An Efficient K-means Clustering Algorithm on MapReduce

Qiuhong Li[1,2], Peng Wang[1,2], Wei Wang[1,2], Hao Hu[1,2], Zhongsheng Li[3], and Junxian Li[1,2]

[1] School of Computer Science, Fudan University, Shanghai, China
[2] Shanghai Key Laboratory of Data Science, Fudan University
[3] Jiangnan Institute of Computing Technology
{09110240012,pengwang5,weiwang1,huhao,09110240011}@fudan.edu.cn, lizhsh@yeah.net

Abstract. As an important approach to analyze the massive data set, an efficient k-means implementation on MapReduce is crucial in many applications. In this paper we propose a series of strategies to improve the efficiency of k-means for massive high-dimensional data points on MapReduce. First, we use locality sensitive hashing (LSH) to map data points into buckets, based on which, the original data points is converted into the weighted representative points as well as the outlier points. Then an effective center initialization algorithm is proposed, which can achieve higher quality of the initial centers. Finally, a pruning strategy is proposed to speed up the iteration process by pruning the unnecessary distance computation between centers and data points. An extensive empirical study shows that the proposed techniques can improve both efficiency and accuracy of k-means on MapReduce greatly.

1 Introduction

Clustering large-scale datasets becomes more and more popular with the rapid development of massive data processing needs in different areas [16]. With the rapid development of Internet, huge amount of web documents appear. As these documents contain rich semantics, such as text and medias, they can be represented as multi-dimensional vectors. To serve the searching and classification of these documents, we need efficient high-dimensional clustering technology. The clustering algorithms for massive data should have the following features:

– It should have good performance when the number of clusters is large. The number maybe more than several thousands, because many applications need to depict the feature of the data in a fine granularity. However, it will increase the cost significantly.
– It can deal with the high-dimensional data efficiently. Computing the distance between high-dimensional centers and data points is time consuming, especially when the large number of clusters is large.

In this paper, we select k-means algorithm to resolve the clustering problem for massive high-dimensional datasets. The k-means algorithm has maintained its popularity for large-scale datasets clustering, it is among the top 10 algorithms in data mining [20]. The k-means algorithm is used in many applications such as multimedia data management, recommendation system, social network analysis and so on.

However, the execution time of k-means is proportional to the product of the number of clusters and the number of data points per iteration. Clustering the massive high-dimensional data set into a large number of clusters is time consuming, even executing on MapReduce. To solve this and other related performance problems, Alsabti et al. [3] proposed an algorithm based on the data structure of the k-d tree and used a pruning function on the candidate centroid of a cluster. However tree structure is inefficient for high-dimension space. For high-dimensional similarity search, the best-known indexing method is locality sensitive hashing (LSH) [15]. The basic method uses a family of locality-sensitive hash functions to hash nearby objects in the high-dimensional space into the same bucket.

In this paper, we use LSH from a different angle, instead of using it as an index. We use it to partition data points into buckets, based on which, the original data points is converted into the weighted representative points as well as the outlier points, named data skeleton. The benefit is two-folds. First, the number of data points used to initialize the centers is reduced dramatically, which can improve its efficiency. Second, during the iteration phase of k-means, we use it to prune off the unnecessary distance computation. In addition, the output of LSH is integer numbers, which makes it a natural choice to generate the "Key" for MapReduce.

A high-quality initialized centers are important for both accuracy and efficiency of k-means. Some initialization algorithms [5] exploit the fact that a good clustering is relatively spread out. Based on it, these approaches prefer the points far away from the already selected centers. However, they need to make k passes over the whole data set sequentially to find k centers, which limits their applicability to massive data: The scalable k-means++ [2] overcomes the sequential nature by selecting more than one centers at an iteration. However, it still faces huge computation when updating the weights for all points. In this paper, we propose an efficient implement of scalable k-means++ on MapReduce, which improve both efficiency and accuracy of center initialization.

Finally, a pruning strategy is proposed to speed up the iteration process of $k-means$. The basic idea is as follows. First, we use the representative points, p, to find the nearest center, say c. If their distance is apparently smaller than that between the representative points and the second nearest center, c is also the nearest center for all data points represented by p. Second, the locality property of LSH is used to make pruning. In other words, for a data point p, we only compute the distance between it and the small number of centers which stays in the buckets near to that of p.

Our contributions are as follows:

- The LSH-based data skeleton is proposed to find representative points for similar points, which can speed up the center initialization phase and iteration phase.
- We propose an efficient implement of scalable k-means++ on MapReduce, which improve both efficiency and accuracy of center initialization.
- A pruning strategy is proposed to speed up the iteration process of k-means.
- We implement our method on MapReduce and evaluate its performance against the implementation of scalable k-means++ in [2]. The experiments show we get better performance than scalable k-means++, both in initialization phase and iteration phase.

The rest of this paper is organized as follows. We discuss related work in Section 2. Section 3 gives the preliminary knowledge. We propose our LSH-based k-means method in Section 4. A performance analysis of our methods is presented in Section 5. We conclude the study in section 6.

2 Related Work

Clustering problems have attracted interests of study for the past many years by data management and data mining researchers. The k-means algorithm keeps popular for its simplicity. Despite its popularity, k-means suffers several major shortcomings such as the need of specified k value and proneness of the local minima. There are many variants of naive k-means algorithm. Ordonez and Omiecinski [18] studied disk-based implementation of k-means, taking into account the requirements of a relational DBMS. The X-means [19] extends k-means with efficient estimation of the number of clusters. Joshua Zhexue Huang [14] proposes a k-means type clustering algorithm that can automatically calculate variable weights. Alsabti et al. [3] proposed an algorithm based on the data structure of the k-d tree and used a pruning function on the candidate centroid of a cluster. The k-d tree fulfils the space partitioning and a partition is treated a unit for processing. The processing in batch can reduce the computation substantially.

The k-means algorithm has also been considered in a parallel environment. Dhillon and Modha [12] considered k-means in the message-passing model, focusing on the speed up and scalability issues in this model. MapReduce [10] as a popular massive-scale parallel data analysis model gains more and more attention and a lot of enthusiasm in parallel computing communities. Hadoop [1] is a famous open-source implementation of MapReduce model. There are many applications on top of Hadoop. Mahout [17] is a famous Apache project which serves as a scalable machine learning libraries, including the k-means implementation on Hadoop. Robson L.F.Cordeiro [8] proposed a method to cluster multi-dimensional datasets with MapReduce. Yingyi Bu proposes HaLoop [7], which is a modified version of Hadoop, and gives the implementation of k-means algorithm on it. Ene et al.[13] considered the k-median problem in MapReduce and gave a constant-round algorithm that achieves a constant approximation.

D. Arthur and S. Vassilvitskii propose k-means++ [5], which can improve the initialization procedure. Scalable k-means++ [6] is proposed by Bahman Bahmani and Benjamin Moseley, which can cluster massive data efficiently.

Yi-Hsuan Yang[21] presented an empirical evaluation of clustering for music search result. The dataset is sampled first to decide the partitions, then m mappers read the data, ignore the elements from the clusters found in the sample and send the rest to r reducers. r reducers use the plug-in to find clusters in the received elements and send the clusters descriptions to one machine which merges the clusters received and get the final clusters.

3 Preliminary Knowledge and Background

3.1 K-means Algorithm and Its Variants

In data mining, k-means is a method of cluster analysis which aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest distance. $X = \{x_1, x_2, ..., x_n\}$ be a set of observations in the d-dimensional Euclidean space. $\|x_i - x_j\|$ denote the Euclidean distance between x_i and x_j. $C = \{c_1, c_2, ..., c_k\}$ be k centers. We denote *the cost of X with respect to C* as

$$\phi_X(C) = \sum_{x \subset X} d^2(x, C) = \sum_{x \subset X} \min_{1 \le i \le k} \|x - c_i\|^2 \tag{1}$$

where $d^2(x, C)$ is the smallest distance between x and all points in C.

The goal of k-means is to find C such that the cost $\phi_X(C)$ is minimized. Clustering is achieved by an iterative process that assigns each observation to its closest center, constantly improving the centers according to the points assigned to each cluster. The process stops when a maximum number of iterations is achieved or when a quality criterion is satisfied. The quality criterion is generally set that $\phi_X(C)$ is less than a predefined threshold.

3.2 MapReduce and Hadoop

MapReduce [11] was introduced by Dean et. al. in 2004. It is a software architecture proposed by Google. The kernel idea of MapReduce is *map* and *reduce*. The *map* phase calls *map* functions iteratively, each time processing an key/value input record to generate a set of intermediate key/value pairs, and a *reduce* function merges all intermediate values associated with the same intermediate key. It is a simplified parallel programming model and it is associated with processing and generating large data sets. Hadoop implements a computational paradigm named MapReduce, where the application is divided into many small fragments of works, every fragment is processed by a *map* function first on any node in the cluster, then the results are grouped by the key and processed by a *reduce* function. Hadoop provides a distributed file system (HDFS) that stores data on the compute nodes.

3.3 Locality Sensitive Hashing

The indexing technique called locality sensitive hashing(LSH) [4] emerged as a promising approach for high-dimensional data similarity search. The basic idea of locality sensitive hashing is to use hash functions that map similar objects into the same hash buckets with high probability. LSH function families have the property that objects that are close to each other have a higher probability of colliding than objects that are far apart. More formally, assume S be the domain of objects, and D be the distance measure between objects.

Definition 1. *A function family $\mathcal{H}=\{h : S \rightarrow U\}$ is called $(r; cr; p_1; p_2)$- sensitive for D if for any v; $q \in S$*

- *if $v \in B(q, r)$ then $P_{rH}[h(q) = h(v)] \geq p_1$,*
- *if $v \notin B(q, cr)$ then $P_{rH}[h(q) = h(v)] \leq p_2$.*

In this paper, we adopt hash function in Eqution 1, considering its simplicity.

Different LSH families can be used for different distance functions. D. Datar et al [9] have proposed LSH families for l_p norms, based on p-stable distributions. When p is 2, it is Eclidean space. Here, each hash function is defined as:

$$h_{a,b}(v) = \left\lfloor \frac{a.v + b}{r} \right\rfloor \tag{2}$$

4 LSH-kmeans for Massive Datasets

4.1 Overview

In this session we propose LSH-kmeans which includes several optimization strategies for large-scale high-dimensional data clustering. Our goal is to optimize both the initialization phase and the iteration phase for k-means on MapReduce. First, we use LSH to get the data skeleton by which the similar points are reduced to a weighted point (Session 4.2). Secondly, we propose an efficient implement of scalable k-means++ on MapReduce, which improve both efficiency and accuracy of center initialization. Furthermore, we reduce the intermediate messages for the MapReduce implementation by adopting coarse granularity of input (Session 4.3). Thirdly, we make use of the low bound property of LSH to prune off the unnecessary comparisons which can guarantee the correctness of the clustering results (Session 4.4).

4.2 Data Skeleton

For massive datasets, it is quite time-consuming for k-means to compute the distance between any point and center pair, especially when k is large. One intuitive idea is that we can group the similar points together first, and so all points in the same group may belong to the same cluster with high probability. We illustrate it in Fig. 1. Let c_1 and c_2 are two center points, and p_1 and p_2 are two points. r_1, $r_{1'}$, r_2 and $r_{2'}$ are the distances between p_1, p_2 and c_1, c_2 respectively. d is the distane between p_1 and p_2. If we know that $r_1 < r_2$ and d is small, it is very likely that $r_{1'} < r_{2'}$. We can generalize it to Theorem 1.

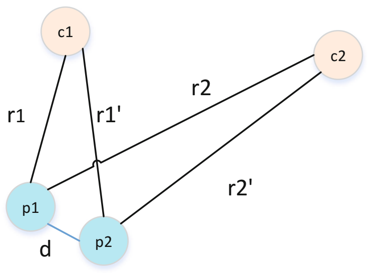


Fig. 1. Points and Centers

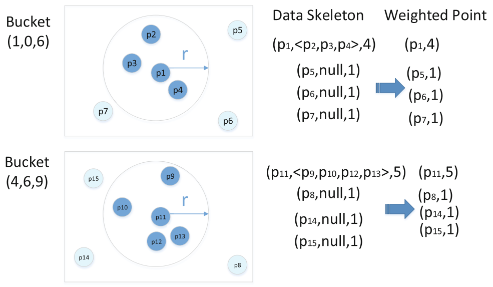


Fig. 2. Data Skeleton

Theorem 1. *Given c_1 and c_2 as two centers, p_1 and p_2 as two points with the distance d, r_1, $r_{1'}$, r_2 and $r_{2'}$ are the distances between p_1, p_2 and c_1, c_2 respectively. If $r_1 < r_2$ and $r_2 - r_1 > 2 * d$, then it holds that $r_{1'} < r_{2'}$.*

Proof. *According to triangle inequality, we have $r_1 - d < r_{1'} < r_1 + d$ and $r_2 - d < r_{2'} < r_2 + d$. Therefore, we have*

$$\begin{aligned} r_{1'} &< r_1 + d \\ &< r_2 - 2d + d && (r_2 - r_1 > 2 * d) \\ &= r_2 - d \\ &< r_{2'} && (r_2 - d < r_{2'}) \end{aligned}$$

■

According to Theorem 1, if we group similar points together, we have large probability to save some unnecessary distance computation. In this paper, we propose data skeleton to achieve this goal.

Each element of data skeleton is a triple $< p_r, L_p, weight >$, where L is the list of points which are represented by p. Formally, for point p, if the distance between p and p_r is smaller than a user-specified threshold, denoted as ε, we add it to L_p. $weight$ is set to $|L_p| + 1$. Data skeleton is computed as follows. First, we use m number of LSH functions to divide the original data set into buckets. All points in each point share the same hash values. From each bucket, we select the center of all points in it as a representative data point, p_r. The distances between all points in this bucket and p_r are computed, and all of them with distance smaller than ε are added into L_p. Other points are named as outlier points, each of which is also an element in data skeleton. L_p of an outlier point is an empty set, and the weight is set to 1. Fig. 2 illustrate it.

Considering that LSH may miss some similar points due to its probabilistic property, we use an iterative process to compute data skeleton with MapReduce. In each iteration, we use a different set of hash functions to find the representative points and outlier points. For the first iteration, the input is the whole dataset. For the rest iterations, the input is the outlier points in the last round. m hash functions in each iteration are selected independently.

Note that, different from the traditional k-means, in which all points have equal weights, in data skeleton, different points have different weights. For representative point, the more points it represents, the higher the weight. For outlier points, the weight is 1. We will use the weighted points to initialize the centers in next section.

4.3 Improving the Seeding Quality

Improving the initialization procedure of k-means is quite important in terms of the clustering quality. The state-of-the-art k-means variants are k-means++ and k-means||. We introduce them respectively.

K-means++ and K-means||. The k-means++ is proposed by Arthur and Vassilvitskii[7], which focuses on improving the quality of the initial centers. The main idea is to choose the centers one by one in a controlled fashion, where the current set of chosen centers will stochastically bias the choice of the next center. The sampling probability for a point is decided by the distance between the point and the center set(Line 3). The distances are considered as weights for sampling. After an iteration, the weights should be changed because of the new centers added into the center set. The details of k-means++ are presented in Algorithm 1. The advantage of k-means++ is that the initialization step itself obtains an $(8 \log k)$-approximation to the optimization solution in expectation. However, its inherent sequential nature makes it unsuitable for massive data set.

Algorithm 1. k-means++ initialization

```
Require: $X$ : the set of points, $k$: number of centers;
1: $C \leftarrow$ sample a point uniformly at random from $X$
2: while $|C| \leq k$ do
3:     Sample $x \in X$ with probability $\frac{d^2(x,c)}{\phi_X(C)}$
4:     $C \leftarrow C \bigcup x$
5: end while
```

The second variant of k-means is k-means||, which is the underlying algorithm for the scalable k-means++ in [6]. The k-means|| improves the parallelism of k-means++ by selecting l centers at one iteration. The k-means|| picks an initial center and computes the initial cost of the clustering. It then proceeds in $\log \psi$ iterations. In each iteration, given the current set of centers C, it samples each x with probability $\frac{l \cdot d^2(x,c)}{\phi_X(C)}$ and obtain l new centers. The sampled centers are then added to C, the quantity $\phi_X(C)$ updated, and the iteration continued. The details are presented in Algorithm 2.

The MapReduce implementation of k-means|| is given in [6], which is called the scalable k-means++. In the rest of this paper, for simplicity, we denote the MapReduce implementation of k-means|| as k-means++ directly.

Algorithm 2. k-means|| initialization

```
Require: $X$ : the set of points,$l$: the number of centers sampled for a time, $k$: number
    of centers;
1: $C \leftarrow$ sample a point uniformly at random from $X$
2: $\psi \leftarrow \phi_X C$
3: for $o(\log \psi)$ do
4:     Sample $C'$ each point $x \in X$ with probability $\frac{l \cdot d^2(x,C)}{\phi_X(C)}$
5:     $C \leftarrow C \bigcup C'$
6: end for
7: For $x \in C$, set $w_x$ to be the number of points in $X$ closer to $x$ than any other point
    in $C$
8: Recluster the weighted points in $C$ into $k$
```

Our Approach. We improve k-means++ with the help of data skeleton. There are two major improvements. First, we use the weighted points in data skeleton to sample the centers. The advantage of sampling centers on the weighted points produced from the data skeleton is that we can acquire a more spread-out center set because the close points are treated as a weighted point. Second, we propose a new sampling implementation on MapReduce.

Our approach includes three steps:

- **Step** 1: Data partitioning and weight initialization
- **Step** 2: Sampling l centers
- **Step** 3: Updating the weights

In first step, we divide the points in data skeleton into $|B|$ disjoint blocks, $(B_1, B_2, \cdots, B_{|B|})$ randomly. Then a random data point is selected as the first center in C. Based on it, we compute the initial sampling weights both each block as well the all points in it. Each block has a unique ID, which is a number within the interval $[1, |B|]$. For each block, the initial sampling weight is the sum of the weights for points in this block. The sampling weight for a weighted point $< x, w_x >$ is $w_x * d^2(x, C)$, denoted as wp_x. It is the smallest distance between x and all centers in C, multiplied by w_x. The sampling weight for block B_i is defined as $\sum_{x \in B_i} wp_x$, denoted as wb_i. Moreover, we compute $\phi_x(C)$, which is the sum of all wb_i's.

A MapReduce job is utilized to partition the data. In Map phase, each point in data skeleton is assigned to a block randomly and the distances for this point to the current centers are calculated. In Reduce phase, we compute the sum of the weights for the points in this block.

Step 2 and 3 are an iterative process. In each round, step 2 generate l centers, and step 3 update the weights of block and points. The iteration continues until we obtain k centers.

In second step, we generate l random numbers randomly within the range $(0, \phi_X(C)]$. For the i-th random number num_i, we can get the corresponding block and points. The corresponding block is B_{n_b} such that $\sum_{j<n_b} wb_j < num_i \leq \sum_{j \leq n_b} w_j$. Within block B_{n_b}, the point p_{n_p} which satisfies

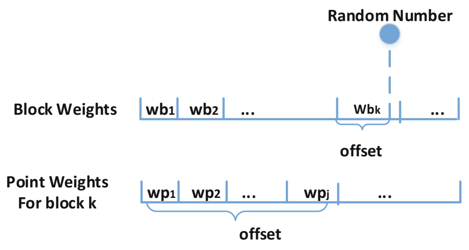


Fig. 3. Sample blocks and points

$$\sum_{j<n_b} wb_j + \sum_{j<n_p} wp_j < num_i \le \sum_{j<n_b} wb_j + \sum_{j\le n_p} wp_j$$

is selected as the i-th center in this iteration. Fig. 3 illustrates it. Also, a MapReduce job is executed in step 2.

Since l new centers are added, the weights for both the block and the points are changed. In third step, we update the weights for blocks and points. This can be implemented by a MapReduce job only with Map phase to update the weights for blocks and for points.

4.4 Pruning Unnecessary Comparisons Using LSH

After k centers are initialized, the iterative phase is conducted to adjust the centers. When k is large, and the data points are high-dimensional, this phase is very time consuming. In this section, we propose a pruning strategy to prune off the unnecessary distance comparisons, which contains two types of pruning. Next, we introduce them respectively.

The first pruning strategy is based on Theorem 1 to reduce the number of data points which needs to find the nearest centers. For each representative point in data skeleton p_r, we first compute its distance to all centers. Let c_1 is the nearest center and c_2 is the second nearest one. If it holds that $d(p_r, c_2) - d(p_r, c_1) > 2\varepsilon$, we can conclude that c_1 is exactly the nearest center for all points represented by p_r. In this case, we need not to compute the distance for all these points.

The second strategy is utilized the local property of LSH to reduce the number of centers to be compared for each data points. Specifically, for a data point, we only compute the distance between it and a few centers which stays in the buckets near to that of this point. It is based on Theorem 2.

Theorem 2. *Given a LSH function:* $h_{a,b}(v) = \lfloor \frac{a\cdot v+b}{r} \rfloor$. *If* $\mid h_{a,b}(v_1) - h_{a,b}(v_2) \mid \ge \delta_h$, *then we have* $d(v_1 - v_2) \ge \frac{(\delta_h-1)\cdot r}{|a|}$.

Proof. *According to definition of LSH, we have* $\mid h_{a,b}(v_1) - h_{a,b}(v_2) \mid = \mid \lfloor \frac{a\cdot v_1+b}{r} \rfloor - \lfloor \frac{a\cdot v_2+b}{r} \rfloor \mid \ge \delta_h$. *We can conclude that* $\mid \frac{a\cdot v_1+b}{r} - \frac{a.v_2+b}{r} + 1 \mid \ge \delta_h$. *Therefore, we have* $\mid \frac{a\cdot(v_1-v_2)}{r} \mid \ge \delta_h - 1$. *We have* $\mid v_1 - v_2 \mid \ge \frac{r(\delta_h-1)}{|a\cos\theta|} \ge \frac{r\cdot(\delta_h-1)}{|a|}$. *Here* θ *is the angle between point* a *and vector* $v_1 - v_2$. ■

We only use one hash function to map all data points and centers into buckets. We use $h(p)$ to denote the bucket ID for point p. For each point p, we first find the bucket, say b_i, which satisfies

- b_i is closest to $h(p)$
- b_i contains at least one center.

In b_i, We randomly select one center, denoted as c. Note that although c and p belong to the nearest buckets, c may not be the closest center for p, due to the probability property of LSH. However, we can use c to prune centers based on Theorem 2.

We calculate the distance from p to c, denoted as $d(p, c)$. From Theorem2, we can get the threshold δ_h to guarantee if the difference of LSH function value is greater than δ_h, the real distance is greater than $d(p, c)$. So we can use only need to compute the distance between p and centers in set $\{c||h(c) - h(p)| < \delta_h\}$.

In fact, these two pruning strategies can be combined. Algorithm 3 shows the pseudo-code of MapReduce job to combine these two pruning strategies. For Map phase, the $< key, value >$ pairs of input represent $< p_r, L_p, weight >$.

We can get the representative point p_r and the points in L_p (Line 3). We record the centers in ordered buckets according to the hash values by a given LSH function when mapper initializes. The results can be shared by all *map* functions. We use binary search to find the closest bucket from the given point p_r and compute the distance from p_r to a point in the bucket. From Theorem 2, we can get the safe pruning threshold δ_h (Line 5 $\sim$ Line 7). We can prune the centers by the threshold δ_h (Line 8 $\sim$ Line 14). From left centers, we get the closet center c' for p_r (Line 15). From Theorem 1, we can conclude that the points in L shares the same center if the distance from p_r to c' is less than $min + 2\varepsilon$. We denote the set of candidate centers as *closeSet*. Because if the distance is greater than $min + 2\varepsilon$, they share the same closest center with p_r. Here min is the distance from p_r to its closest center c' (Line 17 $\sim$ Line 24). We find the closest centers for the points represented by p_r in *closeSet* (Line 25 $\sim$ Line 35). In Reduce phase, new centers are calculated (Line 38 $\sim$ Line 41);

5 Experiments

In this section we present the experimental results of LSH-kmeans. The experiment environment includes a cluster of 14 computers, each of which has two Pentium(R) Dual-Core (2.70GHz) CPU E5400 and 4GB of memory, using Linux. Hadoop version 0.20.3 and Java 1.6 are used as the MapReduce system.

5.1 Dataset and Baseline

We use two datasets. The first is KDDCUP1999 dataset which is publicly available from the UC Irvine Machine Learning repository. The original dataset is comprised of 41 attributes and one class label.

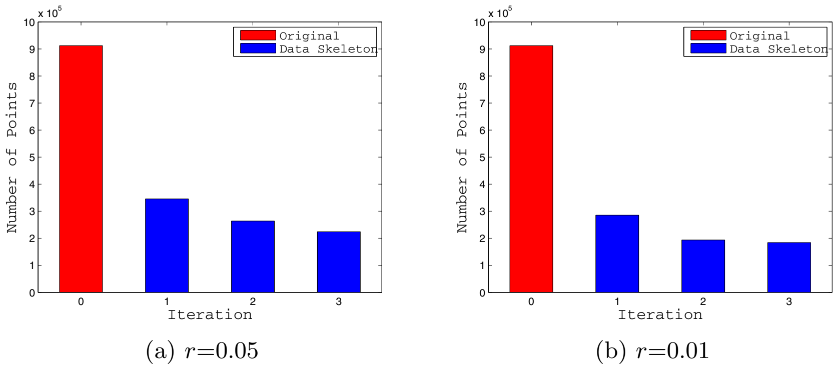
Algorithm 3. LSH-based pruning

```
Require: Set[1:$k$] $C$, parameter $a$, $b$, $r$, $\varepsilon$
 1: Pruning-Map( Key $k$, Value $v$)
 2: begin
 3:     Set $< p_r, L_p, weight > \leftarrow v$
 4:     $hash1 = h(p_r)$
 5:     get $so-far-closest$ from the closest bucket from $hash1$ using binary search;
 6:     $dis = d(p_r, so-far-closest)$
 7:     $\delta_h = \frac{|a| \cdot dis}{r} + 1$
 8:     Set $C' = null$
 9:     for $c$ in $C$ do
10:         $hash2 = \lfloor \frac{a \cdot c + b}{r} \rfloor$
11:         Set $diff \leftarrow abs(hash1 - hash2)$
12:         if $diff \leq \delta_h$ then
13:             $C' = C' + \{c\}$
14:         end if
15:     end for
16:     get closest center $c'$ for $p_r$ in $C'$
17:     $min = distance(p_r, c')$
18:     $closeSet = null$
19:     for $c$ in $C'$ do
20:         $dis2 = d(p_r, c)$
21:         if $| dis2 - min | \leq 2\varepsilon$ then
22:             $closeSet = closeSet + \{c\}$
23:         end if
24:     end for
25:     if $closeSet = null$ then
26:         for $p$ in $L_p$ do
27:             $Output(c', p)$
28:         end for
29:     else
30:         for $p$ in $L_p$ do
31:             get closest center $cen'$ from $closeSet$
32:             $Output(cen', p)$
33:         end for
34:     end if
35:     $Output(c', p_r)$
36: end
37: Pruning-Reduce( Key $k$, Set $values$)
38: begin
39:     $mean = (\sum_{v \in values} v)/sizeof(values)$
40:     $center$ = nearest point from $mean$
41:     $Output(center$, null)
42: end
```

The second dataset is from our music database, which consists of about 1000 MP3 songs downloaded from the Internet, in which most of them are pop songs and the rest are classical and folk music. We extract the key features from the audible data and get the 26-dimension set of points. One point represents a frame of the song. Totally the dataset includes 919711 26-dimensional vectors.

5.2 Experiment Results

Data Reduction of Data Skeleton. In this experiment, we compare the number of data points in original dataset and data skeleton. We execute this process for three iterations. For KDDCUP1999, the time for an iteration about 60s, much less than an k-means iteration (above 600s for k=1500). For Music Frames, the time for an iteration about 130s, much less than an k-means iteration (above 1567s for k=1500). The results are shown in Fig. 4. We see that after data skeleton, the number of data points is reduced dramatically.

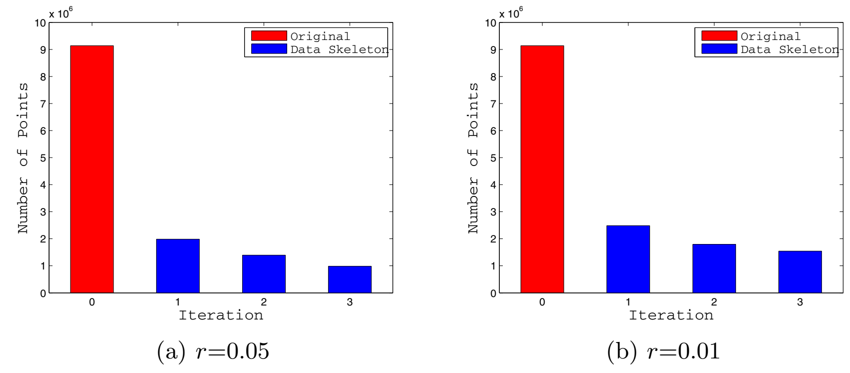


(a) r=0.05 (b) r=0.01

Fig. 4. Data Skeleton For KDDCUP1999

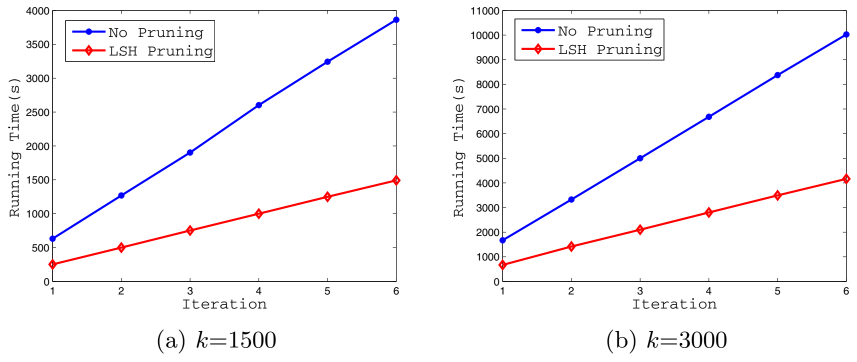
The Center Initialization. We evaluate the quality by two factors with the cost $\phi_X(C)$ in Equation 1. One is the clustering cost after seeding phase, and the other is the convergence property. We use KDDCUP1999 dataset for this evaluation. First we analysis the time performance, which is shown in Fig 7. The time for seeding using LSH-kmeans is about 1/3 the seeding phase of k-means++. The cost comparison is shown in Table 1. It can be seen that our approach generates better cost than k-means++.

The LSH Pruning Performance. To evaluate the performance of the LSH pruning in the iteration phase, we compare the time performance with and without LSH pruning on KDDCUP1999 dataset. The results are shown in Fig. 6. In each iteration, the time using LSH pruning is only 1/3 of the time without pruning. Theorem 1 provides the guarantee for the correctness of the pruning. It can be seen that when k is 3000, the time cost is reduced by about 68%.



(a) r=0.05 (b) r=0.01

Fig. 5. Data Skeleton For Music Frames



(a) k=1500 (b) k=3000

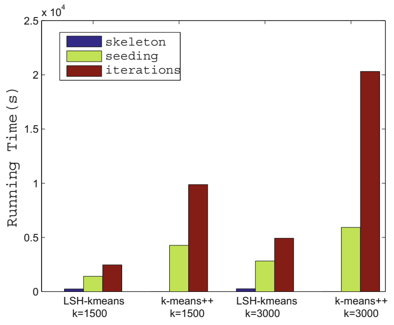
Fig. 6. LSH Pruning Performance

Table 1. Comparison of Clustering Cost (k=3000)

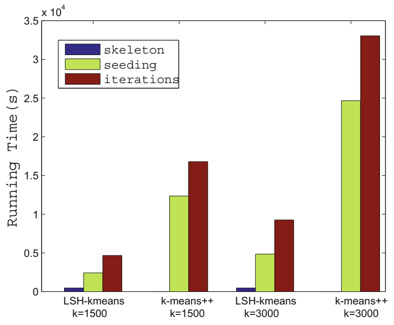
Iteration	Cost of Original Dataset	Cost of Data Sleleton
1	47824.77	47664.18
2	40292.91	40200.01
3	38318.60	38222.02
4	37474.73	37355.58
5	37019.76	36950.85
6	36714.02	36672.52

The Overall Performance Comparisons. We analyze the overall performance of LSH-kmeans compared with k-means++. LSH-kmeans includes three phases, skeleton, sampling centers and iteration. k-means++ includes two phases, sampling centers and iterations. The running time for different phases are shown in Fig. 7. Fig. 7(a) shows four groups of data, which are the running time for different phases in terms of LSH-kmeans and k-means++, with k as 1500 and

3000 respectively. The dataset is KDDCUP1999. As we can see, LSH-kmeans outperforms k-means++ greatly, especially for a larger k ($k = 3000$). The time cost is reduced by 67% when k is 1500, and 76% when k is 3000. We can get the same conclusion for Music Frames from Fig. 7(b). The time cost is reduced by 57% when k is 1500, and 64% when k is 3000.



(a) KDDCUP1999



(b) Music Frames

Fig. 7. Overall Performance Comparisons

6 Conclusion and Future Work

In this paper we propose an improved k-means algorithm to cluster high-dimensional data on MapReduce with the LSH technology. With the increase of k, the number of clusters, the computation cost increases rapidly for high-dimension data. Our method improves the performance of k-means both in the initialization phase and the iteration phase. We implement LSH-kmeans on MapReduce and evaluate its performance on several datasets. The experiment results show that our method can improve the performance dramatically without decreasing the quality.

Acknowledgments. This work was supported by NSFC under grants 60673133, 61103009, 61033010, and the Key Project of Shanghai Municipal Science and Technology Commission (Scientific Innovation Act PlanGrant No.13511504804).

References

1. `http://hadoop.apache.org/`
2. `http://mahout.apache.org/`
3. AlSabti, K., Ranka, S.: An efficient space-partitioning based algorithm for the K-means clustering. In: Zhong, N., Zhou, L. (eds.) PAKDD 1999. LNCS (LNAI), vol. 1574, pp. 355–360. Springer, Heidelberg (1999)
4. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: FOCS, pp. 459–468 (2006)

5. Arthur, D., Vassilvitskii, S.: k-means++: the advantages of careful seeding. In: SODA, pp. 1027–1035 (2007)
6. Bahmani, B., Moseley, B., Vattani, A., Kumar, R., Vassilvitskii, S.: Scalable k-means++. CoRR, abs/1203.6402 (2012)
7. Bu, Y., Howe, B., Balazinska, M., Ernst, M.: Haloop: Efficient iterative data processing on large clusters. PVLDB 3(1), 285–296 (2010)
8. Cordeiro, R.L.F., Traina Jr., C., Traina, A.J.M., López, J., Kang, U., Faloutsos, C.: Clustering very large multi-dimensional datasets with mapreduce. In: KDD, pp. 690–698 (2011)
9. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Symposium on Computational Geometry, pp. 253–262 (2004)
10. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: OSDI (2004)
11. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: OSDI (2004)
12. Dhillon, I.S., Modha, D.S.: A data-clustering algorithm on distributed memory multiprocessors. In: Zaki, M.J., Ho, C.-T. (eds.) KDD 1999. LNCS (LNAI), vol. 1759, pp. 245–260. Springer, Heidelberg (2000)
13. Ene, A., Im, S., Moseley, B.: Fast clustering using mapreduce. In: KDD, pp. 681–689 (2011)
14. Huang, J.Z., Ng, M.K., Rong, H., Li, Z.: Automated variable weighting in k-means type clustering. IEEE Trans. Pattern Anal. Mach. Intell. 27(5), 657–668 (2005)
15. Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: STOC, pp. 604–613 (1998)
16. Kriegel, H.-P., Kröger, P., Zimek, A.: Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. TKDD 3(1) (2009)
17. Mondal, A., Lifu, Y., Kitsuregawa, M.: P2PR-tree: An R-tree-based spatial index for peer-to-peer environments. In: Lindner, W., Fischer, F., Türker, C., Tzitzikas, Y., Vakali, A.I. (eds.) EDBT 2004. LNCS, vol. 3268, pp. 516–525. Springer, Heidelberg (2004)
18. Ordonez, C., Omiecinski, E.: Efficient disk-based k-means clustering for relational databases. IEEE Trans. Knowl. Data Eng. 16(8), 909–921 (2004)
19. Pelleg, D., Moore, A.W.: X-means: Extending k-means with efficient estimation of the number of clusters. In: ICML, pp. 727–734 (2000)
20. Wu, X., Kumar, V., Quinlan, J.R., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G.J., Ng, A.F.M., Liu, B., Yu, P.S., Zhou, Z.-H., Steinbach, M., Hand, D.J., Steinberg, D.: Top 10 algorithms in data mining. Knowl. Inf. Syst. 14(1), 1–37 (2008)
21. Yang, Y.-H., Lin, Y.-C., Chen, H.H.: Clustering for music search results. In: ICME, pp. 874–877 (2009)