(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 106203494 A
(43)申请公布日 2016.12.07

(21)申请号 201610519403.6

(22)申请日 2016.06.30

(71)申请人 电子科技大学
地址 611731 四川省成都市高新区(西区)西源大道2006号

(72)发明人 田玲 罗光春 陈爱国 殷光强

(74)专利代理机构 成都弘毅天承知识产权代理有限公司 51230
代理人 徐金琼

(51)Int.Cl.
G06K 9/62(2006.01)

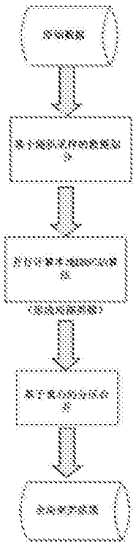
权利要求书1页 说明书6页 附图4页

(54)发明名称

一种基于内存计算的并行化聚类方法

(57)摘要

本发明提供了一种基于内存计算的并行化聚类方法,其主旨在于解决聚类算法DBSCAN在处理海量数据下的效率问题,其方案为:S1:基于简单随机抽样的数据划分,以<ID,Raw_data>作为此阶段的输入,通过对原始数据进行简单随机抽样来完成数据的切分,并将切分的结果保存到不同的RDD;S2:利用内存计算模型在各个计算节点并行执行DBSCAN算法,对不同的RDD中的原始数据进行聚类,产生局部类簇;S3:基于重心合并所有的局部类簇,利用内存计算模型对局部类簇进行合并,从而产生全局聚类结果。本发明基于内存计算模型,通过简单的数据划分方式对原始数据进行切割,极大地提高了算法的处理效率。同时,基于重心距离的局部类簇合并能快速地构建全局类簇,满足了处理大规模数据的用户需求。



CN 106203494 A

1.一种基于内存计算的并行化聚类方法,包括如下步骤:

S1:基于简单随机抽样的数据划分,以<ID,Raw_data>作为此阶段的输入,通过对原始数据进行简单随机抽样来完成数据的切分,并将切分的结果保存到不同的RDD;

S2:利用内存计算模型在各个计算节点并行执行DBSCAN算法,对不同的RDD中的原始数据进行聚类,产生局部类簇;

S3:基于重心合并所有的局部类簇,利用内存计算模型对局部类簇进行合并,从而产生全局聚类结果。

2.根据权利要求书1所述的一种基于内存计算的并行化聚类方法,包括如下步骤:

S11:在集群中启动实现map接口的作业,各个节点以原始数据作为输入,通过自定义的random()函数所产生的随机数作为当前数据的key值,生成带有新的划分标识的原始数据RDD集合;

S12:将S11中产生的RDD集合按照划分标识分解成对应的局部RDD集合。

3.根据权利要求书1所述的一种基于内存计算的并行化聚类方法,包括如下步骤:

S21:对每个局部RDD集合,首先根据数据划分个数确定聚类的邻域半径ε以及阈值MinPts;

S22:选取任一数据对象进行邻域查询,若该对象为核心对象,则将其邻域内所有的数据加入到list中,进行递归调用;若该对象为噪声对象,则将其标记为Noise;

S23:重复S22,直到所有的对象均被标记,同时尽可能的归到某个类簇中,或者找出那些不属于任何类簇的噪声对象;

S24:生成局部类簇RDD。

4.根据权利要求书1所述的一种基于内存计算的并行化聚类方法,包括如下步骤:

S31:计算各个数据划分中局部类簇之间的距离,求得最小值局部d_{min};

S32:根据局部d_{min}求得在整个数据集下的全局D_{min},D_{min}表示两个局部类簇之间的最小距离;

S33:根据D_{min}确定合并阈值σ;

S34:构建重心距离矩阵;

S35:根据合并阈值σ,产生合并序列RDD;

S36:根据合并序列RDD对局部类簇进行合并,从而产生最终的全局聚类结果。

一种基于内存计算的并行化聚类方法

技术领域

[0001] 本发明涉及数据挖掘算法并行化领域，特别涉及一种基于内存计算的并行化聚类方法。

背景技术

[0002] 如今，随着信息技术的不断创新，数据正以爆炸式的速度增长。如何对大规模数据进行有效地处理已然成为一项严峻的挑战。

[0003] 为了能够从海量数据中挖掘出规律信息，找出数据间的区别与联系，数据挖掘作为一门新型学科出现在人们的视线中，并在各行业发挥着重要作用。

[0004] 聚类分析在数据挖掘中占据着举足轻重的地位，得到了人们的广泛关注。聚类通常是按照一定的相似度度量方法，使得相似度较高的一组数据聚在一起。

[0005] DBSCAN算法是1996年由Ester Martin等人提出的基于高密度连接区域的密度聚类方法，它能发现任意形状的类簇，并能够有效地处理噪声点。该算法简单、效率高，目前已被广泛地应用于工业生产与科学研究中。在DBSCAN算法中，类簇中的每个数据对象，对于给定的阈值(MinPts)，该对象的Eps邻域包含的数据对象的个数必须大于等于阈值。因此当相邻区域的对象数不小于阈值时，将继续聚类。

[0006] DBSCAN算法在进行海量数据聚类时，需要数据集载入内存，同时要计算两两对象之间的距离，当数据量过大时会造成内存溢出。如果不将数据集载入内存，频繁的I/O操作会造成算法效率低下。因此，传统的DBSCAN算法无法适用于大规模数据集合的聚类分析。

[0007] 现有的并行DBSCAN算法在进行数据分区时，通常是将原始数据库划分为若干个互不相交的分区，并通过一定的策略保证分区之间的负载均衡，随着数据维数的增加，对高维空间的切分将会消耗大量的时间。同时，在分区边界合并时，针对每个分区，都需要找出位于2m个方向上的边界数据进行边界判定，其中m为数据的维度，这无疑将消耗大量的时间，使得算法的效率不高。

[0008] 弹性分布式数据集RDD是分布式内存计算的抽象应用。RDD是只读、可序列化的并且可以通过persist或cache函数缓存在内存中，减少了大量的磁盘IO，极大地提高了机器学习算法的效率。因此，基于内存计算的DBSCAN算法并行化，可以提高算法处理的效率。

发明内容

[0009] 为了能够更好的解决DBSCAN算法在处理海量数据下的效率问题，本发明提出了一种基于内存计算的并行化聚类方法。它采用内存计算中自定义的RDD算子来实现并行计算，能快 速实现原始数据的划分以及聚类结果的合并。具有更好的运行效率与可扩展性。

[0010] 本发明基于内存计算的并行化聚类方法，包括如下步骤：

[0011] 基于简单随机抽样的数据划分，以<ID,Raw_data>作为此阶段的输入，通过对原始数据进行简单随机抽样来完成数据的切分，并将切分的结果保存到不同的RDD。具体的子流程如下：

[0012] S11:在集群中启动实现map接口的作业,各个节点以原始数据作为输入,通过自定义的random()函数所产生的随机数作为当前数据的key值,生成带有新的划分标识的原始数据RDD集合;

[0013] S12:将S11中产生的RDD集合按照划分标识分解成对应的局部RDD集合。

[0014] S2:并行计算本地DBSCAN算法,利用内存计算模型在各个计算节点并行执行DBSCAN算法,产生局部类簇。具体的子流程如下:

[0015] S21:对每个局部RDD集合,首先根据数据划分个数确定聚类的邻域半径ε以及阈值MinPts;

[0016] S22:选取任一数据对象进行邻域查询,若该对象为核心对象,则将其邻域内所有的数据加入到list中,进行递归调用;若该对象为噪声对象,则将其标记为Noise;

[0017] S23:重复S22,直到所有的对象均被标记,同时尽可能的归到某个类簇中,或者找出那些不属于任何类簇的噪声对象;

[0018] S24:生成局部类簇RDD。

[0019] S3:基于重心合并所有的局部类簇,利用内存计算模型对局部类簇进行合并,从而产生全局聚类结果,具体的子流程如下:

[0020] S31:计算各个数据划分中局部类簇之间的距离,求得最小值局部d_{min};

[0021] S32:根据局部d_{min}求得在整个数据集下的全局D_{min};

[0022] S33:根据D_{min}确定合并阈值σ;

[0023] S34:构建重心距离矩阵;

[0024] S35:根据合并阈值σ,产生合并序列RDD;

[0025] S36:根据合并序列RDD对局部类簇进行合并,从而产生最终的全局聚类结果。

[0026] 本发明因为采用上述技术方案因此具备以下有益效果:

[0027] 与现有技术相比,本发明所提供的一种基于内存计算的并行化聚类方法,能很好地解决大规模数据集聚类的效率问题。采用分布式编程模型,通过简单的数据划分方式对原始数据 进行切割,极大地提高了算法的处理效率。同时,基于重心距离的局部类簇合并能快速地构建全局类簇,满足了处理大规模数据的用户需求。

附图说明

[0028] 图1为本发明方法的框架图;

[0029] 图2为并行计算本地DBSCAN算法流程图;

[0030] 图3为并行计算本地DBSCAN算法示意图;

[0031] 图4为改进的并行局部类簇合并流程图;

[0032] 图5为改进的并行局部类簇合并示意图。

具体实施方式

[0033] 下文与图示本发明原理的附图一起提供对本发明一个或者多个实施例的详细描述。结合这样的实例描述本发明,但是本发明不限于任何实施例。本发明的范围仅由权利要求书限定,并且本发明涵盖诸多替代、修改和等同物。在下文描述中阐述诸多具体细节以便提供对本发明的透彻理解。出于示例的目的而提供这些细节,并且无这些具体细节中的一

些或者所有细节也可以根据权利要求书实现本发明。

[0034] 如上所述，本发明所提供的一种基于内存计算的并行化聚类方法，能很好地解决大规模数据集聚类的效率问题。采用分布式编程模型，通过简单的数据划分方式对原始数据进行切割，极大地提高了算法的处理效率。同时，基于重心距离的局部类簇合并能快速地构建全局类簇，满足了处理大规模数据的用户需求。

[0035] 在执行算法之前，需要根据具体场景对一些参数进行初始化，如DBSCAN算法的邻域半径ε以及阈值MinPts、实际的计算节点数k进行设置，原始数据Raw_data按行存储在HDFS中，格式为<ID,Raw_data>，ID为行号。这里我们以UCI数据集中的3D-Road-Network数据集为例，该数据集中包含了434874条记录，我们从中任意抽取10条记录组成我们的测试数据集Test_Data，数据集的格式为：Road-ID、Longtitude、Latitude以及Altitude组成，具体的数据格式如表1所示。在本例中邻域半径ε取0.1，MinPts取100，k取5。

[0036] 表1 3D-Road-Network数据格式

[0037]

字段	含义	类型	范围
Road-ID	道路ID	int	1~71567
Longtitude	经度	double	0~360
Latitude	维度	double	0~180
Altitude	高度	double	-10~200

[0038] 参考图1，本发明的具体步骤包括：S1：基于简单随机抽样的数据划分，它的思想是：首先根据实际的计算节点确定分区的个数，并在此基础上通过自定义的random()函数，将原始数据随机的输出到各个分片中，同时各个分片的数据个数大致相同。每个分片都相当于一次简单的随机抽样，当每个分片抽取的样本数足够大时，抽取的样本与原始的数据具有相似的分布，并且将划分结果保存到HDFS或其他存储系统中。S2：并行计算本地DBSCAN算法，利用内存计算模型在各个计算节点并行执行DBSCAN算法，产生局部类簇。步骤3：S3：基于重心合并所有的局部类簇，利用内存计算模型对局部类簇进行合并，从而产生全局聚类结果。

[0039] S1：基于简单随机抽样的数据划分，以<ID,Raw_data>作为此阶段的输入，通过对原始数据进行简单随机抽样来完成数据的切分，并将切分的结果保存到不同的RDD中。设计Data_PartitionMap、以及Data_PartitionReduce来完成数据分区。具体的子流程如下：

[0040] S11：在集群中启动实现map接口的作业，以存储在HDFS中的原始数据Raw_data作为输入，输入为(<ID,Raw_data>,k)，取其value值Raw_data。然后使用random()函数产生1到k之间int类型的随机数r_number，将随机数r_number作为key值，输出<key,Raw_data>。

[0041] S12：进入到reduce处理阶段，它将根据上一步输出<key,Raw_data>中的key值合并原始数据Raw_data。相同key值的Raw_data会分发到同一个Reducer中，从而完成对原始数据的划分。因为在并行计算的过程中存在着多个reduce处理过程，最终会将所有的结果合并起来，生成局部RDD数据集，并保存在HDFS中供下阶段使用。

[0042] 参考图2与图3，本发明的S2中并行计算本地DBSCAN算法，将<key,list(Raw_data)>作为此阶段的输入，设计Local_DBSCAN来完成本地DBSCAN的计算，Local_DBSCAN包括

Local_DBSCAN_Map与Local_DBSCAN_ReduceByKey，具体的子流程如下：

[0043] S21：确定参数

[0044] 设定聚类的邻域半径ε为0.1，阈值为20。因为将原始数据分成了5个子集合，因此对于每个集合，它的密度为原始密度的1/5。设置Flag为对象属性标识，Flag的值可以为NOISE、CORE以及BORDER。设置CID为局部类别标识，它的初始值为key_0，并在发现新的类簇后对CID进行更新，依次产生key_1、key_2等等，其中key为步骤1中产生的分区标识。

[0045] S22：邻域查询

[0046] 在集群中启动实现map接口的作业，从局部RDD集合中，从list(Raw_data)中的任一数据对象p开始，进行ε邻域查询，若它的ε邻域的对象数大于阈值20，则该点为核心对象，将 其Flag标记为CORE，否则标记为NOISE；

[0047] S23：密度扩展

[0048] 若p为核心对象，则以p为中心，ε为半径建立类簇，同时将该类簇中的数据对象依次加入到一个容器List中进行递归调用，直至各个局部RDD中所有的对象均被标记，同时尽可能的归到某个类簇中，或者找出那些不属于任何类簇的噪声对象。因此，将局部类簇标识CID与原始数据组成新的对象，即<局部类簇标识，原始数据>，并输出该对象，生成新的局部类簇RDD数据集，并将其保存到HDFS中。输出格式为<Flag,Raw_data>，或者输出<CID,(Flag,Raw_data)>。

[0049] S24：计算各个局部类簇重心

[0050] 在集群中启动实现ReduceByKey接口的作业，读取原始数据，使用saveAsTextFile将局部类簇分别保存到不同的RDD中；计算出list(Flag,Raw_data)或者list(Raw_data)中的数据个数n；对Raw_data中的字段进行切分，并计算出各个字段上的$(\sum_{i=1}^{n} x_i)/n$；构建局部类簇的重心barycenter，输出<key,(barycenter,CID)>。

[0051] 参考图4与图5，本发明的S3中基于重心合并所有的局部类簇，在此阶段，需要对各个分区的局部聚类结果进行合并，从而生成全局聚类结果。由于各个分片之间的分布具有相似性，那么各分片的局部类簇之间也具有一定的相似性。本发明提出一种基于重心的动态数据分区合并策略来对局部类簇进行合并，它的总体思想是：首先根据上阶段求出的各个局部类簇重心求出分片内部各个局部类簇重心之间的距离，并通过快速排序或堆排序得到分片内部类簇之间重心距离的最小值，继而求得在整个数据集合下两两局部类簇重心距离的最小值D_{min}。在进行局部类簇合并之前，可根据D_{min}动态地设定阈值σ，使得σ与D_{min}的关系为$\sigma<<D_{min}$，同时构建重心距离矩阵来统计各个分片间重心的距离，遍历重心距离矩阵中的元素，将小于阈值σ对应的局部类簇加入到合并队列中，并结合贪心算法完成合并序列的构建，从而完成数据CID的更新。设计Partition_Combine来实现数据分区合并。Partition_Combine包含Partition_Combine_ReduceByKey、Partition_Combine_Reduce以及ReLabel_Map，其中Partition_Combine_ReduceByKey用来完成局部d_{min}的获取，Partition_Combine_Reduce用来构建合并序列，ReLabel_Map用来更新类簇标识，形成全局类簇。具体的子流程如下：

[0052] S31：确定局部d_{min}

[0053] 在集群中启动实现ReduceByKey接口的作业，将相同分区的重心数据输入到相同

的ReduceByKey作业中。初始化时将<key,(barycenter,CID)>装入内存,取其value字段构造结构体D_Node＝{CID,barycenter},并将D_Node中的barycenter取出来,求得两两之间的距离d;为了将所有ReduceByKey计算的结果输入到相同的Reduce中,需要自定义统一的标识S,因此输出为<S,(list(barycenter,CID),d_{min})>。

[0054] S32:确定全局D_{min}

[0055] 在集群中启动实现Reduce接口的作业,初始化时将<S,(list(barycenter,CID),d_{min})>装入内存,取其value字段,分别把(list(barycenter,CID)与d_{min}保存起来供下阶段使用;使用堆排序或者快速排序对d_{min}进行排序,得到整个数据集下两两局部类簇重心距离的最小值D_{min},从而进行阈值σ的设定,使得σ<<D_{min};

[0056] S33:确定合并阈值σ

[0057] 在确定全局D_{min}后,可以对局部类簇的合并阈值σ进行定义。当两个局部类簇重心之间的距离小于阈值σ时,可以将该两个局部类簇合并为一个类簇。反之,则不需要合并这两个类簇。受分片个数以及数据集大小的影响,当σ选取过大时,可能会导致部分类被融合,聚类数目偏少。当σ选取过小时,导致部分类之间密度隔离,导致聚类数目过多。若数据量足够大,分片之间的数据分布较为稳定,σ的选取应大于(1/20)D_{min}较为合理。

[0058] S34:构建重心距离矩阵

[0059] 为了能够有效地计算局部类簇重心之间的距离,需要构建重心距离矩阵,矩阵内存储着各个局部类簇之间的距离,比如M_{ij}代表了局部类簇i的重心与局部类簇j的重心之间的距离。在此阶段,通过list(barycenter,CID)构建重心距离矩阵BaryCenter_Matrix。

[0060] S35:构建合并序列

[0061] 在完成重心距离矩阵的构建后,接下来将基于阈值σ对重心矩阵进行搜索,结合贪心算法的思想,首先将重心矩阵中的所有对象均标记为unvisited,然后从矩阵的任意行任意列进行搜索,当发现有值小于阈值σ时,则将对应的CID_i与CID_j加入到list中,将它们标记为visited,并分别从list中选取对象继续进行搜索,找出离list中所有对象最近的CID_k,若满足距离小于阈值σ,则将CID_k加入到list中同时将其标记为visited。重复该过程,直到该list无 法继续扩展为止。在完成一次list构建时,相当于形成了一个全局类簇。接下来,需要从矩阵中任意标记为unvisted的对象进行搜索,完成另一个全局类簇list的构建,依次重复,直到所有的全局类簇被找到。当所有的合并序列list完成构建后,就产生了所有的全局类簇,搜索完成后得到list(merge sequence),因此,此阶段的输出为<G_CID,list(merge sequence)>,其中G_CID为全局类别标识,它的初始值为0,每产生一个合并序列,G_GID的值加1。将<G_CID,list(merge sequence)>生成合并序列RDD数据集,保存到HDFS中,在下阶段可根据该合并序列对类别标示进行更新。

[0062] S36:产生全局聚类结果

[0063] <G_CID,list(merge sequence)>与<CID,(Flag,Raw_data)>作为此阶段的输入,其中list(merge sequence)为合并序列RDD数据集,<CID,(Flag,Raw_data)>为局部类簇RDD数据集。在集群中启动实现ReduceByKey接口的作业,读取输入的<G_CID,list(merge sequence)>,<CID,(Flag,Raw_data)>;将CID在list(merge sequence)中进行搜索,若CID在merge sequence中,则将它的CID改为G_CID,依次重复,直到所有的CID标识更新完成为止;输出为<G_CID,(Flag,Raw_data)>。这样就完成了局部类簇RDD数据集的CID标识的更

新，产生了全局聚类结果，并通过saveAsTextFile将生成的<G_CID,(Flag,Raw_data)>作为新的RDD保存在HDFS中。得到的聚类结果示意如表2所示。

[0064] 表2 聚类结果示意

[0065]

聚类个数	Noise点个数
5	982

[0066] 综上所述，本发明提供了一种基于内存计算的并行化聚类方法，以上说明只是用于帮助理解本发明的方法及其核心思想；同时，对于本领域的一般技术人员，依据本发明的思想，在具体实施方式及应用范围上均有改变之处，综上所述，本说明书的内容不应该被理解为对本发明的限制。因此，在不偏离本发明的精神和范围的情况下，所做的任何修改、等同替换、改进等，均应包含在本发明的保护范围之内。此外，本发明所附的权利要求旨在涵盖落入所附权利要求范围和边界、或者这种范围和边界的等同形式内的全部变化和修改例。

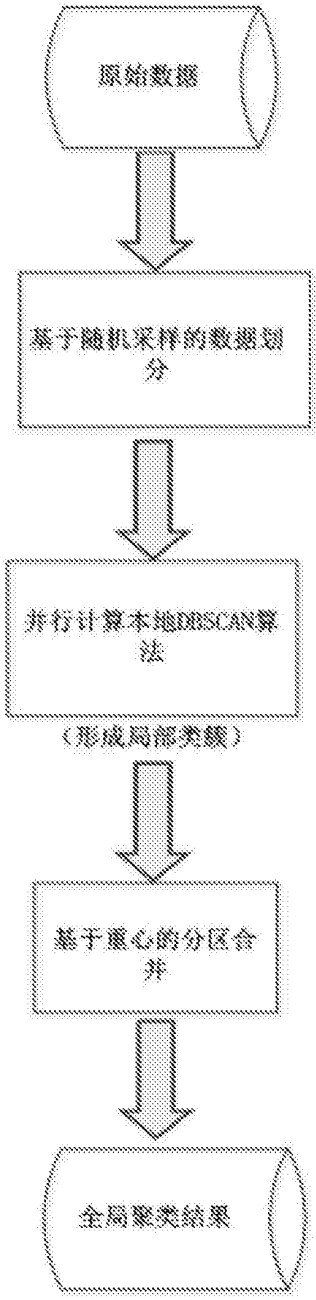

原始数据
基于随机采样的数据划分
并行计算本地DBSCAN算法
（形成局部类簇）
基于重心的分区合并
全局聚类结果

图1

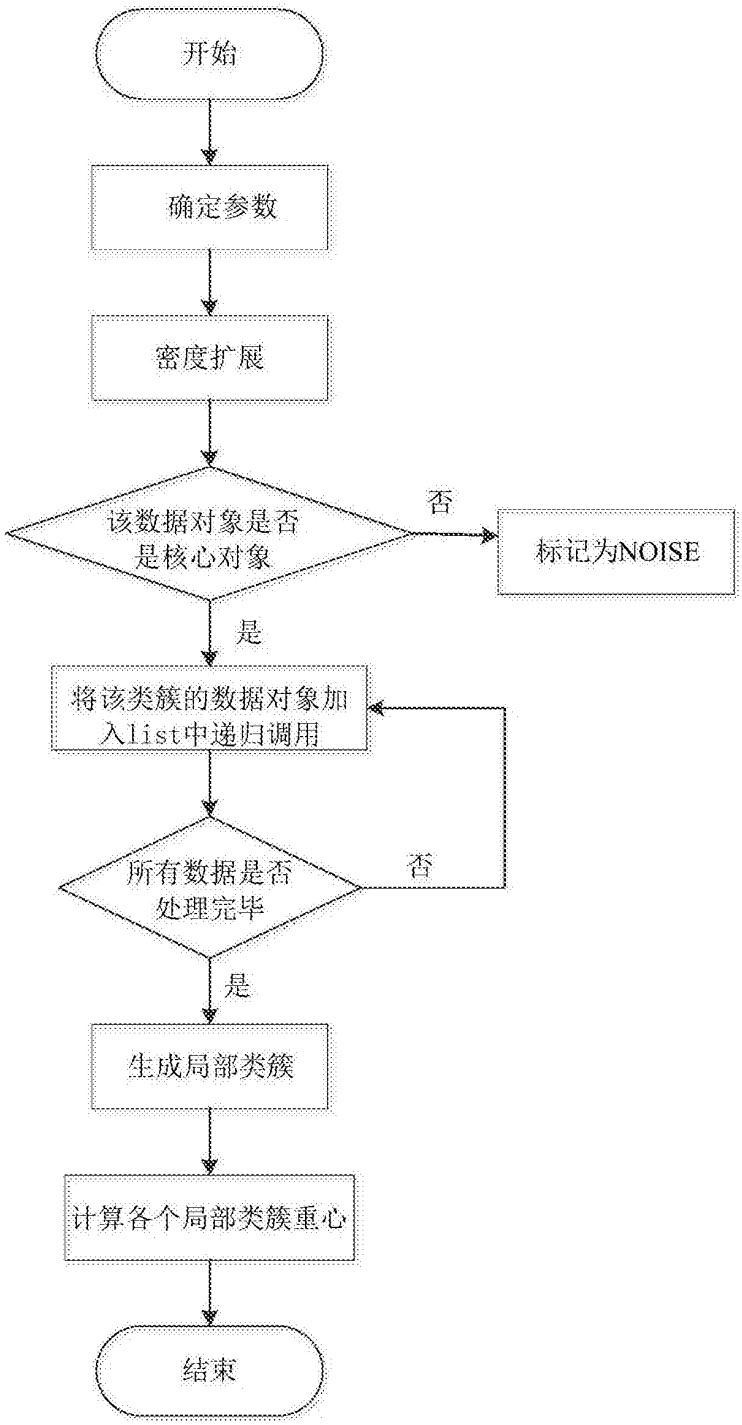

开始
确定参数
密度扩展
该数据对象是否是核心对象
否
标记为NOISE
是
将该类簇的数据对象加入list中递归调用
所有数据是否处理完毕
否
是
生成局部类簇
计算各个局部类簇重心
结束

图2

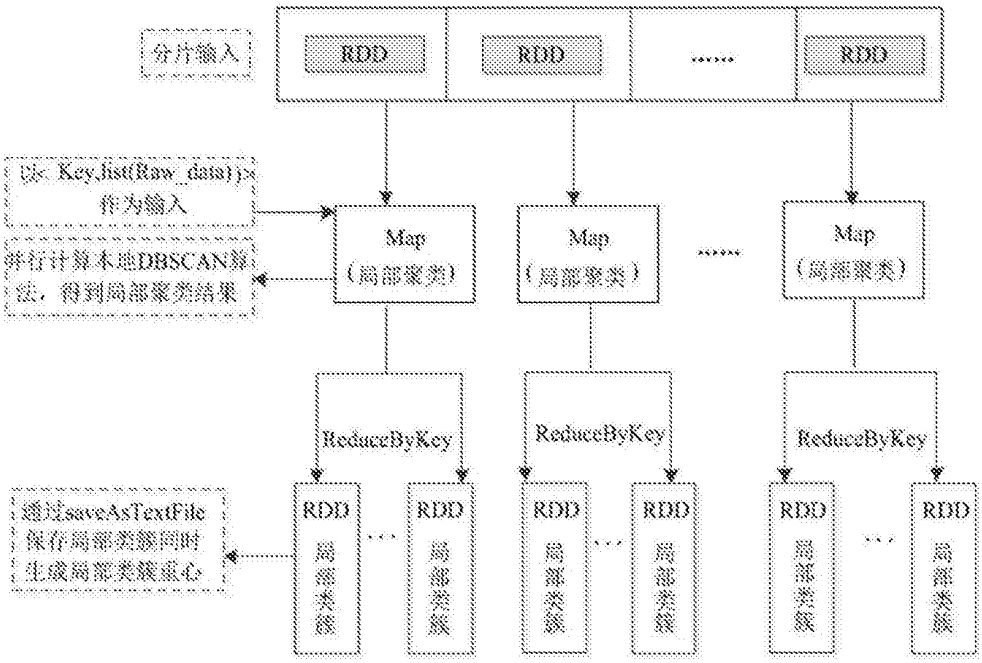

分片输入
RDD
RDD
.....
RDD
以< Key,list(Raw_data)>作为输入
并行计算本地DBSCAN算法，得到局部聚类结果
Map
(局部聚类)
Map
(局部聚类)
.....
Map
(局部聚类)
ReduceByKey
ReduceByKey
ReduceByKey
通过saveAsTextFile保存局部类簇同时生成局部类簇质心
RDD
局部类簇
RDD
局部类簇
RDD
局部类簇
RDD
局部类簇
RDD
局部类簇
RDD
局部类簇

图3

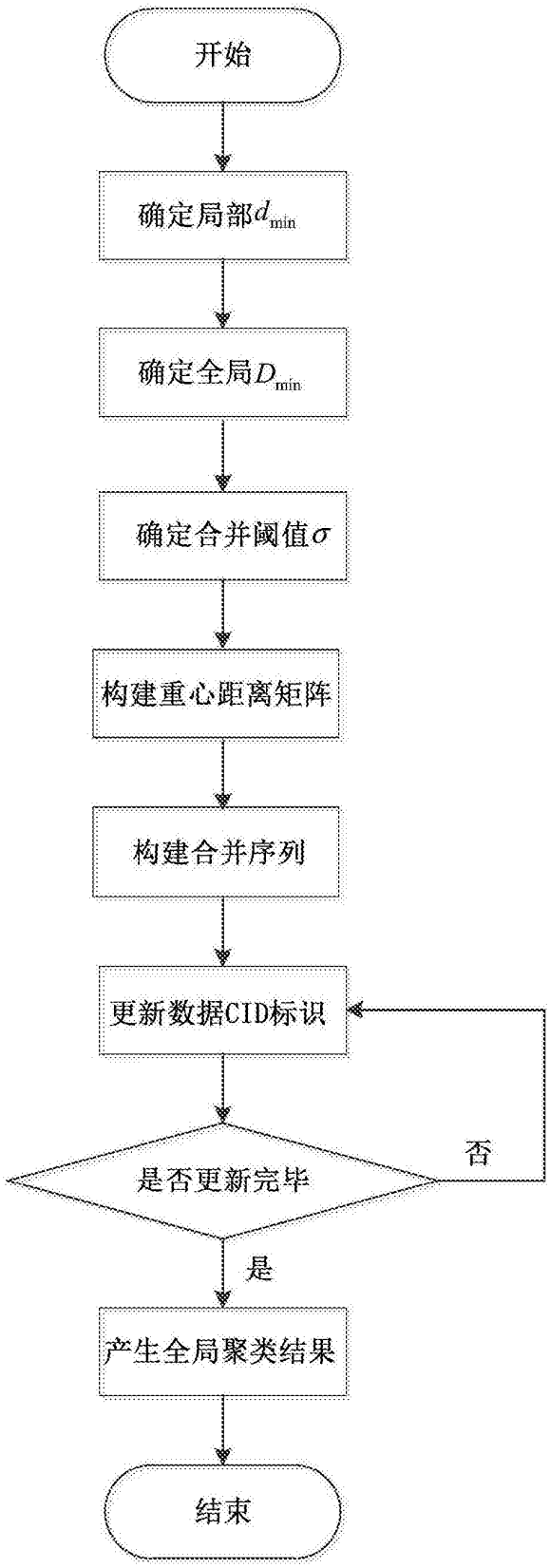

开始
确定局部d_{min}
确定全局D_{min}
确定合并阈值σ
构建重心距离矩阵
构建合并序列
更新数据CID标识
是否更新完毕
否
是
产生全局聚类结果
结束

图4

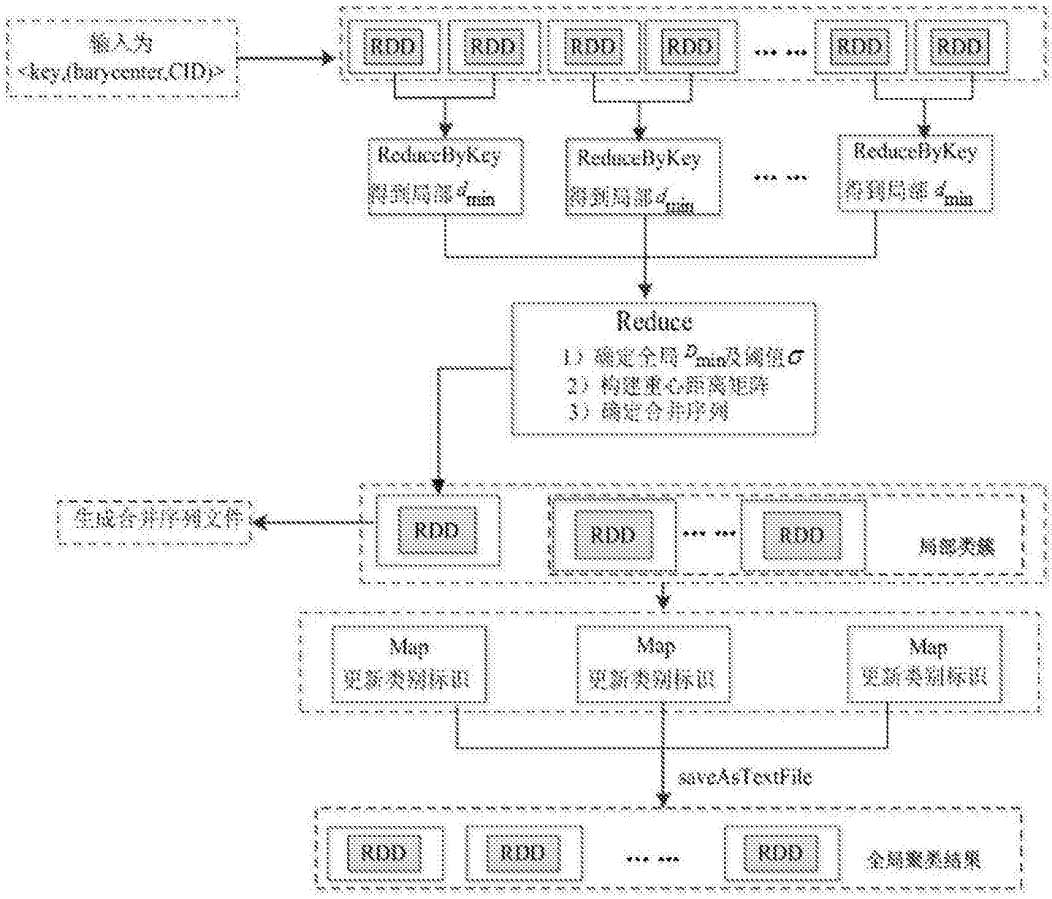

输入为
<key,(barycenter,CID)>
RDD
RDD
RDD
RDD
RDD
RDD
ReduceByKey
ReduceByKey
ReduceByKey
Reduce
2）构建重心距离矩阵
3）确定合并序列
生成合并序列文件
RDD
RDD
RDD
局部类簇
Map
更新类别标识
Map
更新类别标识
Map
更新类别标识
saveAsTextFile
RDD
RDD
RDD
全局聚类结果

图5