vironment, message passing has good portability, supported by almost all paral-lel environments. Meanwhile, it has good scalability and complete asynchronous communication function, which can well decompose tasks according to the require-ments of users, organize data exchange between dierent processes and is applied to scalable parallel algorithms. MPI is an interface mode widely used in various parallel clusters and network environ- ments based on a variety of reliable mes-sage passing libraries. MPI is a message passing parallel programming standard used to build highly reliable, scalable and exible distributed applications, such as work ow, network management, communication services. MPI is a language-independent communications pro- tocol. FORTRAN, C and C++ can directly call the API library. The goals of MPI are high performance, scalability and portability. MPI is a library spec- ication 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 specication designed to support parallel computing in a distributed memory environment. The rst version (MPI-1) was published in 1994 and the second version (MPI-2) was published in 1997[1].Both point-to-point andcollective communication are supported. MPICH is an available and portable im-plementation of MPI, a standard for message passing used in parallel computing.MPI has become the most popular message passing standard for parallel program-ming. There are several MPI implementations among which MPICH is the most popular one. MPICH1 is the original im- plementation of MPICH that implementsthe 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 implementa- tion of MPI that eciently supports dierent computationand communication platforms including commodity clusters, high-speed networks and proprietary high-end computing systems. The standards of MPI are as follows

[2]: -Point-to-point communication

-Collective operations

-Communication contexts

-Process groups.

-Process topologies

-Bindings for FORTRAN 77 and C

-Environmental management and inquiry

-Proling interface

2

*B. 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*.

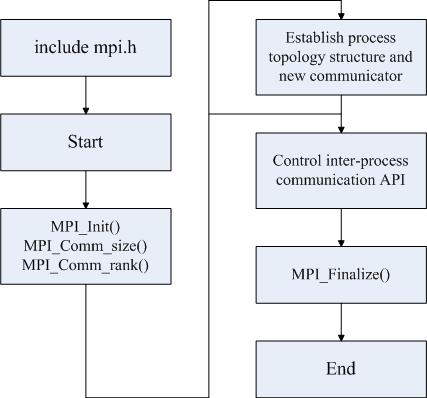
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | TABLE I | |  |
|  |  | MPI BASIC FUNCTIONS | | |  |
|  |  |  |  |  |  |
|  | ***Function*** |  | ***Functionalities*** |  |  |
|  | MPI\_Init |  | Initialization |  |  |
|  | MPI\_Finalize |  | Termination |  |  |
|  | MPI\_Comm\_size |  | 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.



© 2013 ACADEMY PUBLISHER

JOURNAL OF COMPUTERS, VOL. 8, NO. 1, JANUARY

* + Process groups.
  + Communication contexts
  + Process topologies
  + Bindings for FORTRAN 77 and C
  + Environmental management and inquiry
  + Profiling interface

© 2013 ACADEMY PUBLISHER

|  |  |
| --- | --- |
| JOURNAL OF COMPUTERS, VOL. 8, NO. 1, JANUARY 2013 | 13 |

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.

1. 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, *xn* (1≤*n*≤*N*) indicates

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a data | point, | | | | | and | | |  | *ck* | (1 ≤ *k* ≤ *K*) specifies the cluster | |  |
| centroid. | |  |  |  | *x* | − *c* |  |  |  | 2 is the distance measure between | | *x* |  |
|  |  |  |  |  |
|  |  |  |  |  | *n* | *k* |  |  |  |  |  | *n* |  |
| and *ck* . In (2), *μk* | | | | | | | | |  | (1≤*k*≤*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 *μk* (1 ≤ *k* ≤ *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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *N* | *K* | | |  |  |  |  |  |  |  |  |  |
| *J* =∑ ∑ | |  |  |  | *xn* − *ck* | |  |  |  | 2 . | (1) |  |
|  |  |  |  |  |
| *n* =1 *k* =1 | | | |  |  |  |  |  |  |  |  |  |
| *μ k* = | 1 |  |  |  |  | ∑ *x* *n* . | | | | | (2) |  |
| *N k* | | |  |  |  |
|  |  |  | *n*∈ *c k* |  |  |  |  |  |  |

**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 *μk* (1≤*k*≤*K*);
3. Calculate *J* in Formula (1), denoted by *J’*;
4. Assign each object *n* (1≤*n*≤*N*) to the closest cluster;
5. Calculate new centroid of each cluster *μk* 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

© 2013 ACADEMY PUBLISHER

14 JOURNAL OF COMPUTERS, VOL. 8, NO. 1, JANUARY 2013

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 *μk* (1≤*k*≤*K*);
6. Calculate *J* in (1), denoted as *J’*;
7. Assign each object *n* (1 ≤ *n* ≤ *N*) to the closest cluster;
8. Calculate the new centroid of each cluster *μk* 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 *Centroid1* to *Centroidn*. Therefore, our goal is to merge the *n* centroidsets 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*Centroid1*to*Centroidn*(*n\*K* centroids)

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

1. Initialize a centroid set *Centroid* as empty;
2. If any centroid set *Centroidi* (*i=1,…,n*) is empty, exit and the final centroid set is *Centroid*, otherwise, go to step 3;
3. Find a vector of centroids (*c1,…, cK*) with the minimum inner distance, which is defined as in (3), and then delete *ci* from *Centroidi*, in addition, add *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 (*c1,…,* *cK* ) with the minimum inner distance. In addition,

the Euclidean distance is used to calculate the inner distance, denoted as *Distance*. In (3), *ci* comes from

centroid set *C entroid* *i* (*i=1,…,K*) and *c* is the centroid.

|  |  |  |
| --- | --- | --- |
| *K* | − |  |
| *D*(*c1 ,...,cK* )=∑ *Distance* (*ci ,c* ). | | (3) |

*i* =1

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 *Centroidi* (*i=1,…,n*) is empty, the algorithm exits and the final centroid set is *Centroid*, otherwise, find a vector of

centroids (*c1,…,* *cK* ) with the minimum inner distance according to (3), and then delete *ci* from *Centroidi*, in addition, add *c* into *Centroid*, finally, output the centroid

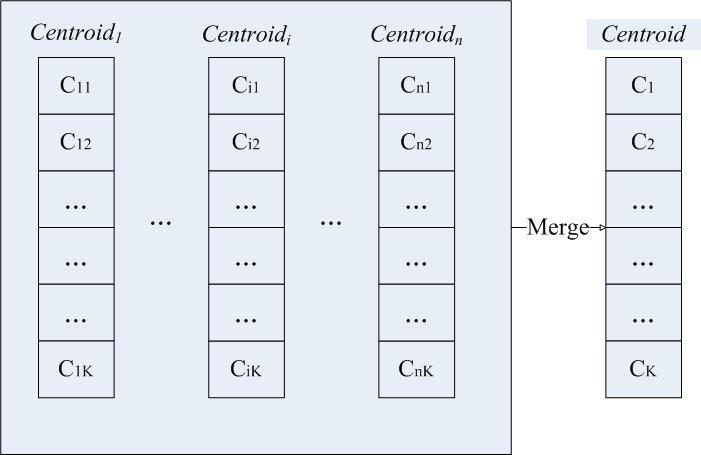
set *Centroid*.

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.

© 2013 ACADEMY PUBLISHER

|  |  |
| --- | --- |
| JOURNAL OF COMPUTERS, VOL. 8, NO. 1, JANUARY 2013 | 15 |

Figure 5. Merging process



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 II

DESCRIPTION OF DATA SETS

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | ***Code*** | |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | ***Number*** | | ***Name of Data*** | | | |  | ***Size(KB)*** | | ***Number of*** | |  |  |  |
|  |  | ***of Data*** | |  |  | ***Sets(.arff)*** | |  | ***Instances*** | |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  | ***Sets*** | |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | 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 | |  |  |  |
|  |  |  |  |  |  |  | TABLE III | | | |  |  |  |  |  |
|  |  | COMPARISON RESULTS BETWEEN MKMEANS AND SKMEANS | | | | | | | | | | | | |  |
|  |  |  |  |  |  |  |  | | |  |  |  | | |  |
|  |  | ***Code*** |  |  |  | ***MKmeans(ms)*** | | | |  | ***SKmeans(ms)*** | | | |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  | ***Number*** |  |  |  | ***Cluster*** |  |  |  |  | ***Cluster*** |  |  |  |  |
|  |  | ***of Data*** |  | ***I/O*** |  |  | ***Total*** | | ***I/O*** |  | ***Total*** | |  |
|  |  |  |  | ***ing*** |  | ***ing*** |  |  |
|  |  | ***Sets*** |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  | |  |  |  |  | |  |
|  |  | 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*** | ***MKmeans(ms)*** | ***SKmeans(ms)*** | ***Speedup*** |  |
| ***of Data*** |  |
|  |  |  |  |
| ***Sets*** |  |  |  |  |
| 1 | 0.2 | 0.0 | 0 |  |
| 2 | 0.2 | 0.0 | 0 |  |
| 3 | 7.6 | 0.0 | 0 |  |
| 4 | 8.8 | 15.6 | 1.7 |  |
| 73 |  |
|  |  |  |  |
| 5 | 45.4 | 46.9 | 1.0 |  |
| 33 |  |
|  |  |  |  |
|  |  |  | 1.0 |  |

© 2013 ACADEMY PUBLISHER

16 JOURNAL OF COMPUTERS, VOL. 8, NO. 1, JANUARY 2013

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 *MCTi* indicates the clustering cost time of running MKMeans in data set *i*

(1 ≤ *i* ≤ 7), *SCTi* 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* = | *SCTi* | . | (4) |  |
|  |  |
|  | *MCT* | |  |  |
|  | *i* | |  |  |

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

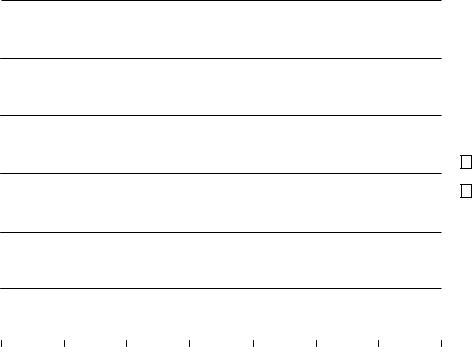
TABLE V

OVERHEADS OF TIME IN SKMEANS AND WKMEANS

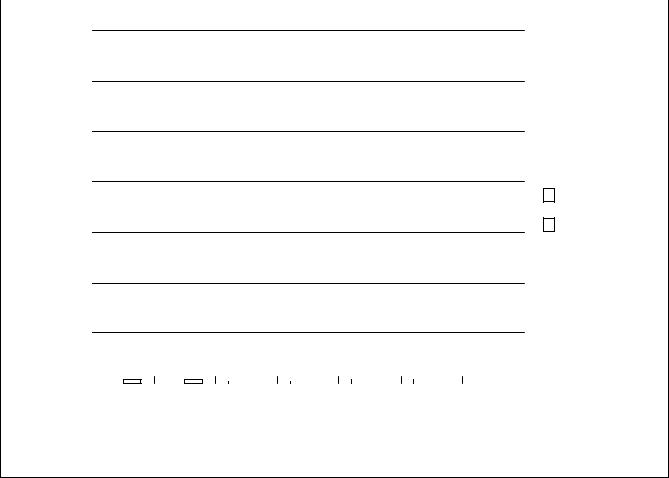
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***Code*** |  |  |  |
|  | ***Number of*** | ***SKmeans(ms)*** | ***WKmeans(ms)*** |  |
|  | ***Data Sets*** |  |  |  |
|  | 1 | 0.0 | 78 |  |
|  | 2 | 0.0 | 78 |  |
|  | 3 | 15.6 | 156 |  |
|  | 4 | 31.2 | 219 |  |
|  | 5 | 93.8 | 562 |  |
|  | 6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

compared to WKmeans. MKmeans and SKmeans share the similar idea, it is hence to conclude that MKmeans is also a stable clustering algorithm.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 300 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| time(ms) | 250 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 200 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 150 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | MKmeans(ms) |  |
| clustering |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | SKmeans(ms) |  |
| 100 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | 0 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | | |  | 2 |  | 3 |  |  |  | 4 |  |  |  | 5 | |  |  |  |  |  | 6 |  |  |  | 7 | | |  |  |  |  |  |  |  |
|  |  |  |  |  | Code Number of Data Sets | | | | | | | | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | |  |  |  |  |  |  |  |  |  | |  |  |  |  |  |  | |  |  |  | |  |  |  |  |  |  |  |  |  |
|  | Figure 6. Clustering time of MKmeans and SKmeans | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7000 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6000 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5000 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | time(ms) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4000 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | SKmeans(ms) |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3000 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | WKmeans(ms) |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | total |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2000 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1000 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | | | | 2 | | 3 | | |  | 4 | | |  | 5 | | | |  |  |  | 6 | | |  | 7 | | | |  |  |  |  |  |  |



Code Number of Data Sets

Figure 7. The total time overheads of SKmeans and WKmeans

1. 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.

© 2013 ACADEMY PUBLISHER

JOURNAL OF COMPUTERS, VOL. 8, NO. 1, JANUARY 2013

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, August1995.
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

17

for Data Mining,” *Data Mining and Knowledge Discovery* *Handbook*, pp. 1269-1277, 2010.

1. C. Blake, E. Keogh, C. Merz, “UCI Repository of machine learning databases,” *Department of Information and* *Computer Science*, *University of California*, 1998.
2. 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 iscurrently 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 iscurrently 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 iscurrently 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 currentlya 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 iscurrently 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.



© 2013 ACADEMY PUBLISHER