Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH **Open Access**

Efficient parallel spectral clustering algorithm design for large data sets under cloud computing environment

Ran Jin[1,2*], Chunhai Kou[1], Ruijuan Liu[1] and Yefeng Li[1]

Abstract

Spectral clustering algorithm has proved be more effective than most traditional algorithms in finding clusters. However, its high computational complexity limits its effect in actual application. This paper combines the spectral clustering with MapReduce, through evaluation of sparse matrix eigenvalue and computation of distributed cluster, puts forward the improvement ideas and concrete realization, and thus improves the clustering speed of the distinctive clustering algorithm. According to the experiment, with the processing data scale being enlarged, the clustering rate is in nearly linear growth, and the proposed parallel spectral clustering algorithm is suitable for large data mining. The research results provide research basis to better design a clustering partition algorithm in large data and high efficiency.

Keywords: Large data; Spectral clustering algorithm; Clustering analysis; Parallel Lanczos; K-means

Introduction

The clustering analysis is an important and active research field in data mining, and the research is about the classification of data objects. In order to conveniently expound and understand the data objects and extract inherent information or knowledge hidden in the data, it is necessary to use cluster analysis technology. Its main idea is to divide the data into several classes or clusters, so as to make the objects in same cluster become the most similar while objects in different clusters vary greatly. On the whole, the algorithm can be divided into partition method, hierarchical method, density method, and model method and so on [1]. Generally, the traditional clustering algorithm has following drawbacks: low efficiency in clustering, long processing time in large data and difficulty in meeting the expected effect. For these problems, a popular research idea is correspondingly formed: combining clustering analysis, parallel computing and cloud computing, and designing an efficient parallel clustering algorithm [2,3]. This paper adopts the classical spectral clustering algorithm as the research foundation of clustering partition algorithm as for large-scale data, analyzes how to dig valuable, understandable data information out of large data in a rapid and efficient way and at low costs. Parallel computing is a process that simultaneously uses various computing resources to solve calculation problem, which has the advantages of speeding up program execution and saving investments. Owing to the clustering, many cheap computers can be used to replace the expensive servers, and the data mining services under the parallel computing environment greatly reduces data processing costs. Besides, the cloud computing can provide scalability, reliability and stability when operating large-scale application in virtual computing environment. Based on the characteristics of cloud computing in large application, namely - distributivity, isomerism and mass data, it is suitable for data intensive application and processing [3,4].

Clustering analysis has following common problems: difficulty in handling mass data and distribution data, difficulty in determining parameters, low efficiency and poor clustering quality. In recent years, some researchers have been focusing on how to accelerate spectral clustering algorithm [5-12]. Fowlkes et al. propose to use the Nyström approximation to avoid calculating the whole similarity matrix. That is to say, they trade accurate similarity values

* Correspondence: ran.jin@163.com
[1]College of Information Science and Technology, Donghua University, Shanghai, P.R.C
[2]School of Computer Science and Information Technology, Zhejiang Wanli University, Ningbo, P.R.C

for shortened computational time. Dhillon et al. presupposed the availability of the similarity matrix and proposes a method which does not use eigenvectors. Although these methods can reduce computational time, they trade clustering accuracy for computational speed gain, and they do not address the bottleneck of memory use. To get rid of the memory capacity limit and computational bottleneck, many people like Yang utilized MPI (Message passing Interface) to build a distributed environment. Nevertheless, MPI mechanism increased the consumption of communication between machines and the network. More importantly, it is more complex if realization program uses MPI to deserialize. After all, it requires the whole cluster communication to be controlled, which is not so convenient and easy comparing with Hadoop. The Hadoop is better in fault tolerance. To make the algorithm work normally in mass data, researchers like *Meng* raised the method of using matrix sparsification - closest method, and finally used the matrix spared through the nearest neighbor method to the parallel implementation of spectral clustering. Finally, by proving the algorithm through learning experience of documents data, they proved that the algorithm can effectively cope with the problem of mass data. In this paper, we first calculate the similar matrix and sparsification according to the data point identification segmentation, then use Lanczos distributed computing and parallel computing to get the feature vector when we store the Laplace matrix in the distributed file system HDFS for calculating the characteristic vector by way of using, finally get clustering results by efficient parallel K-means clustering in terms of the transposed matrix of the feature vector. At each step, different parallel strategies are used in algorithm, and the whole algorithm grows fast.

Paper structure is organized as follows: in Section Relevant concepts and description, the MapReduce paradigm is briefly introduced and traditional spectral clustering algorithm is inspected. In Section Parallel spectral clustering algorithm design based on Hadoop, our design and implementation of PSCA(Parallel Spectral Clustering Algorithm) are presented. Performance evaluation is presented in Section The analysis of experiment and result. In Section Conclusion, conclusion is drawn and future works are discussed.

Relevant concepts and description

From above analysis, we can know that the parallel algorithm design is based on Hadoop, so the users' main job is to design and realize the Map and Reduce functions, including input and output the type of < key, value > key value and specific logic of Map and Reduce functions, etc.

Hadoop platform

With the appearance of Google's MapReduce distributed platform, some calculation of high computational complexity can be completed in acceptable time. Based on MapReduce's thought, Apache foundation developed Hadoop Open Source Project. As an open source project, Hadoop's distributed computing framework can be used to construct cloud computing environment (distributed computing). With the help of the computing power, it can be even distributed to many computing nodes in the cluster, thus realizing the huge computation ability about large data. Hadoop has high data throughput, and realizes the high fault tolerance, high reliability and scalability. It is composed of two main parts: HDFS (distributed file system) and MapReduce programming model. At the same time, by combining spectral clustering, serial traditional algorithm and MapReduce programming model, it is transplanted into Hadoop platform to conduct distributed data mining calculation by adopting corresponding parallel strategy. However, if the Hadoop platform technology is applied to the data mining algorithms, key problem is how to achieve the parallelization implementation of traditional data mining algorithm [13]. Among these modes, MapReduce (mapping and specification) programming model can make the user conveniently develop distributed computing program without caring about details. In the whole operation process, MapReduce model is always using key value of < key, value > to input and output about the form. It simplifies the programming model of parallel computing, and only provides available interface to upper users. Working processes at each stage of MapReduce calculation model is as follows:

(1) Input: An application based on the Hadoop platform and MapReduce framework that often need a pair of Map and Reduce functions by realizing appropriate interface or providing abstract class. It should also specify the locations of both input and output, and other operating parameters. This stage will divide big data under the input directory into several independent data blocks.
(2) Map: MapReduce framework treats the application input as a group of < key, value > key value pairs. At this stage, the framework will call the Map function that user defines to process each < key, value > key value pairs. At the same time, it will create some new intermediate < key, value > key value pairs. The types of the two groups of key value pairs may be different.
(3) Shuffle: In Shuffle stage, in order to ensure that Reduce input is output in sequence that Map has already sequenced, the frame gets all related < key, value > key value pairs in Map output for each Reduce through HTTP; according to the key value, MapReduce framework groups are the input in Reduce stage (There are maybe same key for different Map's outputs).

(4) Reduce: This stage will be full of intermediate data, and for each unique key, implement the user-defined Reduce function. The input parameter is"<key, {list of values} >, and the output is a new < key, value > key value pairs.
(5) Output: This stage will write the result output from Reduce in the designated location of output directory. In this way, a typical MapReduce process is completed.

Traditional spectral clustering algorithm

Spectral clustering algorithm is a dot pair cluster algorithm, and it is first used in computer vision, VLSI design and other fields, and then it is used in machine learning, and rapidly becomes research focus in the field of international machine learning. It has very promising application prospects for data clustering. The idea of this algorithm is derived from the spectrogram partition theory. If each data sample is considered as the vertex V in the chart, give weight value W to edge E between vertex in accordance with similarity degree between samples, then the undirected weighted graph $G=(V, E)$ based on similarity degree can be obtained. So in the graph G, the clustering problem can be transformed into partition problem on graph G. The optimal classification criterion based on graph theory is to make the internal similarity degree of the two subgraphs the largest, and similarity degree between subgraphs the smallest.

The standard serial spectral clustering algorithm steps are as follows:

(1) By computation, obtain the similar matrix $S \in R^{n \times n}$ and then sparse it;
(2) Construct diagonal matrix D;
(3) Compute the standard Laplace matrix L;
(4) Compute k minimum eigenvectors of matrix L, and compose matrix $Z \in R^{n \times k}$ which contains k minimum eigenvectors and are regarded as the columns of the matrix Z;
(5) Standardize it as $Y \in R^{n \times k}$
(6) Use K-Means algorithm to cluster the data point $y_i \in R^k (i = 1, \dots, n)$ into k clusters.

Parallel spectral clustering algorithm design based on Hadoop

In the standard serial spectral clustering algorithms, we know that algorithm computational complexity is mainly presented in the construction of similar matrix, calculation of k minimum feature vector(s) in Laplace matrix and k-means the clustering. The parallel design of spectral clustering algorithm is processed from the above three aspects.

Calculate similar matrixes in parallelized ways

Because the Hadoop MapReduce can provide outstanding distributed computing framework, we realize our parallel spectral clustering algorithm in the Hadoop MapReduce. Firstly, we put the data point $x_1, \dots, x_n$ in HBase chart, which can be accessed by each machine, and the line key (row key) of each data point x_i is set as the subscript $i \in \{1, \dots, n\}$ of the data point. Then we use a Reduce function to automatically distribute the similar values between the calculated data points. For each data point x_i with identification i, Reduce function will only clear those whose subscripts are equal to or bigger than i with the data point of $x_j (j = i, \dots, n)$ and the similar value of x_i. We can call it "the similar value calculation of subscript i". In this way, the similar value between each pair of data points can be calculated only once. The apparent "similar value calculation of subscript i" and "similar value calculation of other subscripts" are independent from each other. Therefore, if we distribute different subscripts to different machines, then "similar value calculation of subscript i" can be operated in distributed environment.

Especially, "similar value calculation of subscript i" needs to calculate the similar value $\{< x_i, x_i >, < x_i, x_{i+1} >, \dots, < x_i, x_n >\}$ of $n - i + 1$ data point pairs. That is to say, the first subscript 1 needs to compute similar value of n data point pairs, and the last subscript n only needs to compute the similar value of a data point, that itself is $< x_n, x_n >$. In order to balance the calculation of similar value, we put the "similar value calculation of subscript 1" and "similar value calculation of subscript n" together, and "similar value calculation of subscript 2" and "similar value calculation of subscript $n - 1$" together, and so on (see Figure 1). When the calculation of similar values is completed, put them back on HBase table and they will be used to calculate the Laplace matrix in later steps. The process of parallel construction of similar matrix can be shown in Algorithm 1.

```
Algorithm 1 parallelized constructing the reduce
function in similarity matrix
  Input: <key, value>, key is the subscript index of data
  point, and value is supposed as null.
  Output:<key', value' > = < key,value>
  1. index = key, another Index = n-key + 1
  2. For $i$ in{index,another Index}
       i_content = get Content From HBase($i$):
       For j = i to $n$ do
         j_content = get Content From HBase($j$);
         sim = compute Similarity(i_content,j_content);
         store Similarity($i$, $j$, sim) into HBase table;
       End For
     End For
  3. Output < key,null>
  4. End.
```

Parallel computing *k* minimum eigenvectors

Lanczos algorithm is an iterative algorithm invented by Cornelius Lanczos. The algorithm was invented and used

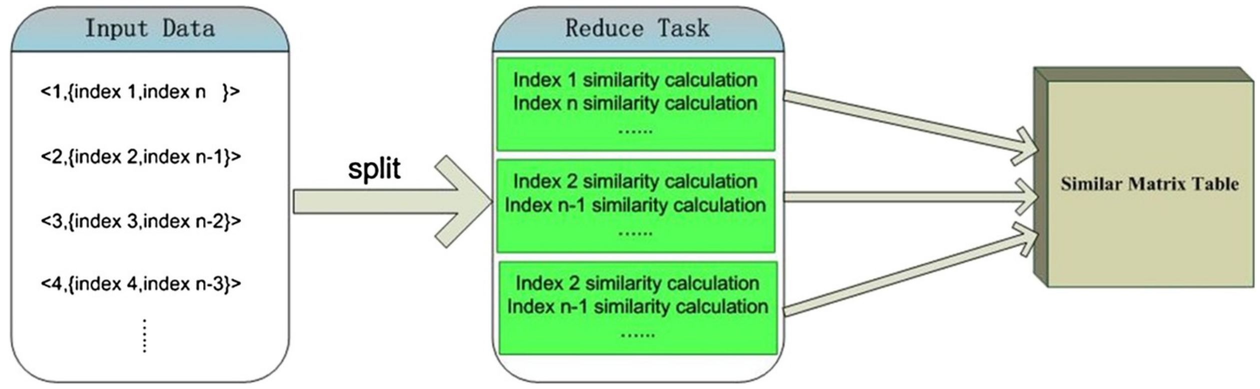
Input Data
<1,{index 1,index n }>
<2,{index 2,index n-1}>
<3,{index 3,index n-2}>
<4,{index 4,index n-3}>
split
Reduce Task
Index 1 similarity calculation
Index n similarity calculation
Index 2 similarity calculation
Index n-1 similarity calculation
Index 2 similarity calculation
Index n-1 similarity calculation
Similar Matrix Table

Figure 1 Parallel computing similar matrix on MapReduce.

to compute the eigenvalue and feature vector of square matrix, or the singular value decomposition of rectangular matrix [14]. Especially for the very large and sparse matrix, Lanczos' algorithm is very effective [15-17]. When calculating the maximum (or minimum) k feature vector of the matrix, the Lanczos is more suitable, for it can find out the k feature vectors by only iterating k times [15,16].

Lanczos transforms the original Laplace matrix L into a real and symmetric tri-diagonal matrix: $T_{mm} = V_m^* L V_m$ with the diagonal elements marked as $\alpha_j = t_{jj}$ and the off-diagonal elements as $\beta_j = t_{j-1j}$. Notice that T_{mm} is a symmetric matrix, so $t_{j-1j} = t_{jj-1}$. Lanczos algorithm is shown in Algorithm 2:

Notice that (x, y) is the dot product of two vectors, and after the iteration, we get a tridiagonal matrix composed of α_j and β_j:

$$T_{mm} = \begin{pmatrix} \alpha_1 & \beta_2 & & & & 0 \\ \beta_2 & \alpha_2 & \beta_3 & & & \\ & \beta_3 & \alpha_3 & \ddots & & \\ & & \ddots & \ddots & \beta_{m-1} & \\ & & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ 0 & & & & \beta_m & \alpha_m \end{pmatrix}$$

After we get the matrix T_{mm}, because T_{mm} is a tridiagonal matrix, it is easy to obtain its eigenvalues and feature vector through other ways (such as QR algorithm). It can be proved that the eigenvalue (feature vector) is the similar value to original Laplacian matrix L's eigenvalue (feature vector).

```
Algorithm 2 Lanczos algorithm

1. $v_1 \leftarrow norm$ is the random vector of 1
    $v_0 \leftarrow 0$
    $\beta_1 \leftarrow 0$
2. Iteration: for $j = 1, 2, \ldots, m$
    $w_j \leftarrow Lv_j - \beta_j v_{j-1}$
    $\alpha_j \leftarrow (w_j, v_j)$
    $w_j \leftarrow w_j - \alpha_j v_j$
    $\beta_{j+1} \leftarrow \|w_j\|$
    $v_{j+1} \leftarrow w_j / \beta_{j+1}$
3. Return
```

From Lanczos' algorithm, we can see that the multiplication Lv_j of matrix and vector is a time-consuming process. If the matrix is put into memory, then L must be removed every time when it is multiplied by a vector, thus consuming a lot of time consumes. The distributed function provided by Hadoop MapReduce and HDF adopts an excellent idea: mobile computing to near the data that is to be operated saves time than to calculation program. We adopt a similar Distributed Matrix to store the matrix L that is to be decomposed on HDFS, and the storage of matrix L on HDFS is according to segmentation. Then Lanczos' each iteration doesn't remove the distributed matrix L on HDFS. On the contrary, what should be moved is a vector (i.e. mobile computing). Every time, the vector vj, which is going to multiply matrix, should be sent to the location that matrix L stores in HDFS, and then the product of vector vj and matrix L on each line (see Figure 2) should be calculated in a parallelized way. The product Lv_j between matrix L

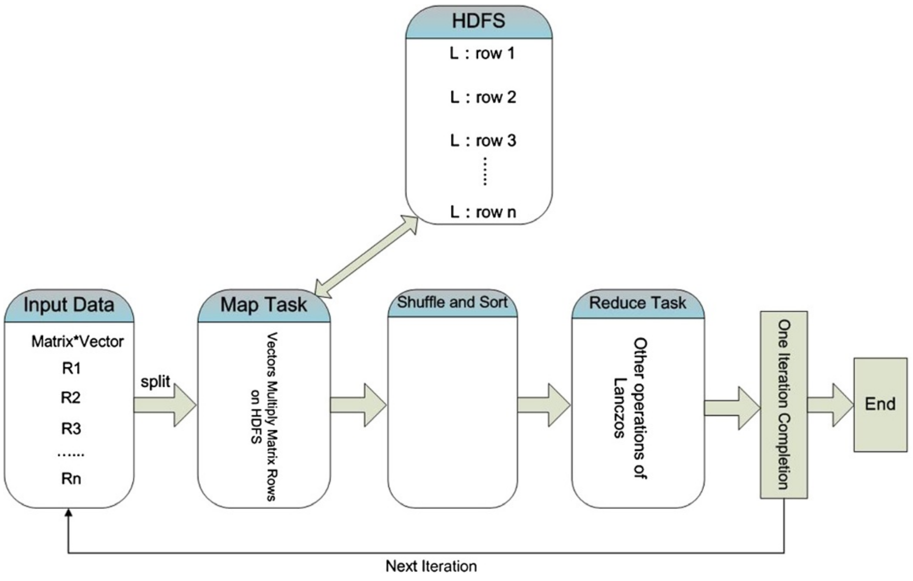

Figure 2 Parallelized Lanczos on MapReduce to calculate Laplacian's feature vector.

and vector v_j is the primary time-consuming operation in Lanczos algorithm. But now with matrix *L*'s distributed storage in HDFS, this operation can be completed by Map/Reduce. If the former *k* feature vector (s) is needed, just send the vector for *k* times to the data storage of matrix for calculation.

Parallelization of K-means clustering

In parallelization of K-means clustering algorithm, a file including initialization *k* cluster (s) center is created, and can be accessed by each machine in the cluster when it is placed on the HDFS. Obviously, the distance calculation between a data point and the *k* center and other data points and *k* center is independent of each other. Therefore, the distance calculation between different data points and *k* center can be performed in parallel in the MapReduce framework. In terms of research on parallel K-means clustering algorithm, there are many achievements, taking literatures [18,19] for instance. In the paper, our designed parallelized K-means clustering algorithm mainly consists of Map function and Reduce function, with Combine operation being added after Map function.

Map function design

The Map function task is to calculate the distance between each record and the center point and remark the focus clustering category. The input is all recorded data for clustering and iterated clustering center from the previous round, with the record data form of < key, value > pairs as < line number, recording line>; each Map function will read the described file of clustering center, and the Map function will calculate the nearest class center to the input recording point and make a new category marking; the form of output intermediate result < key, value > is < cluster category ID, record attribute vector >. The pseudo code of Map function is as follows:

```
void Map(Writable key, Text point){
  The initialization of variable mindis is the possible
  maximum value;
  for(i = 0;i < k;i++){
    if (dis(point, cluster [i]) < mind is){
      mindis = dis(point, cluster[i]);
      current cluster ID = i;}}
  output (current clusterID, point);}
```

When data is large and those objects of each data subset after partition are rather approximate, the middle k value produced in the process of map will be more likely to be repeated. For example, thousands of such records < key j, value j > produced in Map process will be sent through the network to the designated reduce function. It certainly wastes valuable network resources, makes the delay increase, and reduces the I/O performance. Therefore, after the map process is executed, an optional Combiner function is added. Combiner function will firstly merge the output of map function at locality and output < key j, list (value j) > list, and then make use of the partition function hash (key) mod R, halve the intermediate key/value produced by Combiner function into R different partitions, and distribute each partition to the designated reduce function. Figure 3 is the K-means parallel process.

Reduce function design

The task of Reduce function is to calculate the new clustering center in accordance with the intermediate results of Map function, and is for next round of MapReduce Job. The form of input data < key, value > pair is < cluster category ID, {record attribute vector set} >; all the records with same key (i.e., records of same category ID) will receive a Reduce task– accumulate the number of points with same key and the sum of the records and get the average value and then a new clustering center description file; form of the output result < key, value > pair is < cluster category ID, average vector >. The pseudo code of Reduce function is as follows:

```
void Reduce(Writable key, Iterator < Point Writable >
points){
    Initialize the variable num, record the total number
    of samples distributed to the same cluster, the initial
    value is of 0;
    While (points. Has Next()){
        Point Writable current point = points. next();
        Num + =current point. get num();
        for(i = 0;i < dimension;i++){
            sum[i] + =current point. point [i];}
    for(i = 0;i < dimension;i++)
        mean[i] = sum[i]/num;
    out(key, mean);}
```

This iteration continues until each class cluster center is not changed any more, or the iterated number reaches a preset value.

Analysis of complexity of algorithm

Parallel computing of similar matrix

Before giving detailed analysis, assume that the time complexity of computing data points on similar value $S(x_i, x_j)$ is O(l), and assume that m is the number of machines in cluster. It is mentioned that "similar value calculation of subscript i" needs to compute the similar value of $n - i + 1$ data points. We can obtain that the time complexity of "similar value calculation of subscript 1" is O(n), the time complexity of "similar value calculation of subscript 2" is O(n-1), and the like, the time complexity of "similar value calculation of subscript n" is O(l). So the time complexity of computing similar matrix is $O(n + (n - 1) \dots + 1) = O((n^2 + n)/2)$. Because the calculation of similar matrix is evenly executed on m machines, the time complexity of parallel similar matrix calculation is $O((n + (n - 1) \dots + 1)/m) = O((n^2 + n)/(2m))$.

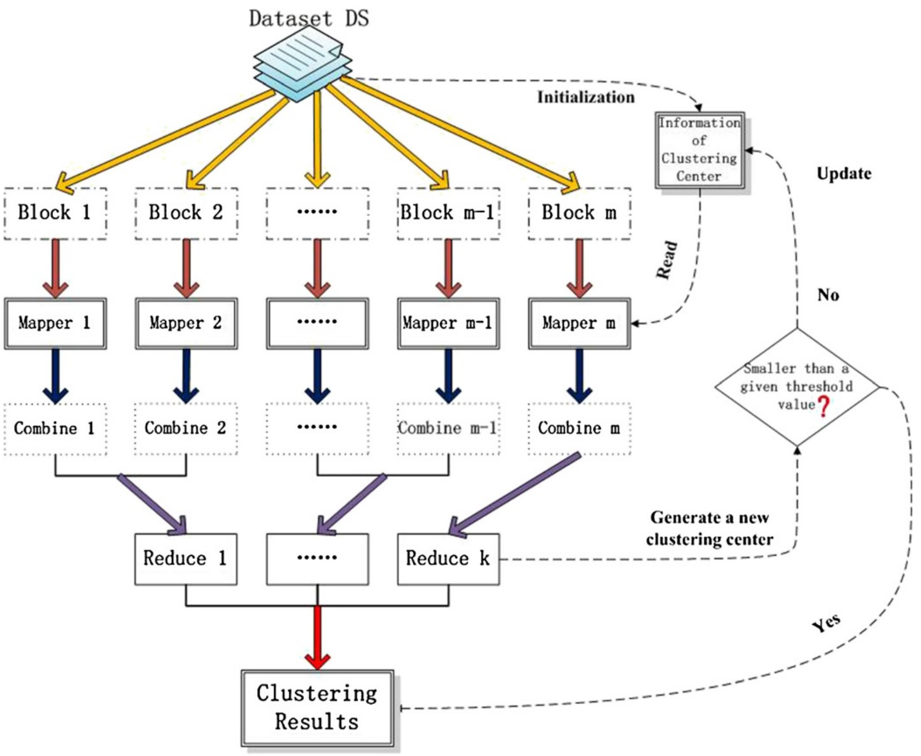

Figure 3 K-means parallel process based Mapreduce.

Parallel computing of k minimum feature vector(s)

Under the non-parallel condition, the time complexity of using Lanczos to compute Laplacian *L*'s *k* vector(s) of different characteristics is $O(kL^{op} + k^2n)$ [20], in which L^{op} is the time that matrix *L* multiplies vector *vj*. Because the matrix *L* has already been segmented into lines and stored on HDFS, multiplication of matrix *L* and vector *vj* is distributed and executed on the machines. And under ideal conditions, the time complexity of each multiplication is L^{op}/m, so the time complexity of parallel computation front *k*feature vector(s) is $O(kL^{op} + k^2n)$.

Parallelization of K-means clustering

New expression form y_i of each data point is *k*-dimensional, hence in each iteration, the distance calculation will be executed between itself and *k* centers. In this way, the distance computing time complexity of each data point is $O(k^2)$. Therefore, the time complexity of iterating the distance calculation of all points each time is $O(nk^2)$. If the condition is ideal, then all the distance calculation of data points is evenly distributed to each machine and in parallel execution, so the time complexity is reduced to $O(nk^2/m)$ * (*numofiterations*).

The analysis of experiment and result

Experimental environment

In this experiment, we use 10 computers to set up the Hadoop cluster. Among them, 8 computers are in dual-core 2.6 GHZCPU, 4 GB memory and operating system of Ubuntu10.04; two in quad core 2.8 GHZCPU, 8 GB memory together with the operating system of Ubuntu10.04. The Hadoop version is 0.20.2, and each machine uses gigabit Ethernet card and is connected through switch machine.

The experiments adopts the classic data set DataSet1 provided by KDD Cup' 99 to test the correctness of the proposed parallel spectral clustering algorithm; we use respectively 10000(Data Set DS1), 50000(Data Set DS2), 100000(Data Set DS3), 1000000(Data Set DS4), 5000000 (Data Set DS5) to verify the superiority of the proposed parallel algorithm, and data samples is the multidimensional data listed in literature [20,21].

In the experiment, both the speedup ration and scaleup ration are deemed as evaluation indicators.

Experimental results

Correctness validation

Table 1 shows the clustering results of data set DataSet1 in stand-alone and the proposed parallel spectral clustering algorithm mode. It can be seen from Table 1, both the proposed parallel spectral clustering algorithm and the serial algorithm have clustering results of higher consistency. The error rate of them is less than 2%, they both achieve a better clustering results and effectiveness, the spectral clustering algorithm proposed in the paper is correct.

Test of speedup ratio

Speedup ratio is defined by parallel computing to reduce the running time and improve the performance. It is an important indicator to verify the performance of parallel computing. The greater speedup ratio is, the less time parallel computing consume relatively, and the higher parallel efficiency and performance improve. Under changing the number of Hadoop cluster nodes, respectively use the results of speedup ratio performance tests according to 10000, 50000, 100000, 1000000, 5000000 pieces of data. Table 2 is the running time of datasets under different nodes. Figure 4 shows the results.

It can be seen from Table 2 and Figure 4, with the increase scale of data set, the algorithm speed-up ratio performance is getting better and better. The reasons are mainly as following: 1) in this paper, the set of < key, value > pair in the stage of Map and Reduce of the proposed parallel spectral clustering algorithm is rather reasonable; 2) we add Combine operation after the stage Map, which greatly reduces the communication costs between the master node and slave nodes. Therefore, as the data quantity becomes large, the speed-up ratio performance will be substantially enhanced.

Table 1 Comparison of clustering accuracy of stand-alone mode and parallel algorithm mode proposed in the paper

Type	Data volume	Stand-alone mode		The proposed parallel algorithm by the paper	
		Correct number	Wrong number	Correct number	Wrong number
Normal	18183	17818	365	17892	291
u2r	267	263	4	265	2
Dos	17408	17132	276	17221	187
R2l	3897	3795	102	3808	89
Probe	4672	4571	101	4600	72
Average error rate		1.98%		1.45%	

Table 2 Comparison of running time

Data volume	Machines	Similar matrix (sec)	Eigenvector (sec)	K-means (sec)	Total time (sec)
DS1 (10000)	1	0.386	0.481	0.156	1.023
	2	0.532	1.099	3.436	5.067
	4	0.186	0.364	1.137	1.687
	6	0.096	0.175	0.582	0.853
	8	0.038	0.065	0.231	0.334
	10	0.025	0.050	0.204	0.279
DS2 (50000)	1	7.251	9.315	2.947	19.513
	2	9.814	12.879	4.139	26.832
	4	3.162	3.963	1.376	8.501
	6	2.299	2.829	1.239	6.367
	8	1.477	1.881	0.910	4.268
	10	1.218	1.555	0.887	3.660
DS3 (100000)	1	19.228	23.982	8.572	51.782
	2	11.234	14.409	4.414	30.057
	4	5.736	6.538	2.246	14.520
	6	4.007	5.587	1.432	11.026
	8	2.965	4.056	0.901	7.922
	10	2.359	3.453	0.654	6.466
DS4 (1000000)	1	7671.580	9603.573	3422.564	20697.717
	2	37.590	46.058	16.678	100.326
	4	19.629	23.755	8.719	53.103
	6	10.126	18.473	6.475	35.074
	8	8.797	13.532	4.865	27.194
	10	6.894	11.415	3.852	22.161
DS5 (5000000)	1	31602.604	39909.984	16630.820	88143.408
	2	150.853	191.559	80.850	423.262
	4	75.164	98.906	39.213	213.283
	6	50.273	70.427	22.087	142.787
	8	40.032	53.142	18.521	111.695
	10	30.841	42.112	13.940	86.893

When the data volume is less than 50000, because in the parallel process, the data volume of each node is not big enough, the speed is smaller than the serial spectral clustering algorithm. However, with the increase of data volume, the speed of parallel algorithm is gradually increased, especially when the data volume is over 1000000, the speedup ratio grows significantly. The running time of stand-alone mode is 3.667 times as long as that of ten computers when dataset volume is 10000. However, it is 1014.39 times when dataset is 5000000. But, it can be seen from Figure 4, when the number of nodes increases to 8 or more, the increasing range of speed-up will narrow. It can be illustrated that the execution efficiency of the parallel spectral clustering algorithm based on Hadoop platform is higher than that of conventional spectral clustering algorithm.

Analysis of scalability

This paper introduces the concept of the efficiency of parallel algorithms. Efficiency of parallel algorithms represents the utilization of a cluster during the execution of parallel algorithms. The formula is $n = {}^{S_p}\!/_{N}$, wherein, S_p represents the speedup ratio, N means the number of cluster nodes. Figure 5 shows the efficiency of parallel algorithms proposed in the paper. For a more general, this paper test the scalability of dataset 100000, 1000000 and 5000000.

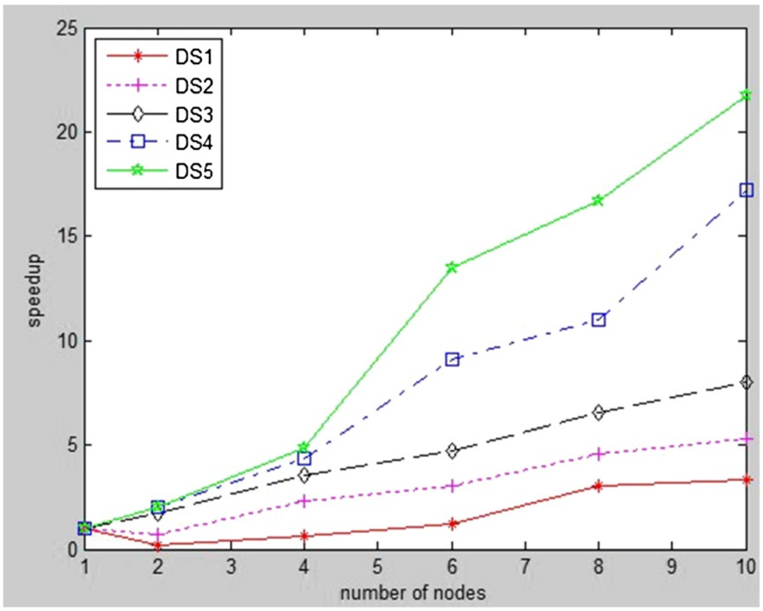

Figure 4 Speedup ratio performance test.

It can be seen from Figure 5, their efficiency curve goes down overall. This is mainly because as the growth of computing nodes in the cluster, the communication overhead increased gradually between nodes. As the data size increases, the efficiency value of parallel algorithm proposed in this paper is larger, namely the better scalability is, the more stable efficiency curve is. Experimental results show that the parallel algorithm proposed in this paper has better scalability in large data sets.

Conclusion

Those data on the Internet exist in vast scale and grow rapidly, so it is urgently required in technology to mine high-value information from the mass data. As a kind of unsupervised learning method, clustering algorithm is a technique commonly used in data statistics and analysis which contains data mining, machine learning, pattern recognition, image analysis, and many other areas. The traditional serial clustering algorithm has two problems

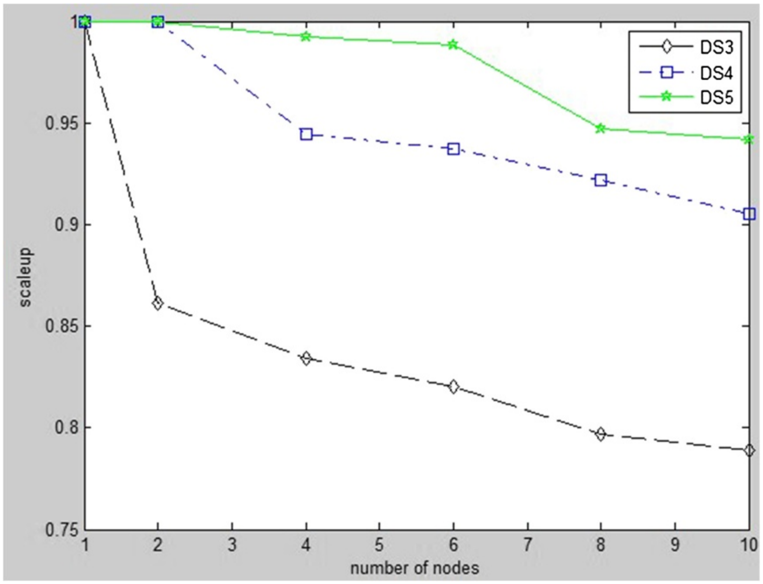

Figure 5 Expansion rate performance test.

and it is difficult to meet the needs of practical applications: the first one is that the speed of clustering is not fast enough and the efficiency is not high; the other one is that in the face of mass data, subject to the limits of memory capacity, it often cannot run effectively. This paper studied the traditional spectral clustering algorithm and designed efficient parallel spectral clustering algorithm. The strategy of parallel spectral clustering algorithm is to compute similar matrix and sparse according to data points segmentation; when computing eigenvectors, store the Laplacian matrix on the distributed file system HDFS, use distributed Lanczos to compute and get the eigenvectors by parallel computation; at last, in terms of the transposed matrix of eigenvectors, adopt the improved parallel K-Means cluster to obtain the clustering results. Through adopting different parallel strategies about each step of the algorithm, the whole algorithm gets linear growth in speed. The experimental results show that the proposed parallel spectral clustering algorithm is suitable for applying in mass data mining. We hope that the research achievements of this paper can provide inspiration and application value for subsequent research developers.

Competing interest
The authors of this paper have no competing interest.

Authors' contributions
The contributions of the paper are twofold: The use of Hadoop to design an improved parallel spectral clustering algorithm for large data sets. The use of speedup ratio and scalability to verify the superiority of the parallel algorithm. All authors read and approved the final manuscript.

Acknowledgement
This work was supported by the Science and Technology Research Program of Zhejiang Province, under grant No.2011C21036, and by the Shanghai Natural Science Foundation under grant No.10ZR1400100, and by Projects in Science and Technique of Ningbo Municipal under grant No. 2012B82003.

Received: 9 April 2013 Accepted: 28 October 2013
Published: 7 November 2013

References
1. Hartigan, JA (1975) Clustering Algorithms. Wiley, USA.
2. Cui J, Li Q, Yang LP (2011) Fast algorithm for mining association rules based on vertically distributed data in large dense databases. Comput Sci 38:216–220
3. Zheng P, Cui LZ, Wang HY, Xu M (2010) A data placement strategy for data-intensive applications in cloud. Comput Sci 33:1472–1480
4. Wang P, Meng D, Zhan JF, Tu BB (2010) Review of programming models for data-intensive computing. J Comput Res Dev 47:1993–2002
5. Fowlkes C, Belongie S, Chung F, Malik (2004) Spectral grouping using the nyström method. IEEE Trans Pattern Anal Mach Intell 26:214–225
6. Dhillon IS, Guan Y, Kulis B (2007) Weighted graph cuts without eigenvectors: a multilevel approach. IEEE Trans Pattern Anal Mach Intell 29:1944–1957
7. Kumar S, Mohri M, and Talwalkar A (2009) Sampling techniques for the nyström method [C]. Paper presented at the 12th conference on artificial intelligence and statistics, University of California, 16–18 April 2009
8. Zhang K, Tsang I, Kwok J (2008) Improve nyström low-rank approximation and error analysis. Paper presented at the 25th International Conference on Machine Learning, Helsinki, 5–9 July 2008
9. Yan D, Huang L, Jordan MI (2009) Fast approximate spectral clustering. Paper presented at the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, 28 June-1 July 2009
10. Gropp E, Skjellum A (1999) Using MPI-2: advanced features of the message-passing interface. MIT Press, USA
11. Song Y, Chen W, Bai H, Lin C, Chang E (2008) Parallel spectral clustering. In: European Conference, ECML PKDD. The joint conference on Machine Learning and Knowledge Discovery in Databases, Belgium, September 2008. Lecture notes in computer science (Lecture notes in artificial intelligence), vol 5212. Springer, Heidelberg, p 374
12. Maschhoff K, Sorensen D (1996) A portable implementation of ARPACK for distributed memory parallel architectures. Paper presented at the 4th Copper Mountain Conference on Iterative Methods, Colorado, 9–13 April 1996
13. Yang C (2010) The research of data mining based on HADOOP. Dissertation, Chongqing University
14. Cullum J, Willboughby RA (1985) Lanczos Algorithms for Large Symmetric Eigenvalue Computations volume I. Birkhauser Boston Inc, USA
15. Golub GH, Loan CFV (1996) Matrix Computations. The Johns Hopkins University Press, Maryland
16. Cullum J, Willboughby RA (1981) Computing eigenvalues of very large symmetric matrices: an implementation of a lanczos algorithm with no reorthogonalization. J Comput Phys 44:329–358
17. Mahadevan S (2008) Fast Spectral Learning Using Lanczos Eigenspace Projections. The 23th national conference on artificial intelligence, Chicago, 13–17 July
18. Zhao WZ, Ma HF, Fu YX, Shi ZZ (2011) Research on parallel K-means algorithm design based on hadoop platform. Comput Sci 38:166–176
19. Niu XZ, She K (2012) Study of fast parallel clustering partition algorithm for large data set. Comput Sci 39:134–151
20. Feng LN (2010) Research on parallel K-Means clustering method in resume data. Dissertation, Dissertation. Yunnan University
21. Jin R, Kou CH, Liu RJ, Li YF (2013) A Co-optimization routing algorithm in wireless sensor network. Wireless Pers Comm 70:1977–1991

doi:10.1186/2192-113X-2-18
Cite this article as: Jin *et al.*: **Efficient parallel spectral clustering algorithm design for large data sets under cloud computing environment.** *Journal of Cloud Computing: Advances, Systems and Applications* 2013 **2**:18.