基于Spark的并行DBSCAN算法的设计与实现

黄明吉 张 倩

(北京科技大学机械工程学院 北京 100083)

摘 要 随着云应用对运行时间和性能水平要求的逐步提高,以及内存价格的持续走低,基于内存的分布式计算框架Spark获得了前所未有的关注。主要研究DBSCAN算法在Spark上并行化的设计与实现,通过整体分析找到算法并行化可能的性能瓶颈,并从Spark的角度设计了并行DBSCAN算法的DAG图,优化了算法的并行化策略,最大化地降低了shuffle频率和数据量。最后将并行DBSCAN算法与单机DBSCAN算法进行性能对比,并通过实验分析不同参数对聚类结果的影响。结果表明,与单机DBSCAN算法相比,基于Spark的并行DBSCAN算法在聚类精度没有明显损失的情况下,数据量在3百万行时运行效率提高了37.2%,且加速比达到1.6。

关键词 Spark,并行DBSCAN算法,DAG,并行化策略

中图法分类号 TP301.6 **文献标识码** A

Design and Implementation of Parallel DBSCAN Algorithm Based on Spark

HUANG Ming-ji ZHANG Qian

(School of Mechanical Engineering, University of Science and Technology Beijing, Beijing 100083, China)

Abstract With the cloud application requiring less running time and higher performance as well as memory prices continuing to decline, the technology of memory-based distributed computing framework, such as Spark, has received unprecedented attention and is widely applied. We designed and implemented parallelized DBSCAN algorithm based on Spark, which can minimize the shuffle frequency and the data amount in shuffle. In order to optimize the algorithm parallelization strategy, we found the bottleneck of algorithm performance through the analysis and described the DAG of parallel DBSCAN algorithm based on Spark. Finally, we compared the performance of parallel DBSCAN algorithm with the DBSCAN algorithm, and analyzed the influence of different parameters on clustering results. Experimental results show that, compared with DBSCAN algorithm, the running time of parallel DBSCAN algorithm based on Spark decreases 37.2% and the speedup reaches 1.6 respectively on a data set containing three million lines without obvious loss of clustering accuracy.

Keywords Spark, Parallel DBSCAN algorithm, DAG, Parallelization strategy

1 引言

聚类分析作为数据挖掘的核心技术之一,是人工智能和机器学习领域的研究热点。从机器学习的角度来讲,聚类属于无监督学习,能够自动发现数据源间的相似性,将有高相似度的数据聚集到一个簇中,将不相似的数据尽可能分开[1]。聚类分析在模式识别、信息检索、计算机仿真、生物信息和市场营销等领域得到了广泛应用。目前,针对不同的应用提出了多种聚类算法,主要包括以K-Means[2]为代表的基于划分的聚类、以STING[3]为代表的基于网格的聚类、以BIRCH[4]为代表的基于层次的聚类,以及以DBSCAN[5]为代表的基于密度的聚类。

然而,面对大规模数据,传统的单机聚类算法无论是在效率上,还是计算复杂度上都已经无法满足海量信息的处理需要。Hadoop,Spark等并行计算技术作为一种计算模型得到了广泛关注,其中由加州伯克利AMP实验室开发的Spark[6]是一个基于内存计算的通用、有弹性的分布式大数据处理框架。针对MapReduce大量网络传输和磁盘I/O导致效率低的缺陷,Spark内存计算的特性使其在大数据环境下能更快速地处理数据和实时查询分析结果,同样地,算法在Spark中的运行速度比Hadoop快10~100倍[6]。Spark使用了弹性分布式数据集(Resilient Distributed Datasets,RDD),保证了容错健壮性和高可伸缩性。

分布式计算技术为大规模数据聚类分析提供了新的研究方向,应用这些技术来进行聚类分析也就成了必然的趋势。Cui等人提出了基于MapReduce的消除迭代依赖的并行K-Means算法[7];赖向阳等人提出了一种MapReduce架构下基于遗传算法的K-Medoids聚类[8];乔少杰等人提出了大规模复杂网络社区并行发现算法DBCS[9];Garg等人提出基于MPI集群环境对BIRCH算法的并行化[10]等。上述聚类算法为适应各自的并行计算环境都设计了很好的并行策略。

在基于密度的聚类算法DBSCAN中,每一个点都会与其

本文受北京市自然科学基金(2112011),中央高校基本科研业务费基金(2050205)资助。

黄明吉(1972-),男,博士,副教授,主要研究方向为数据挖掘、计算机辅助设计,E-mail:huangmingji@ustb.edu.cn;张 倩(1991-),女,硕士生,主要研究方向为数据挖掘、机器学习,E-mail:blockheadzq@163.com(通信作者)。

他数据点进行距离测量，即 DBSCAN 算法依赖于全局数据，不易于并行化实现，因此尽管 Spark 技术已经相当成熟，但是在 Spark 的机器学习库 MLlib[11] 中并没有加入对 DBSCAN 的支持。然而 DBSCAN 算法具有很多其他聚类算法不具有的优势，不需要事先指定聚类个数 K，可以发现任意形状和大小的簇类，对噪声点不敏感且可以在需要时设置过滤噪声点的参数等。因此，若能基于 Spark 完成 DBSCAN 算法的并行化，意义重大。

如上所述，Spark 的机器学习库 MLlib 中并没有加入并行 DBSCAN 算法，本文所做的工作将会是对 Spark 机器学习库的补充，相信也会更方便一般数据分析人员调用和处理大规模海量数据。本文基于 Spark 完成 DBSCAN 算法的并行化的思路主要是先将数据划分到不同的空间，然后在每个空间执行单机 DBSCAN 算法，最后对各个空间的聚类结果进行整合，产生全局聚类结果。为优化并行 DBSCAN 算法，对其进行了性能理论分析，并在 Spark 的角度设计了算法逻辑执行的有向无环图(Directed Acyclic Graph，DAG)，对算法并行化过程做了若干优化。最后，通过实验分析不同参数对聚类效果的影响，将并行 DBSCAN 与单机 DBSCAN 算法做了性能对比，并对并行化算法的性能进行了评定。

2　相关工作介绍

DBSCAN(Density-Based Spatial Clustering of Applications with Noise)是由 Martin Ester，Hans-Peter Kriegel 等人提出的一种基于密度的空间聚类算法。DBSCAN 算法的本质是寻找类簇并不断扩展类簇的过程，直至不再扩大。它将簇定义为密度相连的点的最大集合，能够把具有足够高密度的区域划分为簇，并在有噪声点的数据中发现任意形状的聚类。正是由于 DBSCAN 算法的这些优势，越来越多的研究者开始关注 DBSCAN 算法的并行化研究。Xu 等人采用 dR*-treet 提出了 PDBSCAN 算法[12]；Ali Patwary 提出了 PDSDBSCAN 算法，运用 disjoint-Set 数据结构，打破了数据存取的有序性，取得了更好的负载平衡[13]；Scicluna 提出了基于 FPGA 的并行 DBSCAN 算法[14]；Loh 提出了基于图形计算单元和数据划分的快速 DBSCAN 聚类算法[15]。其中，He 等人提出的 MR-DBSCAN 是基于 MapReduce 的 DBSCAN 算法的并行化[16]，本文就是在此篇文章的基础上基于 Spark 的 DBSCAN 并行化的设计与实现。

3　基于 Spark 的并行 DBSCAN 聚类算法

本文将基于 Spark 的并行 DBSCAN 算法称为 SparkDBSCAN。现分别描述和分析 SparkDBSCAN 算法的并行化策略、时间开销和算法过程优化。

3.1　SparkDBSCAN 算法的并行化策略

如图 1 所示，SparkDBSCAN 算法主要分为 3 步：划分数据空间、执行本地 DBSCAN 算法和产生全局聚类标签。

(1)数据空间划分，将原始数据集划分到多个空间，划分后每个空间中的数据点的数量大致相等；

(2)在每个划分空间内执行本地 DBSCAN 算法，产生局部聚类结果；

(3)产生全局聚类标签，根据各个划分空间中相交数据点的所有局部聚类标签，为每个数据点分配全局聚类标签。

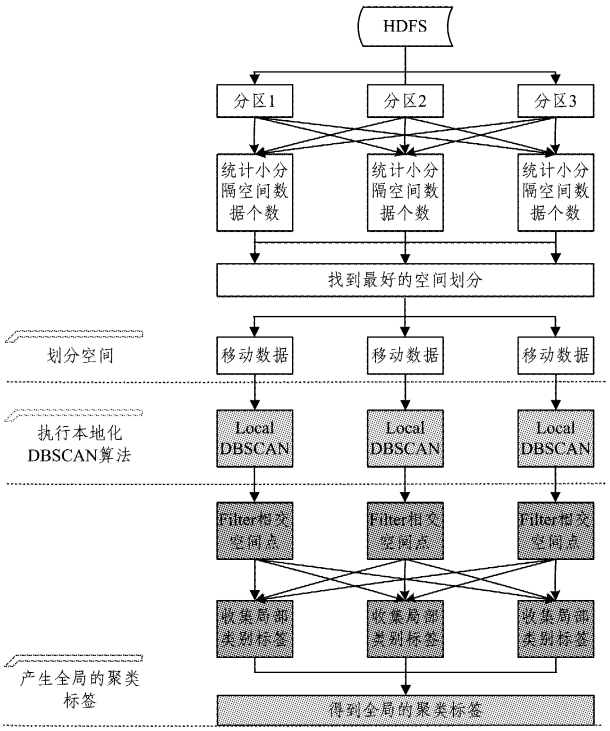

图 1　SparkDBSCAN 算法的逻辑执行示意图

3.1.1　数据空间划分

数据空间划分的主要任务是确定分割超平面，使划分后每个空间中数据量大致相等。数据空间划分的详细步骤如下：

输入：原始数据集 D，最小半径 Eps，每个划分空间最少包含点数 $Points$

输出：最终的划分空间集 P

(1)确定划分空间的数目。设总数据量是 M，则划分空间的数目 N 满足：

$$N= M/Points \tag{1}$$

(2)确定每一维的分割数 K_i，其中 $i\in\{1,2,\cdots,n\}$，n 为数据维数。K_i 满足的条件如下：

$$K_i\approx\sqrt[n]{N}\ \&\&\ N=\prod_{i=1}^{n}K_i \tag{2}$$

(3)确定每一维划分点。设原始数据集的划分空间集为 P_0，通过第 i 维切分得到的划分空间集为 P_i。

1)初始化划分空间集 P_i：

$$P_0=D \tag{3}$$

2)对于 $\forall\, p\in P_{i-1}$，其中，p 为集合 P_{i-1} 的任意一个划分空间。设 p 在第 i 维上的最小边界值为 $p_{\min}$，最大边界值为 $p_{\max}$，p 内数据点的数量为 M_p。划分成 s 个小区间，s 满足的条件为：

$$s=\left\lceil\frac{p_{\max}-p_{\min}}{2*eps}\right\rceil \tag{4}$$

3)j 为 p 上的第 j 个小区间。

对于 $j\in\{1,2,\cdots,s\}$ to s，遍历相加数据点的数目，若区间$(start,end)$的数据量满足条件：

$$\sum_{j=start}^{end}count\geqslant\frac{M_p}{K_i} \tag{5}$$

则把该区间作为新的划分空间加入到 P_i，并令：

$$start=end+1$$

4)重复步骤 2)和步骤 3)，直到最后一维划分完毕。将一

组二维数据集看作一个二维空间，如图 2 的前 3 步，描述了二维数据集的空间划分过程，并被划分为了 6 个空间。

(4)如图 2 第四步所示，空间划分完毕后，每个空间向边缘扩大 *eps* 的距离，产生最终的划分空间集。这样每个空间都会和相邻空间有部分重叠，这对于产生全局聚类标签至关重要。

如图 2 所示，Spark 还将广播划分空间集，以划分空间 id 为 key，将数据移动到各自归属的划分空间中去。

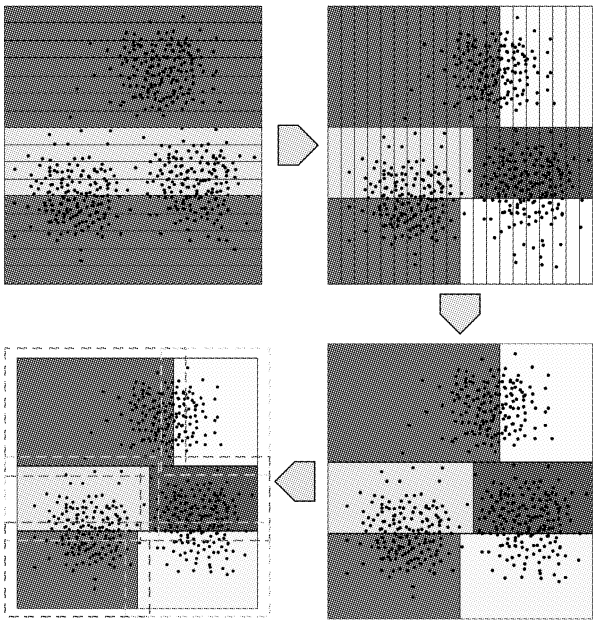

图 2 数据空间划分示意图

3.1.2 执行本地 DBSCAN 算法

数据移动完毕后，在每个划分空间独立执行单机 DBSCAN 算法，得到局部聚类结果，并将数据集的核心点、边界点、噪声点分别标记为 core，border，noise。

3.1.3 产生全局聚类标签

由于执行本地 DBSCAN 算法仅针对各自划分空间的数据点进行，考虑属于同一个类的数据点可能被分到相邻空间中，如图 3 中虚线框内的点，因此得到的是不完整的局部聚类结果。为得到全局聚类结果，要进行数据融合。如图 3 所示，融合不同分区的聚类结果只需关注重叠区域($2 * Eps$ 范围)的数据点即可。设在划分空间 A 中存在聚类 c_1，空间 B 中存在聚类 c_2，本文约定噪声点属于空类 $\emptyset$，则在重叠区域的数据点就存在如下 6 种情况。

(1)数据点 x 在 A 中是噪声点，且在 B 中是噪声点，如图 3 中的数据点 a：

$$x\in\emptyset \text{ in } A \;\&\&\; x\in\emptyset \text{ in } B \tag{6}$$

(2)数据点 x 在 A 中是噪声点，且在 B 中是属于类 c_2 的边界点，或相反，如图 3 中的数据点 b：

$$x:border\in c_1 \text{ in } A \;\&\&\; x\in\emptyset \text{ in } B \tag{7}$$

(3)数据点 x 在 A 中是属于类 c_1 的边界点，且在 B 中是属于类 c_2 的边界点，如图 3 中的数据点 c：

$$x:border\in c_1 \text{ in } A \;\&\&\; x:border\in c_2 \text{ in } B \tag{8}$$

(4)数据点 x 在 A 中是属于类 c_1 的核心点，在 B 中是属于类 c_2 的核心点，如图 3 中的数据点 d：

$$x:core\in c_1 \text{ in } A \;\&\&\; x:core\in c_2 \text{ in } B \tag{9}$$

(5)数据点 x 在 A 中是属于类 c_1 的边界点，在 B 中是属于类 c_2 的核心点，如图 3 中的数据点 e：

$$x:border\in c_1 \text{ in } A \;\&\&\; x:core\in c_2 \text{ in } B \tag{10}$$

(6)数据点 x 在 A 中是属于类 c_1 的核心点，在 B 中是噪声点，如图 3 中的数据点 f。

$$x:core\in c_1 \text{ in } A \;\&\&\; x\in\emptyset \text{ in } B \tag{11}$$

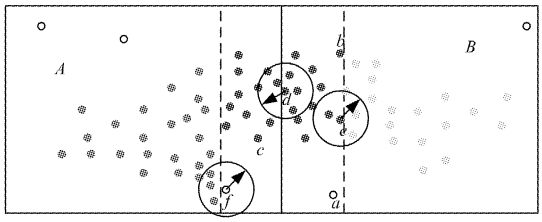

图 3 全局聚类标签产生示意图

边界点和噪声点在 DBSCAN 算法中不具备扩展簇类的能力[5]，因此只需分析(4)(5)(6)这 3 种情况。显然，数据点只要在一个划分空间是核心点，则在全局也是核心点，其连接的在不同划分空间的簇类 c_1 和 c_2 在全局中是一个类。设某个数据点属于 T 个划分空间，在第 t 个空间中属于 c_t 类，若是噪声点，$c_t=\emptyset$，该数据点在全局中所属的簇类 C 如式(12)所示：

$$C=\bigcup_{t=1}^{T} c_t \tag{12}$$

3.2 时间开销分析

本节将在整体上对 SparkDBSCAN 算法进行时间开销分析，以期发现算法的性能瓶颈。

设集群节点数为 S，第 i 台机器为 S_i，集群各机器之间的通信带宽为 B，总的 shuffle 次数为 F，每次 Shuffle Read 的数据集行数为 m_f，维度为 C_f，可知，原始数据集为 $m_1 * C_1$，第 i 台机器处理数据的总时间为 $T(S_i)$，其他时间花销为 $\varepsilon(other)$。

同时假设每台机器的数据都会 shuffle 到其他节点，并且数据量不变，则总的花销时间 $TotleCostTime$ 为：

$$\begin{aligned} TotleCostTime &= \frac{\sum_{f=1}^{F}(S-1)*m_f*C_f}{B}+\max\{T(S_i),1\leqslant i\leqslant S\}+\varepsilon(other)\\ &= ShuffleCost(S)+ComputeCost(S)+OtherTimeCost \end{aligned} \tag{13}$$

其中，$ShuffleCost(S)$ 为所有 shuffle 过程的总时间开销，$ComputeCost(S)$ 为计算的最大开销。因此 $TotleCostTime$ 为实际时间开销 $ActualCostTime$ 的上限，即：

$$ActualCostTime \leqslant TotleCostTime \tag{14}$$

由式(13)可知，SparkDBSCAN 算法的运行时间主要取决于 3 个方面：shuffle 次数、shuffle Read 的数据量和 Local DBSCAN 的执行效率。本文的主要工作之一就是减少 shuffle 的花费时间，尽量寻求最优的 shuffle 时间开销。

3.3 SparkDBSCAN 算法的过程优化

从 Spark 程序运行的角度设计 SparkDBSCAN 算法的逻辑 DAG 图，以期减少 shuffle 次数和每次 shuffle 读写的数据量。如图 4 所示，SparkDBSCAN 算法程序的主体由数据空间划分、本地执行 DBSCAN 算法和产生全局聚类标签组成，

分别以白色、黑色和灰色表示。

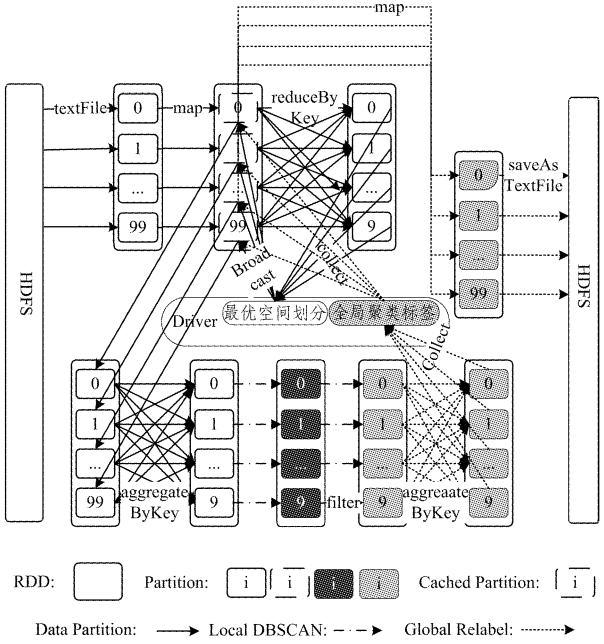

图 4　SparkDBSCAN 算法的逻辑 DAG 图

数据空间划分部分包含两个 shuffle 操作：1）统计每一维属于每个小区间的数据点数；2）将数据移动到各自归属的划分空间中。为了优化性能，对这一部分做了两个改进，第一个使用 RDD 的高性能算子：reduceByKey 和 aggregateByKey，第二个是虚线标注的 RDD 被用到了 3 次，因此将其 Cache 到内存，以最大化减少磁盘的 I/O 次数和 RDD 计算次数。

本地执行 DBSCAN 算法是在数据移动到各自划分空间中以后，在每个划分空间中执行单机 DBSCAN 算法，并过滤出被多个划分空间共同持有的数据。在这一部分中没有 shuffle 过程产生。

之后，判断每个重叠空间的数据是否包含核心点，将每个核心点所属的各个空间中的聚类标签收集起来，这些不同划分空间的聚类标签在全局下肯定是属于同一个类的。这样，最后得到的每个集合的所有局部标签在全局都是同一个簇类。这一部分会有一个 shuffle 过程产生。在此过程中，不涉及数据本身的计算，为减少 shuffle 数据量，提出用数据标签代替数据的方法，如果数据维数是 c，shuffle 过程的数据量将会降低 c 倍，有助于提高程序的运行速度。

本文根据 SparkDBSCAN 算法的逻辑 DAG 图再次优化 SparkDBSCAN 算法程序，最大程度地降低了 shuffle 次数和 shuffle 的数据量，以期最大化减少磁盘 I/O 次数，加快算法的执行速度。

4　实验

本文将 HADOOP HDFS 作为存储系统平台，测试 Spark 集群环境下的 DBSCAN 并行算法的性能，并与单机 DBSCAN 算法做了对比实验。

4.1　实验环境

实验集群包括 1 台 master 节点，3 台 slave 节点，每个计算节点的配置相同，集群环境如表 1 所列。采用东芬兰大学提供的聚类数据集进行实验，分析 SparkDBSCAN 算法聚类精度与参数对聚类的影响，聚类数据集如表 2 所列。

表 1　计算节点配置

配置	配置参数
服务器型号	Dell R720
CPU	Intel(R) Xeon(R) CPU E5-2620 v2 双处理器，共 6 核
内存	64GB
网卡	Broadcom Corporation NetXtreme BCM5720 Gigabit Ethernet PCIe * 4
硬盘读写速度	358.55MB/s
JDK 版本	Jdk1.7
Scala 版本	2.11.4
Hadoop 版本	2.6.47
Spark 版本	2.1

表 2　聚类数据集详表

名称	数据集大小	聚类个数	分布特点
D31[17]	3100 行 * 2 维	31	球状，密度均匀
Jain[18]	373 行 * 2 维	2	密度不均，任意形状
PathBased[19]	300 行 * 2 维	3	环状、球状，密度均匀
Spiral[19]	312 行 * 2 维	3	螺旋形，密度不均

除了上述针对聚类精度实验和参数影响实验的数据集外，还分别采用 50 万行、100 万行、200 万行、300 万行的数据集对其执行速度做了对比实验，并与单机 DBSCAN 算法作了对比。

4.2　聚类精度

本文分别使用 D31，Jain，PathBased，Spiral 4 种数据集进行 SparkDBSCAN 和单机 DBSCAN 的聚类精度对比实验，并得到如表 3 所列的聚类精度。由表 3 可知，SparkDBSCAN 的聚类精度略低于单机 DBSCAN 算法。

表 3　SparkDBSCAN 与单机 DBSCAN 聚类准确度表/%

	SparkDBSCAN	DBSCAN
D31	83.48	83.68
Jain	91.42	92.31
PathBased	97.34	97.72
Spiral	99.36	99.69

为更形象直观地分析 SparkDBSCAN 的聚类精度，对 D31，Jain，PathBased，Spiral 的原始数据集和聚类结果数据集进行图形化处理。其原始数据集效果图为图 5、图 7、图 9、图 11，对应的聚类结果分布图为图 6、图 8、图 10、图 12，五角星表示被判断为噪声点，方形表示误分类点，圆形表示正确分类点。通过比较 4 种数据集的原始数据集和聚类结果图可以看出，聚类错误的点通常处于两个簇边缘或者出现在相对稀疏的簇类，噪声点则通常分布于每个簇类的边缘，这也符合 DBSCAN 基于密度的特点，说明了 SparkDBSCAN 算法具有很高的可信度。

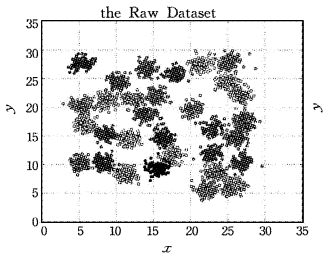

图 5　D31 数据集分布

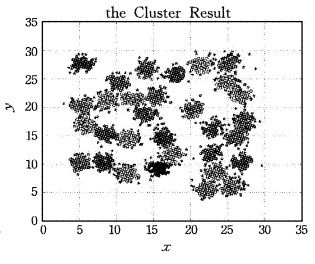

图 6　D31 聚类结果分布

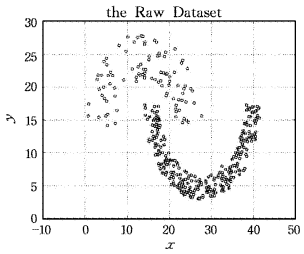

图 7 Jain 数据集分布

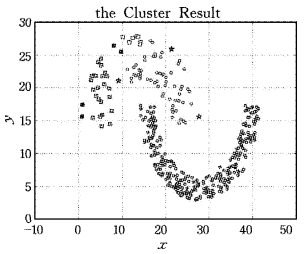

图 8 Jain 聚类结果分布

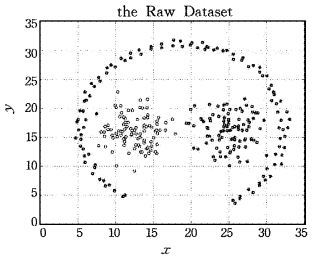

图 9 PathBased 数据集分布

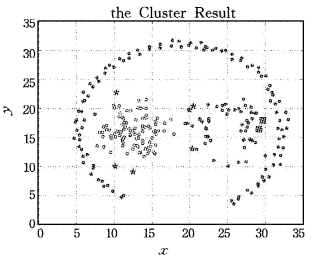

图 10 PathBased 聚类结果分布

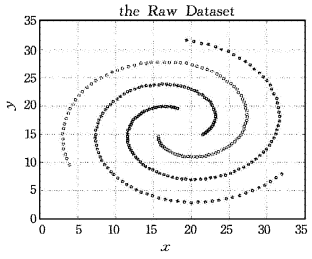

图 11 Spiral 数据集分布

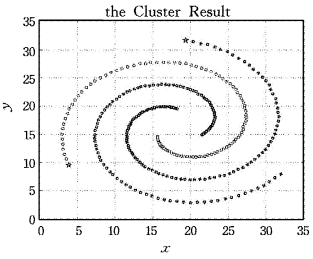

图 12 Spiral 聚类结果分布

对比 4 个数据集的原始分布图(见图 5、图 7、图 9、图 11)还可以看出,D31 的聚类簇较多且簇与簇之间的界限并不明显,交界处易出现误分类数据点,同时簇边缘由于并不紧凑,因此易被 SparkDBSCAN 认为是噪声点;Jain 数据集的簇类密度不均,稠密簇类聚类效果较佳,而稀疏簇类则被聚成了多个类别;PathBased,Spiral 数据集簇类尽管形状不一、数量不同,但是簇内密度高于簇间,SparkDBSCAN 有较好的聚类效果,同样噪声点易出现在簇类边缘的密度相对较稀处,误分类点出现在簇类相交处。由此可知,与单机 DBSCAN 一样,SparkDBSCAN 可以发现任意形状和数量的簇类,但是对于密度分布不均的情况,SparkDBSCAN 聚类效果并未因为并行化而得到改善。

4.3 参数对聚类的影响

最小邻域半径 *Eps* 和邻域内最少包含点数 *MinPts* 是 SparkDBSCAN 算法中最重要的两个参数,它们对聚类精度和聚类个数的影响至关重要,因此观察不同取值下的 *Eps* 和 *MinPts*,对 SparkDBSCAN 算法的聚类效果是非常必要的。图 13－图 16 分别是 *Eps* 和 *MinPts* 在 D31, *Jain*, *PathBased*, *Spiral* 4 个数据集下对聚类的影响。

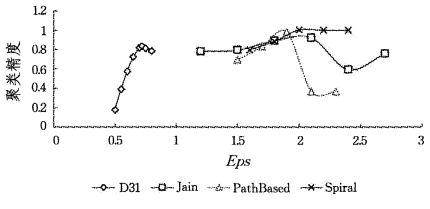

图 13 *Eps* 对聚类精度的影响

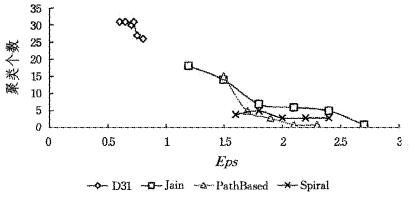

图 14 *Eps* 对聚类个数的影响

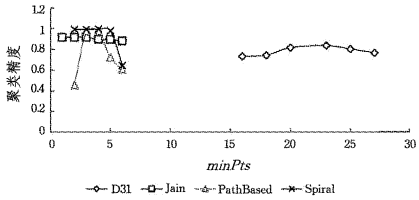

图 15 *MinPts* 对聚类精度的影响

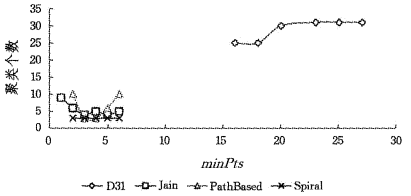

图 16 *MinPts* 对聚类个数的影响

由图 13、图 14 可知,随着 *Eps* 的增大,聚类精度总体呈现先增大后减小的趋势,聚类个数的变化则呈现一定程度的多样性。D31 和 Spiral 数据集保持一定聚类个数后减少,簇类个数较少的 Jain 数据集呈现递减的趋势,PathBased 的聚类个数则是一直减少直到稳定。对于某个数据集,在 *Eps* 较小时产生的聚类个数和本身簇类数即使一致(如 D31 和 Spiral),聚类精度也可能不会很高,这是因为小的 *Eps* 会产生较多的噪声点,从而降低准确度。过大的 *Eps* 会导致原本属于几个类簇的数据点被误聚集到一个类簇中,聚类个数减少,误分类点的数量增大,聚类精度也随之降低。

由图 15、图 16 可知,随着 *MinPts* 的增大,聚类精度呈现先增加后减少的规律。由图 15 可知,小的 *MinPts* 易产生过多的聚类个数,一个类簇被拆分成多个类簇会导致聚类精度的降低。而大的 *MinPts* 使得核心点成立的条件变得苛刻,易出现两种情况:1)一个簇类被误分成几个簇类,导致聚类个数增加;2)满足条件的数据点很少使得总的聚类个数减少。

由以上分析得到,一般来说,*Eps* 对聚类精度的影响更大,*MinPts* 则有控制聚类个数的作用,选取恰当的 *Eps* 和 *MinPts* 才会避免产生过多的噪声点和误分类点。

4.4 性能对比

如图 17 所示,本文对比了 SparkDBSCAN 和单机 DBSCAN 在不同数据量下的运行时间。

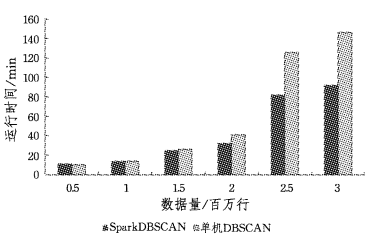

图 17 SparkDBSCAN 和单机 DBSCAN 的运行时间

可以看出,随着数据量的增大,单机 DBSCAN 算法运行

时间呈二次曲线型增长，这主要是因为 DBSCAN 算法的时间复杂度为 $O(n^2)$；而 SparkDBSCAN 算法由于并行性特点，每个计算节点只分担了部分数据，相对而言，对数据量的增大不太敏感，因此其增长幅度也并不明显。当数据量增加到 2.5 百万行和 3 百万行时，SparkDBSCAN 算法的优势更加明显，分别比单机 DBSCAN 算法快 34.6%和 37.2%。

4.5　绝对加速比

绝对加速比是同一个任务运行在单处理系统和并行处理系统中消耗时间的比率，代表了并行化的质量和效率。在不同核数和不同数据量下，SparkDBSCAN 的加速比实验结果如图 18 所示。

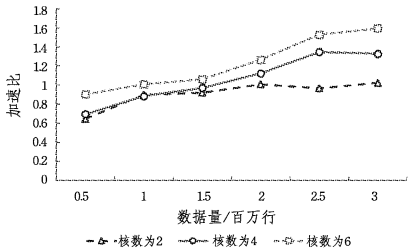

图 18　SparkDBSCAN 的绝对加速比

可以看出，随着数据量的增大，加速比整体呈增大的趋势；随着核数的增加，加速比也随之增大。当数据量增加到 2.5 百万行和 3 百万行时，加速比达到 1.5 和 1.6。另外，在数据量为 50 万和 100 万行时，加速比小于 1，这说明数据量较小时，SparkDBSCAN 较单机 DBSCAN 算法不占优势，其原因主要包括两方面：1）为并行化所做工作的时间开销，如数据空间划分和全局聚类标签产生；2）不可避免的 shuffle 过程带来的跨网络和磁盘 I/O 操作。

结束语　本文基于 Spark 计算框架并行实现了 DBSCAN 算法，补充了 Spark MLlib。结合 Spark 内存迭代的特点，整体分析了算法可能的性能瓶颈，通过设计并行 DBSCAN 算法的 DAG 图，找出可以 Cache 到内存的操作，最大程度地降低了执行过程中的磁盘 I/O 操作，并对 SparkDBSCAN 的逻辑过程进行优化，提出用数据标签代替数据的方法，最大化地降低了 shuffle 频率和 shuffle 数据量，缩短了算法的执行时间。另外，本文实验对比了 SparkDBSCAN 算法与单机 DBSCAN 算法的聚类质量、执行时间，以及 SparkDBSCAN 在不同 CPU 核数和数据量的加速比，并通过实验分析了不同参数对聚类结果的影响。实验结果表明，Spark 并行化后的 DBSCAN 算法，较单机 DBSCAN 算法没有明显损失聚类精度，且获得了较为理想的运行时间和加速比。

在聚类准确度实验中可以看出，SparkDBSCAN 对于密度差别较大的数据集的聚类效果欠佳，因此以后的工作主要是改进现有分区方式，使密度相近的数据划分到同一划分空间，在不同空间使用合适的 Eps 和 $MinPts$，以改善聚类质量。

参 考 文 献

[1] SUN J G, LIU J, ZHAO L Y. Clustering algorithms research [J]. Journal of software, 2008, 19(1): 48-61.

[2] JAIN A K, MURTY M N, FLYNN P J. Data clustering: a review [J]. ACM computing surveys (CSUR), 1999, 31(3): 264-323.

[3] WANG W, YANG J, MUNTZ R. STING: A statistical information grid approach to spatial data mining [C]// VLDB. 1997: 186-195.

[4] ZHANG T, RAMAKRISHNAN R, LIVNY M. BIRCH: an efficient data clustering method for very large databases [C]// ACM Sigmod Record. ACM, 1996: 103-114.

[5] ESTER M, KRIEGEL H P, SANDER J, et al. A density-based algorithm for discovering clusters in large spatial databases with noise [C]// Kdd. 1996: 226-231.

[6] ZAHARIA M, CHOWDHURY M, FRANKLIN M J, et al. Spark: cluster computing with working sets [C]// HotCloud, 2010. Boston MA, US, 2010: 10.

[7] CUI X, ZHU P, YANG X, et al. Optimized big data K-means clustering using MapReduce [J]. The Journal of Supercomputing, 2014, 70(3): 1249-1259.

[8] 赖向阳，宫秀军，韩来明. 一种 MapReduce 架构下基于遗传算法的 K-Medoids 聚类[J]. 计算机科学，2017，44(3)：23-26，58.

[9] 乔少杰，郭俊，韩楠，等. 大规模复杂网络社区并行发现算法[J]. 计算机学报，2017，40(3)：687-700.

[10] GARG A, MANGLA A, GUPTA N, et al. PBIRCH: a scalable parallel clustering algorithm for incremental data [C]// 10th International Database Engineering and Applications Symposium (IDEAS'06). Delhi, India, 2006: 315-316.

[11] MLlib | Spark [EB/OL]. http://spark.apache.org/mllib/.

[12] XU X, JAGER J, KRIEGEL H P. A fast parallel clustering algorithm for large spatial databases [M]// High Performance Data Mining. Springer US, 1999: 263-290.

[13] PATWARY M A, PALSETIA D, AGRAWAL A, et al. A new scalable parallel DBSCAN algorithm using the disjoint-set data structure [C]// Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society Press, 2012: 62.

[14] SCICLUNA N, BOUGANIS C S. FPGA-based parallel DBSCAN architecture [C]// International Symposium on Applied Reconfigurable Computing. Springer International Publishing, 2014: 1-12.

[15] LOH W K, YU H. Fast density-based clustering through dataset partition using graphics processing units [J]. Information Sciences, 2015, 308(C): 94-112.

[16] HE Y, TAN H, LUO W, et al. Mr-dbscan: an efficient parallel density-based clustering algorithm using mapreduce [C]// 2011 IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS). IEEE, 2011: 473-480.

[17] VEENMAN C J, REINDERS M J T, BACKER E. A maximum variance cluster algorithm [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002, 24(9): 1273-1280.

[18] JAIN A K, LAW M H C. Data clustering: A user's dilemma [C]// International Conference on Pattern Recognition and Machine Intelligence. 2005: 1-10.

[19] CHANG H, YEUNG D Y. ROBUST path-based spectral clustering [J]. Pattern Recognition, 2008, 41(1): 191-203.