DOI:10.7544/issn1000-1239.2016.20150616
53(6):1400-1409,2016

EDDPC:一种高效的分布式密度中心聚类算法

巩树凤　　张岩峰
(东北大学计算机科学与工程学院　沈阳　110819)
(shidashufeng@163.com)

EDDPC: An Efficient Distributed Density Peaks Clustering Algorithm

Gong Shufeng and Zhang Yanfeng
(*College of Computer Science and Engineering*, *Northeastern University*, *Shenyang* 110819)

Abstract Clustering is a commonly used method for data relationship analytics in data mining. The clustering algorithm divides a set of objects into several groups (clusters), and the data objects in the same group are more similar to each other than to those in other groups. Density peaks clustering is a recently proposed clustering algorithm published in Science magazine, which performs clustering in terms of each data object's ρ value and δ value. It exhibits its superiority over the other traditional clustering algorithms in interactivity, non-iterative process, and non-assumption on data distribution. However, computing each data object's ρ and δ value requires to measure distance between any pair of objects with high computational cost of $O(N^2)$. This limits the practicability of this algorithm when clustering high-volume and high-dimensional data set. In order to improve efficiency and scalability, we propose an efficient distributed density peaks clustering algorithm—EDDPC, which leverages Voronoi diagram and careful data replication/filtering to reduce huge amount of useless distance measurement cost and data shuffle cost. Our results show that our EDDPC algorithm can improve the performance significantly (up to 40x) compared with naive MapReduce implementation.

Key words density peaks; data clustering; Voronoi partition; MapReduce; big data

摘　要 聚类分析是数据挖掘中经常用到的一种分析数据之间关系的方法.它把数据对象集合划分成多个不同的组或簇,每个簇内的数据对象之间的相似性要高于与其他簇内的对象的相似性.密度中心聚类算法是一个最近发表在《Science》上的新型聚类算法,它通过评估每个数据对象的2个属性值(密度值ρ和斥群值δ)来进行聚类.相对于其他传统聚类算法,它的优越性体现在交互性、无迭代性、无数据分布依赖性等方面.但是密度中心聚类算法在计算每个数据对象的密度值和斥群值时,需要$O(N^2)$复杂度的距离计算,当处理海量高维数据时,该算法的效率会受到很大的影响.为了提高该算法的效率和扩展性,提出一种高效的分布式密度中心聚类算法EDDPC (efficient distributed density peaks clustering),它利用Voronoi分割与合理的数据复制及过滤,避免了大量无用的距离计算开销和数据传输开销.实验结果显示:与简单的MapReduce分布式实现比较,EDDPC可以达到40倍左右的性能提升.

关键词 密度中心;数据聚类;Voronoi分割;MapReduce;大数据

中图法分类号 TP301.6

收稿日期:2015-06-30;**修回日期**:2015-10-29

基金项目:国家自然科学基金项目(61300023,61528203,61272179);中央高校基本科研业务费专项资金项目(N141605001,N120816001)
This work was supported by the National Natural Science Foundation of China (61300023, 61528203, 61272179) and the Fundamental Research Funds for the Central Universities (N141605001, N120816001)

通信作者:张岩峰(zhangyf@cc.neu.edu.cn)

聚类在很多应用中是一种基础的数据分析方法,例如社会网络分析、智能商务、图像模式识别、Web 搜索和生物学等.当前存在很多聚类算法[1],这其中包括基于划分的方法(如 k-medoids[2], k-means[3])、基于层次的方法(如 AGNES[4])、基于密度的方法(如 DBASCAN[5])、基于网格的方法(如 STING[6]),还有面向大数据的快速自适应同步聚类算法 FAKCS[7]等.

密度中心聚类算法[8]是由 Alex 和 Alessandro 最近在《Science》上提出的一个新型聚类算法.它区别于其他聚类算法,基于 2 点假设进行设计:1)聚类中心点的密度不低于它附近点的密度;2)聚类中心点与密度比它大的点(另一个聚类中的点)的距离非常远.在该算法的设计中每个点有 2 个属性:1)密度值 ρ;2)斥群值 δ(与密度值比自己大的点的距离的最小值).密度值 ρ 越大说明该点越有可能是聚类中心,斥群值 δ 越大说明该点越有可能代表一个新的聚类,而只有当密度值 ρ 和斥群值 δ 都较大时,该点才更可能是某一个聚类的密度中心.算法根据这 2 个值的大小来选择密度中心点.

相对于诸多传统聚类算法,密度中心聚类算法具有 3 个优点:

1) 交互性.不同于 k-means 等聚类算法,要求用户在算法执行前指定聚类的个数 k.密度中心聚类算法在计算出每个点的密度值 ρ 和斥群值 δ 之后,由用户根据 ρ 值和 δ 值来确定聚类的个数.

2) 无迭代性.与其他算法(例如 EM clustering 和 k-means)相比,该算法只需要遍历 1 次数据集即可完成,不需要多次迭代.

3) 无数据分布依赖性.算法的聚类形状没有偏倚,可以适用于多种环境下数据的聚类[1].

尽管密度中心聚类算法有以上诸多优点,但是计算密度值 ρ 和斥群值 δ 需要测量数据集中任意 2 点之间的欧氏距离,复杂度为 $O(N^2)$.当处理海量高维数据时,算法的实现涉及到大量的高维欧氏距离计算,造成大量的计算开销,使其无法在单机环境下运行,严重影响了算法的实用性.

针对以上问题,为了提高算法执行效率,本文基于 MapReduce[9]实现了简单分布式密度中心聚类(simple distributed density peaks clustering, SDDPC)算法,该算法虽然可以利用多台节点分布式执行算法,但是仍然需要大量的距离计算开销和数据传输开销.我们把它作为待比较的基准方法,并提出了一种高效的分布式密度中心聚类(efficient distributed density peaks clustering, EDDPC)算法.它首先利用 Voronoi 分割将数据集分成几个互不相交的组,由于对象的密度值 ρ 和斥群值 δ 的计算可能会用到其他组中的数据,本文提出了高效的数据复制过滤模型,将一少部分满足特定条件的数据在分组间复制,过滤掉无用数据.各分组并行地根据分配数据和部分复制数据,在分组内局部计算各点的密度值 ρ 和斥群值 δ,并从理论上保证结果的正确性,从而大大提高了算法的执行效率和可扩展性.

本文的主要贡献如下:

1) 提出了一种基于 MapReduce 的高效分布式密度中心聚类算法 EDDPC,并且在开源的 Hadoop 框架上实现了该算法.

2) 针对 ρ 值计算,设计了一种数据复制模型;针对 δ 值计算,设计了 2 种数据过滤模型,从而减少了数据对象的复制量和距离计算量,提高了算法的执行效率.

3) 在多个真实数据集上对分布式密度中心聚类算法进行了实验和性能评估,实验结果验证了该算法的高效性.

1 相关工作

本节首先简单介绍一下密度中心聚类算法,然后介绍用到的 MapReduce 编程模型和 Voronoi 图数据分割法.

1.1 密度中心聚类算法介绍

密度中心聚类算法[8]基于数据点的密度值 ρ 和斥群值 δ 来对数据集进行聚类.

数据点 i 的密度值 ρ_i 为

$$\rho_i = \sum_j \chi(d_{ij} - d_c), \tag{1}$$

其中,$\chi(x)$是一个函数,当 $x < 0$ 时,$\chi(x)=1$,否则 $\chi(x)=0$;d_{ij} 是点 j 到点 i 的距离;d_c 是一个距离临界值.也就是说,ρ_i 为与点 i 的距离小于 d_c 的点的个数.

点 i 的斥群值 δ_i 为

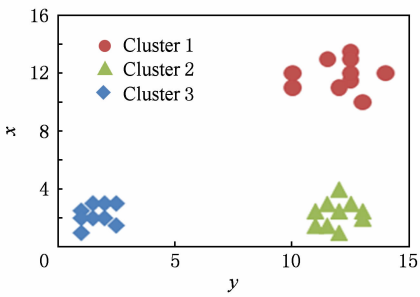
$$\delta_i = \min_{j:\rho_j > \rho_i}(d_{ij}). \tag{2}$$

它代表与密度值比自己大的点的距离的最小值.假设密度比 ρ_i 大的点中,点 j 距离点 i 最近,那么 $\delta_i = d_{ij}$,而点 j 就是点 i 的依附点 σ_i,

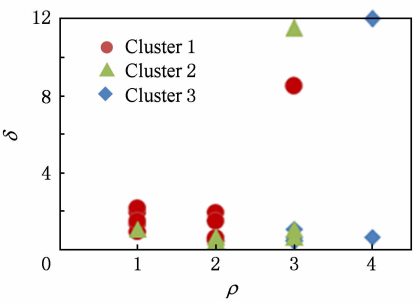
$$\sigma_i = \arg\min_{j \in S, \rho_j > \rho_i}(d_{ij}),$$

说明点 i 可以依附点 j，归属于点 j 所属聚类. 斥群值 δ_i 越小，点 i 距离点 j 越近，这种依附可能性越大，代表点 i 越有可能归属于点 j 所属的聚类；斥群值 δ_i 越大，点 i 距离点 j 越远依附性越小，说明点 i 很有可能是离群点或者是属于另一个聚类. 当某个点 m 的密度值 ρ_m 是所有点中密度最大值，那么点 m 的斥群值 $\delta_m=\max_j(d_{mj})$.

密度中心聚类算法将每个数据点的 ρ 值和 δ 值表示在一个 2 维决策图(decision graph)上. 如图 1(a)展示了一个数据集的分布情况；图 1(b)是根据图 1(a)中每个点的 ρ 值和 δ 值绘制成的决策图. 用户根据决策图中数据点的分布情况，圈出 ρ 值和 δ 值都很大的数据点作为密度中心，即决策图中右上角的部分数据点. 由于在计算 δ 值时已经记录了每个数据点的依附点，可以根据数据点的依附关系并由用户选取的密度中心反推出每个数据点的所属聚类.



(a) Distribution of points



(b) Decision graph

Fig. 1 Density peaks clustering.

图 1 密度中心聚类算法

1.2 基于 Voronoi 图的分割法

Voronoi 图，又叫泰森多边形或 Dirichlet 图，是一个关于空间划分的基础数据结构. 基于 Voronoi 图的划分是指在给定的数据对象集 S 上，选择 M 个对象作为种子对象，这 M 个种子对象按照 2 点之间连线的垂直平分线将数据集 S 分割成 M 个互不相交的组，这 M 个分组称为“Voronoi cells”. 数据集 S 中的每个元素按照距离被划分到距离自己最近的种子对象所在的分组中.

Voronoi 图分割法在并行计算中应用非常广泛，文献[10]以 Voronoi 图分割为基础使用 MapReduce 框架解决在大数据情况下的范围搜索和 KNN 查询，如图 2 显示了一个 Voronoi 图将数据集分割成 8 个组的实例. 文献[11]用 Voronoi 图分割法设计出了一种高效的分布式 KNN 连接算法. 受此启发，本文提出的 EDDPC 算法应用 Voronoi 图分割法对数据集分割，使分割后的数据集分组能够在集群中的节点上局部计算密度值和斥群值. 为方便以后叙述做如下定义：Voronoi 图分割种子对象集合为 $\mathbb{P}$，P_i 表示种子对象 p_i 所在的分组.

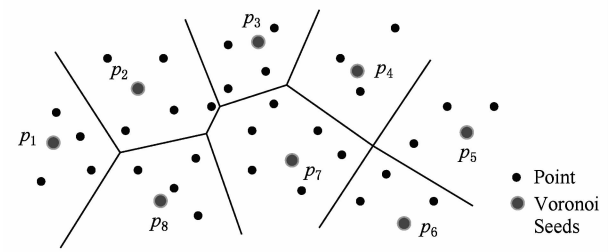


Fig. 2 An example of Voronoi.

图 2 Voronoi 图示例

1.3 MapReduce 与 Hadoop

MapReduce 是一个当前流行的分布式编程模型. MapReduce 提供了 2 个主要函数 *map* 和 *reduce*. 函数 *map* 和函数 *reduce* 由用户自己定义，函数 *map* 根据用户自定义的功能处理输入数据，并且以〈*key*，*value*〉的形式输出. MapReduce 自动收集具有相同 *key* 值的 *value*，并且以〈*key*，*list*(*value*)〉的形式作为 *reduce* 的输入，*reduce* 根据用户定义的功能进行处理，最终以〈*key*，*value*〉的格式输出. MapReduce 的大致工作流程为

$$map(k_1,v_1)\rightarrow list(k_2,v_2),$$

$$reduce(k_2,list(v_2))\rightarrow list(k_3,v_3).$$

Hadoop 是一个实现了 MapReduce 编程模型的开源框架. 在 Hadoop 上的数据默认存储在分布式文件系统 HDFS 上. 本文将基于 MapReduce 模型设计高效的分布式密度中心聚类算法.

2 密度中心聚类在 Hadoop 上的简单实现

本节基于 MapReduce 框架实现基本的分布式密度中心聚类算法 SDDPC.

在 1.1 节已经介绍过，密度中心聚类算法的主要步骤是计算每个点的密度值 ρ 和斥群值 δ，因此分布式密度中心聚类算法的重点是分布式地计算 ρ 值和

δ 值.在得到每个数据点的 ρ 值和 δ 值后,可以将其收集到 1 台机器上绘出决策图进行聚类.接下来描述密度中心聚类算法的 4 个基本计算步骤:1)选择 d_c 长度的预处理阶段(使用 MapReduce);2)计算 ρ 值;3)根据 ρ 值计算出 δ 值;4)绘制决策图并对数据对象聚类.

2.1 预处理:寻找合适 d_c

距离阈值 d_c 在密度中心聚类算法的实现过程中是一个非常重要的参数,如式(1)所示该参数用来计算每个点的密度值 ρ.文献[1]给出了一个 d_c 的经验估计方法,即所有点对距离从大到小排序后的 1%~2%处,因此本文参考该方法来选取 d_c 值.

在大规模数据集下估计 d_c 值是一个开销非常大的任务,即使在分布式环境中,对数据排序也是一个复杂的工作.因此,在本文中应用了采样的理念来对 d_c 值进行估计.由于数据量大,因此我们基于 MapReduce 进行采样,函数 *map* 基于水塘采样(reservoir sampling)方法[12]对任意 2 点的距离进行分布式采样,由于样本数据少,所以可以将采集后的数据发送到单个 *reduce*,由其进行距离计算并排序,然后选择出合适的 d_c 值.

2.2 计算 ρ 值

正如式(1)所示,ρ_o 的计算需要知道对象 o 到其余每个点 $j\in S$ 之间的距离 d_{oj}.对象之间距离的计算在 MapReduce 中可以实现,有 2 种实现方法.在方法 1 中,函数 *map* 选出 1 个对象,然后将其发送给其他每个对象(假设总共有 N 个对象);函数 *reduce* 收集每个对象上由 *map* 发过来的对象,计算该对象与它们之间的距离,并且根据式(1)计算出 ρ 值.显然这种简单的实现需要很大的开销,由于每个对象都需复制到其他对象所在机器进行距离计算,因此 shuffle 开销是 $O(N^2)$,计算开销也是 $O(N^2)$.

为节省 shuffle 开销,方法 2 使用分块技术,将整个数据集 S 分成 n 份互不相交的子集.例如,$S=\bigcup_{1\leqslant k\leqslant n} S_k$,$S_k\cup S_l=\varnothing$,$k\neq l$.同样分组由函数 *map* 来完成,每个子集 S_k 发送给其他的子集.函数 *reduce* 在 2 个子集上进行距离计算和 ρ 值计算,shuffle 开销是 $O(n\times N)$,计算开销是 $O(N^2)$.

由于在方法 2 中需要将数据分块,因此需要 1 个 MapReduce 作业来完成数据的分块,但是仍然比方法 1 效率高,尤其是当 $n\ll N$ 时,shuffle 开销会急剧减少.在方法 2 中每个分块与其他 $n-1$ 块运算,分块中的每个对象 o 就会得到 n 个 ρ 值,因此需要 1 个 MapReduce 作业来合并每个对象 o 的 ρ 值.

2.3 计算 δ 值

δ 值需要根据式(2)计算.算法实现步骤和计算 ρ 值时相似,需要 2 个 MapReduce 作业来完成 δ 值的计算,第 1 个 MapReduce 作业结束之后,每个对象会得到 n 个 δ 值,第 2 个 MapReduce 作业从这 n 个值中选出最小的一个值作为 δ 值.shuffle 开销是 $O(n\times N)$,计算开销是 $O(N^2)$.

2.4 绘制决策图并聚类

将得到的 ρ 值集合和 δ 值集合汇集到 1 台机器上(ρ 值集合和 δ 值集合的大小远小于点数据集 S),并基于得到的 ρ 值和 δ 值绘制决策图.用户根据 ρ 值和 δ 值的分布情况选出绘制在决策图中右上角的部分数据点作为聚类中心,每个点根据计算 δ 值时得到的依附关系,由聚类中心点反推出每个数据点的所属聚类.

3 基于 Voronoi 分割和数据点复制过滤的高效分布式密度中心聚类算法

虽然我们设计了简单的分布式密度中心聚类算法 SDDPC,弥补了单机运行时的缺陷,但是由于存在大量的 shuffle 开销和计算开销,因此并不是一种高效的方法.本节基于 Voronoi 数据分割提出了一种高效分布式密度中心聚类算法 EDDPC.该算法包括 1 个预处理阶段和 2 个完整的 MapReduce 作业.EDDPC 算法将数据分组后,各分组中的数据对象独立并行执行于集群中的各节点中,在分组内部局部计算 ρ 值和 δ 值,避免因计算所有对象间的距离而造成的大量开销.

在预处理阶段需要完成的任务是选择 Voronoi 分割时所需要的种子.2 个 MapReduce 作业分别用来计算 ρ 值和 δ 值.由于对数据分组之后,在某分组中计算某些对象的 ρ 值和 δ 值时需要其他分组的部分对象,因此各分组独立地直接计算 ρ 值和 δ 值将得到错误的结果.例如,对于某数据对象 $o\in P_i$,在计算 ρ_o 时,$|o,q|<d_c$,而 $q\notin P_i$;在计算 δ_o 时,对象 o 的 δ 值依附点 $\sigma_o\notin P_i$.因此在计算 ρ 值和 δ 值时需要对数据进行复制,从而使计算出的 ρ 值和 δ 值成为正确值.为了减少数据的复制量,针对分组内的 ρ 值计算,本文提出了 d_c 复制模型;针对分组内的 δ 值计算,本文提出了 2 个过滤模型.

3.1 种子选择并且计算 d_c

EDDPC 算法应用了 1.2 节提到的基于 Voronoi 图的分组方法.该分组方法使用点之间的距离关系

来进行分组,是一个非常有效的空间分组方法,尤其是在高维空间上.

该分组方法根据种子对象进行分组,因此在分组之前需要先挑选出合适的种子对象.在挑选种子时使用2.1节中选择 d_c 值时使用的水塘采样算法.在挑选种子对象的同时也能够得到 d_c 值,无需额外的 MapReduce 作业.

3.2 计算 ρ 值及其复制模型

经过预处理阶段的操作,可以得到 Voronoi 分组需要的种子对象集 $\mathbb{P}$,然后创建1个 MapReduce 作业对数据集分组并计算 ρ 值.首先根据选出的种子对象进行分组,每个对象 o 与种子对象集 $\mathbb{P}$ 中的每个对象 p_i 计算距离,得到对象 o 到每个种子对象 p_i 的距离 d_{op_i},比较对象 o 到每个种子对象的距离,选出距离 o 最近的种子对象 p_j,将对象 o 分配到 p_j 所在的分组 P_j 中.

分组后,整个数据集分成若干个互不相交的组,因此,在计算对象 o 密度值 ρ_o 时,可能得到的是一个错误的值.如图3(a)所示,o 靠近分组的边缘线时计算得到 $\rho_o=8$,而实际值 $\rho_o=11$.

为了得到正确的 ρ_o 值需要将分组区域 B 中的3个点复制到区域 A 中.据此可以推出每个分组 S_i 中的点不仅仅要包含由 Voronoi 分组后点集 P_i,也要包含本组中的所有对象的邻居集合 R_o.即

$$S_i = P_i \bigcup_{\forall o \in P_i} R_o, \tag{3}$$

其中,$R_o=\{q \mid \forall q \in S, |q,o| < d_c\}$.

在分组之前无法得到对象 o 的 d_c 邻居集合 R_o 包含哪些对象,但是根据 R_o 的定义可知,当 $q \in R_o$ 时,$|o,q| < d_c$.

我们提出保证分组正确计算 ρ 值的对象复制方案.如图3(b)所示,将所有距离边缘点小于 d_c 的点全部发送到相邻的分组中,并由定理1保证其正确性.

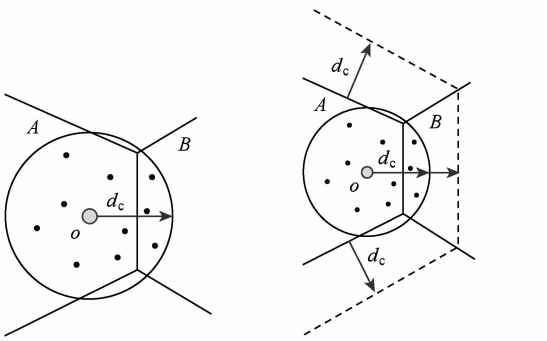


Fig. 3 Example of replication.

图3 数据复制示例

定理1. $\forall o \in P_j$,那么对象 o 可能成为 P_i 中某个对象的 R_o 点而被复制到的 P_i 中的条件为

$$\frac{|o,p_i|^2-|o,p_j|^2}{2|p_i,p_j|} < d_c. \tag{4}$$

证明. 如图4所示,连接点 p_i 和点 p_j,l 为分组 P_i 和 P_j 的分界线,$\forall o \in P_j$,过点 o 做 p_i,p_j 连线的垂线,交 p_i,p_j 连线于点 k,则 $|o,p_i|^2-|p_i,k|^2=|o,k|^2$,$|o,p_j|^2-|p_j,k|^2=|o,k|^2$,因为 $|p_i,k|+|p_j,k|=|p_i,p_j|$,且由 Voronoi 划分定义可知 $|o,l|=\frac{1}{2}|p_i,p_j|-|p_j,k|$,经过合并整理得:

$$|o,l| = \frac{|o,p_i|^2-|o,p_j|^2}{2|p_i,p_j|}. \tag{5}$$

当 $|o,l| < d_c$ 时,结合图3(b),对象 o 可能是属于 P_i 中某个对象 q 的 d_c 邻居集合 R_q.将这些潜在的可能的 d_c 邻居复制到 P_i 中,可以保证分组 P_i 中每个对象 ρ 值计算的正确性. 证毕.

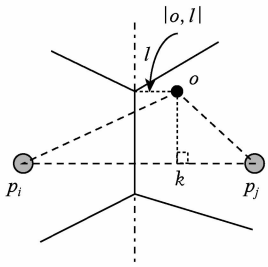


Fig. 4 An example of equation (5).

图4 式(5)示例

式(5)中的 $|o,p_i|$,$|o,p_j|$ 在对数据对象 o 进行分组时已经得出,$|p_i,p_j|$ 在种子选择时算出,由于种子对象数据量少,因此所有种子间的距离计算可以单机完成,并在分布式计算时加载使用.

根据定理1可在分组的同时对数据进行复制,分组完成后每个组内的数据会包含2种数据对象:一种是由 Voronoi 数据分割方法分组后在每个 Voronoi cell 里的对象集 P_i,该集合中的对象称之为原始对象;另一种是为了计算原始对象的密度值 ρ 而将其他组中的数据复制进来的对象集 S_i-P_i,该集合中的对象称之为复制对象.

MapReduce 实现:函数 *map* 首先加载事先采样得到的 Voronoi 种子对象集合 $\mathbb{P}$,然后计算每个对象 o 到所有种子对象 $p_i \in \mathbb{P}$ 的距离 $|o,p_i|$,将该对象发送到距离最近的种子所在的分组中,即对象 o 所属的 Voronoi 分组.同时,函数 *map* 根据式(5)把满足复制条件的对象发送到邻居分组中.函数 *reduce* 负责某个 Voronoi 分组,根据式(1)计算每个

原始对象的 ρ 值(无须计算复制对象的 ρ 值).假设 α 是复制因子,代表平均每个点的副本数量,如果共产生 n 个 Voronoi 分组,那么计算密度值 ρ 需要复制 $\alpha\times N$ 个对象,所以 shuffle 开销是 $O(\alpha\times N)$,而所需的距离计算开销是 $O(\alpha\times N^2/n)$.

3.3 复制过滤计算 δ 值

在得到密度值 ρ 之后,可以根据组内每个对象的 ρ 值和式(2)计算每个对象的 δ 值.只在分组内部计算 δ 值会有一定的局限性,算出的 δ 值并不是实际的 δ 值,而是一个不小于实际 δ 值的局部 δ 值,记为 δ'.如图 5 所示,由于对象 p_2 不在对象 o 所在的分组中,因此对象 o 在分组内部将 p_1 点作为 δ 值依附点,最终算出的 δ' 就比实际的 δ 值大.

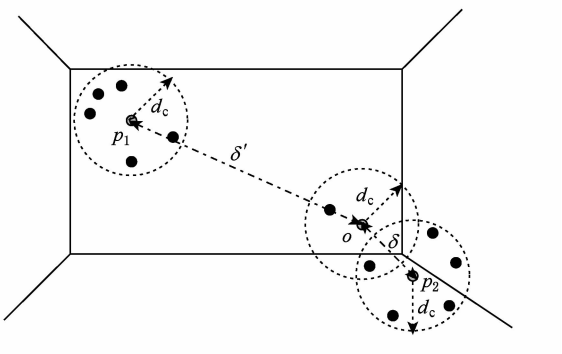


Fig. 5 δ value of object o.

图 5 对象 o 的 δ 值

为了能够求出精确的斥群值 δ 需要第 2 个 MapReduce 作业.如图 5 所示,为得到对象 o 的实际 δ 值,需要将组外的 p_2 复制到对象 o 所在的分组中.因此为了计算每个组中对象的斥群值 δ 需要将一些组外的对象复制到本组中.由式(2)可知,对象 o 的斥群值 δ 是 o 与密度比自己大的对象的最小距离,若某对象的密度值大于分组 P_i 中对象密度的最小值,它便有可能成为 P_i 中某对象 o 的依附点 σ_o.所以我们有如下引理:

引理 1. $\forall q\in P_j$,则对象 q 可能成为分组 P_i 中某个对象的依附点 σ_o 的条件为

$$\rho_q>\min\rho(P_i), \tag{6}$$

其中 $\min\rho(P_i)=\min\{\rho_o\mid\forall o\in P_i\}$.

在整个数据集 S 中符合式(6)的对象可能非常多,当 P_i 中含有整个数据集 S 中密度最小的对象时,需要将整个数据集全部复制到分组 P_i 中.但是,在所有满足式(6)的对象中也有很多对象是冗余的,这些冗余的对象导致一些额外数据复制开销和距离计算开销.因此,为了减少数据的复制量,需要将不必要的对象过滤.

为了计算组 P_i 中每个对象的 δ 值,经过复制后的分组 S_i 不仅包含原始分组对象集 P_i 同时也应该含有每个对象的 σ_o 点,即:

$$S_i=P_i\bigcup_{\forall o\in P_i}\sigma_o. \tag{7}$$

在计算出 ρ 值之后,对本组内的原始对象根据 ρ 值进行局部 δ' 值计算.由于该分组未必包含原始对象 o 的依附点对象 σ_o(最近的密度大于 ρ_o 的对象),所以得到的 δ' 值是一个比实际值大的数值.但是这个值可以作为对象的斥群值 δ 的一个范围值,$\delta\leqslant\delta'$.因此得到:

$$|o,\sigma_o|\leqslant\delta'. \tag{8}$$

根据 1.1 节算法定义,在分组 P_i 中具有最大密度对象的局部 δ 值应该是 $\delta_m=\max_j(d_{mj})$,但是由于 ρ_m 是在整个数据集中并不一定是最大值,因此为了处理方便,我们设为无穷大.除此之外,每个点都会有一个局部 δ' 值,因此本文根据这种情况提出 2 个复制过滤模型:1)用来计算除具有最大 ρ 值对象之外的所有普通对象的斥群值 δ 的普通对象的依附点过滤模型;2)为了计算具有最大 ρ 值对象的斥群值 δ 的局部最大密度对象的依附点过滤模型.

3.3.1 普通对象的依附点过滤模型

抛开分组内部的最大 ρ 值对象(局部密度中心点),只考虑剩余普通对象的 δ 值的计算.

在第 1 个 MapReduce 作业中可以计算出普通对象的 δ' 值,它可以作为实际 δ 值的上界.在分组 P_i 中密度最大的对象的 δ' 值为无穷大,我们记次大的 δ' 为 $\mathrm{smax}\delta(P_i)$,则 P_i 中除局部密度中心点之外所有对象的依附点 σ_o 与分组种子的距离满足如下定理:

定理 2. $\forall o\in P_i,\rho_o\neq\max\rho(P_i)$,记 $U(P_i)$ 为集合 P_i 中的所有普通对象的 δ 值依附点 σ_o 到种子对象 p_i 的距离最大值,那么 $U(P_i)$ 满足不等式:

$$U(P_i)\leqslant B(P_i)+\mathrm{smax}\delta(P_i), \tag{9}$$

其中,$B(P_i)=\max\{|o,p_i|,\forall o\in P_i\}$.

证明. $\forall o\in P_i$,由于 $\mathrm{smax}\delta(P_i)$ 是除最大 ρ 值对象以外最大的 δ' 值,因此组内任何普通点的 δ 值不可能大于该值,所以 $|o,\sigma_o|\leqslant\mathrm{smax}\delta(P_i)$,又因为 $B(P_i)=\max\{|o,p_i|,\forall o\in P_i\}$,所以 $|p_i,o|\leqslant B(P_i)$,由三角不等式 $|\sigma_o,p_i|\leqslant|p_i,o|+|o,\sigma_o|$,所以 $U(P_i)\leqslant B(P_i)+\mathrm{smax}\delta(P_i)$. 证毕.

式(9)说明了,P_i 组内除最大 ρ 值对象的所有普通对象的依附点 σ_o 与该组的种子对象 p_i 的距离

上限为 $B(P_i)+\text{smax}\delta(P_i)$. 由此我们得到如下的推论:

推论 1. $\forall o\in S, o\notin P_i$,则对象 o 可能成为 P_i 中某对象的 σ_o 点而被复制到 P_i 中的条件为

$$\begin{aligned}|o,p_i|&\leqslant U(P_i),\\ \rho_o&>\min\rho(P_i).\end{aligned}\tag{10}$$

以上过滤模型没有复制分组中密度最大对象的依附点,接下来我们提出分组中最大密度对象的 σ_o 过滤模型.

3.3.2 局部最大密度对象的依附点过滤模型

局部最大密度对象的依附点过滤模型同样要找到一个指导对象的依附点复制到组 P_i 的取值范围 $mU(P_i)$,该取值范围不同于普通对象依附点过滤模型中的取值范围 $U(P_i)$. 我们首先给出如下定理:

定理 3. 给定 1 个分组 P_i,对象 m 为该分组的局部最大密度对象,即 $\rho_m=\max\rho(P_i)$,假设其依附点为 σ_m,则:

$$|p_i,\sigma_m|\leqslant\min\{2|m,p_i|+|p_j,u|+|p_i,p_j|\},\tag{11}$$

其中 $u\in P_j$ 且 $\rho_u>\rho_m, p_j\neq p_i, p_j\in P$.

证明. 根据三角不等式可知,$|p_i,\sigma_m|\leqslant|p_i,m|+|m,\sigma_m|$,由 σ_m 的定义可得 $|m,\sigma_m|\leqslant\min\{|m,u|\}$,所以 $|p_i,\sigma_m|\leqslant|m,p_i|+\min\{|m,u|\}$,又因为 $|m,u|\leqslant|m,p_i|+|p_i,u|$, $|p_i,u|\leqslant|p_i,p_j|+|p_j,u|$. 所以 $|p_i,D_m|\leqslant\min\{2|m,p_i|+|p_j,u|+|p_i,p_j|\}$.

证毕.

由此得出如下推论:

推论 2. $\forall o\in S, o\notin P_i$,则对象 o 可能成为 P_i 中局部最大密度对象 m 的依附点 σ_m 而被复制到分组 P_i 中的条件为

$$\begin{aligned}|o,p_i|&\leqslant mU(P_i),\\ \rho_o&>\max\rho(P_i),\end{aligned}\tag{12}$$

其中,$mU(P_i)=\min\{2|m,p_i|+|p_j,u|+|p_i,p_j|\}$.

在 2 个复制模型中涉及到每个分组内的次大 δ' 值 $\text{smax}\delta(P_i)$(最大 δ' 值应该是无穷大)、组内的最大 ρ 值 $\max\rho(P_i)$、组内最小 ρ 值 $\min\rho(P_i)$、组内局部最大 ρ 值对象 m 到种子对象的距离 $|m,p_i|$ 和距离该组的种子对象最远的点的距离 $B(P_i)$,这些数据是在完成第 1 个 MapReduce 作业后需要收集的一些关于分组的信息.

2 个过滤模型中的式(10)和式(12)都以 $|o,p_i|\leqslant\theta$ 的形式出现,其中 $\theta=U(P_i)$ 或 $\theta=mU(P_i)$.

对每个对象决定是否要发送到分组 P_i 中,都要与 p_i 做 1 次距离计算. 因此,当 P 中的对象非常多时,开销也会变大,为此本文给出了一种剪枝的策略.

定理 4. $\forall o\in P_j$, $|o,p_i|<\theta$,则 $|o,p_j|>|p_j,p_i|-\theta$.

证明. 如图 6 所示,由直角三角形斜边大于直角边可知 $|p_i,k|<\theta$, $|p_j,k|>|p_j,p_i|-\theta$,又因为 $|o,p_j|>|p_j,k|$, $|p_j,k|>|p_j,p_i|-\theta$,所以 $|o,p_j|>|p_j,p_i|-\theta$.

证毕.

因此当 $|o,p_j|>|p_j,p_i|-\theta$ 时,不必计算 o 到 p_i 的距离,而直接过滤.

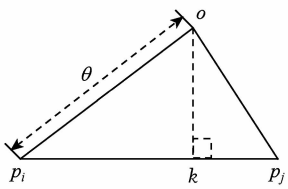


Fig. 6 An example of theorem 4.

图 6 定理 4 示例

MapReduce 实现:在第 2 个 MapReduce 作业中,函数 *map* 根据以上 2 个模型,将满足推论 1 和推论 2 的对象进行相应的复制和发送. 这里需要注意一点,如果某一对象同时满足 2 个条件,为了避免重复计算,只将数据对象复制发送 1 次即可. 在 *reduce* 阶段,只需要比较组内原始对象的密度与复制对象的密度,当密度比自己密度大时计算距离,选择最小距离作为 δ 值,同时记录 δ 值的依附点. 假设 β 是复制因子,代表平均每个点的副本数量,如果共产生 n 个 Voronoi 分组,那么计算斥群值 δ 需要复制 $\beta\times N$ 个对象,所以 shuffle 开销是 $O(\beta\times N)$,而所需的距离计算开销是 $O(\beta\times N^2/n)$.

4 实 验

我们在本地集群和 Amazon EC2 云平台上应用 3 个真实数据集对分布式密度中心算法 SDDPC 和优化的 EDDPC 算法进行了对比实验分析.

4.1 实验环境

Hadoop 本地集群包括 4 台 slave 机器、1 台 master 机器,每台机器配有 Intel I5-4690 3.3 GHz 4Core 处理器、1000 Mbps 以太网卡、1 TB 7200r/m 普通硬盘、4 GB 运行内存. Amazon EC2 集群包括 17 个 m1.medium 节点. 操作系统为 64 位 Ubuntu 14.04 LTS, JDK 版本为 Java1.7, Hadoop 版本为 Hadoop1.2.1.

数据集采用 BigCross[13], facial[14], kdd[15], 其中 BigCross 数据集包含 11 620 300 个点,实验过程中只截取了 50 万个点,每条记录的维度是 57;facial

数据集包含 27 936 个点,每个点 300 维;kdd 数据集包含 145 751 个点,每个点 74 维.

本文所有实验中 SDDPC 算法以 200 个点作为 1 个分块,EDDPC 算法为避免因选择到的种子不同造成分组状态不同,从而导致实验结果不同而对实验效果造成影响,所以 2 种算法均采用 3 次实验取平均值作为最终结果.

4.2 实验结果与分析

4.2.1 整体性能评估

图 7 展示了在 BigCross(随机截取 20 万个点),facial,kdd 数据集上 SDDPC,EDDPC 和原始的密度中心聚类(density peak clustering, DP_Clustering)算法运行时间的比较. EDDPC 种子选取比例为 3‰,运行时间不包括绘制决策图时间. 总体上从图 7 可以看出,EDDPC 算法的运行时间明显要少于 SDDPC 算法,尤其在数据量比较大的 BigCross 和 kdd 数据集上. 在 kdd 数据集上表现最为明显,产生 kdd 运行时间差距的原因是 kdd 数据集中只有 1 个聚类簇,因此在计算 δ 值时 shuffle 开销和计算开销大大减少. 原始 DP_Clustering 算法在单机环境下运行,由于没有 Hadoop 启动时间和多次读文件的时间,因此原始 DP_Clustering 算法在数据量比较小的情况下会比 SDDPC 算法的运行时间快;而在数据量比较大的情况下,原始 DP_Clustering 算法由于是在单机环境下执行,会造成内存溢出,从而无法执行 DP_Clustering 算法.

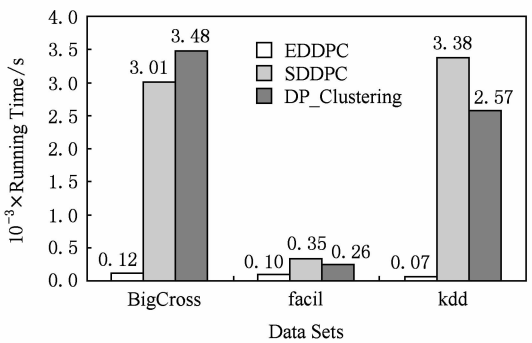


Fig. 7 Running time comparison on three data sets.

图 7 3 种数据集下运行时间比较

4.2.2 数据集大小的影响

为了测试数据集大小变化对算法性能的影响,我们从 BigCross 数据集中分别随机截取了 10 万、20 万、30 万、40 万、50 万个数据对象,分别构成了 5 个大小不同的数据集. 如图 8 显示了 EDDPC 和 SDDPC 算法在这 5 个数据集上运行时间的对比,可以看出改进的 EDDPC 算法时间增长比较缓慢,而 SDDPC 算法以接近平方的速度增长.

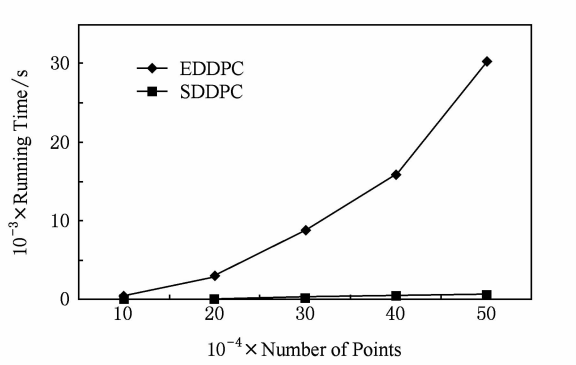


Fig. 8 Running time comparison when varying data size.

图 8 不同数据集大小情况下的运行时间比较

为了分析造成运行时间差距的原因,本文还统计了这 2 种算法在 Hadoop 框架上执行时的通信量(shuffling cost)、每个点的副本个数和点之间距离计算的次数.

图 9 显示了在计算 ρ 值和 δ 值时 EDDPC 和 SDDPC 算法执行过程中每个点的平均副本数量. 其中 EDDPC 算法的副本数量由程序统计而得,并多次统计取平均值;SDDPC 算法的副本数量根据 N/n 计算得到,其中 N 指数据集中点的个数,n 指每个分块中包含的点的个数.

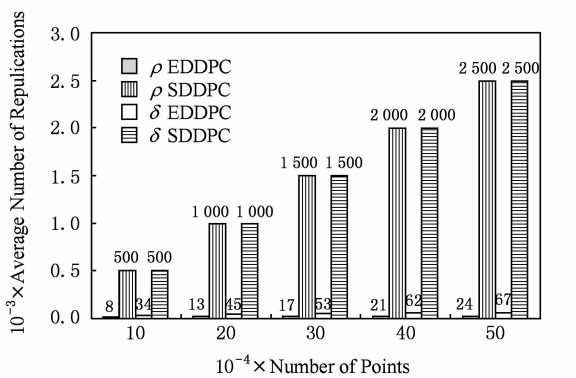


Fig. 9 Comparison of number of points replicas.

图 9 点副本数比较

图 10 显示了在输入数据为 10 万~50 万时 EDDPC 和 SDDPC 算法在 Hadoop 上执行过程中的通信量.

图 11 显示了数据从 100×10^3~500×10^3 时,EDDPC 和 SDDPC 算法在 Hadoop 集群上执行过程中计算 2 点之间距离的计算次数,EDDPC 算法由程序统计,SDDPC 算法根据公式 $2N\times(N-1)$ 计算.

通过分析图 9 可以看到,EDDPC 算法在计算 ρ 值和 δ 值的过程中平均每个点的副本数量比 SDDPC 算法要少,因此 Hadoop 集群中各节点之间的通信量机会变少,如图 10 所示. 同时由于副本

数量的减少和组内各点局部计算距离,从而减少距离计算的次数,如图 11 所示.

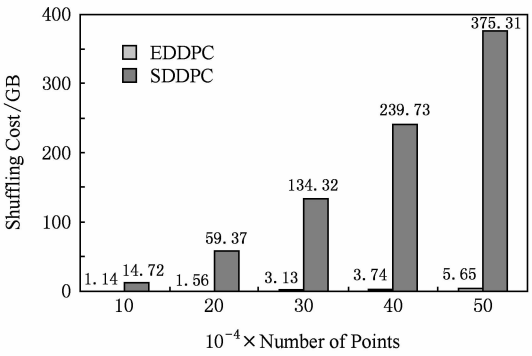


Fig. 10 Shuffling cost comparison.

图 10 Shuffle 通信量比较

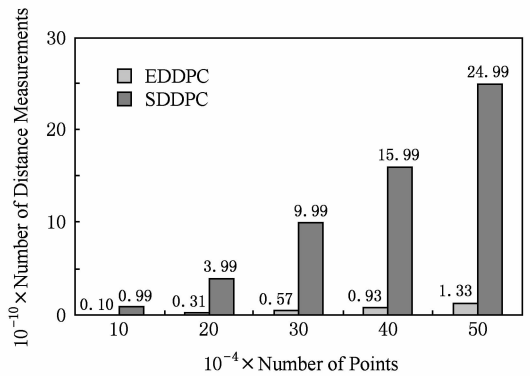


Fig. 11 Comparison of distance measurements.

图 11 距离计算次数比较

4.2.3 分组种子数量的影响

为了探究 EDDPC 算法中种子数量对算法执行效率的影响,本文设计了在相同数据量的点集中相同数据集(facial)下不同种子数量对实验结果造成的影响.

图 12 显示了在种子数量在 5～95 范围内变化时 EDDPC 算法的运行时间,由于选取到的种子不同,造成运行时间不同,因此本实验取 3 次实验的平均值作为本实验的结果值.

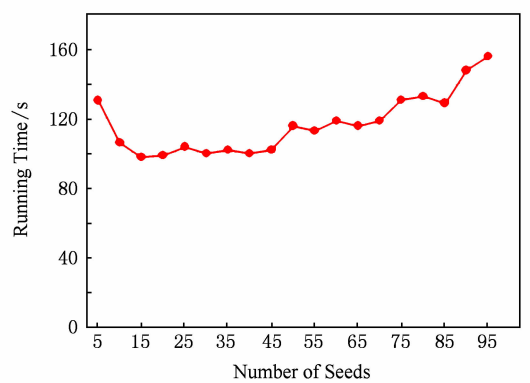


Fig. 12 Running time of EDDPC when varying number of Voronoi partitions.

图 12 EDDPC 中 Voronoi 分组数量不同时的运行时间

由图 12 可以看出,在一定范围内,随种子对象的增多运行时间先变少然后增大. 种子对象在 15～25 时运行时间最少. 由于种子对象的选取可能造成运行时间波动,但是大致趋势仍然是先变小后变大.

我们对图 12 所示现象做如下的分析:当种子对象增多时,数据分组变多,分组内部的数据会减少,因此在计算 ρ 值时分组内部距离计算的开销就会减少. 而在计算 δ 值时,随着分组的增多,$U(P_i)$ 和 $mU(P_i)$ 的值会变小,每个分组内部被复制添加进来的数据对象会减少,因此总体运行时间会减少. 但是随着种子对象的持续增加,分组数量增多,*map* 和 *reduce* 的线程数量就会增加,加大了系统的开销. 同时分组的增多也会使每个对象被复制到其他组的概率变大,因此通信开销也会随之增长,导致运行时间变长.

4.2.4 集群数量的影响

我们在 Amazon EC2 集群上,分别利用 2,4,8,16 个计算节点运行 EDDPC 算法来评估算法扩展性能. 采用 BigCross 数据集中截取的 50 万条数据作为输入数据,实验结果如图 13 所示. 我们以 2 节点运行时间为基准,画出了一条理论最优的扩展性能曲线. 从图 13 可以看出,EDDPC 算法具有不错的扩展性能,虽然性能加速比略低于理论最优的加速比,但是从 4 节点到 16 节点的运行时间几乎是随着节点数量的增加呈线性递减.

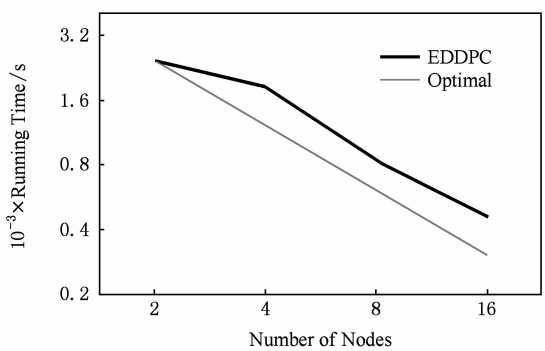


Fig. 13 Running time of EDDPC when varying number of nodes.

图 13 节点不同时的 EDDPC 运行时间

5 结 论

本文发现了密度中心聚类算法的运行效率问

题,并基于 MapReduce 简单实现了分布式密度中心聚类算法——SDDPC. 为了进一步提高 SDDPC 算法的性能,本文基于 Voronoi 图分割的方法提出一种高效的分布式密度中心聚类算法——EDDPC. 为了正确计算 ρ 值和 δ 值,本文分别给出了一个数据对象复制模型和 2 个数据对象过滤模型,将部分其他分组内的必要对象复制到本分组内,这保证了 EDDPC 算法可以在各独立分组内分布式执行 ρ 值和 δ 值的计算. 提出的高效数据复制过滤模型不仅能保证算法执行过程中能够精确计算每个对象的 ρ 值和 δ 值,从而得到与原始 DP_Clustering 算法相等的结果,同时大大减少了数据对象的副本数量和 shuffling 的开销,进而减少了计算量,使 EDDPC 算法能够在分布式情况下高效率执行.

参 考 文 献

[1] Xu Rui, Wunsch D Ⅱ. Survey of clustering algorithms [J]. IEEE Trans on Neural Networks, 2005, 16(3): 645-678

[2] Kaufman L, Peter R. Clustering by Means of Medoids [G] // Statistical Data Analysis Based on the L1 Norm and Related Methods. North-Holland: North-Holland Press, 1987: 405-416

[3] MacQueen J. Some methods for classification and analysis of multivariate observations [C] //Proc of the 5th Berkeley Symp on Mathematical Statistics and Probability. Berkeley, CA: University of California Press, 1967: 281-297

[4] Zhang W, Wang X, Zhao D, et al. Graph Degree Linkage: Agglomerative Clustering on a Directed Graph [M]. Berlin: Springer, 2012: 428-441

[5] Ester M, Kriegel H P, Sander J, et al. A density-based algorithm for discovering clusters in large spatial databases with noise [C] //Proc of ACM KDD'96. New York: ACM, 1996: 226-231

[6] Wang W, Jiong Y, Muntz R. STING: A statistical information grid approach to spatial data mining [C] //Proc of VLDB'97. San Francisco, CA: Morgan Kaufmann, 1997: 186-195

[7] Ying Wenhao, Xu Min, Wang Shitong, et al. Fast adaptive clustering by synchronization on large scale datasets [J]. Journal of Computer Research and Development, 2014, 51(4): 707-720 (in Chinese)
(应文豪, 许敏, 王士同, 等. 在大规模数据集上进行快速自适应同步聚类[J]. 计算机研究与发展, 2015, 51(4): 707-720)

[8] Alex R, Alessandro L. Clustering by fast search and find of density peaks [J]. Science, 2014, 344(1492): 1492-1496

[9] Jeffrey D, Sanjay G. MapReduce: Simplified data processing on large clusters [J]. Communications of the ACM, 2004, 51(1): 107-113

[10] Akdogan A, Demiryurek U, Banaei-Kashani F, et al. Voronoi-based geospatial query processing with MapReduce [C] //Proc of CloudCom '10. Piscataway, NJ: IEEE, 2010: 9-16

[11] Lu Wei, Shen Yanyan, Chen Su, etc. Efficient processing of k nearest neighbor joins using MapReduce [J]. VLDB Endowment, 2012, 5(10): 1016-1027

[12] Jeffery S V. Random sampling with a reservoir [J]. ACM Trans on Mathematical Software, 1985, 11(1): 37-57

[13] Shindler M, Wong A, Meyerson A W. Fast and accurate k-means for large datasets [C] //Proc of NIPS'11. New York: Curran Associates Press, 2011: 2375-2383

[14] Freitas F A, Peres S M, Lima C A M, et al. Grammatical facial expressions recognition with machine learning [C] // Proc of FLAIRS'14. Menlo Park, CA: AAAI Press, 2014: 180-185

[15] Caruana R, Joachims T. Kddcup 04 biology dataset [EB/OL]. 2008 [2015-05-10]. http://kodiak. cs. cornell. edu/kddcup/datasets. html

Gong Shufeng, born in 1991. Master candidate. His main research interests include big data and cloud computing.

Zhang Yanfeng, born in 1982. Associate professor. Member of China Computer Federation. His main research interests include big data and cloud computing.