

基于 MapReduce 的单遍 K -means 聚类算法

唐 浩¹, 杨余旺¹, 辛智斌²

(1. 南京理工大学 计算机科学与工程学院, 江苏 南京 210094;

2. 淮海集团工业有限公司, 山西 长治 046000)

摘 要: K -means 应用于 MapReduce 框架的大数据处理可显著提高 K -means 对大数据集的处理能力。但 K -means 聚类算法需要进行多次迭代才能达到可接受的效果, 并将每次迭代作为一个独立 map 作业执行, 需要读写整个数据集, 从而导致显著的 I/O 消耗, 与 MapReduce 框架的设计理念不符。为此, 提出了一个基于 MapReduce 的单遍 K -means 算法 (MR-SK)。该算法采用流数据单遍算法读取数据, 聚类时采用 K -means++ 初始化 seeding 算法得到初始聚类中心。在理论分析 MRSK 算法复杂度的基础上, 进行了 MRSK 算法的测试验证和相关分析。验证实验结果表明, 相对于基于 MapReduce 和基于数据流的 K -means 聚类算法, 所提出的 MRSK 算法在执行速度和聚类效果方面具有更好的优势。

关键词: MapReduce 框架; 数据聚类; K -means++; Mahout; 单遍技术

中图分类号: TP301.6

文献标识码: A

文章编号: 1673-629X(2017)09-0026-05

doi: 10.3969/j.issn.1673-629X.2017.09.006

A Single-pass K -means Clustering Algorithm with MapReduce

TANG Hao¹, YANG Yu-wang¹, XIN Zhi-bin²

(1. School of Computer Science and Technology, Nanjing University of Science and Technology,

Nanjing 210094, China;

2. Huaihai Industrial Group Co., Ltd., Changzhi 046000, China)

Abstract: The application of fitting K -means into MapReduce framework can greatly improve the processing of K -means on large datasets. But K -means achieves an acceptable clustering effect through multiple iterations. Each iteration is executed as an independent map job, in which the whole dataset must be read and wrote to slow disks, resulting in high I/O overhead, and it is not consistent with the design concept of the MapReduce framework. Therefore, a single-pass K -means clustering algorithm based on MapReduce, called MRSK, is proposed. It reads the data by single-pass and uses the K -means++ seeding algorithm to get the initial cluster center. On the basis of theoretically analyzing the complexity of the MRSK, a series of test and analysis for MRSK is conducted. The experimental results show that compared with the available MapReduce-based and stream-based K -means variants, MRSK performs both faster execution times and higher quality of clustering results.

Key words: MapReduce framework; data clustering; K -means++; Mahout; single-pass

0 引 言

K -means^[1] 聚类算法是数据挖掘领域中的一种重要方法, 由于计算简单、收敛速度快, 在机器学习、图像处理等领域应用广泛。近年来, 随着数据量的不断攀升, 传统的 K -means 聚类分析方法已无法满足大数据^[2]的需求, 所以亟需对传统的 K -means 方法做出改进以支持大规模数据分析。

MapReduce^[3]是由谷歌提出的一种大数据并行计算框架。MapReduce 的 3 个主要特点: 模型使用方便、适用于大型数据处理、可扩展并内置容错, 使其成为一种理想的大数据处理框架^[4]。如今, K -means 已经成功地应用在了 MapReduce 框架中。

Zhao W 等提出的基于 MapReduce 的 K -means 聚类算法, 是一种快速聚类方案, 利用了运行在 Ha-

收稿日期: 2016-11-04

修回日期: 2017-03-09

网络出版时间: 2017-07-11

基金项目: 国家自然科学基金资助项目 (61640020); 江苏省科技支撑计划 (BE2012386, BE2011342); 江苏省农业自主创新项目 (CX(13)3054, CX(16)1006); 江苏省重点研发计划 (BE2016368-1); 深圳市战略性新兴产业发展专项资金项目 (JCYJ20130331151710105)

作者简介: 唐 浩 (1990-), 男, 硕士生, 研究方向为云计算和大数据处理; 杨余旺, 教授, 研究方向为云计算和大数据处理、网络编码、传感器网络。

网络出版地址: <http://www.cnki.net/kcms/detail/61.1450.TP.20170711.1457.086.html>

doop^[5]框架上的 Mahout^[6]项目,称之为 PKMeans^[7]。艾隆等提出的基于数据流的 K-means 算法,利用了基于数据流的单通技术^[8],称之为 SKMA^[9]。

上述解决方案为了达到可接受的结果,都需要对整个数据集进行多次迭代,而 MapReduce 本质上不支持迭代。在每次迭代中,整个数据集必须从磁盘加载到内存中,数据集经过处理后,输出时必须再一次写入磁盘,因此,每一次迭代都会产生大量和 I/O 相关的操作,如磁盘 I/O、数据序列化等等,严重影响了程序的执行速度。所以这些方案并不适合 MapReduce 框架。Hadian 和 Shahrivari 为了共享内存,实现并行计算,提出了一种并行的单遍聚类算法^[10],但是该算法没有使用 MapReduce 框架。

为此,文中提出一种基于 MapReduce 的单遍 K-means 聚类算法(MRSK)。该算法利用流数据单遍算法读取数据,在数据聚类过程的 map 阶段采用 K-means++初始化 Seeding 算法^[11]得到中间簇中心集;在 reduce 阶段,采用支持权重的 K-means++初始化 Seeding 改进算法进行中间簇中心集处理,并获得初始聚类中心,再利用 K-means 进行聚类计算,得到最终聚类结果。

1 K-means++与单遍算法

1.1 K-means++初始化 Seeding

假设一个数据集有 n 项,每项有 d 个特征, k 为聚类中心数,则这个数据集可以用矩阵表示为 $M_{n \times d}$ 。定义 $M_{n \times d}$ 中第 i 行、第 j 列元素为 $m_{i,j}$,数据集第 i 项可以用向量 \vec{m}_i 表示。设 k 为簇个数,簇中心的集合为 $C = \{C_1, C_2, \dots, C_k\}$, $D(x)$ 为数据项 x 到已被选定的离它最近的簇中心的距离。

K-means 是一种简单快速且高效的聚类算法^[12]。质心的计算如下:

$$\text{Center}(C_i) = \frac{1}{|C_i|} \sum_{\vec{m}_j \in C_i} \vec{m}_j \quad (1)$$

初始聚类中心的选取对 K-means 影响很大,但是 K-means 随机选取初始聚类中心,无法保证聚类效果。K-means++通过初始化 Seeding 算法^[13],可以提供一个较好的初始聚类中心。

通过式(2)初始聚类中心:

$$\frac{D(\vec{m}_j)^2}{\sum_{\vec{m}_j \in M} D(\vec{m}_j)^2} \quad (2)$$

初始化 Seeding 算法的步骤为:

算法 1: K-means++中心初始化。

1: 执行 K-means++(M, k);

2: 从数据集 M 中随机选择一个初始中心 C_1 ;

3: $\hat{C} \leftarrow \{\hat{C}_1\}$;

4: while $|\hat{C}| \neq k$ do

利用式(2)选取数据项 \vec{m}_i 且 $\vec{m}_i \in M - \hat{C}$; $\hat{C} \leftarrow \hat{C} \cup \{\vec{m}_i\}$;

5: end while

6: 返回 \hat{C} ;

7: 结束

1.2 单遍算法

单遍算法^[14]是流式数据聚类的经典方法。该算法依据数据输入顺序依次处理数据,依据当前数据与现有类的匹配度大小,判定该数据归入已有类或另外创建一个新类。

利用欧氏距离作为匹配度判定依据,欧式距离公式如下:

$$d(\vec{m}_i, \vec{m}_j) = |\vec{m}_i - \vec{m}_j| = \sqrt{\sum_{p=1}^d (m_{i,p} - m_{j,p})^2} \quad (3)$$

在 $d(\vec{m}_i, \vec{m}_j)$ 小于阈值时,将点归入 $d(\vec{m}_i, \vec{m}_j)$ 值最小的类; $d(\vec{m}_i, \vec{m}_j)$ 大于阈值时,创建一个新类。

2 MRSK 算法

2.1 算法流程

提出 MRSK 方法,是为了解决现有基于 MapReduce 方案中因多次迭代而引起的 I/O 消耗问题。在 MRSK 方法中,将整个数据集分割成更小的,可存于每台机器内存的数据块,并分发这些数据块到可用的机器。然后,每个块由所处的机器独立聚簇,每个块经过并行处理后产生若干中间簇中心,之后将中间簇中心的集合从每台机器中抽取出来作为一个更小的数据集装载进单独一台机器的主存中,最终的聚类中心从中间簇中心集中通过重新聚类产生。

每个块所在的机器在处理数据时采用单遍法,对当前到达的数据进行聚类,依据匹配度大小,将数据判为已有类或创建一个新类。采用单遍算法,计算过程中整个数据集只被读取了一次。

由于每个块的处理是相互独立的,每个块可以在一个专用的 map 任务中处理。也就是说,每个 map 任务选择一个数据集分块,并在数据块上执行 K-means++初始化 Seeding 算法,得到中间簇中心和每个簇中心的权重,然后各聚类中心和各中心的权重作为结果输出。Map 过程完成后,所有加权中心形成中间簇中心集,并作为 map 输出。这个中间簇中心集会分配给一个单独的 reduce 任务。Reduce 任务得到中间簇中心和它们的权重后,会采用改进的支持权重的 K-means++初始化 Seeding 算法得到初始簇中心,然后通过改进的 K-means 算法得到最终的聚类中心。MRSK 算法的完整流程如算法 2 所示。

算法 2: MRSK 算法。

1: 执行 MRSK(M, k)

2: Map 过程:

3: 将数据集 M 分割为 d 块, 构成集合 $M = M_1, M_2, \dots, M_d$;

4: 对于集合 M 中的每个 M_i 执行步骤 5;

5: 从 M_i 中随机选择一个初始中心 \hat{C}_i ;

6: $\hat{C} \leftarrow \{\hat{C}_i\}$;

7: while $|\hat{C}| \neq k$ do

利用式(2)选取数据项 \vec{m}_i 且 $\vec{m}_i \in M - \hat{C}$; $\hat{C} \leftarrow \hat{C} \cup \{\vec{m}_i\}$;

8: end while

9: return \hat{C} ;

10: Map 任务结束;

11: Reduce 任务:

12: Map 任务的输出构成中间簇中心集 C_w ;

13: 从集合 C_w 中随机选择一个初始中心 \hat{C}_1 ;

14: $\hat{C} \leftarrow \{\hat{C}_1\}$;

15: while $|\hat{C}| \neq k$ do

利用式(4)选取数据项 \vec{m}_i 且 $\vec{m}_i \in M - \hat{C}$;

$\hat{C} \leftarrow \hat{C} \cup \{\vec{m}_i\}$;

16: end while

17: return \hat{C} ;

18: 利用式(5)对 \hat{C} 进行迭代计算, 直到簇中心不再改变, 此时最终簇中心集为 C_f ;

19: return C_f ;

20: 结束

2.2 MRSK 聚类

把整个执行过程放在 MapReduce 框架中。数据块处理放在 map 阶段, 中间簇中心集的处理放在 reduce 阶段。如前所述, 提出方案的主要优势是使用了数据流单遍算法, 降低了迭代过程中的 I/O 消耗, 同时也提高了执行速度。

在这两个阶段中, 均使用了 K -means++ 算法, 这是因为 K -means++ 的初始化 Seeding 算法有助于提高算法的聚类效果。对于每个数据块, 利用 K -means++ 算法取得在此数据块上的 k 个聚类中心。因此, 如果有 c 个数据块, 在每个数据块上使用 K -means++ 后, 将会生成一个拥有 $k \cdot c$ 个元素的中间簇中心集合。这个集合就是 map 阶段的输出。

此外, 使用聚类中心的权重来帮助提高 MRSK 的精确度。每个聚类中心分配的数据项的个数表明了该中心的重要性。因此, 对于获得中间簇中心, 必须拓展 K -means++ 算法以支持加权。为了达到此目的, 需要改进初始化聚类中心计算方法(式 2)和聚类中心计算方法(式 1)。

在 K -means++ 算法中, 利用式(4)初始化聚类中

心, 在之后的迭代过程中, 使用式(5)计算簇中心。

在提出算法中, 对 K -means++ 算法进行如下改进, 以使其支持加权。设 $W(\vec{m}_i)$ 作为 \vec{m}_i 的权重, $W(\vec{m}_i)$ 定义如下:

$$W(\vec{m}_i) = |S(\vec{m}_i)| \quad (4)$$

其中, $S(\vec{m}_i)$ 为以 \vec{m}_i 为质心的元素构成的集合。初始化簇中心利用式(5):

$$\frac{D(\vec{m}_i)^2 \cdot W(\vec{m}_i)}{\sum_{m_j \in M} (D(\vec{m}_j)^2 \cdot W(\vec{m}_j))} \quad (5)$$

之后的迭代过程中, 簇中心的选取使用式(6):

$$\hat{C}_i = \text{Center}(C_i) = \sum_{m_j \in C_i} \frac{(\vec{m}_j \cdot W(\vec{m}_j))}{W(\vec{m}_j)} \quad (6)$$

3 算法复杂度分析

典型的聚类算法有三种类型的复杂度: 时间复杂度、I/O 复杂度和空间复杂度。MRSK 有两个过程: map 阶段处理数据块和 reduce 阶段在中间簇中心集上运行 K -means++ 算法。为确定每一种类型的复杂度, 必须考虑这两个阶段。对于 MRSK 复杂性的分析, 提出了以下假设: k 为簇的个数, n 为数据集中数据项的数量, c 为每个数据块的大小, p 为可以并行处理的数据块数, I 为直到收敛为止总的迭代次数。值得注意的是, 假设计算和存储一个数据项的时间复杂度为 $O(1)$ (在大部分相似的文献中也做了这样的假设)。

3.1 数据块大小设定

为了实现聚类, MRSK 需要两个重要的参数: 聚簇个数 k 和数据块大小 c 。如何设置块大小需要考虑, 理论上, 块大小最小为 k , 最大为 n , 但这两个极端值都不切合实际。

块大小为 c 时, 数据块总数为: $\text{num}(\text{chunk}) = n/c$, 在 map 阶段执行完后, 生成的簇中心数为: $\text{Crad}(C_w) = k \cdot n/c$ 。Reduce 阶段在一个单独的机器中执行, 但通常情况下, 同一个 Hadoop 集群中作为数据节点的机器配置相同。所以有: $\text{Crad}(C_w) = k \cdot n/c = c \Rightarrow c = \sqrt{k \cdot n}$

因此, 可以得出内存利用率最高的块大小为 $\sqrt{k \cdot n}$, 而且, 通常情况下块大小不应该大于这个值。在大数据集上, 优先考虑的是内存利用率, 所以选择块大小为 $\sqrt{k \cdot n}$ 。

但是如果数据集不是非常大, 块大小可以被设置为较小的值, 如 $k \cdot \log n$ 或者 k 的倍数。因为块较小, 可以产生更多的中间簇中心, 从而可以更好地代表原始数据集, 得到的聚类结果也会更好。

3.2 I/O 复杂度

Map 阶段的 I/O 复杂度为 $O(n/p)$ 。事实上,整个数据集只读了一次。Reduce 阶段的 I/O 复杂度为 $O(k \cdot n/c)$ 。因此,MRSK 的 I/O 复杂度为 $O(n/p + k \cdot n/c)$ 。如果 $c = \sqrt{k \cdot n}$,I/O 的复杂度将变为 $O(n/p + \sqrt{k \cdot n})$ 。

3.3 空间复杂度

MRSK 的空间复杂度与块大小以及可并行处理的数据块的数目相关。Map 阶段为了保持 p 个数据块在内存中并行处理,所需空间为 $O(p \cdot c)$ 。Reduce 阶段为了存储中间簇中心所需空间为 $O(k \cdot n/c)$ 。因此 MRSK 的空间复杂度为 $O(p \cdot c + k \cdot n/c)$ 。最优情况下, $c = \sqrt{k \cdot n}$,空间复杂度为 $O(p \cdot \sqrt{k \cdot n})$ 。

3.4 时间复杂度

在 map 阶段,会有 n/c 个数据块需要处理,处理时,每个块上要运行 K -means++。最优情况下 K -means++结果的复杂度为 $O(\log k)$ 。因此,map 阶段耗时 $O(k \cdot n/p)$ 。在 reduce 阶段,在中间簇中心集有数据项 $k \cdot c/p$,因此此阶段运行 K -means++算法耗时为 $O(k^2 \cdot n/c)$ 。综上所述,整个 MRSK 算法的时间复杂度为 $O(k \cdot n/p + k^2 \cdot n/c)$ 。如果 $c = \sqrt{k \cdot n}$,时间复

杂度为 $O(k \cdot n/p + k \cdot \sqrt{k \cdot n})$ 。值得注意的是,虽然 reduce 阶段也可以并行执行,但是,因为中间簇中心集相对于输入数据集非常小,它可以非常快地在一个单独机器中执行,所以在 reduce 阶段并不需要采用 MapReduce 作业进行处理。

3.5 与相关工作进行复杂度对比

在前面提到的 PKMeans 和 SKMA 中,PKMeans 利用了 Apache Hadoop Mahout 项目,是标准 K -means 在 MapReduce 框架中的实现;SKMA 算法将原始数据集分割为多个小的子集,在每个子集上利用 K -means#算法选择 k 个中心构成中间簇中心集,然后利用 K -means++算法^[11]从中间簇中心集中选出最终的 k 个中心。

表 1 分别列出了 PKMeans、SKMA 和 MRSK 的三种复杂度。在大数据集中,数据量 n 很大,所以聚类算法要达到收敛,需要经过多次迭代,从而导致 I 很大,而且容易得到 $n \gg I > k > p$ 。可以发现,在 I/O 复杂度上,PKMeans 最高,MRSK 与 SKMA 相当;但是在时间复杂度和空间复杂度上 MRSK 均比其他两种算法低。通过比较可以得出,相对于另外两种算法,MRSK 复杂度最低。

表 1 复杂度比较

| 算法名称 | 时间复杂度 | 空间复杂度 | I/O 复杂度 |
|---------|---|------------------------------------|---|
| PKMeans | $O(I \cdot n/p)$ | $O(d \cdot n)$ | $O(I \cdot (n/p + k))$ |
| SKMA | $O(\log k \cdot k(n/p + \sqrt{k \cdot n}))$ | $O(\log k \cdot \sqrt{k \cdot n})$ | $O(n/p + \sqrt{k \cdot n})O(n/p + I \cdot k)$ |
| MRSK | $O(k \cdot (n/p + \sqrt{k \cdot n}))$ | $O(p \cdot \sqrt{k \cdot n})$ | $O(n/p + \sqrt{k \cdot n})O(n/p + I \cdot k)$ |

4 实验与结果分析

4.1 实验设置

实验中,使用 6 台服务器搭建 Hadoop 集群,每台机器 CPU 为 Intel Xeon E5520 * 2,内存 16 G。机器上安装 Centos6. 5 (64 位) 系统,Open JDK 7, CDH5 (Cloudera Hadoop 5) 和 Mahout0. 8。

主要目标是比较 MRSK 算法和另外两种算法的执行速度和精确度。为了比较三种算法对于大数据集的处理能力和效果,生成了四个大型合成数据集,分别包括 10,50,100,和 500 个簇。每个数据集拥有 10 亿数据项。每个数据项有 5 个特征,标准偏差为 10,因此,最优聚类时,SSE 大约为 $5 \cdot 10^{11}$ 。

4.2 评价指标

在实验中,主要比较在相同情况下三种算法的执行速度和聚类效果。执行速度通过执行时间来比较。聚类效果通过误差平方和函数 SSE (Sum of Squares for Error) 来比较。SSE 显示簇的紧凑性,SSE 值低,意味着聚类效果更好。SSE 计算公式如下:

$$SSE(C,M) = \sum \text{mind}(\vec{m_i}, C_j)^2 \tag{7}$$

4.3 实验结果分析

标准 K -means 和 SKMA 算法,利用 Apache Mahout 项目中基于 MapReduce 框架的版本。图 1 为 k 增加时 PKMeans、SKMA 和 MRSK 的执行时间变化图。图 2 为 k 增加时三种算法的 SEE 变化图。

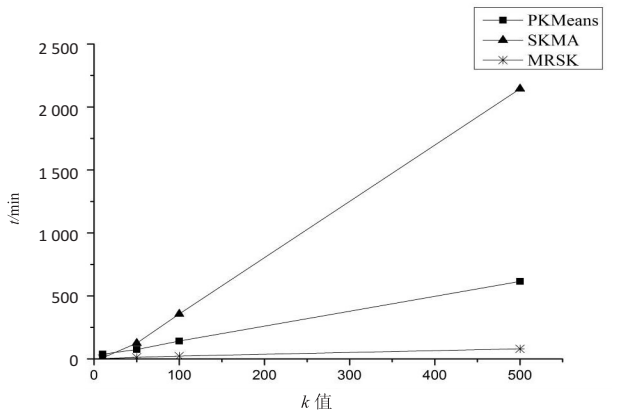


图 1 k 和时间的关系

从图 1 的结果可以看出,相对于其他方案,MRSK

的速度更快。尤其是 k 大于 100 时, MRSK 的速度远快于另外两种算法。在图 2 中, MRSK 的聚类效果也比另外两种算法要好, 随着 k 值增大, SSE 越来越接近 $5 \cdot 10^{11}$, 而 $5 \cdot 10^{11}$ 正是预估的理论值。图中, PKMeans 的聚类效果一直比另外两种算法差。 $k = 10$ 时, SKMA 和 MRSK 的 SSE 值大小相当, k 大于 10 时, MRSK 的 SSE 值逐渐低于 SKMA。而且, 随着簇的增加, MRSK 一直保持着较低的 SSE 值。

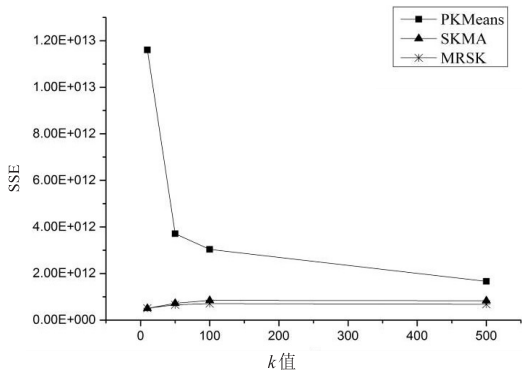


图 2 k 和 SSE 的关系

实验结果表明, MRSK 在保证聚类效果的前提下, 降低了 I/O 开销, 提高了执行速度。

5 结束语

针对 K -means 的迭代性质并不能在 MapReduce 框架中被模型化, 需要反复读写磁盘导致极高 I/O 开销的问题, 提出了基于 MapReduce 的单遍 K -means 算法。理论分析和实验结果均表明, MRSK 有效提高了算法执行速度, 降低了 I/O 开销, 为处理大数据集上的大量聚簇问题提供了一个有益的解决思路 and 方案。下一步的研究重点是利用包括 k -d 树在内的临近搜索方法, 进一步降低算法的时间复杂度。

参考文献:

[1] Wang L, Tao J, Ranjan R, et al. G-Hadoop: MapReduce across distributed data centers for data-intensive computing [J]. Future Generation Computer Systems, 2013, 29(3): 739

-750.

[2] 王 珊, 王会举, 覃雄派, 等. 架构大数据: 挑战、现状与展望[J]. 计算机学报, 2011, 34(10): 1741-1752.

[3] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[C]//Conference on symposium on operating systems design & implementation. [s. l.]: USENIX Association, 2004: 107-113.

[4] Dean J, Ghemawat S. MapReduce: a flexible data processing tool[J]. Communications of the ACM, 2010, 53(1): 72-77.

[5] White T. Hadoop: the definitive guide[M]. [s. l.]: [s. n.], 2010.

[6] Thangavel S K, Thampi N S, Johnpaul C I. Performance analysis of various recommendation algorithm using apache Hadoop and mahout [J]. International Journal of Scientific & Engineering Research, 2013, 4(12): 279-287.

[7] Zhao W, Ma H, He Q. Parallel k-means clustering based on MapReduce [C]//IEEE international conference on cloud computing. [s. l.]: IEEE, 2009: 674-679.

[8] 谢国富, 王文成. 单遍数据读取的 GPU 上的多片元效果绘制[J]. 计算机学报, 2011, 34(3): 473-481.

[9] Ailon N, Jaiswal R, Monteleoni C. Streaming k-means approximation [C]//Conference on neural information processing systems. Vancouver, British Columbia, Canada: [s. n.], 2009: 10-18.

[10] Hadian A, Shahrivari S. High performance parallel k-means clustering for disk-resident datasets on multi-core CPUs[J]. Journal of Supercomputing, 2014, 69(2): 845-863.

[11] Arthur D, Vassilvitskii S. k-means++: the advantages of careful seeding[C]//Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms. [s. l.]: Society for Industrial and Applied Mathematics, 2007: 1027-1035.

[12] Hartigan J A, Wong M A. A K-means clustering algorithm [J]. Applied Statistics, 2013, 28(1): 100-108.

[13] Bahmani B, Moseley B, Vattani A, et al. Scalable k-means++ [J]. Proceedings of the VLDB Endowment, 2012, 5(7): 622-633.

[14] 张培伟. 基于改进 Single-Pass 算法的热点话题发现系统的设计与实现[D]. 武汉: 华中师范大学, 2015.