文章编号:1674-2974(2020)10-0100-11 DOI:10.16339/j.cnki.hdxbzkb.2020.10.012

异构分布式环境中的并行离群点检测算法

王习特†,朱宗梅,于雪苹,白梅
(大连海事大学 信息科学技术学院,辽宁 大连 116000)

摘 要:离群点检测是数据挖掘领域研究的热点之一,主要目的是识别出数据集中异常但有价值的数据点.随着数据规模不断扩大,使得处理海量数据的效率降低,随即引入分布式算法.目前现有的分布式算法大都用于解决同构分布式的处理环境,但在实际应用中,由于参与分布式计算的处理机配置的差异,现有的分布式离群点检测算法不能很好地适用于异构分布式环境.针对上述问题,本文提出一种面向异构分布式环境的离群点检测算法.首先提出基于网格的动态数据划分方法(Gird-based Dynamic Data Partitioning,GDDP),充分利用各处理机的计算资源,同时根据数据点的空间位置信息进行数据划分,可有效减少网络通信.其次基于GDDP算法,提出了异构分布式环境中并行的离群点检测算法(GDDP-based Outlier Detection Algorithm,GODA).该算法包括2个阶段:在每个处理机本地,按照索引中数据点的顺序进行过滤,通过2次扫描得到离群点候选集;判断候选离群点需要进行网络通信的处理机,使用较低网络开销得出全局离群点.最后,通过大量实验验证了本文提出的GDDP和GODA算法的有效性.

关键词:离群点检测;异构分布式;网格;数据划分
中图分类号:TP391 **文献标志码**:A

Parallel Outlier Detection Algorithm in Heterogeneous Distributed Environment

WANG Xite†,ZHU Zongmei,YU Xueping,BAI Mei
(College of Information Science and Technology,Dalian Maritime University,Dalian 116000,China)

Abstract:Outlier detection is one of the hotspots in the field of data mining. The main purpose is to identify the abnormal but valuable data points in the data set. With the expansion of data scale,the efficiency of processing massive data is reduced,and then a distributed algorithm is introduced. At present,most of the existing distributed algorithms are used to solve the homogeneous distributed processing environment. However,in practical applications,due to the differences in processor configuration involved in distributed computing,the existing distributed outlier detection algorithms cannot be well applied to heterogeneous distributed environments. In view of the above problems,this paper proposes an outlier detection algorithm for heterogeneous distributed environments. Firstly,a grid-based dy-

* **收稿日期**:2019-12-26
基金项目:国家自然科学基金资助项目(61602076,61702072,61976032),National Natural Science Fundation of China(61602076,61702072,61976032);辽宁省自然科学基金资助项目(20180540003),National Natural Science Fundation of Liaoning Province;中国博士后科学基金面上项目(2017M611211,2017M621122);国家重点研发计划项目课题(2017YFC1404606);中央高校基本科研业务费专项资金(3132019202)
作者简介:王习特(1987—),男,辽宁大连人,大连海事大学副教授,博士
† 通讯联系人,E-mail:wangxite@dlmu.edu.cn

namic data partitioning (GDDP) method is proposed, which makes full use of the computing resources of each processor and divides the data according to the spatial location information of the data points, which can effectively reduce the network communication. Secondly, based on the GDDP algorithm, a parallel GDDP-based Outlier Detection Algorithm (GODA) is proposed. The algorithm consists of two stages: in each processor, filtering according to the order of the data points in the index and obtaining the outlier candidate set by two scans; determining the candidate outliers requiring network communication and using low network overhead leads to global outliers. Finally, the effectiveness of the proposed GDDP and GODA algorithms is verified by a large number of experiments.

Key words: outlier detection; heterogeneous distribution; grid; data partition

离群点检测是数据管理领域的重要研究内容之一，其主要目的是识别出数据集中与其他数据存在显著差异的数据对象. 随着对离群点的不断研究,出现了多种离群点描述方式. 其中,最直观的方式是通过度量数据空间中数据点间的距离来量化数据的离群程度. 具体地,基于距离的离群点可定义为:给定参数 k 和 r,对于数据集 P 中任意数据点 p,若与点 p 的距离小于等于 r 的数据点个数少于 k,则 p 为离群点[1].

随着数据库中数据量的不断增加，有关离群点的研究已从集中式处理环境转向分布式处理环境,以提高海量数据的离群点检测效率[2]. 但是,现有的分布式算法大都用来解决同构分布式环境中的离群点检测问题. 在实际应用中,参与分布式计算的处理机大都存在配置上的差异，各处理机的性能可能存在较大差异. 这也就使得现有的针对同构分布式环境设计的离群点检测算法不能很好地适用于异构分布式环境. 因此,迫切需要设计一种高效的异构分布式环境中的并行离群点检测算法.

本文主要针对异构分布式环境下的离群点检测问题,提出一种动态数据划分方法和相应计算方法,旨在提高算法的执行效率. 具体地,本文的主要贡献包括以下几个部分:

1) 提出一种基于网格的动态数据划分方法 GDDP (Gird-based Dynamic Data Partitioning). 该方法依据异构分布式环境中各处理机计算能力的差异动态分配计算任务，实现了各处理机计算任务的负载均衡,加快了计算效率. 同时,考虑了数据集中数据点所处的空间位置信息，尽可能将空间上相邻的数据点划分至同一处理机,降低网络开销.

2)基于 GDDP 划分方法,提出了异构分布式环境中并行的离群点检测算法 GODA (GDDP-based Outlier Detection Algorithm). 该算法主要包括两个过程:①本地离群点计算.首先通过索引对每个处理机本地的数据集进行管理，然后根据过滤规则实现数据集中非离群点的过滤，从而快速计算出本地离群点;②全局离群点计算.利用每个候选离群点和其所在数据块与相邻数据块的距离关系，判断需要进行网络通信的处理机，然后统一将候选点发送到相应的处理机上,从而得到全局离群点.

3)分别利用真实数据集和人工合成数据集进行实验，验证了本文提出的 GDDP 和 GODA 算法的有效性.

1　相关工作

本节主要从集中式和分布式两方面对离群点检测的相关研究进行概述.

1.1　集中式离群点检测

目前，大多数离群点研究都是面向集中式环境的. 研究内容主要包括两个方面:离群点定义和离群点检测方法. 首先,离群点定义方面的研究致力于给出一种描述或者形式化定义离群点的方法. Hawkins 等[3]最早给出离群点的形象化描述,认为离群点是数据集中与其他点差距很大、且怀疑与其他数据点并非由同一机制生成的点. 为了更明确数据点是否为离群点,提出基于模型的离群点定义,以数据的分布模型为参考，将数据集中不符合分布模型的数据点定义为离群点[4]. 至今,研究人员给出了多种不同的离群点定义标准,如:基于距离的离群点、基于聚类的离群点[5]、基于密度的离群点[6]. 其中,基于距离的离群点定义能直观地反应离群点本质进而得到广泛应用.

其次，离群点检测是指根据某种离群点判定方

式，从数据集中快速找到符合该判定方式的数据点的过程,判定方式不同,相应的离群点计算方法也不同. 具体地,文献[1]基于距离的离群点定义,提出一种嵌套循环方式，对数据集中每个数据点进行距离计算,以找出最终的离群点. 这种方法可以应对多维数据集,但实际的计算效率有待优化. 之后,Bay 等[7]提出 ORCA 算法,利用缓冲区给出一种修剪规则,实现对非离群点的过滤,加快检测效率. 此外,有些学者利用空间索引加速离群点的计算过程. 文献[8] 提出了基于 R 树的离群点检测算法,文献[9]给出了基于 k–d 树的离群点检测算法.

1.2 分布式离群点检测

为了解决传统集中式离群点检测算法在大规模数据中检测效率低的问题，许多研究人员提出了有效的分布式算法，而现有的分布式离群点检测算法都是面向同构分布式环境的. Hung 等[10]首先给出分布式环境中基于距离的离群点检测算法，将数据均匀划分至各处理机，然后对传统嵌套循环算法进行优化,实现分布式的离群点计算过程. 文献[11]首先提出了 GBP 数据划分方法,利用网格空间索引结构对数据进行管理和划分,然后基于 GBP 划分方法给出分布式离群点检测算法 DLC. 同样地,针对同一离群点定义,文献[12]也提出了相应的分布式离群点计算方法，在处理具有偏态分布的数据集时非常有效. 文献[13]首先提出了 BDSP 数据划分算法,可以有效均衡各计算节点的工作负载，同时实现良好的过滤效果，然后利用 BDSP 方法提出基于距离的分布式离群点检测算法 BOD,有效减少网络开销,缩短了离群点的计算时间. 此外,文献[14]还给出了利用 GPU 并行地计算离群点的策略.

虽然已有一些优秀的分布式离群点检测方法，但这些方法都是针对同构分布式环境设计的，在异构分布式环境中无法保证高效的离群点检测效率.

2 问题描述

本文主要研究在异构分布式环境中，如何并行地计算出数据集内所有基于距离的离群点. 下面,具体地给出基于距离的离群点的形式化定义,其中,表 1 列出了本文使用的符号及其含义.

一般地，给定一个具有 d 维属性值的数据集合 P. 对于 P 中任一数据点 p 可表示为：p<p[1],p[2],p[3],…,p[d]>,那么对于数据集 P 中任意两个数据点 p_1 和 p_2 之间的欧式距离的计算公式为：

$$\mathrm{dist}(p_1,p_2)=\sqrt{\sum_{i\in[1,d]}(p_2[i]^2-p_1[i]^2)} \tag{1}$$

表 1 符号列表

Tab.1 List of symbols

符 号	含义
P	数据集
$\|P\|$	数据集大小
p	数据点
r	查询距离阈值
k	查询数量阈值
d	数据维度
c	从节点
C	从节点集合
b	数据块
B	数据块集合
g	网格块

定义 1 （r–近邻集合）. 给定数据集合 P 和用户定义的查询半径 r,点 p_i 的近邻集合是指 P 中所有到 p_i 的距离不大于 r 的点的集合,即 $N(p_i,r)=\{p_j \mid p_j\in P,\mathrm{dist}(p_i,p_j)\leqslant r\}$.

定义 2 （(r,k)–离群点）. 给定距离阈值 r 和查询邻居阈值 k，对于数据集 P 中的任意一数据点 p，若到数据点 p 的距离小于等于 r 的数据点个数小于 k,即 $|N(p_i,r)|<k$,则数据点 p 被称为(r,k)–离群点.

3 异构分布式中的离群点查询处理框架

本文主要针对异构分布式环境，其整体上包含一台主控制节点和数目固定的多个从节点 $C=\{c_1,c_2,\cdots,c_n\}$. 由于各节点间数据迁移量较小,本文不考虑网络异构，其异构环境特指每个参与计算的从节点的配置各不相同，使得它们在计算能力上存在差异. 另外，主控节点和每一个从节点都通过网络互连,可以进行网络通信. 每两个相邻的从节点之间也可相互通信. 为了防止主控节点在计算过程中发生拥塞,在数据划分时将不给主控节点分配计算任务，即在整个基于距离的离群点计算过程中，主控节点主要负责计算任务的调度，大部分的离群点计算任务都将由从节点完成.

根据图 1 中的查询处理框架，异构分布式环境中的基于距离的离群点检测任务可以被描述为:对

于给定的数据集 P、查询距离阈值 r 和查询数量阈值 k,通过充分利用异构分布式计算框架中各异构处理机的计算能力，快速地计算出给定数据集合 P 中的所有基于距离的离群点.

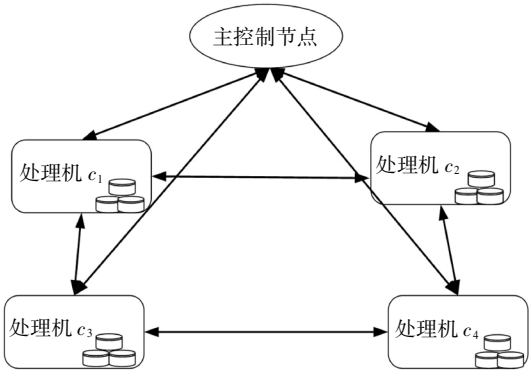


图 1　查询处理框架

Fig.1　Query processing framework

具体地,整个过程大致可分为 3 个阶段:

1)数据划分阶段. 对于给定数据集 P,主控制节点首先依据各异构处理机的计算能力的不同对数据集进行划分，然后将划分后的子数据集发送至相应的处理机.

2)局部离群点计算阶段. 各个参与计算的处理机在接收到对应的数据子集合后，在本地进行局部计算,得到局部离群点,最后将它们发送到主控制节点.

3)全局离群点计算阶段. 主控制节点根据接收到的局部离群点的位置信息判断其需要通信的从节点，然后将它们发送到相应处理机再次进行距离计算,得出最终的全局离群点.

4　基于网格的动态数据划分方法

在离群点检测之前,对数据集进行有效的划分,可以显著提高离群点检测效率. 本文提出的基于网格的动态数据划分方法 GDDP 保证了计算资源的充分利用,具有较高的负载均衡性.

4.1　计算能力评估

在分布式计算系统中为了能够实现整个系统的高可用性，一般需要实现各个参与计算的处理机之间负载均衡. 在异构分布式环境中,各处理机间的计算能力不同,为了能对计算任务进行合理的分配,需要对每个处理机的计算能力进行评估.

首先，查询处理框架中的主控制节点会将相同的一份测试数据集发送给每个参与计算的从节点，进行相同的任务计算，然后记录每个处理机完成任务的时间. 其中,处理机完成任务的时间越短,说明其计算能力越强;反之,其计算能力越弱.

为了便于后续基于网格的数据集的划分，本文给出一种有效的评分机制. 首先,将工作节点的计算能力划分为 m 个等级,即 $\beta_1,\beta_2,\cdots,\beta_m$,对应得分分别为 $1,2,3,\cdots,m$,相应的工作节点的计算能力依次增强;其次,求出每个从节点的计算能力与当前最大计算能力的比值,所得结果在(0,1]内,其中,将(0,1]划分为 m 等份,表示(0,1/m],(1/m,2/m],…,(m−1/m,1]，所得比值在各区间内的处理机的等级分别为 $\beta_1,\beta_2,\cdots,\beta_m$;最后,根据每台处理机的计算能力等级得出相应的评分. 例如，在一个异构分布式系统中有 c_1~c_5 5 台从节点,为简化示例,此处等级数 m=4,由以上方法得出各处理机的计算能力评估表如表 2 所示.

表 2　处理机的计算能力评估

Tab.2　Evaluation of the calculation ability of the processors

处理机	完成时间	速度	比值(速度/最大速度)	等级	得分
c_1	5	1/5	0.4	β_2	2
c_2	2	1/2	1	β_4	4
c_3	7	1/7	0.29	β_2	2
c_4	3	1/3	0.67	β_3	3
c_5	10	1/10	0.2	β_1	1

4.2　数据集划分

传统的数据划分方法对维度过高的数据集进行处理时，会产生巨量的划分子块，致使检测效率低下，由此本文决定对给定的多维数据集中数据分布最均匀的二维空间利用网格进行数据划分，经数据划分后会形成多个二维的数据子块.

首先确定数据集中每一维数据需划分为几段. 考虑在异构分布式环境中，各从节点的计算能力不同,由此具体的划分段数为:

$$\text{Segment}=\left\lceil\sqrt{\sum_{i=0}^{n}\text{comp\_Grade}(c_i)}\right\rceil \tag{2}$$

其中,comp_Grade（c_i）表示处理机 c_i 的计算能力得分,n 表示处理机的数目. 例如,根据表 2 中对 5 台异构处理机 c_1 ~ c_5 计算能力的评估结果,结合公式(2)可得划分段数 Segment 值为 4,因此,数据集将被映射到一个 4×4 规模的网格结构内.

其次,为了判断数据在哪个维度上分布最均匀,

本文给出一种衡量标准. 具体地,数据在第 d 维上分布的均匀程度为:

$$\text{single\_}U(d)=\sum_{i=1}^{n}(\text{num}(\text{segment}(i))-\mu)^2 \quad (3)$$

single_$U(d)$的值越小,说明数据在第 d 维上分布越均匀. 其中,num(segment(i))表示数据在第 d 维上的第 i 个划分段内的实际分布数量;μ 表示数据在第 d 维上每个划分段数内的平均分布数量,它等于数据集的大小与划分段数 Segment 的比值;n 取值为划分段数 Segment 的值.

接下来,数据在由 d_1 和 d_2 维所组成的二维空间内的分布均匀程度用 multi_U (d_1,d_2) 来描述. 同样地,multi_$U(d_1,d_2)$的值越小,表明数据在该二维空间内分布越均匀. 具体地,multi_$U(d_1,d_2)$的计算公式表示为:

$$\text{multi\_}U(d_1,d_2)=\sum_{i=1}^{n}(\text{num}(g_i)-\mu)^2 \quad (4)$$

式中:num(g_i)表示数据在当前 d_1 和 d_2 维空间内所建立的网格索引中第 i 个网格内的实际分布数量;n 表示二维网格中网格的数目,其值等于划分段数 Segment 的平方;μ 表示数据在每个网格内的平均分布数据,其值等于数据集的总量与网格数目的比值. 由以上数据分布均匀程度的两个衡量过程,可确定给定数据集中数据分布最均匀的二维空间,从而完成数据集的划分. 具体地,对于给定的具有 d 个维度的数据集的划分过程大致为:首先,根据划分段数 Segment 将数据集的每一维度划分成 s 个区间,然后统计数据在每个区间内的分布情况,并根据计算得出数据在每一维上的 single_U (d) 值,选择 single_U (d) 值最小的$\left\lfloor \frac{d}{2} \right\rfloor$个数据维度作为候选目标;其次,在候选维度里面,根据划分段数 Segment,依次选取两个维度建立二维网格,并评估数据在二维网格内的分布情况;最终,根据 multi_$U(d_1,d_2)$值最小的二个维度上对应的网格对数据集进行划分,可以得到多个不相交的划分数据子块.

4.3 数据块的分配

当数据集被分成多个不相交的数据子块后,紧接着需将其合理地分配到各异构的从节点上.

4.3.1 分配顺序

根据基于距离的离群点定义,为了减少分布式计算过程中各处理机之间的通信代价,需要将空间中邻近的数据点尽量分配到同一台工作节点上. 因此,在数据块的分配过程中首先需要考虑数据集划分后各数据块之间的相邻关系,这样才能尽可能将空间中相邻的数据对象分配到同一台工作节点上. 由于划分时用到的是二维网格,本文则按照网格的行序将相邻的数据块依次进行分配即可保证数据块之间是相邻的,如图 2 中箭头所指方向即为数据块的分配顺序.

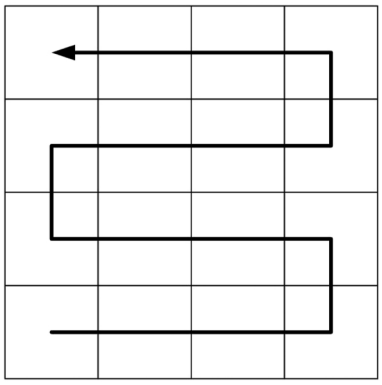


图 2 数据块的分配顺序

Fig.2 The allocated order of data blocks

4.3.2 分配规则

在本文所指的异构分布式环境中,对于参与计算的异构处理机,按照其计算能力由大到小顺序排列,之后优先选择计算能力强的处理机进行数据块的分配. 如下算法 1 给出了具体的数据块分配过程.

算法 1 数据块分配算法.

输入:数据块集合 $B=\{b_1,b_2,\cdots,b_b\}$,从节点集合 $C=\{c_1,c_2,\cdots,c_n\}$;

输出:分配给每个从节点的数据块集合

1. 初始化队列 H 为空;
2. 将从节点集合 C 中的节点按照计算能力由强到弱的顺序加入到队列 H 中
3. WHILE 队列 H 不为空 DO
4. 取队列 H 中的第一个节点 c_i;
5. 计算节点 c_i 需要的数据量 num(c_i);
6. 初始化集合 B_{temp} 为空;
7. FOR 数据块集合 B 中的每一个数据块 b_i DO
8. 用 num(b_i)记录数据块 b_i 的数据量;
9. IF 集合 B_{temp} 为空 或 num(b_i)≤num(c_i) DO
10. 将数据块 b_i 加入到集合 B_{temp} 中并将数据块 b_i 从集合 B 中删除;
11. num(c_i)= num(c_i) − num(b_i);
12. ELSE
13. BREAK;
14. ENDFOR
15. 将节点 c_i 从队列 H 中删除;
16. ENDWHILE

经过上述分配过程,可以完成数据集中大多数数据的初分配,并且分配后大多数处理机都能获得小于等于其实际所需的数据量.

例如,给定多维数据集 P,且$|P|$=120. 如表 2 所示,有 5 台异构的从节点参与离群点的计算,对应的 c_1~c_5 处理机顺序为:c_2、c_4、c_1、c_3、c_5. 图 3 所示为数据集划分的结果,则数据块的分配顺序为 b_1~b_{16}. 首先对处理机 c_2 进行数据块分配,计算可得其所需数据量为 $\frac{4}{2+4+2+3+1}\times 120 = 40$. 优先处理数据块 b_1,由于块内数据量 5 小于 c_2 当前所需数据量 40,则将 b_2 分配给处理机 c_2,同时更新 c_2 当前所需数据量为 35,最终,数据块 b_1、b_2、b_3、b_4、b_5 都可被分配给 c_2,当对数据块 b_6 进行分配时,由于处理机 c_2 目前所需要的数据量为 1,小于数据块 b_6 的数据量,所以停止对处理机 c_2 的数据块分配. 同理,可依次完成处理机 c_4、c_1、c_3、c_5 的数据块分配任务.

8 b_{16}	11 b_{15}	4 b_{14}	6 b_{13}
7 b_9	6 b_{10}	12 b_{11}	5 b_{12}
6 b_8	5 b_7	11 b_6	10 b_5
5 b_1	8 b_2	9 b_3	7 b_4

图 3　数据集划分结果

Fig.3　The partition result of the data set

具体的分配结果如表 3 所示. 这样经过数据块的分配后,只有处理机 c_5 所分配的数据量略大于它实际所需要的数据量. 同时,只有数据块 b_{16} 未被分配,把它暂时留在主控制处理机上.

表 3　分配结果

Tab.3　Allocation result

处理机	所需的数据量	分配的数据块	分配后的数据量
c_2	40	b_1,b_2,b_3,b_4,b_5	39
c_4	30	b_6,b_7,b_8,b_9	29
c_1	20	b_{10},b_{11}	18
c_3	20	b_{12},b_{13},b_{14}	15
c_5	10	b_{15}	11

4.4　动态数据分配

动态分配旨在对存储于主控制节点上的剩余的未被分配的数据块进行数据分配. 为了对空闲处理机进行准确的任务分配,主控制节点需记录从开始执行计算任务到有空闲节点出现所经过的时间,即当前空闲的处理机 c_i 完成初始的分配任务的时间 t,以及在处理时间 t 内每个处理机已经处理的数据量 complete_num(c_i),由此可计算出各处理机在当前所划分的数据集中进行离群点检测任务时的实时处理速度 v_i(其中,$v_i = \frac{\text{complete\_num}(c_i)}{t}$);同时,用当前未被处理的数据总量除以各处理机处理速度之和还可预估出剩余的所有计算任务完成的时间 T;最后,将时间 T 和空闲处理机的实时处理速度 v_i 相乘,即可得出当前空闲工作节点所应分配的数据量. 当存储在主控制节点上的数据都被分配后,动态分配过程结束.

这种动态数据分配方法依据各处理机的实际处理速度进行计算任务的分配,尽可能地避免了空闲处理机的等待,并充分利用了整个异构分布式环境中各处理机的计算资源,加快了离群点的检测速度.

5　GODA 算法描述

本节主要描述了 GODA 算法的具体细节. 整个过程分为 2 个阶段:1)本地离群点计算,即利用每个处理机的计算能力计算出本地离群点;2)全局离群点计算,即通过全局通信确定最终离群点结果集.

5.1　本地离群点计算

本小节主要阐述 GODA 算法中一种有效的面向多维大规模数据集的本地离群点计算方法.

5.1.1　索引构建

由于网格索引、R 树索引等一般的空间索引无法应对数据集维度的增长,并且根据离群点的形象化描述,在给定的数据集中,大部分数据点都是非离群点,而离群点只占据数据集的很少一部分,因此,本文索引构建采用如下方法.

在给定数据集中随机选取一个数据点 p,其中,点 p 在很大程度上为非离群点,以点 p 为中心,计算数据集中其他数据点到点 p 的距离,依据距点 p 由近及远的顺序对数据进行管理,如果数据点 q 距离中心点 p 很远,则说明点 q 很可能是离群点.

5.1.2　本地离群点计算方法描述

首先,对于给定数据集 P 中的数据点 p,自定义数据结构 node 如下:

1)给定数据集 P 中的一个数据点 p;

2)数据点 p 在给定数据集 P 中的唯一标识,用 $p.id$ 表示;

3)数据点 p 在给定数据集 P 中的邻居情况,用 $p.nn$ 表示;

4)数据点 p 与其第 k 个近邻点的距离,用 $p.r$ 表示.

其中,$p.nn$ 是一个大小为 k (用户给定的查询数量阈值)的列表,用于存储数据集 P 中与数据点 p 的距离不大于用户给定的查询距离阈值 r 的数据点 q 的标识以及点 q 与点 p 间的距离 $\mathrm{dist}(p,q)$;对于 $p.r$, 当 $p.nn$ 的大小为 k 时,$p.r$ 等于存储在 $p.nn$ 中最大的距离值;当 $p.nn$ 的大小小于 k 时,$p.r$ 的值被设为$+\infty$.

在离群点计算过程中, 根据点 p 的node 结构中 $p.r$ 的值与查询距离阈值 r 的大小关系即可判断当前数据点 p 是否为离群点. 例如, 若 $p.r$ 的值小于等于查询距离阈值 r,那么数据点 p 是非离群点;若 $p.nn$ 列表未存满,且 $p.r$ 取值为$+\infty$,则点 p 是离群点.

根据上述自定义的 node 结构,给出非离群点过滤定理.

定理 1 对于当前扫描到的数据集 P 中的数据点 q,如果它和内存中存储的数据点 p 之间的距离满足 $\mathrm{dist}(p,q)\leqslant r-p.r$,则数据点 q 为非离群点.

证 根据 $\mathrm{dist}(p,q)\leqslant r-p.r$ 可得 $p.r\leqslant r$,则说明内存中的数据点 p 为非离群点,且在 $p.r$ 范围内数据点p 的邻居个数至少为 k 个. 又因为数据点 p 和数据点 q 之间的距离满足 $\mathrm{dist}(p,q)+p.r\leqslant r$,则说明数据点$p$ 及其邻居都是数据点 q 的邻居,也即数据点 q 的邻居个数大于给定的数量阈值 k,因此根据基于距离的离群点定义可得数据点 q 为非离群点.

证毕.

根据索引中数据点的顺序及定理 1 中的过滤规则, 每个处理机通过扫描两次数据集可实现本地离群点的计算过程. 算法 2 给出了本地离群点检测的具体过程.

算法 2 本地离群点检测算法.

输入:查询距离阈值 r,查询数量阈值 k,本地存储的数据集 P;

输出:本地离群点集合

```
1. 初始化队列 H 为空;
2. FOR 数据集 P 中的每个数据点 p DO
3.   初始化 p 的信息,p.nn 为空,p.r 为+∞;
4.   IF H 为空 或者 IsNoInlier(p) THEN
5.     将数据点 p 加入到队列 H 中;
6.   ENDIF
7. ENDFOR
8. FOR H 中的每一个数据点 q DO
9.   将满足 q.r 不大于查询半径 r 的数据点从 H 中删除;
10. ENDFOR
11. FOR 数据集 P 中的每个数据点 p DO
12.   DeleteInlier(p);
13. ENDFOR
14. 队列 H 中的数据点全部都是离群点,输出;
```

第一次扫描数据集时, 需要将当前扫描到的不能确定是非离群点的数据点的 node 结构存储到内存中,以致于第一次扫描结束后,内存中至少存储着所有的本地离群点的信息. 由此,判定数据点 p 不是非离群点的 IsNoInlier(p)函数的算法如下:

```
IsNoInlier( )
输入:查询距离阈值 r,查询数量阈值 k,查询数据点 p,队列 H;
输出:数据点 p 的查询结果
1. FOR H 中的每一个数据点 q DO
2.   IF dist(p,q)≤r-p.r THEN
3.     数据点 p 为非离群点,返回 FALSE;
4.     BREAK;
5.   ENDIF
6.   IF dist(p,q)≤r THEN
7.     利用临时变量 oldr 记录数据点 q 的 q.r 值,并将 p 更新到 q 的
       q.nn 列表中;
8.     IF oldr>r 并且 q.r≤r THEN
9.       以 0.5 的概率将数据点 q 从 H 中删除;
10.    ENDIF
11.    将 q 更新到 p 的 p.nn 列表中;
12.    IF p.r≤r THEN
13.      数据点 p 为非离群点,返回 FALSE;
14.    ENDIF
15.  ENDIF
16. ENDFOR
17. IF p.r>r THEN
18.   返回 TRUE;
19. ENDIF
```

第二次扫描数据集时, 即从所有的候选本地离群点集合中找出真正的本地离群点,其中,删除内存中非离群点的 DeleteInlier(p)函数的算法如下:

```
DeleteInlier( )
输入:查询距离阈值 r,查询数量阈值 k,数据点 p,队列 H;
输出:队列 H
1. FOR H 中的每一个数据点 q DO
2.   IF dist(p,q)≤r THEN
3.     将 p 更新到 q 的 q.nn 列表中;
4.     IF q.r≤r THEN
5.       将数据点 q 从队列 H 中删除;
6.     ENDIF
7.   ENDIF
8. ENDFOR
```

总体上, 本小节基于距离的本地离群点计算方法可以通过扫描较少次数的数据集完成离群点的检测过程,在一定程度上减少了 IO 代价,同时,还可以很好地应对数据集中数据维数的增长.

复杂度分析:记本地数据集的大小为 N. 对于本地离群点的计算,构建索引所需时间复杂度为 $O(N\cdot \lg N)$,两次扫描数据集所需的时间复杂度为 $O(|H|\cdot N)$. 因此,算法 2 的时间复杂度为 $O(N\cdot \lg N+|H|\cdot N)$.

5.2 全局离群点计算

在完成 5.1 节的计算之后,本地离群点被发送到主控制节点形成离群点候选集, 此后由主控制节点判断其所需进行网络通信的处理机, 进而计算出真正的离群点.

对于给定的数据集 P, 利用 GDDP 划分方法对数据分布最均匀的二维空间(假设是 d_1 和 d_2 维所组成的空间)划分后所得的数据块 b 可表示为 $b=[b.\min,b.\max]$. 其中,$b.\min$、$b.\max$ 分别表示数据块 b 的下界和上界,则对于数据块 b 中的任意数据 p,都有 $b.\min[d_1]\leqslant p[d_1]\leqslant b.\max[d_1]$,$b.\min[d_2]\leqslant p[d_2]\leqslant b.\max[d_2]$. 进一步,有如下引理:

引理 1 [15] 对于数据块 b 中的任意本地离群点 p,如果 $\forall i\in\{d_1,d_2\}$,$p[d_1]-b.\min[d_1]>r$ 且 $b.\max[d_2]-p[d_2]\geqslant r$,则数据点 p 是全局离群点.

对于离群点候选集中满足引理 1 的数据点 p,可得出其一定是全局离群点, 而对于候选集中剩余的非确定的本地离群点, 需和其所在数据块的相邻数据块进行计算,以确定是否为真正的离群点.

定义 3 点与数据块的最小距离. 给定数据点 p 和数据块 b,则点 p 与块 b 的最小距离:

$$\min\_dist(p,b)=\sqrt{\sum_{i\in\{d_1,d_2\}}(z[i])^2} \tag{5}$$

其中:

$$z[i]=\begin{cases} b.\min[i]-p[i], & b.\min[i]>p[i] \\ p[i]-b.\max[i], & p[i]>b.\max[i] \\ 0, & \text{其他} \end{cases} \tag{6}$$

显然,对于块 b 的近邻块集合 $N(b)$中的块 b_i,若 $\min\_dist(p,b_i)\leqslant r$,说明块 b_i 中可能存在点 p 的近邻,则将点 p 发送到 S_b 集合中,最终将 S_b 中的数据点发送到块 b_i 所在的处理机上. 算法 3 展示了具体的全局离群点计算过程.

算法 3　全局离群点检测算法.

输入:数据块 b 中的本地离群点集合 L_b,数据块 b 的近邻块集合 $N(b)$,距离阈值 r,数量阈值 k;

输出:本地离群点集合 L_b 中的全局离群点

1. FOR $N(b)$中的每一个数据块 b_i DO
2.　初始化集合 S_b;
3.　FOR 本地离群点集合 L_b 中的每一个数据点 p DO
4.　　IF $\min\_dist(p,b_i)\leqslant r$ THEN
5.　　把数据点 p 加入到集合 S_b 中;
6.　　ENDIF
7.　ENDFOR
8. 将集合 S_b 发送到数据块 b_i 所在的节点上;
9. ENDFOR
10. FOR L_b 中的每个数据点 p DO
11.　根据各数据块返回的结果,计算数据点 p 的近邻数目 $nn(p)$;
12.　IF $nn(p)<k$ THEN
13.　　将数据点 p 加入到全局离群点集合中;
14.　ENDIF
15. ENDFOR

复杂度分析:对于全局离群点的计算,记本地候选离群点的个数为 η,根据公式(2)~(4)可将总数据集 P 划分为 b 个数据块, 将本地候选离群点与相邻数据块中的数据点依次计算,因此,算法 3 的时间复杂度为 $O(\eta\cdot(8/b)\cdot|P|)$. 不难发现,在分布式环境中,一个数据块的相邻块通常分布在不同的处理机上,因此可以并行执行离群点的计算, 使得处理效率大大提高.

6 实验分析

6.1 实验环境

本文实验所搭建的异构分布式计算系统共包含 5 台处理机,其中 1 台主控节点,4 台从节点,它们的具体配置如下.

主控制节点:Intel Core i3-7020U CPU,4GB 内存,500GB 硬盘,操作系统为 Windows 10;从节点 1:Intel Core i5-8265U CPU,8GB 内存,256GB 硬盘,操作系统为 Windows 10; 从节点 2:Intel Pentium(R) 2020M 2.4 GHz CPU,4GB 内存,500GB 硬盘,操作系统为 Windows 10; 从节点 3:Intel Core i3 2100 3.1GHz CPU,8GB 内存,500GB 硬盘, 操作系统为 Windows 10;从节点 4:Intel Core i3-6006U CPU,4GB 内存,125GB 硬盘,操作系统为 Windows 10.

本文分别用真实数据集和合成数据集对所提出的 GODA 算法与 PENL 算法、BOD 算法进行对比实验.

6.2 真实数据集中的实验结果分析

实验所用的真实数据集为森林覆盖数据集 Covertype,包含 581 012 条数据,每条数据有 10 个维度,每维属性值的范围为[0,100]. 表 4 和表 5 展示了 3 种算法在该数据集中的处理时间和通信量.

表 4 真实数据集中的处理时间

Tab.4 Processing time in real data sets

查询阈值	GODA 的处理时间/s	BOD 的处理时间/s	PENL 的处理时间/s
$k=6, r=12$	228	326	1 232
$k=6, r=16$	212	304	1 303
$k=8, r=16$	236	335	1 287
$k=10, r=16$	245	347	1 398

表 5 真实数据集中的通信量

Tab.5 Traffic in real data sets

查询阈值	GODA 的通信量/个	BOD 的通信量/个	PENL 的通信量/个
$k=6, r=12$	1 183	1 462	3 947 154
$k=6, r=16$	1 052	1 278	3 947 154
$k=8, r=16$	1 097	1 320	3 947 154
$k=10, r=16$	1 146	1 395	3 947 154

如表 4 和表 5 所示，本文提出的 GODA 算法采用了基于网格的动态数据划分方法 GDDP，可以根据各从节点的计算能力进行数据划分，实现异构分布式环境中各处理机的负载均衡，加快检测效率；同时，考虑点的空间位置，将相邻的点尽可能划分至同一节点，大大节省了网络开销. 相比之下，BOD 算法和 PENL 算法主要针对同构分布式环境，导致各节点数据分配不均衡，降低处理速度；PENL 算法采用基于数据量的划分方法，对于处理机的数据分配，都需与其他处理机上的数据集比较来确定最终结果，导致数据集重复传输，增加了网络开销.

另一方面，通过对比表 4 和表 5 中 1、2 行的实验结果，可以发现，当 k 不变，随着距离阈值 r 的增大，数据集中离群点数目逐渐减少，GODA 算法和 BOD 算法的处理时间和通信量也逐步减少. PENL 算法的处理时间与这两种算法趋势相同，而通信量维持不变. 这主要是因为 PENL 算法需要将所有处理机上的数据集进行两两之间的比较，网络传输量与处理机的数目和数据集的规模有关，与查询无关. 通过对比 3、4 行实验结果可以看出，随着查询阈值 k 的增大，离群点的数目变多，导致 GODA 算法的处理时间和通信量都随之增加. 但是，不管查询阈值 k 和距离阈值 r 如何变化，与 BOD 算法和 PENL 算法相比，本文提出的 GODA 算法在异构分布式环境中能更有效地进行离群点的检测.

6.3 人工合成数据集中的实验结果分析

由于真实数据集规模有限，数据维度也无法变化，所以本节使用人工合成的数据集对 3 种算法进行有效性验证. 首先，所使用的合成数据集为聚簇分布数据，各维取值范围为 0~10 000. 具体地，对于含有 n 个数据对象的数据集，随机生成 $n/10\,000$ 个聚簇点，每个聚簇内平均包含 1 000 个数据对象，并以聚簇点为中心呈高斯分布. 表 6 给出了实验相关变量的默认值和变化范围.

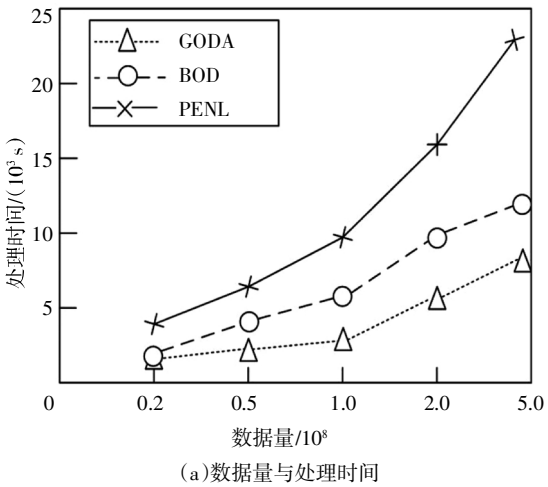
表 6 人工合成数据集中的参数设置

Tab.6 Parameter settings in a synthetic dataset

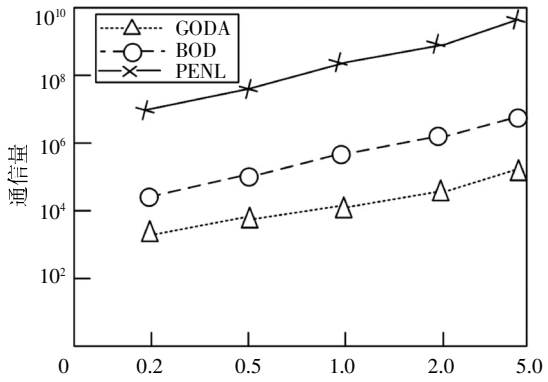
参数	默认值	变化范围
数据量	10^8	$0.2\times10^8 \sim 5\times10^8$
数据维度	10	8 ~ 16
查询数量阈值 k	50	40 ~ 60
查询距离阈值 r	16	14 ~ 18

图 4 描述了数据量对 3 种检测算法的影响. 随着数据量的增加，上述算法的处理时间都会变长，对应的通信量也随之变多. 但通过对比可以发现，GODA 算法和 BOD 算法的处理时间变化的速度较慢，而 PENL 算法的处理时间的增长趋势较快，也会增大网络开销. 由此说明 GODA 算法的查询效率要高于 BOD 算法和 PENL 算法的查询效率.

图 5 显示了数据维度对 3 种算法检测效果的影响. 在图 5(a)中，随着数据维度的升高，PENL 算法和 GODA 算法的处理时间呈现出线性变化，而 BOD 算法处理时间的增长趋势明显较快. 通过图 5 (b)可以发现，3 种算法的通信量都不受维度的影响.



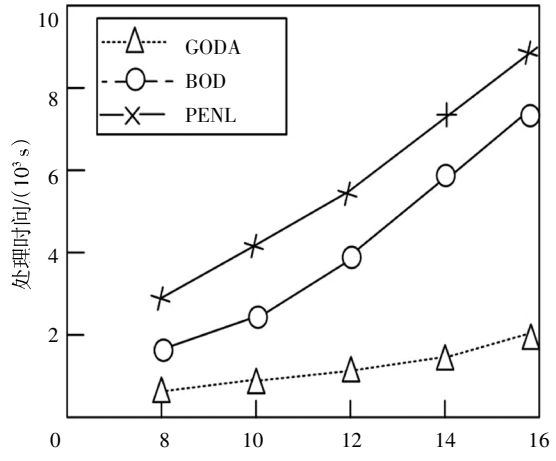
(a)数据量与处理时间



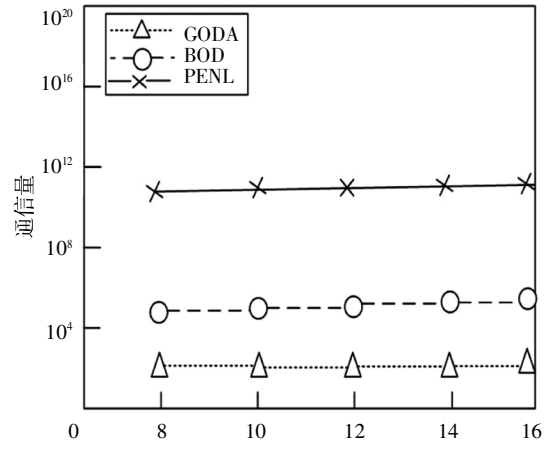
(b)数据量与通信量

图 4　数据量的影响

Fig.4　The effect of data volume



(a)维度与处理时间

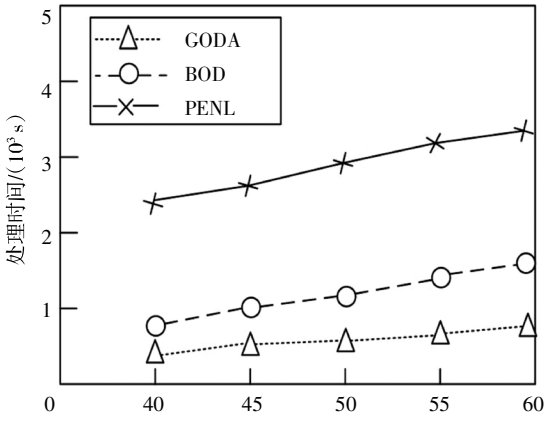


(b)维度与通信量

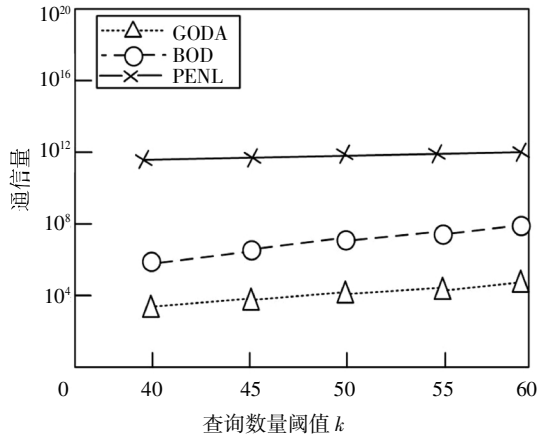
图 5　维度的影响

Fig.5　The effect of dimensionality

图 6 表述了查询阈值 k 的变化对 3 种算法的影响. 在图 6(a)中,查询阈值 k 增大时,3 种算法的处理时间都变长,这是因为 k 增大,数据集中离群点数目增多,增加了算法的处理时间. 但与 PENL 算法相比,由于 GODA 算法和 BOD 算法在本地计算时良好的过滤措施,使其处理时间小于 PENL 算法. 通过图 6(b)可得,随着查询阈值 k 增大,PENL 算法的通信量不变,而对于 GODA 算法和 BOD 算法,当 k 增大,使得数据集中离群点的数目增多, 从而使其通信量稍微增加.



(a)查询阈值 k 与处理时间



(b)查询阈值 k 与通信量

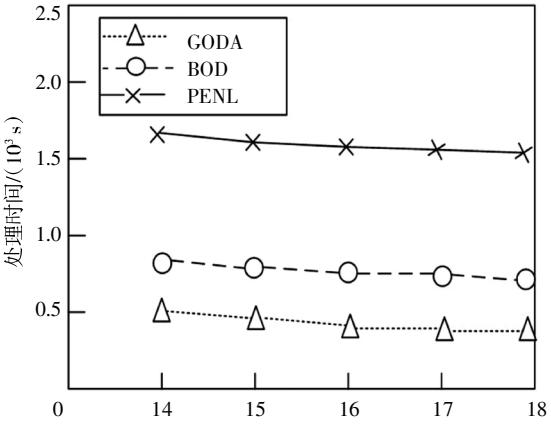
图 6　查询阈值 k 的影响

Fig.6　The effect of query threshold k

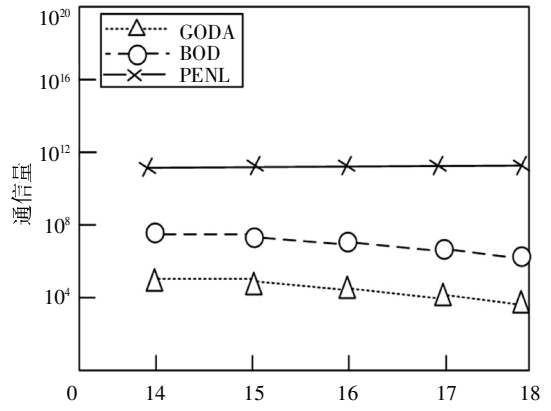
图 7 描述了距离阈值 r 的变化对 3 种算法性能的影响. 在图 7(a)中,当距离阈值 r 不断变大,数据集中离群点数目变少, 使得所有算法的处理时间都变短. 由图 7 (b) 可以发现, 随着距离阈值 r 增大, PENL 算法的通信量不变, 而对于 GODA 算法和 BOD 算法,由于 r 变大,数据集中离群点数目减少,使其通信量也稍微变少.

通过上述对比实验, 验证了本文提出的 GODA 算法的优秀性能. 与 PENL 算法和 BOD 算法相比, GODA 算法可以很好地解决异构分布式环境中基于

距离的离群点检测问题.



(a)距离阈值 r 与处理时间



(b)距离阈值 r 与通信量

图 7 距离阈值 r 的影响

Fig.7 The effect of distance threshold r

7 结 论

本文针对常见的异构分布式环境中的离群点检测问题,首先,提出有效的数据划分方法 GDDP,保证了各处理机计算能力的充分利用,并通过考虑数据点的空间位置信息减少网络通信. 其次,基于 GDDP 方法,提出了 GODA 算法,在每个处理机本地,通过构建索引进行数据管理,使用过滤规则进行批量过滤,加快本地离群点计算. 再次,通过异构分布式计算框架中各节点间的网络通信,计算出全局离群点. 最后,通过实验验证了 GDDP 和 GODA 算法的有效性.

参考文献

[1] KNORR E M, NG R T. Algorithms for mining distance-based outliers in large datasets [C]//Proceedings of the 24th International Conference on Very Large Data Bases. New York, USA, 1998: 392—403.

[2] 郑明玲,蒋句平,袁远,等. 一种面向大规模计算机的监控管理系统[J]. 湖南大学学报(自然科学版),2015,42(4):107—113.
ZHENG M L, JIANG J P, YUAN Y, *et al*. A monitoring and management system for the large-scale computer[J]. Journal of Hunan University(Natural Sciences), 2015, 42(4): 107—113. (In Chinese)

[3] ATKINSON A C, HAWKINS D M. Identification of outliers[J]. Biometrics, 1981, 37(4): 860.

[4] FITRIANTO A, RANA S, MIDI H, *et al*. Effects of a single outlier on the coefficient of determination: An empirical study[J]. Empowering the Applications of Statistical and Mathematical Sciences, 2015, 1643: 409—413.

[5] WANG X T, SHEN D R, BAI M, *et al*. Cluster-based outlier detection using unsupervised extreme learning machines[C]//Proceedings of ELM-2015 Volume 1. Berlin: Springer International Publishing, 2016: 135—146.

[6] BREUNIG M M, KRIEGEL H P, NG R T, *et al*. LOF: Identifying Density-Based Local Outliers [C]// Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data. Dallas, Texas, USA: ACM, 2000: 93—104.

[7] BAY S D, SCHWABACHER M. Mining distance-based outliers in near linear time with randomization and a simple pruning rule[C]// Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Texas, USA: ACM, 2003: 29—38.

[8] YANG P, HUANG B. An efficient outlier mining algorithm for large dataset [C]// International Conference on Information Management. IEEE Computer Society, 2008: 41—46.

[9] BUZZI-FERRARIS G , MANENTI F . Outlier detection in large data sets[J]. Computers & Chemical Engineering, 2011, 35(2): 388—390.

[10] HUNG E, CHEUNG D W. Parallel mining of outliers in large database [J]. Distributed and Parallel Databases, 2002, 12(1): 5—26.

[11] BAI M, WANG X T, XIN J C, *et al*. An efficient algorithm for distributed density-based outlier detection on big data[J]. Neurocomputing, 2016, 181: 19—28.

[12] YAN Y Z, CAO L, KULHMAN C, *et al*. Distributed local outlier detection in big data [C]//Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Halifax, Canada, 2017: 1225—1234.

[13] 王习特,申德荣,白梅,等. BOD:一种高效的分布式离群点检测算法[J]. 计算机学报,2016,39(1):36—51.
WANG X T, SHEN D R, BAI M, *et al*. BOD: An efficient algorithm for distributed outlier detection [J]. Chinese Journal of Computers, 2016, 39(1): 36—51. (In Chinese)

[14] ANGIULLI F, BASTA S, LODI S, *et al*. GPU strategies for distance-based outlier detection [J]. IEEE Transactions on Parallel & Distributed Systems, 2016, 27(11): 3256—3268.

[15] 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.
WANG S, WANG H J, QIN X P, *et al*. Architecting Big data: challenges, studies and forecasts [J]. Chinese Journal of Computers, 2011, 34(10): 1741—1752. (In Chinese)