

Research on Dynamic Clustering Algorithm Based on Spark Framework

Botao Zhang, Jianhua Li, Lei Fan

School of Information Security Engineering, Shanghai Jiao Tong University, Shanghai

Email: retropoktan@sjtu.edu.cn, lijh888@sjtu.edu.cn, fanlei@sjtu.edu.cn

Received: Nov. 4th, 2016; accepted: Nov. 21st, 2016; published: Nov. 24th, 2016

Copyright © 2016 by authors and Hans Publishers Inc.

This work is licensed under the Creative Commons Attribution International License (CC BY).

<http://creativecommons.org/licenses/by/4.0/>



Open Access

Abstract

In the era of big data, with the rapid growth of data size, the requirements of data processing increase constantly. It has put forward many effective algorithms for data stream clustering these years. However, with the continuous development of social technology, single machine environment has been difficult to meet the needs of data mining. Cluster environment is used more for information collection and data processing, the traditional clustering algorithm does not adapt well to the new processing requirements. This paper made some improvements from the data stream clustering algorithm D-Stream, used the big data processing framework Spark and designed a dynamic data clustering algorithm PDStream based on distributed architecture. The new algorithm is proved to be more efficient and able to perform dynamic clustering tasks under distributed architecture from the results of experiment.

Keywords

D-Stream, PDStream, Spark, Dynamic Clustering Algorithm

基于Spark的动态聚类算法研究

张伯涛, 李建华, 范磊

上海交通大学信息安全工程学院, 上海

Email: retropoktan@sjtu.edu.cn, lijh888@sjtu.edu.cn, fanlei@sjtu.edu.cn

收稿日期: 2016年11月4日; 录用日期: 2016年11月21日; 发布日期: 2016年11月24日

摘 要

针对数据流的聚类算法,近年来取得了有效的进展,出现了许多卓有成效的算法。随着信息采集技术的进步,需要处理的数据量越来越大,需要研究针对数据流的并行聚类算法。本文基于串行的数据流聚类算法D-Stream作出并行化改进,用通用的大数据处理框架Spark设计了一个基于分布式架构运行的动态数据聚类算法PDStream。实验结果表明,该算法具有更高的效率和良好的扩展性,能够实现分布式架构下的流数据动态聚类。

关键词

D-Stream, PDStream, Spark, 动态聚类算法

1. 引言

当今社会信息化的发展,在生物工程、新能源、企业医疗等新领域出现了海量的数据。由于这些数据持续不断产生,随着时间动态无限增长,近似呈现一种“流”的形态,因此此类数据被称为流数据。针对流数据进行数据挖掘是当前的热门课题。聚类作为数据挖掘领域的一个重要分支,一直以来倍受关注并得到了学者们广泛、深入的研究。它作为一种无监督的学习方式,在数据挖掘过程中能根据数据间的相似性将待处理对象划分为一个或若干个簇,在网络入侵检测、挖掘潜在客户和分析市场情况等方面具有重要应用。

针对流数据的聚类分析因其诸多特点变得相对困难,主要体现在三个方面:1) 低时空复杂度。数据流源源不断地产生且高速到达,它的海量、无限性要求算法不能够存储全部数据,每个数据项的处理也不能花费过长时间,算法在线处理速度应尽量和数据流的流速相匹配。2) 增量的处理新数据。由于流数据数量巨大,数据量未知,因此近乎不可能重复计算原有数据,算法必须能够在已有结果基础上增量处理新的数据。3) 正确、快速的处理离群点。数据流具有不断演化的特性,随着时间的推移数据流可能会不断地改变趋势,算法应能够清晰、快速地识别离群点,且对离群点做出正确的处理。

近年来,已经有许多优秀的流数据聚类算法诞生,他们有各自的处理要求和特点。CluStream [1]算法是一个基于层次的流数据聚类算法,它首次将数据流的处理框架分为在线层和离线层,并引入聚类特征向量来表示聚簇,且采用预先定义的距离阈值来衡量数据项是否属于已知类。DenStream [2]是一个基于密度的流数据聚类算法,该算法将满足一定密度阈值的数据点划分为一个簇,且能够发现球状簇,但该算法对输入参数敏感,参数的变化将严重影响聚类结果,且聚类过程中需要对每个数据对象进行邻域检查,因而计算时间复杂度较大。D-Stream [3]是一个基于网格密度的算法,它将空间划分成一个个离散的网格,将进入系统中的数据映射到对应的网格中,然后操纵这些网格的信息,并将密度较大的相邻网格合并为一个聚簇。DENGRIS-Stream [4]同样是一个基于网格密度的流数据聚类算法,但是它额外加入了滑动窗口,使得聚簇结果消除了历史数据的影响。ExCC [5]是一个针对混合属性设计的算法,属性中包含数字和非数字,它对相邻网格的概念进行了重新定义,使得它适应非数字属性,并且依据数据流速的不同来决定消除历史数据的规模。FlockStream [6]是一个基于密度的算法,它基于群体智能[7],给每个输入数据指定一个代理,代理在一个提前定义好的虚拟空间中移动,当两个代理相遇则将他们合并为一个聚簇,该算法合并了在线层和离线层,在任何时间都可能生成新的聚簇。以上这些流聚类算法有各自适用的环境,在实际运用中需要针对业务的特点合理进行选择。

随着集群技术、并行计算与分布式计算的发展, Hadoop [8] 分布式计算框架和 Spark [9] 内存计算框架等的广泛应用, 为解决大规模流数据的实时挖掘带来了曙光。由于传统流聚类算法大都不适用分布式环境, 本文提出了一种 PDStream 算法, 它基于 Spark 中的 Spark Streaming [10] 模块, 对传统的数据流聚类算法 D-Stream 做出了相关改进, 它保持了 D-Stream 算法的优越性, 并在此基础上能够适应分布式环境。

2. 相关工作

2.1. 基于网格的流式聚类算法

假定数据空间 S 包含 d 个维度, 则 $S = S_1 \times S_2 \times \cdots \times S_d$, 其中 S_i 表示第 i 维子空间。在 D-Stream 算法中, 数据空间 S 划分为一个个网格, 假定每个子空间 S_i 被划分为 P_i 个部分, 即 $S_i = S_{i,1} \cup S_{i,2} \cup \cdots \cup S_{i,P_i}$, 则整个数据空间 S 可以划分的网格数为: $N = \prod_{i=1}^d P_i$ 。

其中, 网格 g 由 $S_{1,j_1} \times S_{2,j_2} \times \cdots \times S_{d,j_d}$ 组成, $j_i = 1, \dots, P_i$, 网格 g 坐标定义为: $g = (j_1, j_2, \dots, j_d)$ 。

数据流中的数据 $x_1, x_2, \dots, x_k, \dots$ 分别在时间 $t_1, t_2, \dots, t_k, \dots$ 进入系统, 数据 x_i 的时间戳为 $T(x_i) = t_i$ 。其中每一个数据 $x_i = (x_{i,1}, x_{i,2}, \dots, x_{i,d})$ 包含 d 维数据, 则数据 x 可以映射到网格 $g(x)$ 中, 即

$$g(x) = (j_1, j_2, \dots, j_d), x_i \in S_{i,j_i}$$

随着时间的推进, 数据流输入的数据规模将非常庞大, 历史数据的影响力依然存在。假定新数据的聚类更需要关心, 历史数据的影响力需要逐渐变小, 则有如下定义。

定义 2.1: 定义衰减系数 $\lambda (0 < \lambda < 1)$, 对于数据 x , 它在时间 $T(x) = t_c$ 进入系统, 则它在时间 t 的密度系数为

$$D(x, t) = \lambda^{t-t_c}$$

每个网格 g 在时间 t 对应的密度系数为 g 包含的所有数据 x_i 的密度系数之和, 即

$$D(g, t) = \sum_{x \in g} D(x, t)$$

定义 2.1 定义了网格 g 关于时间 t 的密度系数。对于每个时刻 t , 如果网格 g 没有接受新的数据, 则它的密度系数会进行衰减。如果网格 g 最后一次在 t_i 时刻接收了数据并更新了密度系数, 在 t_n 时刻 ($t_n > t_i$) 再次接收了一个数据, 则密度系数 $D(g, t_n)$ 可被更新为

$$D(g, t_n) = \lambda^{t_n-t_i} D(g, t_i) + 1$$

这样仅在网格接收新数据的时刻更新网格密度系数, 节省了大量的计算时间, 提高了效率。有了密度系数, 对网格 g 的密度进行划分是基于以下的观察:

命题 2.1 让从 0 时刻到 t 时刻到达的所有数据记录成为一个集合, 则:

$$1) \sum_{x \in X(t)} D(x, t) \leq \frac{1}{1-\lambda}, t = 1, 2, \dots$$

$$2) \lim_{t \rightarrow \infty} \sum_{x \in X(t)} D(x, t) = \frac{1}{1-\lambda}$$

命题 2.1 表明系统中所有数据记录的密度总和永远不会超过 $\frac{1}{1-\lambda}$ 。由于存在 $N = \prod_{i=1}^d P_i$ 个网格, 因此每个网格的平均密度不会超过 $\frac{1}{N(1-\lambda)}$ 。基于这个观察, 有以下定义:

定义 2.2 在时刻 t , 如果网格 g 的密度系数满足:

- 1) $D(g, t) \geq \frac{C_m}{N(1-\lambda)} = D_m$, 则网格 g 为稠密网格, 它的 status 标签设为 dense。其中, $C_m > 1$ 。
- 2) $D(g, t) \leq \frac{C_l}{N(1-\lambda)} = D_l$, 则网格 g 为稀疏网格, 它的 status 标签设为 sparse。其中, $0 < C_l < 1$ 。
- 3) $\frac{C_l}{N(1-\lambda)} \leq D(g, t) \leq \frac{C_m}{N(1-\lambda)}$, 则网格 g 为过渡网格, 它的 status 标签设为 transitional。

对网格的密度进行定义后, 相邻的密度较大的网格将被合并到一个簇中, 对于相邻网格有如下定义:

定义 2.3 考虑两个网格 $g_1 = (j_1^1, j_2^1, \dots, j_d^1)$ 和 $g_2 = (j_1^2, j_2^2, \dots, j_d^2)$, 如果存在 k , $1 \leq k \leq d$, 使得:

- 1) $j_i^1 = j_i^2, i = 1, \dots, k-1, k+1, \dots, d$,
- 2) $|j_k^1 - j_k^2| = 1$

那么 g_1 和 g_2 被称为相邻网格, 他们在第 k 维上相邻。

2.2. D-Stream 算法框架

D-Stream 是一种典型的基于网格密度的聚类算法, 它把空间区域 S 划分为一个个离散的网格, 将新来的数据映射到对应的网格。这样不需要关心原始数据, 只要对网格特征向量数据进行相关操作, 最终得出聚类结果。D-Stream 算法框架分为在线部分和离线部分, 在线部分用于接收数据流, 将数据对应的网格信息进行更新; 离线部分定期取出在线部分的数据, 通过聚类算法进行聚类 and 调整, 最终将结果输出。

D-Stream 在线部分负责接收数据流, 更新数据所在网格的特征向量。对于每个网格的当前状态, 在 D-Stream 算法中, 有如下特征向量进行表示:

定义 2.4 网格 g 的特征向量为一个四元组: $\{t_i, D, \text{status}, \text{label}\}$, 代表含义分别为:

- t_i : 网格 g 最后接收到数据的时间。
- D : 即为 $D(g, t_i)$ 的值, 记录网格 g 最后更新后的密度系数
- status: 表示网格 g 密度的划分, 取值范围是 $\{\text{sparse}, \text{transitional}, \text{dense}\}$
- label: 表示网格的聚簇标签

每当一个新数据 x 在 t_n 进入系统时, 首先找到 x 所属的网格 g , 并更新网格 g 加入数据 x 后的特征向量: 记录数据 x 的时间戳为 t_n , 并依据 $D(g, t_n) = \lambda^{t_n - t_i} D(g, t_i) + 1$ 更新网格 g 的密度系数, 同时将 t_i 更新为 t_n , label 依据 D 的值并对比阈值 D_m 和 D_l 确定值为 sparse、transitional 或 dense。这样, 在线层完成了空间中网格特征向量的更新, 为离线部分的聚类过程做好了准备工作。

D-Stream 算法的主要思想就是将密度较大的相邻网格连成聚簇, 由于生成聚簇的计算量较大, 实时维护一个聚类结果是相当困难的。基于这个事实, 算法离线部分在每隔 ΔT 时间, 对聚簇结果进行一次维护。D-Stream 的离线部分就是基于这个思想, 考虑当前时间为 T , 在 $T - \Delta T$ 之前, 算法已经维护了一个聚簇结果, 而在 ΔT 时间内, 一批新的数据进入了系统, 算法将这批数据对原有聚簇产生的影响增量地加入其中, 产生新的聚簇结果。

一个稠密网格长期不接受数据, 可能会变成稀疏网格; 一个稀疏网格接收一些新的数据, 也可能升级为一个过渡网格或稠密网格, 因此, 一个关键点是决定离线部分生成聚簇的时间间隔 ΔT 的长度, 如果 ΔT 太大, 那么数据流的动态变化不能很好地识别过来; 如果 ΔT 太小, 则导致离线部分频繁计算从而增加了负载。因此, 根据 D-Stream [2] 中的描述使用以下命题来确定 ΔT :

命题 2.2: 对任何稠密网格 g ，从稠密网格成为一个稀疏网格所需的最少时间是 $\delta_0 = \left\lceil \log_{\lambda} \left(\frac{C_l}{C_m} \right) \right\rceil$ ；而对任何稀疏网格 g ，从稀疏网格成为一个稠密网格所需的最少时间是 $\delta_1 = \left\lceil \log_{\lambda} \left(\frac{N - C_m}{N - C_l} \right) \right\rceil$ 。

因此， ΔT 设置为 $\min\{\delta_0, \delta_1\}$ ，这样能够识别一个网格状态的任何变化。

2.3. Spark 和 Spark Streaming

Spark 是加州大学伯克利分校的 AMP 实验室所开源的类似 MapReduce 的通用并行计算框架，它拥有 MapReduce 的优点，并且由于 Spark 将中间计算结果的读写操作都在内存中进行，从而不需要读写 HDFS，也不负责数据的存储，这样能够更好地优化迭代 MapReduce 工作的负载。

Spark 对于数据的处理是基于弹性数据集 RDD，它是一个不可变的带分区的记录集合。Spark 为 RDD 提供了两类操作：转换和动作，转换是对原有的 RDD 进行一些变换来生成一个新的 RDD，而动作则是返回一个结果。一个 job 可以拆分成 RDD 经过一系列的变换操作得到最终的结果，也就是可以拆分成 RDD 组成的有向无环图 DAG (如图 1)，并将这个 DAG 作为一个 job 提交给 Spark 执行。

Spark Streaming 是一种构建在 Spark 上的实时计算框架，它扩展了 Spark 处理大规模流式数据的能力，它的基本原理是将输入数据流以时间片 Δt 为单位拆分成块，然后以类似 Spark 批处理的方式将每块数据作为 RDD 生成一个 job 进行处理，最终结果也返回多块。

Spark Streaming 具有高容错性，每一批输入到 RDD 中的数据都会在内存中备份，当某个结点发生故障导致数据丢失时，Spark Streaming 计算框架会找到备份的数据并在其他结点上重新计算得到最终的结果。

随着大数据的发展，人们对大数据处理提出了更高的要求。原有的批处理框架 MapReduce 由于其所有中间结果数据的读写都需要经过 HDFS 文件系统，因此启动一次 job 的代价很高，难以满足实时性要求。而 Spark Streaming 是 Spark 处理流式数据的框架，使用基于内存的 Spark 作为执行引擎，具有高速的执行效率和极低的延迟，对流数据的处理有极好的支持。

3. 分布式流聚类算法 PDStream 设计

结合 Spark Streaming 框架和 D-Stream 算法，本节提出一种基于分布式环境下的流数据聚类算法 PDStream。该算法将数据空间 S 划分为不同的块，对每个块运用 D-Stream 算法进行处理，得到该块的聚簇结果，最后将所有块的聚簇进行合并，生成最终的聚簇结果。

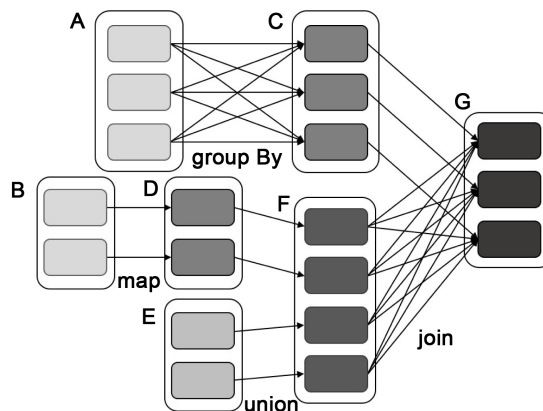


Figure 1. Execution of spark job

图 1. Spark Job 执行过程

3.1. 相关定义

定义 3.1 数据空间 S 包含 d 个维度, $S = S_1 \times S_2 \times \cdots \times S_d$, 而对每个子空间 S_i 的一个连续子集 S_{i,j_i} , $S_{1,j_1} \times S_{2,j_2} \times \cdots \times S_{d,j_d}$ 可以组成一个较小的数据空间 K , 该空间被定义为块。

定义 3.2 块 K 在 d 维空间中可以表示为 $K = S'_1 \times S'_2 \times \cdots \times S'_d$, 将每个子空间 S'_i 划分为 P'_i 个部分, 则整个块 K 可以划分的网格数为 $N_k = \prod_{i=1}^d P'_i$ 。设 g_p 表示块 K 中的网格 g 在第 p 个维度的相邻网格数, $0 < p \leq d$ 。若存在 $g_p < 2$, 则 g 是块 K 的一个边界网格。

定义 3.3 若构成块 K_1 和块 K_2 的第 p 维子空间 S_{p,k_1} 和 S_{p,k_2} 在第 p 维上相邻, 则块 K_1 和块 K_2 是相邻块。

3.2. PDStream 算法流程

在算法运行之前, 将空间 S 划分为 p 个互不相交的块, 即 $S = K_1 \cup K_2 \cup \cdots \cup K_p$, $K_i \cap K_j = \emptyset$, $1 \leq i < j \leq p$ 。对每个块 K_i 分别运行 D-Stream 算法。PDStream 算法流程如算法 1。

对于块 K_i 处理模块, 运行 D-Stream 算法, 它分为在线层和离线层。在线层 $K_i_online_component$ 负责接收数据, 保存网格的特征向量; 离线层 $K_i_offline_component$ 每隔 ΔT 时间生成 K_i 内部的聚簇。当所有块 K_i 的聚簇生成完毕后, 所有分块 K_i 的聚簇结果需要进行合并, 合并算法过程如算法 2。

Algorithm 1. PDStream

算法 1. PDStream

```

1:   $t = 0$ 
2:  while data stream is active do
3:     $x = (x_1, x_2, \dots, x_d)$  arrive at time  $t$ 
4:    calculate block  $K_i$  that contains  $x$ 
5:    send  $x$  to the  $K_i\_online\_component$ 
6:    if  $t \bmod \Delta T = 0$ 
7:      on each module  $K_i$ , call  $K_i\_offline\_component$ 
8:      Merge sub-clustering result
9:    end if
10: end while

```

Algorithm 2. Merge sub-clustering

算法 2. 合并分块聚簇

```

1:  for block  $K_i$  :
2:    for border grid  $g_i$ , get the neighbouring grid  $g_j$  on block  $K_j$ 
3:      if ( $g_i$  is labelled as  $c_{i,k_i}$  in block  $K_i$ ) and ( $g_j$  is labelled as  $c_{j,k_j}$  in block  $K_j$ )
4:        if  $|c_{i,k_i}| < |c_{j,k_j}|$ 
5:          label all grids in  $c_{i,k_i}$  as in  $c_{j,k_j}$ 
6:        else
7:          label all grids in  $c_{j,k_j}$  as in  $c_{i,k_i}$ 
8:        end if
9:      if ( $g_i$  is labelled as  $c_{i,k_i}$  in block  $K_i$ ) and ( $g_j$  is labelled as NO_CLASS)
10:         add grid  $g_j$  to cluster  $c_{i,k_i}$ 
11:      else
12:        if ( $g_j$  is labelled as  $c_{j,k_j}$  in block  $K_j$ ) and ( $g_i$  is labelled as NO_CLASS)
13:          add grid  $g_i$  to cluster  $c_{j,k_j}$ 
14:        end if
15:      end if
16:    end for
17:  end for

```

3.3. 基于并查集的优化

应当注意到，算法中经常涉及到将两个聚簇进行合并或者将某个网格合并到聚簇的操作。在实际代码编写中，每个网格需要设置它所属的聚簇标签，当需要将聚簇 c_1 合并到 c_2 中时，程序会将 c_1 中所有的网格所属的聚簇标签修改为 c_2 ，当这样的操作一旦大量增加，将会成为算法的瓶颈。使用并查集可以对上述操作进行优化，避免大量的修改。并查集是一种森林结构，用于处理一些不相交集合并及查询问题，核心思想是用一个代表元素表示集合中的所有元素，即森林中每棵树的根结点表示该树的所有结点，并且进行查找操作时可以进行路径压缩，将查找路径中每个结点直接连接至该树的根节点，提高再次查找的效率。当需要将结点 p 和结点 q 合并至一个集合时，先找到结点 p 的祖先结点 p' 和结点 q 的祖先结点 q' ，将 p' 变为 q' 的孩子结点，这样就将两颗树代表的集合进行了合并，新树的根 q' 节点即为集合合并后的代表结点。

如图 2，有三个块： K_i ， K_j 和 K_l ，其中块 K_i 中有聚簇 c_1 、 c_2 ， K_j 中有聚簇 c_3 、 c_4 、 c_5 ， K_l 中有聚簇 c_6 、 c_7 。

首先对于每个块内的聚簇，在森林 F 中生成自身结点进行初始化，表示每个聚簇自成一个集合。在进行相邻块聚簇的合并操作时，对于块 K_i ，遍历它的右边界网格时，发现聚簇 c_2 和相邻块 K_j 中的 c_3 可以进行合并，于是在 F 中找到 c_2 和 c_3 ，通过并查集的查找操作发现他们的祖先就是本身，并且并不属于同一个集合，则可将 c_2 作为 c_3 的孩子结点，表示聚簇 c_2 和 c_3 已经进行了合并，合并后的聚簇代表元为 c_3 。在遍历块 K_i 下方的边界网格时，再次发现聚簇 c_2 可以和块 K_l 中的 c_6 进行合并，于是在森林 F 中找到 c_2 ，将 c_6 变为 c_2 的孩子节点。再次查找 c_6 的祖先结点时，通过路径压缩可以直接把 c_6 变为 c_3 的孩子结点。上述过程如图 3 所示。

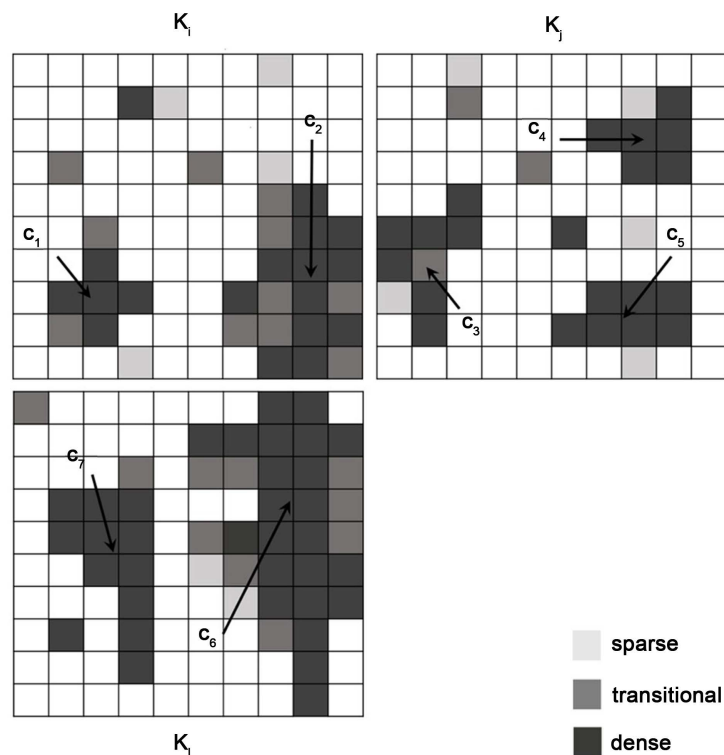


Figure 2. Clusters in blocks
图 2. 块中生成的聚簇

下面给出算法伪代码。在分块合并前，需要将所有块中的聚簇以结点形式加入到 F 中，以此来对 F 进行初始化，伪代码如算法 3。

初始化工作完成后，分属不同块的聚簇需要进行合并，下面给出将块 K_i 中的聚簇 p 和块 K_j 中的聚簇 q 合并的过程(算法 4)。

该算法的第一行调用了 FindRoot 函数，该函数可以查找一个节点所在树的根节点。在利用树结构递归查找根节点的同时，可以使用路径压缩技术进行优化，将该路径上的所有结点直接变为该树根结点的孩子。下面给出该函数伪代码(算法 5)。

完成所有的合并操作后， F 中树的个数即为最终形成的聚簇数，最后结合 F 将聚簇结果输出。

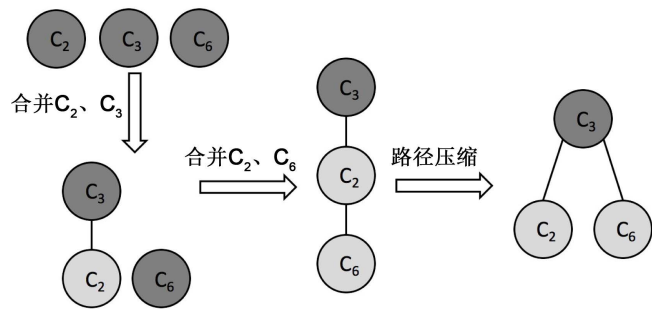


Figure 3. Use union find to merge clusters
图 3. 并查集合并聚簇

Algorithm 3. Init (clusters_List)

算法 3. 并查集初始化

```
1: set forest  $F$  as {}
2: for block  $K_i$ 
3:   if block  $K_i$  has clusters  $\{c_{i,1}, c_{i,2}, \dots, c_{i,m}\}$ 
4:     for each cluster  $p$ 
5:       add  $p$  as a node to  $F$ 
6:       set  $p$ .father =  $p$ 
7:     end for
8:   end if
9: end for
```

Algorithm 4. Merge (p, q)

算法 4. 合并聚簇(p, q)

```
1:  $p' = \text{FindRoot}(p)$ 
2:  $q' = \text{FindRoot}(q)$ 
3: if  $p' \neq q'$ 
4:   label  $p'$  as a child of  $q'$ 
5: end if
```

Algorithm 5. Find Root (p)

算法 5. 寻找树根(代表元)

```
1: if  $p$ .father =  $p$ 
2:   return  $p$ 
3: else
4:    $p$ .father = FindRoot( $p$ .father)
5:   return  $p$ .father
6: end if
```

4. 实验结果

PDStream 算法运行在分布式环境下, 该环境使用三台机器构建一个集群, 机器的硬件环境为 4 核 CPU 处理器, 16G 内存, 在集群无其他任务运行的条件下进行实验。Spark 框架支持多种语言接口, 我们使用 java 语言编写 Spark Job, 并提交到集群中运行。在下面进行的所有实验中参数设置保持一致: $C_m = 3.0$, $C_l = 0.8$, $\lambda = 0.998$ 。

4.1. 实验数据

我们使用两份数据集进行实验。第一份是一个真实的数据集 KDD-CUP 99 [11], 由 MIT 林肯实验室在 1998 年美国国防部高级规划署建立了模拟美国空军局域网的一个网络环境, 收集了 9 周时间的 TCP 连接数据, 仿真各种用户类型、各种不同的网络流量和攻击手段, 使它就像一个真实的网络环境。这些原始数据被分为两个部分: 7 周时间的训练数据, 以及 2 周时间的测试数据, 这些数据可以测试聚类算法的准确度。林肯实验室主页上提供的数据集有 500,000 条连接记录, 这个数据集中包含 5 个聚簇, 每个聚簇代表着一类异常连接或正常连接记录, 分别是 DOS, R2L, U2R, PROBING 以及 NORMAL, 每一条连接记录中包含 42 个属性。根据[3], 将其中的 34 个连续的属性作为聚类的输入, 最终聚簇结果应当为 5, 我们将运行 PDStream 算法验证结果的正确性。

另一份数据集是人工生成的二维数据集, 为了与 D-Stream 算法进行对比, 我们模拟[3]中使用的人工数据集, 数据量为 85K, 聚簇数量设置为 4。这个数据集主要用来观察 PDStream 输出的聚簇结果是否准确, 并且是否可以消除历史数据的影响。

4.2. 聚类的准确度

首先使用 KDD-CUP 99 作为实验数据集, 将它分别运行单机运行 D-Stream 算法、单机运行 PDStream 算法、多线程运行 PDStream 算法。如图 4, 可以看出在单机情况下 PDStream 和 D-Stream 正确率保持一致, 而在分布式多线程环境下, PDStream 算法的正确率依然能够得到保证, 并且可以从图中看出, 随着线程的增加, 准确度并没有明显提高, 线程数为 2 的时候甚至会下降, 这说明块的划分数目、划分方式以及聚簇的分布对算法结果有着很大的影响。

在人工数据集中, 一台机器上以 4 个线程运行 PDStream 算法, 数据量为 85 K, 以 1 K/s 的速率将数

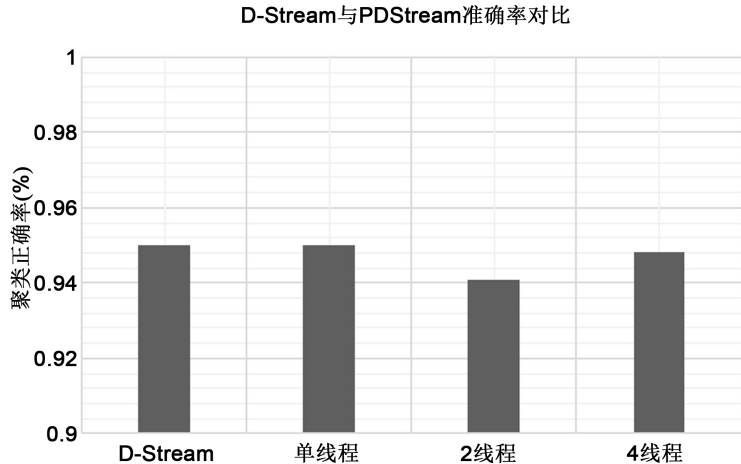


Figure 4. Accuracy comparison of D-Stream and PDStream

图 4. D-Stream 与 PDStream 准确率对比

据传入系统，数据集如图 5 所示，将区域分为 4 个簇。

为了验证算法可以显示聚簇随着时间动态变化的准确性，首先不加入衰减，输出最后的聚类结果如图 6 所示，从图中可以清楚地看到数据形成了 4 个簇。

现在再次运行算法，并在其中加入衰减参数 λ ，选取 $t = 10\text{ s}$ 、 $t = 50\text{ s}$ 、 $t = 85\text{ s}$ 三个时间点来观察聚簇的生成情况，分别如图 7、图 8、图 9 所示。观察这几幅图可以发现，由于衰减的存在，数据流的聚簇结果会随着时间发生变化，消除了历史数据的影响。从图中还可以很直观地看到 PDStream 聚类算法能够得到当前时间正确的聚类结果，验证了算法的准确性。

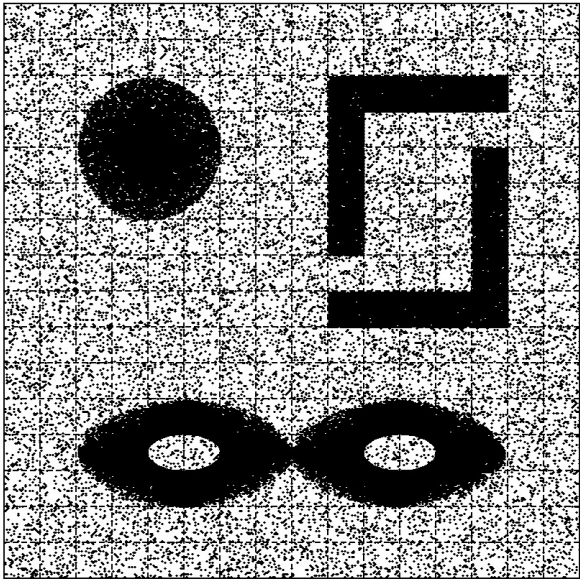


Figure 5. Artificial datasets including four clusters
图 5. 包含四个簇的人工数据集

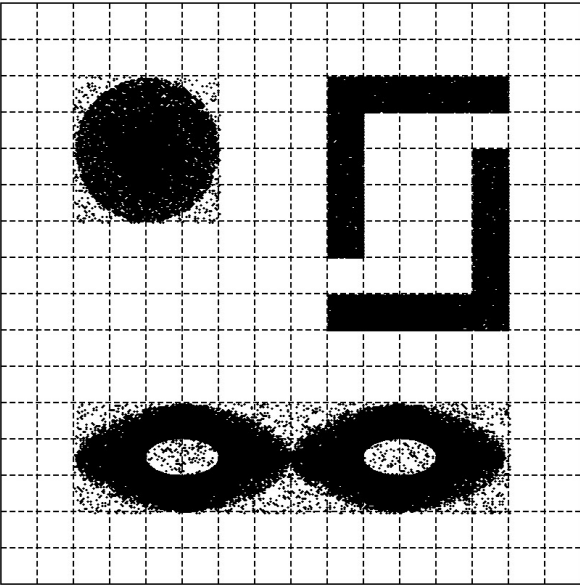


Figure 6. Result of clustering without attenuation
图 6. 无衰减下的聚类结果

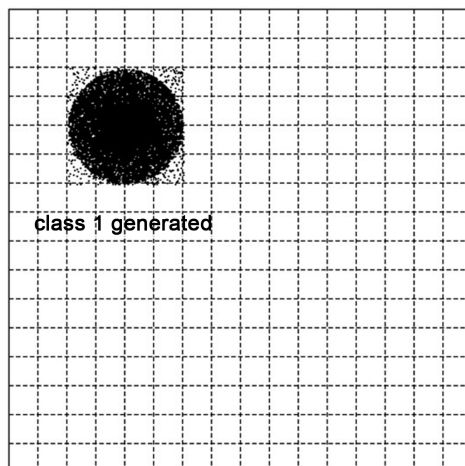


Figure 7. Result of clustering at $t = 10$ s
图 7. $t = 10$ s 时聚簇结果

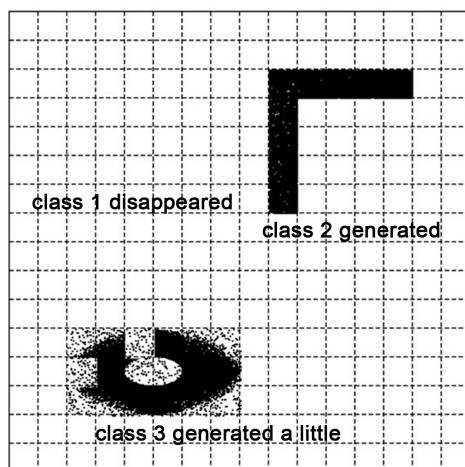


Figure 8. Result of clustering at $t = 50$ s
图 8. $t = 50$ s 时聚簇结果

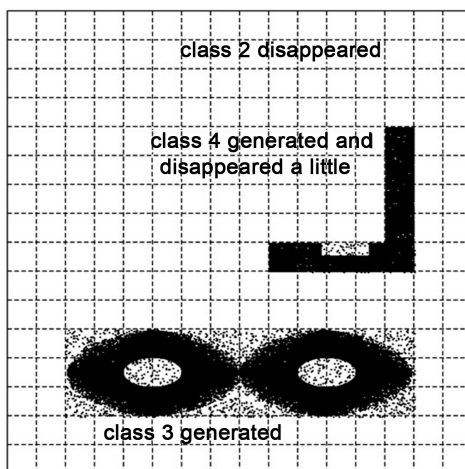


Figure 9. Result of clustering at $t = 85$ s
图 9. $t = 85$ s 时聚簇结果

4.3. 性能表现

由于加入了基于并查集的优化，PDStream的运行效率较D-Stream得到了显著提升。在KDD-CUP 99数据集上分别运行单机运行D-Stream算法和PDStream算法，记录不同数据量下生成聚簇的时间，如图10所示。从图中可以看出，随着数据量的增多，PDStream在时间效率上的优势越明显。

观察多线程下PDStream的性能表现，在人工数据集中分别以单线程、2线程、4线程、8线程运行PDStream算法，统计不同数据量下聚簇的生成时间，如图11所示。

从图中可以看出，随着线程的增加，聚簇生成的时间明显缩短，在数据量达到85K时，2线程相比单线程运行时间上缩短了接近20%，而4线程缩短了40%，可见分布式环境下分块中运行聚簇算法使得效率得到提升。但由于本文所用的人工数据集聚簇较少，数据空间较小，并且随着线程数目的增加，块的划分数目变多，块之间聚簇的合并操作成了算法新的瓶颈，因此从图中可看出8线程较4线程环境下的时间效率并无提升，反而有所下降，可见针对不同规模的数据集，需要根据实际情况合理规划线程数，使得算法达到最理想的运行效果。

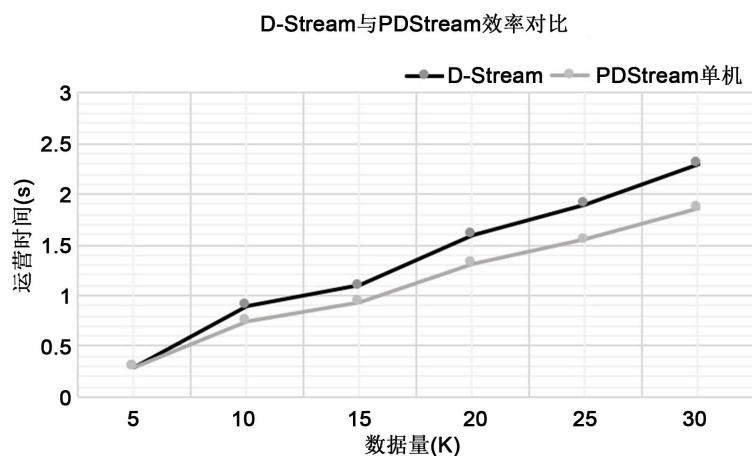


Figure 10. Efficiency comparison of D-Stream and PDStream

图 10. D-Stream 与 PDStream 效率对比

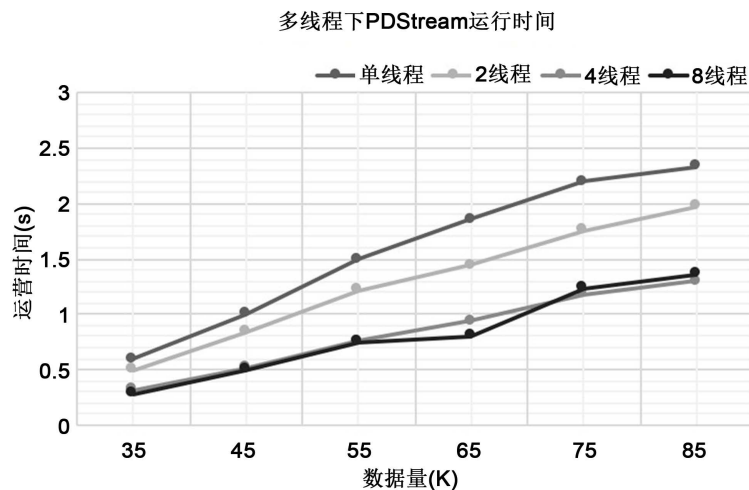


Figure 11. PDStream execution time in multithread

图 11. 多线程下 PDStream 运行时间

5. 总结

本文设计了一种基于 Spark Streaming 的分布式流聚类算法 PDStream, 该算法在 D-Stream 的基础上, 对数据集进行了分块并行处理, 在聚类结果得到保证的情况下, 加入了基于并查集的优化算法, 极大地提高了算法的运行效率。实验结果表明 PDStream 在运行效率上相比 D-Stream 有较大的提高, 并能适用于分布式环境, 从而说明了 PDStream 算法的可行性。

基金项目

上海市科委基础研究重点项目(13JC1403501)。

参考文献 (References)

- [1] Aggarwal, C.C., Han, J., Wang, J., et al. (2003) A Framework for Clustering Evolving Data Streams. *Vldb*, **29**, 81-92. <https://doi.org/10.1016/b978-012722442-8/50016-1>
- [2] Cao, F., Ester, M., Qian, W., et al. (2006) Density-Based Clustering over an Evolving Data Stream with Noise. *Siam International Conference on Data Mining*, Bethesda, 20-22 April 2006, 328-339.
- [3] Chen, Y. and Tu, L. (2007) Density-Based Clustering for Real-Time Stream Data. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Jose, August 2007, 133-142.
- [4] Amini, A. (2012) DENGRI-Stream: A Density-Grid Based Clustering Algorithm for Evolving Data Streams over Sliding Window. *International Conference on Data Mining and Computer Engineering*.
- [5] Bhatnagar, V., Kaur, S. and Chakravarthy, S. (2014) Clustering Data Streams Using Grid-Based Synopsis. *Knowledge & Information Systems*, **41**, 127-152. <https://doi.org/10.1007/s10115-013-0659-1>
- [6] Forestiero, A., Pizzuti, C. and Spezzano, G. (2013) A Single Pass Algorithm for Clustering Evolving Data Streams Based on Swarm Intelligence. *Data Mining & Knowledge Discovery*, **26**, 1-26. <https://doi.org/10.1007/s10618-011-0242-x>
- [7] Eberhart, R.C. (2001) Swarm Intelligence.
- [8] Hadoop, W.T. (2010) The Definitive Guide. *O'reilly Media Inc Gravenstein Highway North*, **215**, 1-4.
- [9] Zaharia, M., Chowdhury, M., Franklin, M.J., et al. (2010) Spark: Cluster Computing with Working Sets. 10.
- [10] 夏俊鸾, 邵赛赛. Spark Streaming: 大规模流式数据处理的新贵[J]. 程序员, 2014(2): 44-47.
- [11] 张新有, 曾华荣, 贾磊. 入侵检测数据集 KDD CUP99 研究[J]. 计算机工程与设计, 2010, 31(22): 4809-4812.

期刊投稿者将享受如下服务:

1. 投稿前咨询服务 (QQ、微信、邮箱皆可)
2. 为您匹配最合适的期刊
3. 24 小时以内解答您的所有疑问
4. 友好的在线投稿界面
5. 专业的同行评审
6. 知网检索
7. 全网络覆盖式推广您的研究

投稿请点击: <http://www.hanspub.org/Submission.aspx>

期刊邮箱: csa@hanspub.org