

分布式数据流聚类算法及其基于 Storm 的实现

万新贵,李玲娟,马 可

(南京邮电大学 计算机学院,江苏 南京 210003)

摘 要:为了提高数据流聚类算法的效率,设计并提出了基于质心距离和密度网格的数据流聚类算法—CDD-Stream,并通过对其网格结构的更新实施了并行化策略,进而设计并提出了一种分布式数据流聚类算法—DCD-Stream (Distributed Centroid Distance D-Stream)。该算法分为在线和离线两个部分,在线部分实时接收数据流,利用局部节点和全局节点实现了网格结构更新的并行化,完成了整体网格结构的增量更新;离线部分基于网格结构的更新结果进行全局聚类,并存储网格帧,供用户查询历史簇。充分利用 Storm 快速实时处理数据流并显著提高数据流挖掘算法性能的优势,设计并实现了基于 Storm 的 DCD-Stream 算法实现方案。该方案通过内存数据库 Redis 和消息中间件 Kafka 的应用对 DCD-Stream 算法的拓扑进行了合理部署与实现。对比验证实验结果表明,相对于其他算法,DCD-Stream 算法在数据流对象上有相当高的聚类精度和更好的时效性,基于 Storm 的 DCD-Stream 算法实现方案是可行且有效的。

关键词:数据流聚类;分布式;质心距离;密度网格;Storm

中图分类号:TP311

文献标识码:A

文章编号:1673-629X(2017)07-0150-06

doi:10.3969/j.issn.1673-629X.2017.07.034

Distributed Data Stream Clustering Algorithm and Its Implementation with Storm

WAN Xin-gui, LI Ling-juan, MA Ke

(School of Computer, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: In order to improve the efficiency of data stream clustering algorithm, a data stream clustering algorithm based on centroid distance and density grid (named as CDD-Stream) has been designed and proposed, and a distributed data stream clustering algorithm DCD-Stream (Distributed Centroid Distance D-Stream) has been designed and proposed through adopting the parallelization strategy of updating grids into CDD-Stream algorithm. The algorithm has been divided into on-line part and off-line part. The online part is responsible for receiving data streams in real time and realizing the parallel updating of the grid structures by using local and global nodes. The off-line part finishes global clustering based on the updated results of grids, and stores grid frames which allows user to query the historical clusters. By making full use of Storm's fast real-time processing of data stream and improving the performance of data stream mining algorithm significantly, a scheme of implementing DCD-Stream algorithm on Storm platform has been designed and implemented. It uses memory database Redis and messaging middleware Kafka to deploy and realize the topology of DCD-Stream algorithm reasonably. The experimental results have shown that compared with other algorithm, DCD-Stream algorithm has considerable clustering quality and better clustering timeliness on data stream objects, and it is practical and effective for implementing DCD-Stream algorithm based on Storm.

Key words: data stream clustering; distributed; centroid distance; density grid; Storm

0 引言

随着物联网、移动互联网以及社交网络等的广泛应用^[1],数据量急速增长,变化速度也越来越快,数据流^[2]成为数据挖掘任务的新对象,数据流的处理方式也由传统的单节点处理逐渐转变为分布式^[3]处理。传

统的集中式数据挖掘已经不能适应这种分布式环境,针对分布式数据流挖掘的研究成为重点。

数据流聚类是数据流挖掘技术的一个重要分支^[4],经典的基于距离进行聚类的数据流聚类算法,如 C. C. Aggarwal 等提出的数据流聚类算法 Clustream^[5]

收稿日期:2016-07-21

修回日期:2016-11-04

网络出版时间:2017-04-28

基金项目:国家自然科学基金资助项目(61302158,61571238)

作者简介:万新贵(1991-),女,硕士研究生,CCF 会员(E200041361G),研究方向为流数据挖掘;李玲娟,教授,CCF 会员(E200015276M),研究方向为数据挖掘、信息安全、分布式计算。

网络出版地址: <http://cnki.net/kcms/detail/61.1450.TP.20170428.1703.076.html>

和 HPStream 算法^[6],均存在不能发现任意形状簇的问题;Cao F 等提出的 DenStream 算法^[7]很好地解决了该问题;Chen Y 等提出的基于密度与网格的数据流聚类算法 D-Stream^[8]显著提高了 DenStream 算法的效率。

针对 D-Stream 算法存在的一些缺陷,提出基于质心距离和密度网格的算法—CDD-Stream。该算法通过对参数设置、簇边界判定以及历史数据存储等方面的改进,有效提升了 D-Stream 算法的性能。为适用于分布式处理环境,基于 CDD-Stream 算法,进一步设计并提出了分布式数据流聚类算法—DCD-Stream。该算法分为在线和离线两个部分。在线部分负责实时处理数据流,并行化实现网格结构的增量更新;离线部分负责合并初始簇结构和最新网格结构,进行全局聚类,生成簇结构。

Storm^[9-10]作为一种典型的分布式数据流处理系统,与传统的批量数据处理系统相比,在实时性方面具有显著优势。由此可以得出,将分布式数据流挖掘算法部署到 Storm 上可以提高算法效率,取得更好的算法效果。基于 Storm 的编程模型,设计了 DCD-Stream 算法基于 Storm 的实现方案,通过使用内存数据库 Redis^[11]和消息中间件 Kafka^[12],完成了 DCD-Stream 算法在 Storm 上的部署与实现。

1 概述

1.1 相关概念

对涉及到的相关概念^[8-9]作如下解释:

(1)数据向量:假设数据流到达的时刻为 t ,则 t 时刻到达的 d 维数据向量表示为 $\mathbf{x}_t = (x_1, x_2, \dots, x_d)$ 。

(2)密度网格:给定 d 维空间 S ,将 S 按维度均匀划分为密度网格。每一维空间均被划分为 p 个部分,因此数据空间 S 共被划分为 $N = \prod p_i (i \in [1, d])$ 个密度网格。

(3)密度衰减系数:基于密度衰减系数 $\lambda \in (0, 1)$,网格单元的密度会随着时间不断衰减,衰减过程体现出数据流的变化特性。在时刻 t_0 到达的数据点 x 在时刻 t 的密度系数 $D(x, t)$ 为 λ^{t-t_0} 。

(4)网格单元密度和网格单元性质:一个网格单元的密度是由密度衰减系数和网格单元内数据点个数决定的;网格单元的密度决定了网格单元的性质,网格单元按性质被划分为:稠密网格、过渡网格和稀疏网格。

(5)特征向量:每个网格单元均采用一个 7 元组 $\langle t_g, t_m, D, \text{label}, \text{count}, \overrightarrow{\text{CF1}^x}, \text{status} \rangle$ 作为该网格单元的特征向量,用于存储概要数据结构。其中, t_g 是网格单元 g 更新数据后时间; t_m 是网格单元 g 被作为松

散网格从 grid_list 中移除的最后时间; D 是网格单元 g 最终更新的密度; label 是网格单元 g 的类(簇)标签; count 则记录了网格单元 g 的数据点总数; $\overrightarrow{\text{CF1}^x}$ 记录了所有数据点向量的代数和; status 标签用于移除松散网格。

(6)网格组:网格单元的集合。

(7)网格簇:如果网格组 G 内部每一个网格都是稠密网格且每个外部网格或者是一个稠密网格或者是一个过渡网格,那么 G 就是一个网格簇。

(8)质心距离:两个邻接网格单元 g_i, g_j 的质心 C_i, C_j 之间的欧氏距离称为网格的质心距离。

(9)加权平均密度:一个网格单元的数据量与网格组总的数据量的比值为该网格单元的密度权值,该密度权值与网格单元密度的乘积构成网格单元的加权平均密度。

(10)时间间隔 gap :网格簇的调整周期。随着网格单元对数据流的持续接收,网格单元的性质会发生变化,网格簇的结构需要按周期调整,合理的调整周期可以有效提升算法效率。

1.2 CDD-Stream 算法设计

CDD-Stream 算法总体分为在线、离线以及历史簇查询三个部分,算法流程如图 1 所示。

在线部分负责处理数据流,更新网格单元的特征向量,生成初始化簇结构并在此基础上按周期检测和删除松散网格。在该部分,CDD-Stream 算法通过计算加权平均密度,实现了网格单元参数的动态设置,解决了因为先验知识缺乏导致算法参数设置不合理的问题。

离线部分负责按周期调整簇结构,同时基于金字塔时间模型存储网格帧,为历史簇查询部分提供依据。在调整簇结构阶段,基于网格单元的质心距离的簇边界判定算法,在一定程度上避免了由于网格单元的误删导致簇边界的丢失,提高了算法的聚类精度。

在历史簇查询部分,用户可以根据实际需求,设置查询时间段,基于离线部分存储的网格帧进行历史簇查询。

1.3 Storm 系统

Storm 最初是由 Back Type 公司研发的, Twitter 公司于 2011 年将其开源。与批量数据处理系统如 Hadoop 不同, Storm 是一个分布式的可靠、可容错和可扩展的流式数据实时处理系统,可以用来实时处理新数据和更新数据库;与一般流式数据处理系统如 S4^[13]相比, Storm 在消息处理反馈和记录完全处理这两方面有新的突破,并且 Storm 采用弱中心化的结构^[10]进行任务分配,该模式显著降低了节点之间通信和同步的代价。

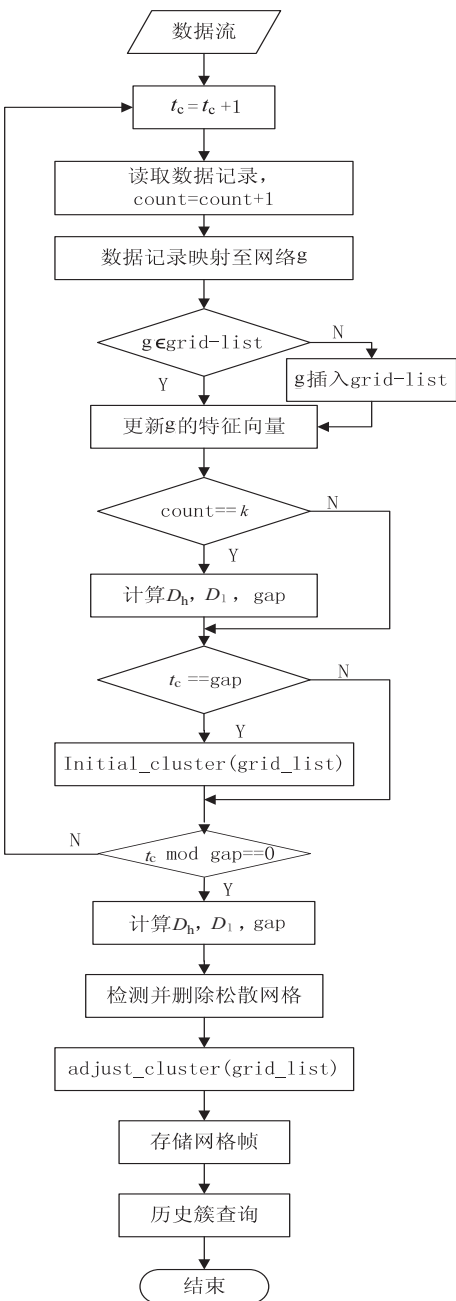


图 1 CDD-Stream 算法流程图

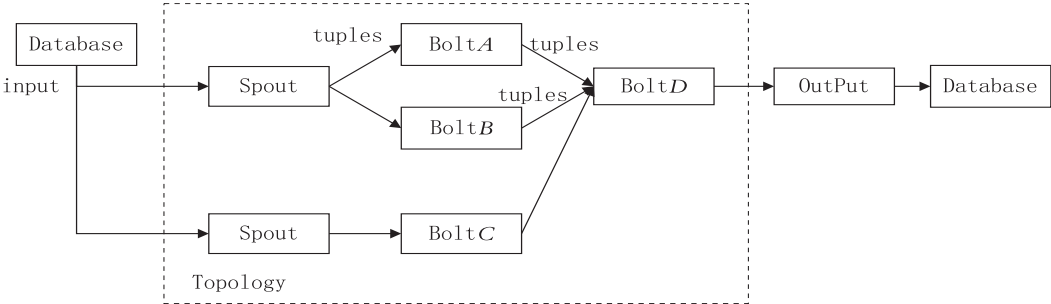


图 2 Storm 集群的拓扑

Storm 集群任务的完成主要依靠各种不同的组件,各组件负责不同的任务,Spout 组件负责集群输入流的处理,Bolt 组件接收 Spout 传递过来的数据流,并以指定方式处理。Storm 集群可以看作是一系列组件构成的有向无环图,称为拓扑 (Topology),拓扑一旦提交不能自动终止^[10]。Storm 集群的拓扑如图 2 所示。

Storm 集群采用弱中心化主从结构,集群分为三类节点:Zookeeper 节点、主节点 Nimbus 和从节点 Supervisor。Zookeeper 节点负责与 Storm 集群交互协作;Nimbus 节点负责提交任务,通过 Zookeeper 节点向工作节点指派任务和状态监控;Supervisor 节点负责启动 Worker 进程,执行拓扑任务。

2 分布式数据流聚类算法—DCD-Stream

2.1 算法的分布式思想

DCD-Stream 分为在线和离线两个部分。在线部分实时接收数据流,利用局部节点和全局节点实现了网格结构更新的并行化,实现整体网格结构的增量更新;离线部分基于网格结构的更新结果进行全局聚类,并存储网格帧,供用户查询历史簇。算法的总体处理过程如图 3 所示。

预处理样本产生初始参数,局部网格维护节点负责接收数据流,全局网格维护节点合并局部网格节点的网格结构,完成网格结构的增量更新。当满足时间间隔 gap 时,全局网格维护节点将整体网格结构发送至初始化节点以及计算节点;合并节点接收网格增量结果,并将其与已有的簇结构进行合并生成最新簇结构;基于最新簇结构,根据金字塔时间模型存储网格帧,提供后期进行历史簇查询的依据。

2.2 网格结构更新的并行化策略

网格结构的在线增量更新任务由多个局部网格节点水平并行化执行,即数据流被平均发送至各局部网格维护节点,当局部网格维护节点的数据量满足一定

条件时,局部网格维护节点将各自的网格结构发送至全局网格维护节点。

分布式环境要求算法在不影响算法准确度的情况下尽量减少节点之间的通信,局部网格维护节点的

合并条件直接影响了算法的通信消耗。考虑到全局网格维护节点是以时间间隔 gap 为周期发送网格结构的,因此局部网格维护节点也按周期 w 发送网格结构至全局网格维护节点。w 的大小设置既要保证网络结

构的更新结果被及时发送,又要尽可能降低各节点之间的通信消耗,因此基于簇结构调整周期 gap 定义局部网络维护节点发送周期 w 。具体定义如下:

$$w = (gap/2) * v * 1s$$

其中, v 为数据流的流速。

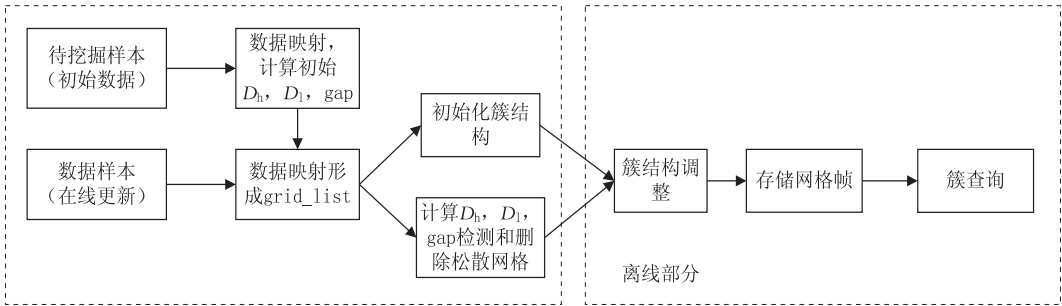


图3 DCD-Stream 算法模块划分

局部网络维护节点的伪代码如算法 1 所示。

算法 1:局部网络结构在线增量更新算法。

输入:数据流 DS , 初始参数;

输出:局部网络结构 $G' = \{g_1, g_2, \dots, g_m\}$, 网格 g 的特征向量为 $\langle t_g, t_m, D, label, count, CFI^+, status \rangle$ 。

```
1:while 数据流未结束 do
2:读取数据记录  $x = (x_1, x_2, \dots, x_d)$ ,  $count++$ 
3:计算该数据点对应网格  $g$  的 id, 将数据映射至网格  $g$ 
4:if  $g \notin grid\_list$ 
5:将  $g$  插入  $grid\_list$ , 更新网格单元  $g$  的特征向量
6:end if
7:if count 为  $w$  的倍数
8:向全局节点发送  $G'$ 
9:end if
10:end while
```

全局网络维护节点的伪代码如算法 2 所示。

算法 2:全局网络结构合并算法。

输入: n 个局部网络结构 $\{G_1, G_2, \dots, G_n\}$;

输出:全局网络结构 G 。

```
1:接收局部网络结构  $\{G_1, G_2, \dots, G_n\}$ 
2:以特征向量中的  $t_g$  为依据, 在所有网络结构中选取距离当前时间最远的网格单元  $g$ 
3:遍历所有网格单元
4:if 存在网格组  $p = (p_1, p_2, \dots, p_i)$  与所选网格单元  $g$  的 id 相同
5:以  $t_g$  为基准, 网格单元  $g, p_1, p_2, \dots, p_i$  分别作为密度个体, 进行密度衰减计算, 计算过程与数据点的计算过程相似
6:得出最终合并网格  $p'$  的密度, 将网格  $p'$  插入  $G$ 
7:else
  直接将网格单元插入  $G$ 
8:end if
```

2.3 DCD-Stream 算法基于 Storm 的实现方案

DCD-Stream 算法在 Storm 集群上的拓扑如图 4 所示。

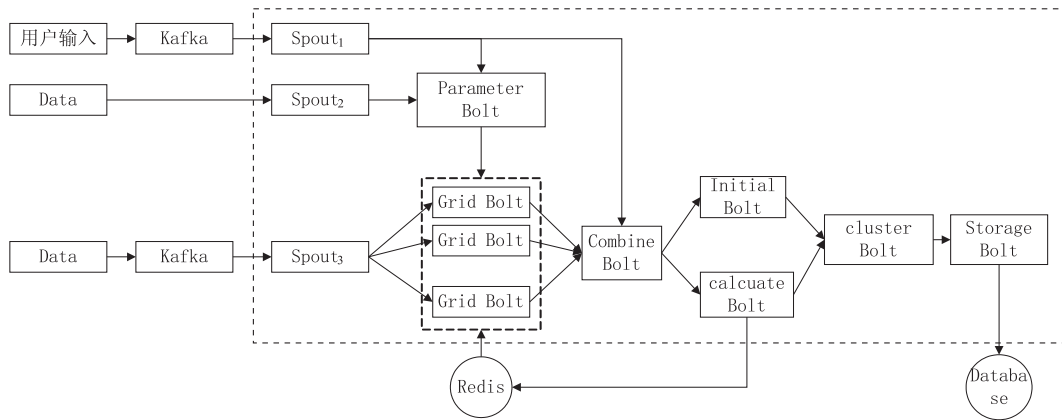


图4 DCD-Stream 算法在 Storm 上的拓扑

方案使用 Redis 存储中间结果,使用 Kafka 作为消息中间件传递消息。

Redis 是一种高性能的键值存储系统,用于缓存或存储数据。有别于传统数据库,Redis 存储的数据均在内存中,磁盘仅用于持久化操作;相对于类似的内存数据库如 Memcached,Redis 提供了更丰富的数据类型,它的另一特点是主从结构的支持,主服务器的数据

可以被复制到多个从服务器,有效地实现了读写分离。

Kafka 最初是由 LinkedIn 公司开发的,之后成为 Apache 的子项目,它是一种分布式发布订阅消息系统。相对于传统的企业消息系统,Kafka 的高吞吐量、持久化机制、分布式结构、自平衡机制以及多场景支持决定了它在大数据处理系统中的重要地位。Kafka 根据 Topic 对消息进行归类,Producer(消息发送者)和

Consumer(消息接收者)负责实现 Kafka 注册的接口。数据从 Producer 发送到 Broker, Broker 再分发注册到系统中的 Consumer, 其中 Broker 的作用类似于缓存。

Spout₁ 和 Spout₂ 分别接收用户输入和用于计算初始化参数的数据, 并将其发送至 Parameter Bolt 进行初始化参数的计算, Spout₁ 同时发送至 Combine Bolt 为网格的合并提供参数; Spout₃ 从 Kafka 中接收待挖掘的数据流, 将其发送至所有 Grid Bolt 进行网格结构的在线增量更新。

Parameter Bolt 完成初始参数的计算, 并将其发送至各 Grid Bolt, 为网格特征向量的更新提供依据。

Grid Bolt 接收 Spout₃ 的待挖掘数据流, 从 Redis 中读取最新的算法参数和最新的网格结构, 将数据流映射至相应网格单元, 同时更新网格单元的特征向量, 实现网格结构的在线增量更新, 并且按周期发送网格结构至 Combine Bolt。

Combine Bolt 负责合并所有 Grid Bolt 发送的网格结构, 以时间间隔 gap 值为依据发送总体网格结构至 Initial Bolt 和 Calculate Bolt, 用于完成簇结构的初始化和算法参数的动态更新。

Initial Bolt 在第一个 gap 周期完成簇结构的初始化, 并将结果发送至 Cluster Bolt。

Calculate Bolt 接收来自 Combine Bolt 的整体网格结构, 计算最新的算法参数, 并且检测删除松散网格, 形成最新的网格结构, 将其存入 Redis。

Cluster Bolt 为合并节点, 基于 Calculate Bolt 提供的最新网格结构和 Initial Bolt 提供的初始化簇结构进行聚类, 更新簇结构, 结果发送至 Storage Bolt。

Storage Bolt 基于金字塔模型存储簇结构的概要信息, 为历史簇查询提供依据。

3 实验结果及分析

3.1 实验环境与数据集

为了验证 DCD-Stream 算法的正确性和时效性, 以及该算法基于 Storm 实现方案的可行性和有效性, 设计了基于单机环境和集群环境的对比实验, 测试了 DCD-Stream 算法在两种环境下的精确度, 以及不同环境相同并行度下的效率。

算法的实验环境如下:

单机环境: eclipse_4. 5. 0、JRE1. 7. 0_13、Windows7、2. 13 GHz、6 GB 内存。

集群环境: 虚拟机模拟 1 个 Nimbus 节点, 2 个 Supervisor 节点, 操作系统 centos6. 4、JRE1. 7. 0_13、Zookeeper-3. 4. 6、Storm 小 0. 9. 1、Kafka2. 8. 1、redis-2. 4. 5。

数据集为数据入侵检测数据集 KDD-CUP99, 该数

据集^[14]共有 41 维属性, 其中 34 维连续属性, 7 维离散属性, 实验选取其 10% 样本集作为测试数据(采用 34 维连续属性), 共有数据记录 311 029 条。所有的实验数据都进行了归一化处理。

3.2 精确度测试及分析

为了测试 DCD-Stream 算法的聚类精度是否受线程数以及物理节点数的影响, 对比实验分为: 单机环境下的 CDD-Stream 算法测试、单机环境线程数为 2 的 DCD-Stream 算法测试、Storm 集群环境线程数分别为 2 和 4 的 DCD-Stream 算法测试。其中, 不同线程针对的是 Grid Bolt 的线程数。对于以上所有测试情况, 以数据集大小不同为基准测试 5 次取其平均值。图 5 为对算法精确度的测试结果。

由图 5 可以看出, 单机多线程 DCD-Stream 算法、集群多线程 DCD-Stream 算法的精确度与 CDD-Stream 算法相差不大, 只有小幅波动, 精确度基本保持不变, 说明算法本身的精确度不受分布式处理环境的影响。

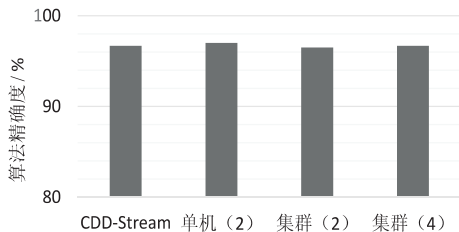


图 5 算法精确度测试结果

3.3 效率测试及分析

在精确度测试的基础上, 设计了对比实验进行 DCD-Stream 算法的效率测试。将单机环境与集群环境下算法的 Grid Bolt 线程数均设置为 4, 测试不同数据量所需的处理时间, 结果如图 6 所示。

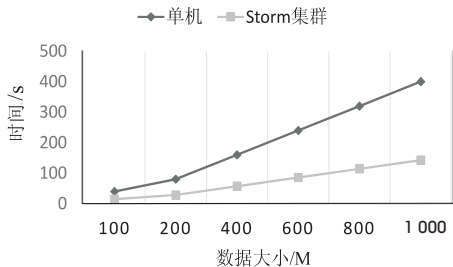


图 6 算法效率测试结果

由图 6 可以看出, 随着数据量的增加, 单机环境下算法运行时间的增长幅度明显高于集群环境, 由此可以得出, DCD-Stream 算法基于 Storm 的实现是可行的, 且在算法时效性方面具有更显著的提升。

4 结束语

为更好地解决分布式数据流聚类问题, 基于改进的密度和网格聚类算法—CDD-Stream, 提出了网格结构更新的并行化策略, 设计并提出了分布式数据流聚

类算法—DCD—Stream,在保证算法聚类精度的情况下提高了算法效率。基于 Storm 的编程模型,设计了基于 Storm 的 DCD—Stream 算法实现方案,通过使用内存数据库 Redis 和消息中间件 Kafka,完成了 DCD—Stream 算法拓扑的部署和实现。基于 KDDCUP99 数据集的对比实验结果及其分析表明,DCD—Stream 算法在数据流对象上具有稳定的聚类精度和更好的时效性,基于 Storm 的 DCD—Stream 算法实现方案是可行且有效的。

参考文献:

[1] Wang Y Z, Jin X L, Cheng X Q. Network big data: present and future[J]. Chinese Journal of Computers, 2013, 36(6): 1125–1138.

[2] Babcock B, Babu S, Datar M, et al. Models and issues in data stream systems [C]//Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems. [s. l.]: ACM, 2002: 1–16.

[3] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107–113.

[4] Huang Lei. Mining stream data: a survey[J]. Journal of Software, 2004, 15(1): 1–7.

[5] Aggarwal C C, Han J, Wang J, et al. A framework for clustering evolving data streams[C]//Proceedings of the 29th international conference on very large data bases. [s. l.]: [s. n.], 2003: 81–92.

(上接第 149 页)

提升了贸易信用度。

参考文献:

[1] 鲁旭. 基于跨境供应链整合的第三方物流海外仓建设[J]. 中国流通经济, 2016, 30(3): 32–38.

[2] 李向阳. 促进跨境电子商务物流发展的路径[J]. 中国流通经济, 2014, 28(10): 107–112.

[3] 曹旭光, 王金光, 刘希全. 跨境电子商务的物流商业模式及其创新途径[J]. 对外经贸实务, 2015(10): 93–96.

[4] 何影. 2016 进口跨境电商行业发展趋势分析[EB/OL]. 2016. <http://mt.sohu.com/20160515/n449517962.shtml>.

[5] 吕赛. 基于复杂适应性系统的众包翻译平台的模型与仿真[J]. 计算机系统应用, 2015, 24(2): 7–13.

[6] Ghadiyaram D, Bovik A C. Massive online crowdsourced study of subjective and objective picture quality[J]. IEEE Transactions on Image Processing, 2016, 25(1): 372–387.

[7] Zheng Haichao, Li Dahui, Hou Wenhua. Task design, motivation, and participation in crowdsourcing contests[J]. International Journal of Electronic Commerce, 2014, 15(15): 57–88.

[6] Aggarwal C C, Han J, Wang J, et al. A framework for projected clustering of high dimensional data streams[C]//Proceedings of the thirtieth international conference on very large data bases. [s. l.]: [s. n.], 2004: 852–863.

[7] Cao F, Ester M, Qian W, et al. Density-based clustering over an evolving data stream with noise[C]//SIAM international conference on data mining. [s. l.]: [s. n.], 2006: 328–339.

[8] Chen Y, Tu L. Density-based clustering for real-time stream data[C]//Proceedings of the 13th ACM SIGKDD international conference on knowledge discovery and data mining. [s. l.]: ACM, 2007: 133–142.

[9] Storm, distributed and fault-tolerant realtime computation [EB/OL]. 2011. <http://storm.apache.org/>.

[10] Leibiusky J, Eisbruch G, Simonassi D. Getting started with storm[M]. [s. l.]: O'Reilly Media, Inc., 2012.

[11] 曾泉匀. 基于 Redis 的分布式消息服务的设计与实现[D]. 北京: 北京邮电大学, 2014.

[12] ApacheKafka: a high-throughput, distributed, publish-subscribe messaging system [EB/OL]. 2014. <http://kafka.apache.org/>.

[13] Neumeyer L, Robbins B, Nair A, et al. S4: distributed stream computing platform [C]//IEEE international conference on data mining workshops. [s. l.]: IEEE, 2010: 170–177.

[14] 王洁松, 张小飞. KDDCup99 网络入侵检测数据的分析和预处理[J]. 科技信息, 2008(15): 409–410.

[8] Dissanayake I, Zhang J, Gu B. Task division for team success in crowdsourcing contests: resource allocation and alignment effects[J]. Journal of Management Information Systems, 2015, 32(2): 8–39.

[9] 叶晨, 王宏志, 周小田, 等. 基于众包的电子商务数据实体分类系统[J]. 计算机研究与发展, 2013, 50(S): 405–409.

[10] 张利娜, 郑桂玲. Web2.0 环境下电子商务“众包式零售”模式探讨[J]. 电子商务, 2013(11): 54–55.

[11] Zhou J H, Management S O. The research of fresh food and e-commerce industry under the crowdsourcing logistics pattern[J]. Science Technology & Industry, 2016, 16(5): 33–35.

[12] Chen C, Pan S, Wang Z, et al. Using taxis to collect citywide E-commerce reverse flows: a crowdsourcing solution[J]. International Journal of Production Research, 2016(8): 1–12.

[13] 孙洋洋. 基于众包模式的档案馆信息资源协同共建研究[J]. 浙江档案, 2015(11): 17–21.

[14] Swahney N S. The globe brain [M]. Philadelphia: Wharton School Publishing, 2007: 50.