·先进计算与数据处理· 文章编号: 1000-3428(2017)04-0021-07 文献标志码: A 中图分类号: TP311

基于KD树划分的云计算DBSCAN优化算法

陈广胜[1,2],程逸群[1,2],景维鹏[1,2]

(1.东北林业大学 信息与计算机工程学院,哈尔滨 150040;
2.黑龙江省林业生态大数据存储与高性能(云)计算工程研究中心,哈尔滨 150040)

摘 要: 在并行RDD-DBSCAN算法的数据划分和区域查询过程中会对数据集进行重复访问,降低了算法效率。为此,提出基于数据划分和融合策略的并行DBSCAN算法(DBSCAN-PSM)。利用KD树进行数据划分,实现数据分区与区域查询步骤的合并,从而减少数据集的访问次数以及降低I/O过程对算法效率的影响。采用判定数据点自身属性的方式,对标注为边缘点的数据进行融合,避免全局标记的额外时间开销。实验结果表明,DBSCAN-PSM算法相比RDD-DBSCAN算法可节省18%左右的运行时间,适用于处理海量数据聚类问题。

关键词: 聚类;DBSCAN算法;Spark平台;数据划分;数据融合

中文引用格式: 陈广胜,程逸群,景维鹏. 基于KD树划分的云计算DBSCAN优化算法[J]. 计算机工程,2017,43(4):21-27.

英文引用格式: Chen Guangsheng,Cheng Yiqun,Jing Weipeng. DBSCAN Optimization Algorithm Based on KD-tree Partitioning in Cloud Computing[J]. Computer Engineering,2017,43(4):21-27.

DBSCAN Optimization Algorithm Based on KD-tree Partitioning in Cloud Computing

CHEN Guangsheng[1,2],CHENG Yiqun[1,2],JING Weipeng[1,2]

(1.College of Information and Computer Engineering,Northeast Forestry University,Harbin 150040,China;
2.Heilongjiang Province Engineering Technology Research Center for Forestry Ecological Big Data Storage and High Performance(Cloud) Computing,Harbin 150040,China)

【Abstract】 The parallel RDD-DBSCAN algorithm has a repeated access to the data set in the data partition and region query steps,which reduces the efficiency of the algorithm. Aiming at the above problems,a parallel DBSCAN algorithm based on data partitioning and fusion stragy(DBSCAN-PSM) is proposed. It imports the KD-tree to partition the data,merges the partition and region query steps,reduces the number of access to the data set and decreases the influence of I/O on the algorithm. Data fusion method is realized by determining the clustering characteristics of the spatial boundary points,which avoids the time overhead of global markup. Experimental results show that DBSCAN-PSM algorithm runs faster than RDD-DBSCAN by 18%. It can deal with mass data clustering problem more effectively.

【Key words】 clustering;DBSCAN algorithm;Spark platform;data partitioning;data fusion

DOI: 10.3969/j.issn.1000-3428.2017.04.004

0 概述

大数据是近年来计算机领域兴起的热点研究方向,通过聚类可以解决诸如机器学习、数据挖掘、生物信息分析等诸多大数据领域的问题[1]。聚类是研究分类问题的重要方法,通过聚类分析可以将样本中具有相同或者相似特征的项归为一类,而将不具有该特征的项排除在外。主流的聚类方法包括基于划分的聚类方法,如K-means;层次聚类方法,如CURE和BIRCH等;基于统计模型的方法,如EM算法等;基于密度的方法,如DBSCAN,OPTICS等。在基于密度的方法中,DBSCAN是较为典型的一种,它以超球状区域内数据对象的数量来衡量此区域密度的高低,能够发现任意形状的聚类并有效识别噪声点。

传统的聚类方法基于串行方式,目前已经不能

基金项目:黑龙江省自然科学基金重点项目(ZD201403);林业公益性行业科研专项(201504307)。
作者简介:陈广胜(1969—),男,教授、博士,主研方向为大数据存储、云计算;程逸群,硕士研究生;景维鹏,副教授、博士。
收稿日期:2016-03-18 修回日期:2016-05-18 E-mail:nefujwp@163.com

满足对海量数据的分析需求。并行化的 DBSCAN 算法随之出现,文献[2]介绍了一种基于主从模型的 P-DBSCAN 算法,实现了聚类过程的并行化操作。但由于主从模式自身限制,这种并行化方法仍然较为繁琐。随着 Hadoop 开源大数据平台的兴起,广泛开展了基于 MapReduce[3] 的多种聚类算法并行化工作。文献[4]提出基于 MapReduce 模型的 K-means 聚类集成算法,改进共协关系矩阵得出聚类结果。文献[5]提出基于 MapReduce 模型的 DBSCAN 算法,将 DBSCAN 聚类过程通过 Map/Reduce 方式描述并且在 Hadoop 平台上得以实现。文献[6]介绍了 MR-DBSCAN 算法,基于 MapReduce 模型,通过引入数据划分策略进一步提高算法效率,但随着数据量的成倍增加,MapReduce 模型中间迭代过程产生的频繁 I/O 操作正成为制约算法效率的关键。在 DBSCAN 聚类过程中,由于数据划分和区域查询操作对数据集的访问,因此制约作用尤其明显。

文献[7]在 Apache Spark 中引入了弹性分布式数据集(Resilient Distributed Datasets,RDD)。RDD 是一种容错、并行的数据结构,支持基于内存的跨集群存储,从而满足低延迟特征。由于 Spark 基于内存这一优势,许多聚类算法的并行化工作正更多地基于 Spark 平台展开。文献[8]介绍一种基于 Spark 平台的空间密度聚类算法,利用 RDD 实现了大规模空间数据的快速聚类。虽然 DBSCAN 同为基于密度的聚类算法,但目前仍然鲜见基于 Spark 平台的 DBSCAN 算法改进工作。

文献[9]提出一种基于 RDD 的 DBSCAN 算法 RDD-DBSCAN。RDD-DBSCAN 将算法分为数据划分、本地聚类、全局标记 3 个步骤,并在 Spark 平台成功实现该算法。数据划分是一种常见的算法并行化手段,RDD-DBSCAN 在数据划分处理中运用二叉空间分割方式,将数据划分抽象为二叉树构建。这种划分处理较简单,未能考虑数据集自身的统计特性,因此在很多时候不能满足对数据集的有效分割需求。文献[10]介绍了一种密度聚类算法中的数据划分方式,利用树型数据结构完成划分。RDD-DBSCAN 对于聚类步骤本身改进较小。该算法的数据划分与区域查询步骤独立进行,增加了对数据集的读写次数,影响算法效率。此外,RDD-DBSCAN 在聚类后的数据融合过程中采用对所有数据点均进行操作的全局标记方式,影响算法执行效率。

综上,本文在文献[9]算法的基础上,提出一种改进的基于数据划分和融合策略的并行化 DBSCAN 算法(DBSCAN-PSM)。通过改进数据划分方式将数据划分过程抽象为 KD 树的构建过程,同时将区域查询提前至数据划分过程,合并数据划分与区域查询步骤。在数据融合过程中,通过判定边界点的特性,避免对所有数据点的操作。

1 DBSCAN 算法分析

DBSCAN 算法是一种基于密度的空间数据聚类算法,可以发现某一密度条件下任意形状的聚类。DBSCAN 中的基本概念定义如下:

Eps:邻域半径,用于判定核心点和噪声点。

MinPts:最小点集,即若某点为核心点,则其邻域内点的个数必须超过 *MinPts*。

核心点:邻域内点数目超过 *MinPts* 的点。

直接密度可达:存在点 p,q。若 q 在 p 的 *Eps* 邻域内,且 p 为核心点,则称 p,q 之间直接密度可达。

密度可达:某一区域内存在一连串的点 P_i($i=1,2,\cdots,n$),令 $P=P_1$,$P_n=Q$,P_i 到 P_{i-1} 均密度可达,则记 P 和 Q 密度可达。

DBSCAN 基于这样一个事实:一个聚类可以由其中的核心点唯一确定,将算法表述为:

1) 对于任一给定的核心点 p,均有区域 D 中所有从 p 密度可达到的对象 o 所组成的集合,该集合构成一个完整的聚类 C,且必有 $p\in C$。

2) 对于给定的聚类 C 和任意核心点 p,C 等价于集合。

为确定聚类,DBSCAN 从区域 D 中任取某个点 p,并从 p 开始查找 D 中满足 *Eps* 和 *MinPts* 且从 p 密度可达的所有点。若 p 是核心点,则有 p 的 *Eps* 邻域内包含的点数目大于 *MinPts*。若 p 是边界点,则 p 的 *Eps* 邻域内包含的对象数目小于 *MinPts*,即没有对象从 p 密度可达,p 暂时被标记为噪声点,算法继续处理 D 中的下一个任选点。在算法执行过程中,一个核心点密度可达的所有对象需要反复由区域查询(GetNeighbors)实现。

2 DBSCAN-PSM 算法

如前文所述,区域查询步骤是确定核心点密度可达对象的重要步骤,DBSCAN 算法的 I/O 开销很大一部分来源于该步骤。文献[9]中这种查询使用 R 树,但是基于全局的 *Eps* 和 *MinPts* 值的查询将产生大量的内存和 I/O 开销,严重影响算法效率。如图 1 所示,灰色三角点与黑色方块点为 2 种不同类簇。当点在区域 D 中的分布极不均匀时,这种情况尤为严重。

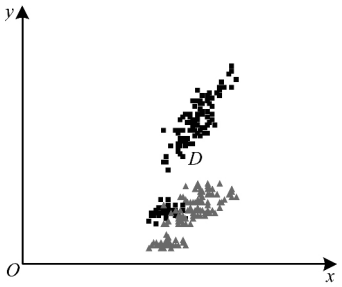

图 1　极端分布情况

由于数据划分步骤中同样需要对数据集进行 I/O操作，因此考虑到 DBSCAN 算法的特性，在并行化过程中，可以将数据划分与区域查询步骤合并，合理规避反复 I/O 操作带来的时间开销。从并行化角度来看，若仅简单地将 DBSCAN 算法用 Spark 平台实现而不进行数据划分操作，将全部数据集均放入一个 RDD 中，则难以将其全部调入内存，Spark 自身具有的低延迟特性将不能被有效发挥。所以，在合并数据划分与区域查询步骤时很有必要采用新的数据划分策略。

2.1　数据划分策略

文献 [9] 在数据划分步骤中运用 BSP 策略[11]。BSP 是一种基于二叉树的空间划分方法，它基于这样一个事实: 任何平面都可以将空间划分成 2 个半空间。重复该过程可构建一颗二叉树，每个叶子节点即被分割的空间。这种方法的空间划分策略较随意，基本上采用典型的二分法。在数据集划分过程中，有可能使空间划分与数据集分布不完全对应，从而降低并行化效率。此外，文献 [9] 中的算法依然需要进行独立的区域查询(GetNeighbors) 操作，尽管其使用了 R 树降低该过程的复杂度，且仅在被划分后的分区中独立进行，但仍然需要耗费大量时间。

针对这些问题，本文在区域查询中引入 KD 树[12]。KD 树是在 K 维空间中对数据点进行划分的一种数据结构。从本质上看，KD 树是一颗平衡二叉树，是二进制空间分割树的特殊情况。本文将数据划分过程抽象为 KD 树的构建过程，每次产生的2 个叶节点即为当前步骤数据划分产生的 2 个划分区域。

本文将区域查询步骤提前至数据划分过程中，与数据划分进行合并，在产生分区的过程中，利用 KD 树同时进行查询操作。虽然此处需要对数据集进行全局查询，耗费时间比独立在分区内的区域查询时间更多，但是由于减少了对数据集的访问次数，可大大节约 I/O 时间，利用 KD 树的查询可将时间复杂度从 $O(n^2)$ 降至 $O(n\mathrm{lb}n)$。该策略避免了文献 [9] 中对于数据集的重复访问和数据划分的时间开销。

将一组二维数据集看做一个二维空间。如图 2 所示，虚线 1 为第 1 次划分的空间分割线，将平面分为 A，B 2 个子区域。虚线 2 为子区域 B 中第2 次划分的空间分割线，将区域 B 分为 B_1，B_2 2 个子区域。

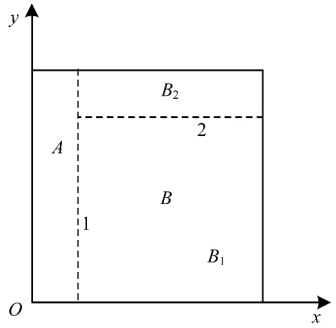

图 2　基于 KD 树确定分割线的示意图

显然，KD 树构建中的主要任务是确定图中的分割超平面。一般确定策略有依据某一维度上的中位数、平均值和方差等。针对数据集的特性，考虑到每一个划分中的数据集点应基本相等，本文采用依据方差的划分方式。

分割超平面的确定步骤如下:

1) 设空间 R^n，$S_i^2(i=1,2,\cdots,n)$ 为某维度上的方差。分割域 *Split* 满足:

$$Split=\{\max S_i^2 \mid x=1,2,\cdots,n\}$$

2) 根据确定的分割域 *Split*，统计 *Split* 域所有数据点值，则分割点 *Node* 满足:

$$Node=\begin{cases}\dfrac{X_{PointIn(Split)/2}+X_{PointIn(Split)/2+1}}{2}, & Split\ \text{域当前维度点为偶数}\\ X_{PointIn(Split)}, & Split\ \text{域当前维度点为奇数}\end{cases}$$

3) 由 *Node* 确定超平面分割数据集。将小于 *Node* 的所有数据点视作 KD 树左子树，大于 *Node* 的所有数据点视作 KD 树右子树。

4) 在 KD 树左右子树中继续执行步骤 1) ~步骤 3)，直至满足划分停止条件。停止划分时的叶节点即为数据集的最终划分。

划分停止条件应当与 DBSCAN 特性和数据集特性有关。由于划分后的分区应当保证每个分区都能独立进行 DBSCAN 操作，即该分区内的所有点均被标记为核心点或噪声点，因此分区停止条件为:

1) 当分区小于 $2Eps$ 时，如图 3 所示，设分区为 p。当分区长度小于 Eps 时，分区内任意一点都必须在分区合并后才能得到完整的聚类结果。这会使分区的合并操作十分复杂，并且严重影响并行化效率和聚类质量。因此，当分区长度小于等于 $2Eps$ 时，停止分区。

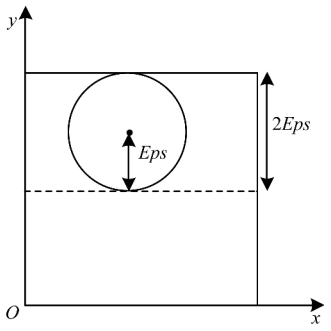

图 3 分区示意图

2) 考虑到 RDD 的特性,此处引入 *MemPts* 参数。依据 RDD 的特性,若分区包含的数据点超过内存容量,超出的部分将被写入磁盘中,不利于发挥 Spark 平台低延迟性的特点。因此,每个分区不应超过处理节点的内存容量,即可以将所有数据都放入内存中。*MemPts* 应当被设定为适应 Spark 集群中内存容量最小的节点所能容纳的值。

3) 考虑到 KD 树在极端情况下会造成分割过细,进而导致层数过多,因此,此处对数据划分层数加以限制。基于优化的 KD 树搜索策略,本文将层数限制在 10 以内。

这样的分区策略在一次对数据集的操作过程中完成了区域查询和数据划分操作,避免了对数据集的重复作用。根据 KD 树特性划分的分区可以最大限度地保证每个分区内的数据点数目相近,同时依据数据集特征的划分可以尽量避免分区操作带来的聚类质量下降,最终提高并行化水平。

2.2 数据融合策略

在分区后,算法在每个分区独立进行 DBSCAN 聚类操作,由于独立的分区操作仅针对分区内的数据点进行,因此得到的聚类是不准确和不完整的。为得到全局的聚类值,有必要在独立的 DBSCAN 操作后对分区进行融合操作。文献 [9] 中通过同时对分区和数据点的标记实现融合,需要对全部数据进行操作,效率较低。考虑到分区融合的操作大多涉及分区边缘点,因此,本文在文献 [13] 提出方法的基础上,进一步简化归并融合方法,采用不理会分区标记,直接对分区边缘所在点进行判定的方法。

当点位于分区边界 *Eps* 范围外时,若满足 *Eps* 范围内点数目不低于 *MinPts*,则形成聚类且与相邻分区不发生关系,因此,分区边界 *Eps* 范围外的点被认为不是边缘点。

当点位于分区边界 *Eps* 范围内时,聚类可能存在以下情况: 1) 该点恰好被标记为本分区内的某个聚类的边界点,这种情况不需要进一步操作。2) 该点被标记为核心点,这种情况可能出现将本属于一个聚类的点划分到 2 个相邻分区当中。因此,需要对 2 个聚类进行合并操作。3) 该点被标记为噪声点,但在分区合并后,该点 *Eps* 半径内的数据点个数满足 *MinPts*,即可形成新的聚类。因此,在合并后需要继续判定是否会产生新的聚类。本文将分区边界 *Eps* 范围内的点标记为边缘点,并对边缘情况 2) 和情况 3) 分别讨论。

1) 2 个聚类的合并

考虑到分区过程中可能将本属于一个聚类的点划分到 2 个相邻分区中,如图 4 所示,虚线内的点为边界点集。在融合过程中,应当对此类点加以归并。

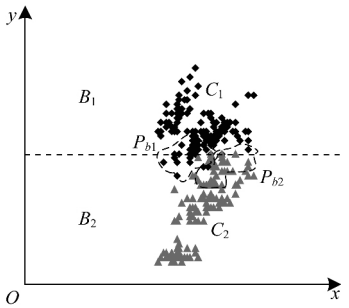

图 4 同一聚类被划分到不同分区的情况

当且仅当:

(1) a 和 b 位于相邻的分区 B_1,B_2。C_1,C_2 为 2 个聚类 $a \in C_1$,$b \in C_2$。

(2) $DISTANCE(a,b) \leqslant Eps$。*Eps* 表示该次聚类的邻域值。$B_1$,$B_2$ 分区边界点集为 P_{b1},P_{b2},有:

$$\overline{d} = \frac{\sum_i \sum_j DISTANCE(p_i, q_j)}{N_{b1} \times N_{b2}} \leqslant Eps$$

其中,C_1 和 C_2 可被视为同一聚类。

2) 噪声点产生的新类

在划分过程中,分区 B 附近可能存在一些点,在该分区的聚类过程中,该点被标记为噪声点,但在合并后该点 *Eps* 半径内的数据点个数满足 *MinPts*,因此,这些点会形成新的聚类。在分区融合过程中,对此类点也应予以考虑。

设 B_1,B_2 为 2 个相邻分区,B_1,B_2 分区边界点集为 P_{b1},P_{b2}。令 P_{b1},P_{b2} 中噪声点集为 PN,当且仅当:

(1) $\exists p_0 \in PN$, $\exists p_i \in PN$ $(i = 1, 2, \cdots, n, n \geqslant MinPts)$。

(2) $DISTANCE(p_0, p_i) \leqslant Eps$。

其中,p_0 可认为是新的核心点; p_0,p_i 构成新的聚类 C。

2.3 算法实现

在引入数据划分策略后,DBSCAN 的并行化效率得到进一步提高。基于 Spark 平台的 DBSCAN 建

立在数据集划分的基础上。对数据集进行划分后，DBSCAN 在每一个被划分的数据集上独立运行。这样可以不对 DBSCAN 算法进行较大改动，同时最大限度提高并行程度。

3　实验验证

3.1　测试平台与数据集

本文使用 5 台浪潮英信 I8000 刀片服务器（Xeon E5-2620 v2 6 核 2.10 GHz 处理器，8 GB 内存，200 GB 硬盘），采用 Tenda TEG1024G 千兆以太网交换机连接，运行 Spark 1.5.0 版本。

本文使用 Scilearn [14] 数据集工具包中的 Samples Generator [15] 工具产生测试数据集。为有效验证不同数据规模下的聚类效果，本文选取包含 10 万行～500 万行数据的数据集。数据集和聚类情况如表 1 所示。

表 1　数据集和聚类情况

编号	行数/10^5	聚类数	编号	行数/10^5	聚类数
1	1	4	6	25	10
2	5	3	7	30	8
3	10	5	8	35	7
4	15	2	9	45	3
5	20	5	10	50	4

3.2　DBSCAN-PSM 聚类结果分析

本文分别运用串行 DBSCAN，RDD-DBSCAN 和 DBSCAN-PSM 测试上述 10 万行、150 万行、250 万行和 350 万行的数据集。对于聚类 i，计算 P_{ij}。P_{ij} 为聚类 i 中成员属于类 j 的概率，有 $P_{ij}=\frac{m_{ij}}{m_i}$，其中，$m_i$ 为聚类 i 中所有成员个数；m_{ij} 为 i 中成员属于 j 的个数。由此引入 *Purity* 和 *Entropy* 评价指标。指标 *Entropy* 衡量聚类熵值，定义如下：$Entropy=-\sum_{i=1}^{K}\frac{m_i}{m}\sum_{j=1}^{L}P_{ij}\mathrm{lb}P_{ij}$。指标 *Purity* 衡量聚类纯度，根据上文中的 P_{ij} 定义，定义 $P_i=\max(P_{ij})$。整个聚类的纯度定义如下：$Purity=\sum_{i=1}^{K}\frac{m_i}{m}P_i$。

聚类结果的 *Purity* 和 *Entropy* 指标如图 5、图 6 所示。可见在聚类数目较少时，RDD-DBSCAN 和 DBSCAN-PSM 均有较好的聚类质量，与串行 DBSCAN 差别较小。但是在 250 万和 350 万行数据集中，由于聚类个数较多，RDD-DBSCAN 和 DBSCAN-PSM 聚类质量相比串行算法均有一定程度的下降。这主要是由于聚类数据较多时，数据划分对于聚类结果的影响更大，但 DBSCAN-PSM 与 RDD-DBSCAN 相比，聚类质量并无明显变化，说明基于 KD 树的数据划分方式没有进一步影响聚类质量。

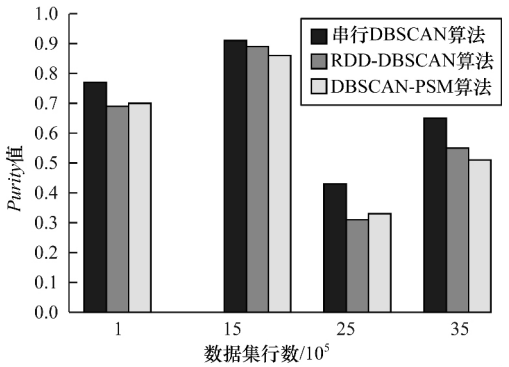

图 5　聚类结果的 *Purity* 指标

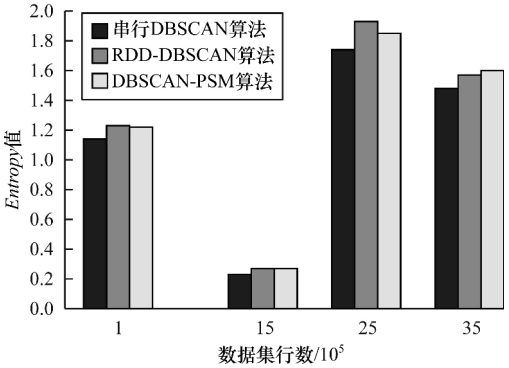

图 6　聚类结果的 *Entropy* 指标

3.3　Spark 平台下改进 DBSCAN 算法效率的验证

本文实验并行算法采用 5 个节点，双路 Xeon E5-2620v2 处理器，共 60 核。对照的串行 DBSCAN 采用单节点 12 核处理器。分别运用串行 DBSCAN，MR-DBSCAN，RDD-DBSCAN 和 DBSCAN-PSM 进行聚类操作，测试上述数据集。算法执行时间如图 7 所示。

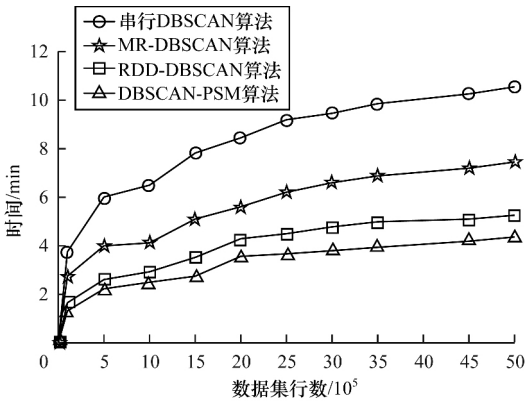

图 7　单机与并行 DBSCAN 算法聚类效率比较

在相同的数据规模上，与串行 DBSCAN 相比，MR-DBSCAN，RDD-DBSCAN 和 DBSCAN-PSM 效率均有提高。通过算法并行化能减少运算时间，同时由于 RDD-DBSCAN 和 DBSCAN-PSM 均采用基于内存的 Spark 框架，因此算法效率较 MR-DBSCAN 显著提高。

在数据集规模较小时，DBSCAN-PSM 效率提高并不明显，这主要是由于在数据集规模较小时，BSP 和 KD 树 2 种划分策略的差异并不明显。且在数据集规模有限的前提下，数据划分和区域查询操作重复访问 I/O 的影响也较有限，因此，算法效率提升有限。但当数据集规模进一步增大时，由于减少了对数据集的访问，DBSCAM-PSM 相较于 RDD-DBSCAN 效率更高。在数据集规模比较大时，I/O 操作的影响较为明显，DBSCAN-PSM 效率提升较大，可以更有效地应对海量数据处理的场景。

3.4 不同核环境对改进 DBSCAN 算法效率的影响

实验选取 100 万行数据集，在单节点，2 核 ~ 12 核运行环境下测试，结果如图 8 所示。无论是 RDD-DBSCAN 还是 DBSCAN-PSM，核心数对算法效率影响较为突出。这是由于 Spark 自身的调度策略所决定。当节点数一定且可用核心数较多时，Spark 可以将聚类任务分配至更多 CPU 核心运行，从而提高算法并行化程度。但是在单节点环境下，当核心数大于 8 后，在数据集行数有限时，Spark 调度策略不会启用所有的 CPU 核进行算法运算，核心数的进一步提升对算法效率的影响有限。因此，在此基础上若想进一步提高算法效率，需要对 Spark 调度算法进行改进，但是与 RDD-DBSCAN 相比，改进数据划分方法后的 DBSCAN-PSM 在相同核心数目下的算法执行时间均有减少，即减少 I/O 操作对于算法改进是必要的。

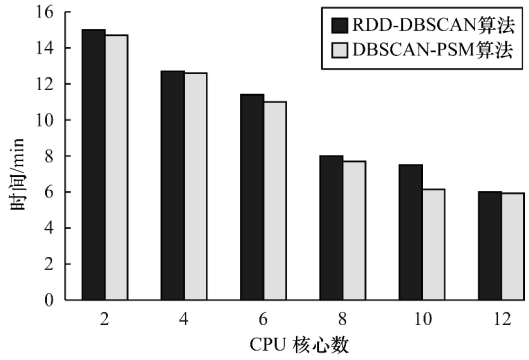

图 8 可用 CPU 核心数对算法效率的影响

3.5 不同 *Eps* 参数选择对改进 DBSCAN 算法效率的影响

Eps 参数是 DBSCAN 算法的重要参数之一。不同 *Eps* 参数对于算法效率也有影响。本文实验采用 5 个节点、60 核。如图 9 所示，当输入 *Eps* 参数增大时，算法效率受到显著影响。这是由于在数据划分策略中，要求每个分区最小长度大于 2*Eps*，因此在 *Eps* 增大时，分区的最小长度相应增加，所得到的数据划分数目相对减少。在这种情况下，算法的并行化程度降低，算法用时大幅增加。但与未改进前的 RDD-DBSCAN 相比，由于数据划分方式的变化，使得数据分区更为合理，不至于造成数据的过度划分或过少划分，因此 DBSCAN-PSM 由 *Eps* 造成的影响显著降低。在 *Eps* 较小时，由于 2 种划分方式产生的分区结果相近，因此这种影响尚不明显。但在 *Eps* >0.6 时，基于 KD 树的数据划分方法开始表现出明显优势，算法执行时间比 RDD-DBSCAN 大约减少 16%。

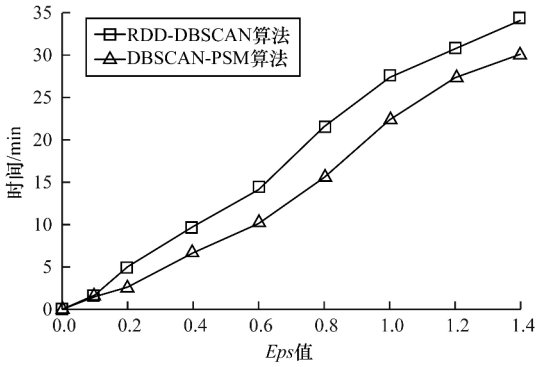

图 9 *Eps* 值对算法效率的影响

3.6 核间通信对改进 DBSCAN 算法效率的影响

由于并行化算法在集群中运行，因此 CPU 核间通信开销也是算法效率的重要因素之一。本文采用单节点，2 核 ~ 12 核进行实验。通信开销占算法执行总时间的比例如图 10 所示。

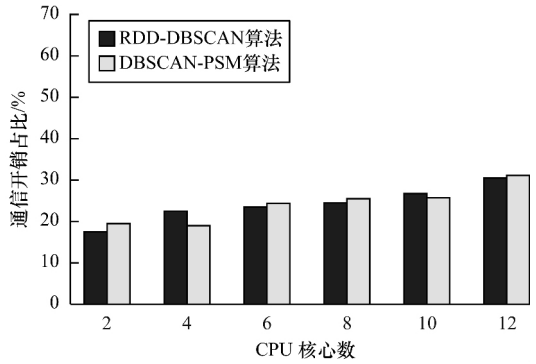

图 10 通信开销所占算法执行总时间的比例

文献 [9] 已经证明，在节点增加时，通信开销对于 RDD-DBSCAN 算法影响较小。改进后的 DBSCAN-PSM 算法与 RDD-DBSCAN 相比，虽然对数据划分方式做了一定程度上的更改，但 DBSCAN-PSM 的通信开销没有显著变化。随着 CPU 核心数的增加，通信开销总体呈上升趋势，但仍保持在 20% 以

下,说明合并的数据划分和区域查询步骤对于通信开销并无明显影响,算法可以较好地适应不同节点的并行化环境。

4　结束语

本文基于数据划分的思想,提出 DBSCAN-PSM 算法,利用 KD 树进行数据划分,同时简化数据融合进一步提高算法效率,通过处理海量数据的聚类,使其更加有效地应用于云计算环境中。下一步工作重点为: 1) 改进现有分区方式,在不同分区中采用不同的 *Eps* 值以进一步提高聚类质量。2) 利用数据集的统计特征,自动选取 *Eps* 和 *MinPts* 值,提高算法自动化水平。3) 针对不同数据集类型应用不同的分区方式,进一步提高并行化水平。

参考文献

[1] 孙吉贵,刘　杰,赵连宇. 聚类算法研究 [J]. 软件学报,2008,19(1):48-61.

[2] Chen Min, Gao Xuedong, Li Huifei. Parallel DBSCAN with Priority R-tree [C]//Proceedings of IEEE International Conference on Information Management and Engineering. Washington D. C., USA: IEEE Press, 2010: 508-511.

[3] Wikipedia. Mapreduce [EB/OL]. (2015-04-04). http://en.wikipedia.org/wiki/MapReduce.

[4] 冀素琴,石洪波. 基于 MapReduce 的 K-means 聚类集成 [J]. 计算机工程,2013,39(9):84-87.

[5] Dai Biru, Lin I C. Efficient Map/Reduce-based DBSCAN Algorithm with Optimized Data Partition [C]//Proceedings of IEEE International Conference on Cloud Computing. Washington D. C., USA: IEEE Press, 2012: 59-66.

[6] He Yaobin, Tan Haoyu, Luo Wuman, et al. MR-DBSCAN: An Efficient Parallel Density-based Clustering Algorithm Using MapReduce [C]//Proceedings of IEEE International Conference on Parallel & Distributed Systems. Washington D. C., USA: IEEE Press, 2011: 473-480.

[7] Zaharia M, Chowdhury M, Das T, et al. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing [C]//Proceedings of Usenix Conference on Networked Systems Design & Implementation. San Jose, USA: USENIX Association, 2012: 2.

[8] 李璐明,蒋新华,廖律超. 基于弹性分布式数据集的海量空间数据密度聚类 [J]. 湖南大学学报(自然科学版),2015,42(8):116-124.

[9] Cordova I, Moh T S. DBSCAN on Resilient Distributed Datasets [C]//Proceedings of International Conference on High Performance Computing & Simulation. Washington D. C., USA: IEEE Press, 2015: 531-540.

[10] 于亚非,周爱武. 一种改进的 DBSCAN 密度算法 [J]. 计算机技术与发展,2011,21(2):30-33.

[11] Berger M J, Bokhari S H. A Partitioning Strategy for Nonuniform Problems on Multiprocessors [J]. Computers, 1987, 100(5): 570-580.

[12] Wikipedia. KD-tree [EB/OL]. (2015-04-10). https://en.wikipedia.org/wiki/K-d_tree.

[13] 周水庚,周傲英,曹　晶. 基于数据分区的 DBSCAN 算法 [J]. 计算机研究与发展,2000,37(10):1153-1159.

[14] Scikit-learn. Dataset Loading Utilities [EB/OL]. (2015-11-09). http://scikit-learn.org/stable/datasets/.

[15] Pedregosa F, Varoquaux G, Gramfort A, et al. Scikit-learn: Machine Learning in Python [J]. Journal of Machine Learning Research, 2011, 12(1): 2825-2830.

编辑　陆燕菲

(上接第 20 页)

[13] 蔡颖琨,谢昆青,马修军. 屏蔽了输入参数敏感性的 DBSCAN 改进算法 [J]. 北京大学学报(自然科学版),2004,40(3):480-486.

[14] Xu X, Ester M, Kriegel H P, et al. A Distribution-based Clustering Algorithm for Mining in Large Spatial Databases [C]//Proceedings of the 14th International Conference on Data Engineering. Washington D. C., USA: IEEE Press, 1998: 324-331.

[15] 夏鲁宁,荆继武. SA-DBSCAN: 一种自适应基于密度聚类算法 [J]. 中国科学院大学学报,2009,26(4):530-538.

[16] 罗　可,林睦纲,郝东妹. 数据挖掘中分类算法综述 [J]. 计算机工程,2005,31(1):3-5.

[17] Quinlan T R. C4.5: Programs for Machine Learning [M]. San Mateo, USA: Morgan Kaufamann, 1993.

[18] Han J, Kamber M. Data Mining: Concepts and Techniques [M]. San Mateo, USA: Morgan Kaufmann, 2000.

[19] Breunig M, Kriegel H P, Ng R, et al. LOF: Identifying Density-based Local Outliers [C]//Proceedings of ACM SIGMOD International Conference on Management of Data. New York, USA: ACM Press, 2000: 93-104.

[20] Ester M, Kriegel H P, Sander J, et al. Incremental Clustering for Mining in a Data Warehousing Environment [C]//Proceedings of the 24th International Conference on Very Large Data Bases. San Mateo, USA: Morgan Kaufmann, 1998: 323-333.

编辑　陆燕菲