Chapter 97
One More Efficient Parallel Initialization Algorithm of K-Means with MapReduce

Bingliang Lu and Shuchao Wei

Abstract Because the main deficiencies of a k-means++ algorithm is its internal orderliness, which restricts its applicability in the field of big data processing, we propose an initialization algorithm called pk-means++ based oversampling technology. The initial cluster centers that are obtained using the new algorithm are proved to be very close to the desired cluster centers used for iterative algorithms. It is implemented based on MapReduce of Hadoop, and the experimental results demonstrate that the improved MapReduce pk-means++ algorithm is much more efficient than random and k-means++ initialization algorithm used in k-means based MapReduce and can reach a good approximation.

Keywords Cloud computing • Hadoop • MapReduce • Clustering • k-means • pk-means++

97.1 Introduction

Cluster analysis is a very important area of research on data mining [1, 2], and with advances in technology, big data analysis and application represent general trends. Thus, clustering studies face many new problems and challenges, such as massive data analysis and new computing environments. K-means algorithms are good clustering methods in terms of speed and simplicity in the field of data mining. They have been identified as one of the top ten algorithms for data mining [3]. Some researchers have achieved parallelization of k-means algorithms using MapReduce, but the initial part of it that is used is a largely random initialization method, which is easy to fall into local optima and might still not be well parallelized. (The initial value of the inappropriate choice would result in local optimal solution. For example, the distance of selected points randomly is very close. In this case, we have several randomly chosen initial values and then selected the optimal solution.) Thus, the focus of this study is to search for a better parallel initialization algorithm for k-means

B. Lu (✉) • S. Wei
School of Computer, Shenyang Aerospace University, 110136, Shenyang, China
e-mail: bingliang_lu@163.com; scwade@foxmail.com

W.E. Wong (ed.), *Proceedings of the 4th International Conference on Computer Engineering and Networks*, Lecture Notes in Electrical Engineering 355,
DOI 10.1007/978-3-319-11104-9_97

algorithms to improve the quality and overall performance (we try to find a more perfect and simply realization method of k-means on distributed platforms).

In this area, Arthur and Vassilvitskii proposed an initialization method called k-means++ [4], which obtained excellent theoretical guarantees for clustering performance. Using this method, the convergence time of Lloyd's iteration was reduced by means of a good set of starting centers. But the main deficiencies of the k-means++ algorithm is its internal orderliness, so it is not parallelizable. This fact is more serious in the field of big data processing [5].

Research efforts are being devoted to improving and realizing a k-means++ parallel method based on the MapReduce programming model. One parallel version of the initialization algorithm called pk-means++ is proposed.

97.2 The of K-Means Algorithm

First, some notation is established. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a data set in the d-dimensional Euclidean space. Next, let K be a number of clusters of k-means, and the value of K is specified based on experience (this is not the focus of this paper).

Let $C = \{c_1, c_2, \ldots, c_n\}$ be a set of points and $\|xi - xj\|$ denote the Euclidean distance between xi and xj. The k-means method generates k centers by optimizing the criterion of minimum squared error (MSE, the cost of clustering), which is given by $\phi_X(C) = \sum_{x \in X} \min_{c \in C} ||x - c||^2$.

The ultimate goal of a k-means clustering algorithm is to sample a new set C that includes k centers in order to minimize $\phi_X(C)$. Finally, the data set X is divided into K data sets Y, and $Y = \{Y_1, \ldots, Y_k\}$, $\bigcup_{i=1}^{k} Y_i = X$, $Y_i \cup Y_j = \varnothing$.

One study has proven that finding $\phi_X(C)$ is an NP-hard problem [6]. A variety of options have been researched on providing approximate solutions. Among them, Lloyd's algorithm is widely used in k-means algorithms.

Next, we discuss the algorithm of k-means++ in more detail (Sect. 3). Then, in Sect. 3.1, we give a new initialization algorithm. Finally, a more precise overview of the MapReduce mode is given in Sect. 3.2.

97.3 Parallel Initialization Algorithm Design of PK-Means++

The main idea of k-means++ is that the first centroid is chosen randomly, and then subsequent centers are chosen one by one from the remaining data points [4]. The k-means++ initialization algorithm is presented as follows:

Algorithm 1 : k-means++

1. $C \leftarrow \phi$
2. Choose one center x as c_1 uniformly at random from X, $C = C \bigcup \{x\}$.
3. Repeat

   Choose x ∈ X with probability $d^2(x,C)/\phi_X(C)$

   $C = C \bigcup \{x\}$
4. Until k centers are chosen, Output: $C = \{c_1, c_2 \ldots, c_k\}$.

Formally, k-means++ samples a single point in each pass. It runs very fast in practice, and if the data are clustering well [7], it can guarantee a solution will be found that is an O (log k) approximation to the optimal k-means [6].

But the major downside of the k-means++ is that it is applicable only to big data: a parallel initialization algorithm is proposed based on the k-means++ algorithm and oversampling technology; we call it pk-means++.

97.3.1 Basic Idea of PK-Means++

The main idea of the new initialization algorithm is that it chooses O (k) points in each pass and obtains a nearly optimal solution after a logarithmic number of passes. Finally, O (k log n) points will be left, and experimental results show that a constant number of passes suffices in practice.

Then we set $< \text{num}[i], c_i >, i = 1, 2, \ldots, k$ as the weight of every centroid selected**;** it represents the number of objects in set X to the centroid point of c_i. Next, we recluster these points using k-means++ initialization. Next, the points are reclustered into k initial centers, and Lloyd's iteration can continue. Finally, we use a standard k-means algorithm to complete the clustering.

97.3.2 MapReduce Implementation of PK-Means++

MapReduce is a programming model for efficient distribution, but also for processing big data sets. A typical MapReduce program consists of three stages: the Mapper stage, Shuffle stage, and Reduce stage. Also, the data set should have the following characteristics [5]: it can be broken into many small data sets, and every small data set can be processed completely in parallel.

In a k-means algorithm, the distance from each element to the centroid is calculated independently, with different elements not being linked to each other in the course of operation. Thus, the parallelized version of Lloyd's iterations based MapReduce modules can be easily realized. The basic idea is that each iteration

starts a MapReduce process. According to the computing needs of MapReduce, the data are stored by row and can be sliced by row, with no correlation between the chip data. However, this is not the focus of the present study, though it has been achieved by some researchers [5, 8], and it will not be discussed any further.

According to the preceding analysis, the MapReduce implementation of the pk-means++ algorithm should be divided into two stages, Mapper and Reduce. First, in the Mapper stage, complete steps 1–7 of the pk-means++ algorithm, including generating a temporary data set and calculating the weight of each element of the set. Reduce the stage to complete the remaining part of the algorithm. To obtain the final set of initial cluster centroids C, we integrate k-means++ algorithm with reduce methods. Please refer to the specific implementation which is described in what follows.

The Mapper of pk-means++ initialization

Input : k , the number of clusters $X = \{x_1, x_2, \ldots, x_n\}$,a set of data points.

Output: $< num[i], c_i >, i = 1, 2, ..., k$ **;** $num[i]$ presents that the number of objects in set X to the centroid point of c_i .

1. $C \leftarrow \phi$
2. Choose one center x as c_1 uniformly at random from X , $C = C \bigcup \{x\}$
3. for $i = i; i \le O(k \log n); i++$ do
4. $num[i] = 0$
5. $\varphi \leftarrow \phi_X(C)$
6. for $O(\log \varphi)$ times do
7. $C' \leftarrow$each point x is choosed from X with probability $f \cdot d^2(x, C) / \phi_X(C)$ respectively.
8. $C \leftarrow C \bigcup C'$
9. End for
10. for $i = i; i \le O(k \log n); i++$ do
11. find the nearest center $c_i \in C$ for x_i then $num[i]++$
12. output $< num[i], c_i >$ and $TempC = C$;

As described above, several expensive distance computations can be carried out with MapReduce. Step 7 can be performed in the following manner: each Mapper can sample independently and merge the intermediate results. Step 8 will be given a small set C of centers, and we can compute the value as follows: each Mapper job of Hadoop works on an input data split and then merges these intermediate results from all Mappers to obtain an output for Reducer.

The task of the Reducer function is to update, based on the output of the Mapper function, update the cluster centers for the next round of the Mapper function to use.

Meanwhile, calculate the standard measure for the main function to determine whether the iteration is over. The specific description of the Reducer function is as follows:

The Reducer of pk-means++ initialization

Input : k , the number of clusters ;The output of Mapper: $< num, c >$, *TempC*

Output: $C = \{c_1, c_2 \ldots, c_k\}$

1. $C = \varnothing$; $X = TempC$
2. Choose one center x as c_1 uniformly at random from X , $C = C \bigcup \{x\}$
3. while |C| is lesser than K, do
   - 3.1 Choose x *from* X with the probability of $d^2(x, C) / \phi_X(C)$
   - 3.2 $C = C \bigcup \{x\}$
5. Until k centers are chosen ; output: $C = \{c_1, c_2 \ldots, c_k\}$
6. Continue the process with the normal k-means clustering algorithm

The preceding MapReduce procedure is called in the main function. Each iteration is applied to a new job until the square error, which is calculated by the old centers before iteration and the new centers obtained after iteration, is less than the given threshold value; then the iteration ends.

97.4 Experimental Results and Analysis

The parallel experiments are performed on a homogeneous Hadoop cluster running the latest stable version of Hadoop 1.2.1. The cluster consists of four machines with one master node and three slave nodes. Each node has one Intel Core i5-2400 3.10 GHz Quad-Core CPU, 4 GB RAM, 500 GB hard disk, Intel 82551 10/100 Mbps ethernet controller. The operating system of each node is CentOS-6.3 server 32 bit and per Hadoop daemon is allocated 1 GB memory. This cluster consists of one TaskTracker and one DataNode daemon running on each slave and a single NameNode and JobTracker daemon on the master. Two map slots and two reduce slots are configured on each node. The experiments are conducted on the KDDCup1999 data set. This is a real data set whose size is 4.8 M; the points of the data set are 42 dimensions in Euclidean space. In the experiments, it is assumed that when the initialization method is finished, Lloyd's iterations are continued implicitly.

Let us first analyze the run time of different initialization procedures in the tables. Comparing the data in Table 97.1, it is found that when one suitable parameter f is selected for the pk-means++ algorithm, the convergence speed of the clustering algorithm used by pk-means++ is the fastest and the total time using the random k-means++ is at least one-fourth that of the other algorithm. Combined

Table 97.1 Total clustering time (minutes) of k-means

Initialization algorithm	*k* (number of clusters)	
	200[a]	*500*[a]
Random	320.0	474.8
k-means++	408.5	1,027.2
pk-means++, $f = 0.5$ k	66.5	44.5
pk-means++, $f = 1.0$ k	73.6	87.2
pk-means++, $f = 2.0$ k	64.2	82.5
pk-means++, $f = 4.0$ k	79.5	104.2

[a]The experimental results can be easily observed and contrasted

Table 97.2 Clustering cost (MSE-scaled down by 10^{10}) of different algorithms for $r = 5$

Initialization algorithm	*k* (number of clusters)	
	200[a]	*500*[a]
Random	6.7E + 7	6.5E + 7
k-means++ (alg-1)	7.2	2.0
pk-means++, $f = 0.5$ k (alg-2)	18.5	5.3
pk-means++, $f = 1.0$ k (alg-3)	7.5	2.2
pk-means++, $f = 2.0$ k (alg-4)	5.1	1.6
pk-means++, $f = 4.0$ k (alg-5)	5.75	1.52

[a]The experimental results can be easily observed and contrasted

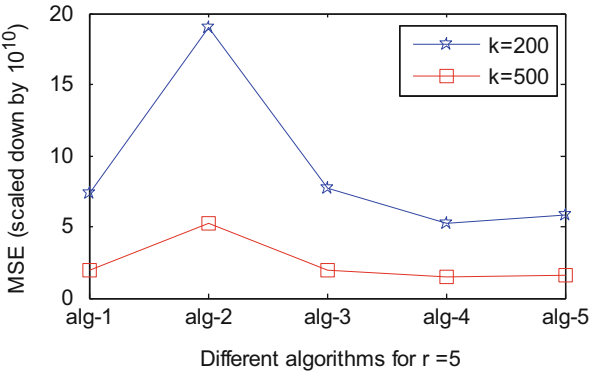

Fig. 97.1 Clustering cost (MSE) of different algorithms for $r = 5$

with the final clustering quality analysis we found that, when $f = 2.0$ k, the performance of the pk-means++ clustering algorithm is the best.

As described in Table 97.2 and Fig. 97.1, it is clear that pk-means++ outperforms k-means++ by orders of magnitude. When the coefficient $f = 0.5$ k, the convergence value of the objective function is higher than that of the k-means++ algorithm. But when the coefficient $f > 1$ k, the objective function value of the algorithm exhibits a linear downward trend, suggesting that the effect of clustering with an increasing f value will be significantly improved. By setting the value of k to 200 and 500, we find that the larger the value of k clusters is, the smaller the convergence value is.

To describe the experimental data of Table 97.2 in a more intuitive way, the following description uses a line chart. As described in Fig. 97.1:

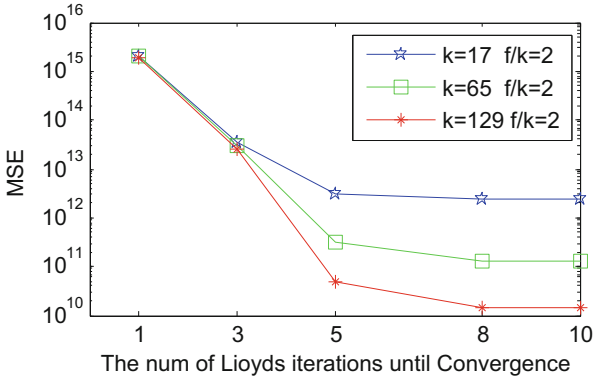

Fig. 97.2 Effect of different parameter settings on final cost of algorithm

Next, Let us evaluate the influence of different settings of parameter carefully. The experiment is conducted with a changing value of a k-based data set that represents a 10 % sample of KDDCup1999. The detailed data are as follows:

As described in Fig. 97.2, when $f = 2$ k, we find that the MSE shows a downward trend with the increased number of passes. Further, it is found that even a small number of passes is enough to reduce the final convergent cost substantially. The total clustering cost is the cost after the completion of Lloyd's iteration, and when, after a certain value of r is reached, the r value is changed again, and the change in the clustering cost becomes very negligible.

Conclusion

In this paper, we develop an efficient k-means++ initialization algorithm with MapReduce, and a parallelized version based on MapReduce called pk-means++ is proposed. The standard k-means++ initialization is also applied to the Reducer phase of pk-means++.

For the Reducer of MapReduce projects, the new algorithm saves considerable communication and I/O costs. Extensive experiments on real data were conducted. The results indicate that the proposed MapReduce pk-means++ algorithm is much more efficient and random than k-means++ and demonstrates that the improved MapReduce pk-means++ algorithm is much more efficient and can obtain good approximations.

References

1. Moise D, Shestakov D, Gudmundsson G, Amsaleg L. Indexing and searching 100 m images with map-reduce. In: Proceedings of the 3rd ACM Conference on Multimedia Retrieval; ACM, New York; 2013. p. 17–24.
2. Xu Z, Ke Y, Wang Y, Cheng H, Cheng J. A model-based approach to attributed graph clustering. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data; ACM, New York; 2012. p. 505–16.

3. Wu X, Kumar V, Ross Quinlan J, Ghosh J, Yang Q, Motoda H, McLachlan GJ, Ng A, Liu B, Yu PS, Zhou Z-H, Steinbach M, Hand DJ, Steinberg D. Top 10 algorithms in data mining. Knowl Inf Syst. 2008;14(1):1–37.
4. Arthur D, Vassilvitskii S. k-means++: the advantages of careful seeding. In: Proceeding of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms; Society for Industrial and Applied Mathematics, Minneapolis,. 2007; p. 1027–35.
5. Zhao W, Ma H, He Q. Parallel k-means clustering based on mapreduce. In: Proceedings of the 1st International Conference on Cloud Computing; Springer, Berlin, Heidelberg; 2009. p. 674–79.
6. Aloise D, Deshpande A, Hansen P, Popat P. NP-hardness of Euclidean sum-of-squares clustering. Mach Learn. 2009;75(2):245–48.
7. Bahmani B, Moseley B, Vattani A, Kumar R, Vassilvitskii S. Scalable k-means++. PVLDB. 2012;5(7):622–33.
8. Lloyd SP. Least squares quantization in PCM. IEEE Trans Inf Theory. 1982;28(2):129–36.