Parallel *K*-Means Clustering Based on MapReduce

Weizhong Zhao[1,2], Huifang Ma[1,2], and Qing He[1]

[1] The Key Laboratory of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences
[2] Graduate University of Chinese Academy of Sciences
{zhaowz,mahf,heq}@ics.ict.ac.cn

Abstract. Data clustering has been received considerable attention in many applications, such as data mining, document retrieval, image segmentation and pattern classification. The enlarging volumes of information emerging by the progress of technology, makes clustering of very large scale of data a challenging task. In order to deal with the problem, many researchers try to design efficient parallel clustering algorithms. In this paper, we propose a parallel k-means clustering algorithm based on MapReduce, which is a simple yet powerful parallel programming technique. The experimental results demonstrate that the proposed algorithm can scale well and efficiently process large datasets on commodity hardware.

Keywords: Data mining; Parallel clustering; K-means; Hadoop; MapReduce.

1 Introduction

With the development of information technology, data volumes processed by many applications will routinely cross the peta-scale threshold, which would in turn increase the computational requirements. Efficient parallel clustering algorithms and implementation techniques are the key to meeting the scalability and performance requirements entailed in such scientific data analyses. So far, several researchers have proposed some parallel clustering algorithms [1,2,3]. All these parallel clustering algorithms have the following drawbacks: a) They assume that all objects can reside in main memory at the same time; b) Their parallel systems have provided restricted programming models and used the restrictions to parallelize the computation automatically. Both assumptions are prohibitive for very large datasets with millions of objects. Therefore, dataset oriented parallel clustering algorithms should be developed.

MapReduce [4,5,6,7] is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real-world tasks. Users specify the computation in terms of a *map* and a *reduce* function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks. Google and Hadoop both provide MapReduce runtimes with fault tolerance and dynamic flexibility support [8,9].

In this paper, we adapt k-means algorithm [10] in MapReduce framework which is implemented by Hadoop to make the clustering method applicable to large scale data. By applying proper <key, value> pairs, the proposed algorithm can be parallel executed effectively. We conduct comprehensive experiments to evaluate the proposed algorithm. The results demonstrate that our algorithm can effectively deal with large scale datasets.

The rest of the paper is organized as follows. In Section 2, we present our parallel k-means algorithm based on MapReduce framework. Section 3 shows experimental results and evaluates our parallel algorithm with respect to speedup, scaleup, and sizeup. Finally, we offer our conclusions in Section 4.

2 Parallel *K*-Means Algorithm Based on MapReduce

In this section we present the main design for Parallel K-Means (PKMeans) based on MapReduce. Firstly, we give a brief overview of the k-means algorithm and analyze the parallel parts and serial parts in the algorithms. Then we explain how the necessary computations can be formalized as map and reduce operations in detail.

2.1 *K*-Means Algorithm

K-means algorithm is the most well-known and commonly used clustering method. It takes the input parameter, k, and partitions a set of n objects into k clusters so that the resulting intra-cluster similarity is high whereas the inter-cluster similarity is low. Cluster similarity is measured according to the mean value of the objects in the cluster, which can be regarded as the cluster's "center of gravity".

The algorithm proceeds as follows: Firstly, it randomly selects k objects from the whole objects which represent initial cluster centers. Each remaining object is assigned to the cluster to which it is the most similar, based on the distance between the object and the cluster center. The new mean for each cluster is then calculated. This process iterates until the criterion function converges.

In k-means algorithm, the most intensive calculation to occur is the calculation of distances. In each iteration, it would require a total of (nk) distance computations where n is the number of objects and k is the number of clusters being created. It is obviously that the distance computations between one object with the centers is irrelevant to the distance computations between other objects with the corresponding centers. Therefore, distance computations between different objects with centers can be parallel executed. In each iteration, the new centers, which are used in the next iteration, should be updated. Hence the iterative procedures must be executed serially.

2.2 PKMeans Based on MapReduce

As the analysis above, PKMeans algorithm needs one kind of MapReduce job. The map function performs the procedure of assigning each sample to the closest center while the reduce function performs the procedure of updating the new centers. In order to decrease the cost of network communication, a combiner function is developed to deal with partial combination of the intermediate values with the same key within the same map task.

Map-function The input dataset is stored on HDFS[11] as a sequence file of <key, value> pairs, each of which represents a record in the dataset. The key is the offset in bytes of this record to the start point of the data file, and the value is a string of the content of this record. The dataset is split and globally broadcast to all mappers. Consequently, the distance computations are parallel executed. For each map task, PKMeans construct a global variant *centers* which is an array containing the information about centers of the clusters. Given the information, a mapper can compute the closest center point for each sample. The intermediate values are then composed of two parts: the index of the closest center point and the sample information. The pseudocode of map function is shown in Algorithm 1.

Algorithm 1. map (*key*, *value*)

Input: Global variable *centers*, the offset *key*, the sample *value*
Output: <*key*', *value*'> pair, where the *key*' is the index of the closest center point and *value*' is a string comprise of sample information

```
1. Construct the sample $instance$ from $value$;
2. $minDis = Double.MAX\_VALUE$;
3. $index = -1$;
4. For i=0 to $centers$.length do
       $dis = ComputeDist(instance, centers[i])$;
       If $dis < minDis$ {
           $minDis = dis$;
           $index = i$;
       }
5. End For
6. Take $index$ as $key'$;
7. Construct $value'$ as a string comprise of the values of different dimensions;
8. output $< key', value' >$ pair;
9. End
```

Note that Step 2 and Step 3 initialize the auxiliary variable *minDis* and *index*; Step 4 computes the closest center point from the sample, in which the function *ComputeDist* (*instance*, *centers*[*i*]) returns the distance between *instance* and the center point *centers*[*i*]; Step 8 outputs the intermediate data which is used in the subsequent procedures.

Combine-function. After each map task, we apply a combiner to combine the intermediate data of the same map task. Since the intermediate data is stored in local disk of the host, the procedure can not consume the communication cost. In the combine function, we partially sum the values of the points assigned to the same cluster. In order to calculate the mean value of the objects for each cluster, we should record the number of samples in the same cluster in the same map task. The pseudocode for combine function is shown in Algorithm 2.

Algorithm 2. combine (*key*, V)

Input: *key* is the index of the cluster, V is the list of the samples assigned to the same cluster
Output: $< key', value' >$ pair, where the *key*' is the index of the cluster, *value*' is a string comprised of sum of the samples in the same cluster and the sample number

```
1. Initialize one array to record the sum of value of each dimensions of the samples contained in
   the same cluster, i.e. the samples in the list $V$;
2. Initialize a counter $num$ as 0 to record the sum of sample number in the same cluster;
3. while($V$.hasNext()){
       Construct the sample $instance$ from $V$.next();
       Add the values of different dimensions of $instance$ to the array
       $num$++;
4. }
5. Take $key$ as $key$';
6. Construct $value$' as a string comprised of the sum values of different dimensions and $num$;
7. output $< key', value' >$ pair;
8. End
```

Reduce-function. The input of the reduce function is the data obtained from the combine function of each host. As described in the combine function, the data includes partial sum of the samples in the same cluster and the sample number. In reduce function, we can sum all the samples and compute the total number of samples assigned to the same cluster. Therefore, we can get the new centers which are used for next iteration. The pseudocode for reduce function is shown in Algorithm 3.

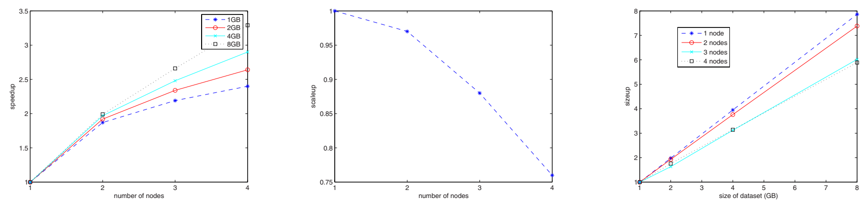
Algorithm 3. reduce (*key*, V)

Input: *key* is the index of the cluster, V is the list of the partial sums from different host
Output: $< key', value' >$ pair, where the *key*' is the index of the cluster, *value*' is a string representing the new center

```
1. Initialize one array record the sum of value of each dimensions of the samples contained in the
   same cluster, e.g. the samples in the list $V$;
2. Initialize a counter $NUM$ as 0 to record the sum of sample number in the same cluster;
3. while($V$.hasNext()){
       Construct the sample $instance$ from $V$.next();
       Add the values of different dimensions of $instance$ to the array
       $NUM$ += $num$;
4. }
5. Divide the entries of the array by $NUM$ to get the new center's coordinates;
6. Take $key$ as $key$';
7. Construct $value$' as a string comprise of the $center$'s coordinates;
8. output $< key', value' >$ pair;
9. End
```

3 Experimental Results

In this section, we evaluate the performance of our proposed algorithm with respect to speedup, scaleup and sizeup [12]. Performance experiments were run



(a) Speedup (b) Scaleup (c) Sizeup

Fig. 1. Evaluations results

on a cluster of computers, each of which has two 2.8 GHz cores and 4GB of memory. Hadoop version 0.17.0 and Java 1.5.0_14 are used as the MapReduce system for all experiments.

To measure the speedup, we kept the dataset constant and increase the number of computers in the system. The perfect parallel algorithm demonstrates linear speedup: a system with m times the number of computers yields a speedup of m. However, linear speedup is difficult to achieve because the communication cost increases with the number of clusters becomes large.

We have performed the speedup evaluation on datasets with different sizes and systems. The number of computers varied from 1 to 4. The size of the dataset increases from 1GB to 8GB. Fig.1.(a) shows the speedup for different datasets. As the result shows, PKMeans has a very good speedup performance. Specifically, as the size of the dataset increases, the speedup performs better. Therefore, PKMeans algorithm can treat large datasets efficiently.

Scaleup evaluates the ability of the algorithm to grow both the system and the dataset size. Scaleup is defined as the ability of an m-times larger system to perform an m-times larger job in the same run-time as the original system.

To demonstrate how well the PKMeans algorithm handles larger datasets when more computers are available, we have performed scaleup experiments where we have increased the size of the datasets in direct proportion to the number of computers in the system. The datasets size of 1GB, 2GB, 3GB and 4GB are executed on 1, 2, 3 and 4 computers respectively. Fig.1.(b) shows the performance results of the datasets. Clearly, the PKMeans algorithm scales very well.

Sizeup analysis holds the number of computers in the system constant, and grows the size of the datasets by the factor m. Sizeup measures how much longer it takes on a given system, when the dataset size is m-times larger than the original dataset.

To measure the performance of sizeup, we have fixed the number of computers to 1, 2, 3, and 4 respectively. Fig.1.(c) shows the sizeup results on different computers. The graph shows that PKMeans has a very good sizeup performance.

4 Conclusions

As data clustering has attracted a significant amount of research attention, many clustering algorithms have been proposed in the past decades. However, the enlarging data in applications makes clustering of very large scale of data a challenging task. In this paper, we propose a fast parallel k-means clustering algorithm based on MapReduce, which has been widely embraced by both academia and industry. We use speedup, scaleup and sizeup to evaluate the performances of our proposed algorithm. The results show that the proposed algorithm can process large datasets on commodity hardware effectively.

Acknowledgments. This work is supported by the National Science Foundation of China (No.60675010, 60933004, 60975039), 863 National High-Tech Program (No.2007AA01Z132), National Basic Research Priorities Programme (No.2007CB311004) and National Science and Technology Support Plan (No.200-6BAC08B06).

References

1. Rasmussen, E.M., Willett, P.: Efficiency of Hierarchical Agglomerative Clustering Using the ICL Distributed Array Processor. Journal of Documentation 45(1), 1–24 (1989)
2. Li, X., Fang, Z.: Parallel Clustering Algorithms. Parallel Computing 11, 275–290 (1989)
3. Olson, C.F.: Parallel Algorithms for Hierarchical Clustering. Parallel Computing 21(8), 1313–1325 (1995)
4. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. In: Proc. of Operating Systems Design and Implementation, San Francisco, CA, pp. 137–150 (2004)
5. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Communications of The ACM 51(1), 107–113 (2008)
6. Ranger, C., Raghuraman, R., Penmetsa, A., Bradski, G., Kozyrakis, C.: Evaluating MapReduce for Multi-core and Multiprocessor Systems. In: Proc. of 13th Int. Symposium on High-Performance Computer Architecture (HPCA), Phoenix, AZ (2007)
7. Lammel, R.: Google's MapReduce Programming Model - Revisited. Science of Computer Programming 70, 1–30 (2008)
8. Hadoop: Open source implementation of MapReduce, `http://lucene.apache.org/hadoop/`
9. Ghemawat, S., Gobioff, H., Leung, S.: The Google File System. In: Symposium on Operating Systems Principles, pp. 29–43 (2003)
10. MacQueen, J.: Some Methods for Classification and Analysis of Multivariate Observations. In: Proc. 5th Berkeley Symp. Math. Statist, Prob., vol. 1, pp. 281–297 (1967)
11. Borthakur, D.: The Hadoop Distributed File System: Architecture and Design (2007)
12. Xu, X., Jager, J., Kriegel, H.P.: A Fast Parallel Clustering Algorithm for Large Spatial Databases. Data Mining and Knowledge Discovery 3, 263–290 (1999)