2016 届研究生硕士学位论文　　学校代码：10269

学　　号：51131201015

華東師範大學

基于弹性分布式数据集的流数据聚类分析

院　　系：信息科学技术学院计算机系

专　　业：计算机软件与理论

领　　域：复杂信息处理与数据库

指导教师：章炯民　副教授

论文作者：张 媛

2016 年 3 月完成

Dissertation for Master Degree, 2016 School Code: 10269

No: 51131201015

East China Normal University

Analysis of the Clustering Algorithm on Data Stream using Resilient Distributed Datasets

Department: Information Science and Technology

Major: Computer Software and Theory

Research Area: Complex Information Processing and Database

Supervisor: Associate Prof. Jiongmin Zhang

Student Name: Yuan Zhang

Mar., 2016

华东师范大学学位论文原创性声明

郑重声明：本人呈交的学位论文《基于弹性分布式数据集的流数据聚类分析》，是在华东师范大学攻读硕士/博士（请勾选）学位期间，在导师的指导下进行的研究工作及取得的研究成果。除文中已经注明引用的内容外，本论文不包含其他个人已经发表或撰写过的研究成果。对本文的研究做出重要贡献的个人和集体，均已在文中作了明确说明并表示谢意。

作者签名：_____ **日期：** 年 月 日

华东师范大学学位论文著作权使用声明

《基于弹性分布式数据集的流数据聚类分析》系本人在华东师范大学攻读学位期间在导师指导下完成的硕士/博士（请勾选）学位论文，本论文的研究成果归华东师范大学所有。本人同意华东师范大学根据相关规定保留和使用此学位论文，并向主管部门和学校指定的相关机构送交学位论文的印刷版和电子版；允许学位论文进入华东师范大学图书馆及数据库被查阅、借阅；同意学校将学位论文加入全国博士、硕士学位论文共建单位数据库进行检索，将学位论文的标题和摘要汇编出版，采用影印、缩印或者其它方式合理复制学位论文。

本学位论文属于（请勾选）

（ ）1.经华东师范大学相关部门审查核定的“内部”或“涉密”学位论文*，于 年 月 日解密，解密后适用上述授权。

（ ）2.不保密，适用上述授权。

导师签名_____ 本人签名_____

年 月 日

* “涉密”学位论文应是已经华东师范大学学位评定委员会办公室或保密委员会审定过的学位论文（需附获批的《华东师范大学研究生申请学位论文“涉密”审批表》方为有效），未经上述部门审定的学位论文均为公开学位论文。此声明栏不填写的，默认为公开学位论文，均适用上述授权）。

硕士学位论文答辩委员会成员名单

姓　名	职　称	单　位	备　注

摘 要

随着互联网应用的普及和深入，其所产生的数据急剧膨胀，且其中的许多数据都是动态的流式数据，需要及时处理和分析。对于流数据的聚类分析，国外学者已进行了一些探索和研究，目前已存在一些可用的流数据聚类算法，但这些算法仍然普遍存在诸多问题，例如，不能反映流数据的演化过程、无法识别任意形状的聚簇、对海量数据聚类效率不理想等等。

近年来，随着各种新型并行计算平台的出现和不断完善，聚类分析在并行计算平台上的实现得到了广泛的关注和认可，为提高聚类效率提供了新的有效的途径。例如，Spark 平台上的 K-Means Streaming 流数据聚类分析算法，但是，由于 Spark 平台的发展历史相对较短，其上的流数据聚类分析算法实现尚不多见，我们仅发现上述一例。

本文对经典的基于密度的聚类算法 DBSCAN 算法作了改进，提出了基于网格思想的聚类算法 GDBSCAN，在保留 DBSCAN 算法可以挖掘任意形状的簇的特性的前提之下，降低了其时间复杂度。其次，通过数据点有效时间的概念反映流数据的演化过程，结合 Spark 的 RDD （Resilient Distributed Datasets，弹性分布式数据集）内存计算的优势，本文进一步给出了 GDBSCAN 算法的 Spark 并行化实现 RDDGD-Stream，用于实时高效地对流数据进行聚类分析。此外，为了进一步提高算法的效率，RDDGD-Stream 还设计了基于网格数据点数目的重分区方法，平衡集群各节点的计算负载。

为了检验 GDBSCAN 和 RDDGD-Stream 算法的有效性，我们设计了多组实验，从聚类效率（运行时间和加速比）、演化性、聚类质量等多个方面加以考察。实验结果表明 GDBSCAN 和 RDDGD-Stream 算法的执行效率有明显的提高，聚类质量也有一定程度的提高。

关键词： 数据挖掘；流数据；聚类; RDD; DBSCAN; Spark

ABSTRACT

With the popularization and development of Internet applications and the rapid growth of the generated data, most of the data is dynamic data stream which needs to be processed and analyzed in time. Foreign scholars have carried out some exploration and research on the clustering algorithms of data stream. At present, there have been some available clustering algorithms of data stream but there are still many problems, for examples, couldn't reflect the evolving process of data stream or couldn't find the clusters of arbitrary shape, or are of low efficiency and so on.

In recent years, with the appearance and improvement of new parallel computing platform, the realization of the clustering algorithms on them has been widely concerned and recognized. It provides a new effective way to improve the efficiency of clustering, such as K-Means Streaming that is a clustering algorithm of data stream on Spark. However, due to the short development history of Spark platform, the clustering algorithms of data stream based on Spark are still not many and we only found one case.

In this paper, we improves classical density-based DBSCAN algorithm based on the idea of grid method to propose an algorithm GDBSCAN which reduces the time complexity under the premise of preserving the properties of finding the clusters distributed in arbitrary shape. Secondly, the effective time of data point is defined to reflect the evolving process of data stream. And combining the advantages of RDD, we provide a parallel implementation RDDGD-Stream of GDBSCAN algorithm on Spark which is used to cluster the data stream efficiently in real time. In addition, in order to further improve the efficiency of the algorithm, RDDGD-Stream also designed a repartitioning method based on the number of data points in grids to balance the computing load of each node of the cluster.

In order to validate the effectiveness of the GDBSCAN and RDDGD-Stream algorithms, we design a multi set of experiments to investigate from the clustering efficiency (running time and speedup), evolution, and clustering quality and so on. The experimental results show that the efficiency of GDBSCAN and RDDGD-Stream

algorithm is significantly improved, and the clustering quality is improved to a certain extent.

KEY WORDs: Data Mining; Data stream; Clustering; RDD; DBSCAN; Spark

目录

第1章 绪论

1.1 研究背景

二十世纪八十年代后期，电子技术的高速发展推动了计算机的广泛普及以及数据存储设备的大量供应，极大地促进了信息管理和数据库技术的发展，数据库已从简单的原始文件记录发展为结构型的支持查询和事务处理的数据库管理系统。数据的爆炸式增长和长时间的积累，使得高级数据分析开始引领各领域的发展方向，人类进入了数据挖掘时代[1]。数据挖掘即从海量数据中挖掘出潜在有用的信息的过程。

之后，随着智能设备的普及以及互联网技术的迅猛发展，网络应用渗透到人们生活的方方面面。天气监测、股票交易、零售业务和社交网络等应用产生了大量与传统的静态数据不同的数据形式，称这种数据形式为流数据[2]。流数据具有大规模、多样化、持续变化、快速增长的特性[3]。传统的数据挖掘技术已不适用于流数据的挖掘。如何快速有效地从流数据中挖掘出隐含的有助于人们决策分析的知识，已经成为企业界和研究机构的重点研究目标。基于流数据的聚类分析是挖掘方法中的重要内容，必然成为学术界亟待发展的热点。

1.2 流数据挖掘的研究现状

1.2.1 国内外研究现状

1998 年，Henzinger 等人在论文“Computing on data stream”中第一次提出流数据处理模型[4]。1999 年开始，在数据挖掘与数据库领域的各大顶级会议中发表大量有关流数据处理的文章，其主要研究范围包括流数据的聚类分析、流数据的分类分析和异常值检测等。流数据挖掘技术有广泛的现实应用前景，例如金融网点的交易和欺诈预测、网络入侵监测、互联网通讯等等都是流数据挖掘的重要研究问题。流数据聚类分析作为流数据挖掘的一个重要方法也成为热点研究课题。

对于流数据的聚类分析，国外学者已进行了一些探索和研究，目前已存在一些可用的流数据聚类算法[5]，但这些算法仍然普遍存在诸多问题。这些算法针对

流数据的特点对传统聚类算法做了改进：在文献[6][7]中提出了通过一次扫描在有限的空间实现流数据的 k-means[8]聚类；文献[7]改进了文献[6]中 k-means 需事先指定聚簇数目的缺陷，使聚簇数目可自适应变化。但这两种算法都只能分析当前的流数据，不能反映流数据的演化过程。CluStream[9]采用了双层框架，将流数据聚类过程分为在线部分和离线部分：在线部分负责实时处理新到来的数据并生成概要统计，离线部分利用生成的概要统计信息生成聚簇，从而解决了这个问题。但这几个算法均采用了 k-means 的基本思想，仅能挖掘球形聚簇，不能识别任意形状的聚簇。基于密度的流数据聚类算法 D-Stream[10]借鉴了 CluStream 算法的双层框架，解决了识别任意形状聚簇的问题，但对维度较高或者快速增长的流数据，聚类效率会明显下降。

针对维度较高或快速增长的流数据，迫切需要高速、有效的流数据聚类算法。分布式集群为大规模的流数据聚类提供了思路，其中最为典型的是基于 Hadoop MapReduce[11]框架的并行聚类。但是，基于 Hadoop MapReduce 框架的流数据聚类将中间结果保存在磁盘，新的流数据到来后，要多次存取磁盘中的中间结果，进行再聚类，I/O 操作频繁，会有一定延迟。

为了提高集群的并行计算能力，提高吞吐量，Matei 等人提出了弹性分布式数据集（Resilient Distributed Datasets，RDD）[12]。RDD 是一个共享内存模型，基于内存计算，不必频繁的存取磁盘，因此运算性能有了明显提升。目前，在 Spark 平台上已经实现了 K-Means Streaming[13]聚类算法，与传统的 K-Means 算法有相同的缺陷，无法识别任意形状的聚簇。

1.2.2 存在的问题

通过对目前国内外流数据聚类算法的深入研究，发现其仍存在如下问题：

首先，在流数据的聚类过程中，数据点在一段时间内的存在是有意义的，如果过期的历史数据不及时处理，一直保留在内存中，将会耗用大量内存资源，并且在较长一段时间后，已无存在意义的历史数据会影响流数据的聚类质量和聚类效率；如果简单的放弃历史数据而只对当前流入的数据集进行处理，则不能挖掘出流数据的演化过程。因此，如何及时删除历史数据释放内存，并且反映流数据的演化过程，对提高聚类质量是非常重要的。

其次，流数据具有高维度、速度快、连续性的特点，这就要求流数据的聚类分析算法有较高的处理速度。目前，大部分的企业应用已不再是单机作业，集群为数据挖掘技术带来了新的机遇。最典型的是各类数据挖掘技术在 Hadoop 平台的实现和应用，但是在 Hadoop 平台上，数据分析的中间结果保存在磁盘，需要多次从磁盘读写数据，频繁的 I/O 操作，降低了处理效率。因此，如何通过减少 I/O 操作提高集群的处理速度，对提高聚类分析效率是非常重要的。

最后，流数据是多样化和快速变化的，隐藏在其中的聚簇具有任意形状，这就要求流数据聚类算法能高效地识别流数据中任意形状的聚簇。目前，大部分的流数据聚类算法无法识别任意形状的聚簇，聚类质量较低。因此，如何实时高效地挖掘出流数据中任意形状的聚簇，对提高聚类质量是非常重要的。

1.3 本文的主要研究内容

DBSCAN 算法是一个基于密度的经典聚类算法，可以挖掘任意形状的簇，但时间复杂度较高。本文基于网格方法的思想对 DBSCAN 算法进行改进，提出 GDBSCAN 算法，保留了 DBSCAN 算法可以挖掘任意形状的簇的特性，并降低了时间复杂度。但对于海量的流数据聚类，GDBSCAN 算法仍显的力不从心。

近年来，随着各种新型并行计算平台的出现和不断完善，聚类分析在并行计算平台上的实现得到了广泛的关注和认可，为提高聚类效率提供了新的有效的途径。Spark 是一个新崛起的基于 RDD 的可扩展的集群平台，可与目前流行的大数据平台无缝结合，为大数据提供了通用算法的标准库和相同的部署方案。内存计算是 Spark 的最主要的优势，使 Spark 在集群解决方案和大数据处理领域极具竞争力。由于 Spark 平台的发展历史相对较短，其上的流数据聚类分析算法实现尚不多见，我们仅发现 K-Means Streaming 算法这一例。

本文结合 GDBSCAN 算法的特点和 RDD 的内存计算的优势，提出一种基于 Spark 平台的流数据聚类算法 RDDGD-Stream 算法，具体如下：

1) 利用传统的网格划分算法对数据空间进行初始划分，将其等分成边长不小于 2* ε（邻域半径）的网格。流数据输入系统后，将其映射到相应的网格。
2) 利用基于数据点的有效时间的淘汰算法，删除过期的历史数据，筛选出需要聚类的网格和数据点，一方面提高了计算效率，另一方面反映了流数据的动

态演化过程。

3) 提出基于数据点数目的重分区方法。重分区方法基于数据点数目对数据重新分区，均衡集群中各节点的计算负载，从总体上缩短计算时间。

4) 结合 RDD 的内存计算优势，基于 GDBSCAN 算法在 Spark 平台上实现并行聚类，从而实现流数据的实时高效聚类，挖掘流数据中任意形状的聚簇。

1.4 本文结构

本文各部分内容和安排如下：

第一章绪论，主要阐述本论文的研究背景、流数据挖掘和聚类的发展现状、存在的问题以及本文的主要研究内容。

第二章聚类分析概述，详细分析传统的数据挖掘和聚类方法以及流数据挖掘和流数据聚类方法的特点、常用算法、优缺点以及应用场景。

第三章 Spark 计算平台，详细分析云计算的相关知识、Spark 平台的弹性分布式数据集、工作机制和 Shuffle 机制。

第四章 RDDGD-Stream 算法的设计与实现，详细分析 RDDGD-Stream 算法和 GDBSCAN 算法的基本思路、执行流程以及实现过程。

第五章实验与实验结果分析，设计多组实验，从聚类效率（运行时间和加速比）、演化性、聚类质量等多个方面验证 GDBSCAN 算法和 RDDGD-Stream 算法的有效性，并对实验结果进行了分析。

第六章总结与展望，总结全文，概括算法 RDDGD-Stream 和 GDBSCAN 的优势，并说明现在算法的一些不足，提出就这些不足之处将展开的工作。

第2章 聚类分析概述

2.1 数据挖掘技术

在这信息大爆炸的时代，如何从信息的汪洋大海中及时发现潜在的有用信息，提高信息利用率，是人类面临的严峻挑战。面对这一挑战，数据挖掘技术应用而生，而且是近些年来发展迅速的数据库新技术之一。数据挖掘是从高性能计算、人工智能、统计学等多领域发展前进的，是一门跨领域的研究课题。

2.1.1 数据挖掘的概念

DM（Data Mining，数据挖掘）是从海量的、杂乱无章的初始数据集中，推导出隐含的对人类有帮助的知识和信息的过程。它的总体目标是通过一定的算法，从大量杂乱无章的数据集中抽取模式，发现数据变化的规律、趋势以及数据间的相互制约关系，作出易于理解的归纳性推理，指导决策者进行科学决策和分析，降低各类未知的风险，做出正确决策。

2.1.2 数据挖掘的过程

数据挖掘的过程除数据分析步骤外，还包含数据信息管理、数据预处理、模型建立与推理、复杂度分析、信息知识的发现以及可视化信息等方面。

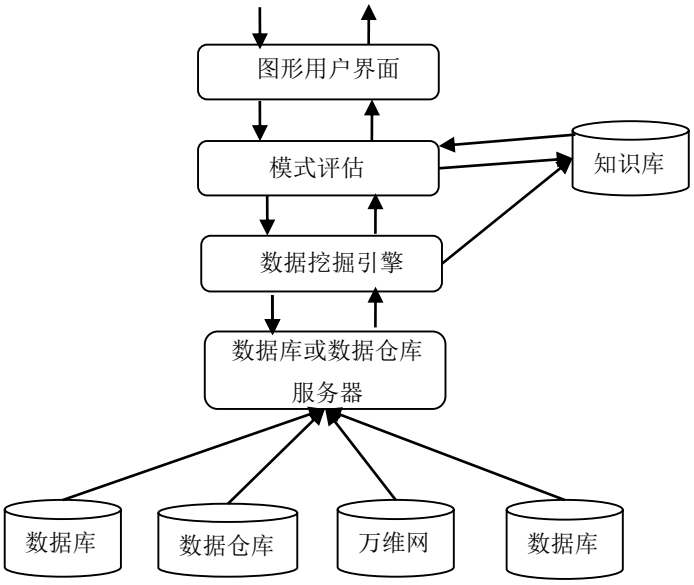


图 2.1 经典的数据挖掘框架

图 2.1 是经典的数据挖掘框架，其过程可大致如下划分：

第一阶段，数据准备阶段：在分析前必须对原始数据进行筛选与填充。原始数据集必须足够大，这样才能保证要发现的模式存在于原始数据集中，并且还要保证简洁，这样才能使处理时间限定在可接受范围内，所以在分析前必须要对原始数据集进行预处理，从中提取必要的挖掘项，并删除或增加缺少的数据记录。

第二阶段，数据挖掘阶段：借助各类挖掘工具，从预处理后的数据集中发现潜在的模式，主要步骤包括数据挖掘任务的确定、挖掘技术和算法的选择。数据挖掘主要的任务有以下六种：

1) 关联规则学习：发现数据记录集中各数据项之间的相互关联型。
2) 聚类分析：启发式地搜索对象集合中相似对象的聚类簇。
3) 分类分析：将还未分类的数据对象归类到已知分类簇结构中。
4) 聚合分析：将数据信息进行归纳分析并通过可视化技术展示出来。
5) 回归分析：定量地分析变量间依赖关系的统计分析方法。
6) 异常点检测：采用特征值匹配的方法查找数据对象集中不同于其他数据对象的对象。

第三阶段，结果表达与解释阶段：将数据挖掘分析所获得的知识用于解释已发生的情况、预测未来可能出现的现象。数据挖掘的结果也可以利用书面语言、图像、列表等各种可视化技术进行展示，这样有助于人们理解抽象的信息，并直

接使用。所以，进行数据挖掘的分析人员可以利用相关背景知识和所研究领域的前沿技术，来确定结果的表达形式，进而使得挖掘出的知识，能更加形象化地表现出来。

以上这几步工作，在现实应用场景中，一般都是反复进行的。事实上，很多数据分析方面的专家在经过大量的分析实践后，都会得出这个结论：在数据挖掘的整个过程中，数据预处理和结果表达消耗了数据挖掘整个流程的百分之八十的时间，其中主要有数据筛选、数据格式转换、记录整合、结果分析以及实现结论的可视化展示。随着机器性能和各种算法执行效率的提高，这两方面的需求逐步提升。

2.2 流数据挖掘分析

随着信息技术的高速发展，由传感器、网络应用等产生的数据已不仅仅局限于像数据库记录、文件等传统的静态数据，一种以流形式传输的数据越来越广泛的出现在各类应用领域。因此，流数据挖掘成为数据挖掘中一个新的热点研究课题，并逐渐发展为各领域的重要分析工具。

2.2.1 流数据的定义及特点

流数据是一系列数据元素的有序序列，这些数据元素按顺序到达流数据处理中心。假设有序数据元素序列为 $O_1,O_2,\ldots,O_i,\ldots$，这是一个具有时间戳的有序多维度的对象集合，对应的时间序列为 $T_1,T_2,\ldots,T_i\ldots$，其中数据对象 O_i 在 T_i 时刻到达，如果 $T_i<T_j$，说明数据对象 O_i 比 O_j 先到达。每一个数据对象元素 O_i 是一个 d 维向量，记为 $O_i=(O_{i1},O_{i2},\ldots,O_{id})$，分别代表数据对象 O_i 的 d 个属性值。

流数据的特点如下，与之前的数据形式有很大的区别：

1) 流数据的数据量是海量的、惊人的。
2) 流数据的潜在规模是无界的，而存储介质能存储的数据相对流数据来说是非常有限的。
3) 流数据是实时变化、按序到达的。
4) 变化频率快。

5) 无法预知将要处理的新数据的到达次序。

2.2.2 流数据挖掘的特点

传统的存储于数据库和文件中的数据以静态形式存在，而流数据可以看作是动态的，它们具有大规模、快速变化、有序性和潜在无限等特性。传统的数据挖掘技术主要是处理以静态形式存在的数据集，它们可以被多次且随机访问。根据流数据的自身特性可知，传统方法已不适用于新型数据领域，针对流数据的挖掘技术须适应流数据的自身特性。

由于流数据具有动态性，传统的先收集数据再处理的挖掘技术已失去作用，针对流数据的挖掘技术必须能够收集数据和分析处理同时进行，必须占用最少的内存资源，以最快的速度从不间断到达的数据中挖掘出感兴趣的模式，适应实时应用的需求。此外，由于收集的数据集的时效性和分析处理速度的局限性，挖掘出精确的分析结果是不太可能的，一般来讲，针对流数据的数据挖掘仅仅能获得近似结果。目前来说，对流数据的挖掘分析研究还处于初级阶段，但是由于流数据在不同领域的广泛应用和良好的发展前景，流数据挖掘技术已经成为数据分析领域的一个热点研究方向。

流数据的这些特性和实时应用的需求使流数据挖掘算法极富挑战性。根据流数据的上述诸多特性，流数据挖掘算法主要有如下几个特点：

1) 流数据挖掘算法要求时间复杂度要低。流数据的变化频率非常快，对时效性要求高，所以流数据挖掘算法的处理速度相应地要非常快。
2) 可以聚类任意形状的簇。现在大部分的流数据聚类算法都仅能识别球状簇，不能识别任意形状的簇，但是流数据的形状极具不确定性，随着流数据的输入，会出现不同形状的簇，所以流数据挖掘算法要能实时识别任意形状的簇。
3) 流数据挖掘分析的结果是近似性的。传统的数据挖掘分析要求分析结果要精确，但是流数据的数据收集时间和处理速度的局限性，无法达到传统挖掘分析的精确性，分析结果只能是近似的。
4) 流数据挖掘要求空间复杂度要低。流数据挖掘对时效性要求很高，所以一般数据都直接存储在内存中，内存的存储能力有限，所以对流数据挖掘要求尽量占用少的内存。

2.3 传统的聚类分析

“物以类聚，人以群分”，聚类分析是人们发现和探索事物之间关联性的一种方法，具有非常广泛的应用场景。在地理信息系统中，聚类分析可以发现具有相似地貌和土壤的区域，有助于矿产资源的开采；在商业市场中，聚类分析可帮助市场调查人员根据消费者的购买记录等信息，对消费者分类，进行针对性的产品推广；在医学研究中，病理研究员可以对病原体基因进行聚类，发现相似病原体，有助于医疗技术的进步。

在最初的分类学中，研究人员主要依据从业经验和专业领域知识对样本进行分类。随着科学技术的快速发展，人类对世界的认知水平逐步提升，对知识的分类也越来越精细化，仅仅依靠个人经验已不足以实现精确分类，所以需要将定性和定量的分析方法结合起来进行系统地分类。于是各类数学分析工具被引用到分类学中，并逐渐发展为数值分类学这一新学科。再后来，随着多元分析方法也被引进和使用，聚类分析逐渐从数值分类学中剥离出来，逐步发展为一个相对独立的新学科。由于各类应用领域中产生的信息量越来越大，聚类分析已成为一个异常活跃的研究课题[14]。

2.3.1 聚类分析的概念

一个类是由互相之间都相似的数据对象组成的集合，不同类中的数据对象是不相似的。聚类就是将一组物理或抽象的数据对象集，按照它们的各个属性值划分成多个不同类的分析过程。由此可见，聚类分析就是发现各个数据样本之间有价值的关联。在诸多应用场景中，属于同一聚类中的数据对象经常被看成一个对象来分析处理。聚类分析是一种无指导学习的方法，而分类属于有指导学习，它们之间最本质的区别是：在分类时，必须事先知道数据对象的分类属性值，将每个新的数据对象标记为已存在的类别；而聚类是由数据驱动，自动划分成事先未知的类。

聚类分析是人类一个重要的认知活动。从儿童时期开始，人们就不断的完善自己潜意识的分类能力，学习如何识别不同的物体，如树和草、鸡和鸭等。聚类分析在模式识别、市场分析、图像处理、生物研究和金融分析等场景中已有长足

发展。聚类分析是数据挖掘中的一个重要模块，既可以作为其他数据挖掘分析算法的一个数据预处理步骤，也可以作为一个单独的工具发现数据分布的深层信息、分析各数据类的特点、针对感兴趣的某些类进行更深入地分析研究[15]。

2.3.2 聚类分析算法

在聚类分析算法的实际应用中，为了解决不同的聚类问题，产生了各类聚类算法，如下[16]：

- 基于划分的方法

将包含 n 个数据对象的样本集划分为 m（m<n）个分组，每个分组代表一个簇。这 m 个分组中，任意一个分组最少应有一个数据对象并且任意一个数据对象必须存在于且仅存在于某一个分组。

该算法首先通过一定的选择规则创建一个初始划分，k 个划分中都有代表聚类点，再将其他数据对象点划分到距离初始聚类点最近的聚簇中，然后通过不断地循环迭代，改变分组策略，使得每次迭代后每个分组内的数据对象更相似或相近，而不同分组中的数据对象则更不同。

- 基于层次的方法

这种方法是对某一特定的数据对象集按照层级分解，直到符合规定条件。可分为如下两种方法：1) 凝聚法是初始阶段每个数据对象均为一个分组，按照特定规则逐渐地合并这些分组，直到满足某一条件或合并到层次顶端，即从底向上。2) 分裂法是从所有数据对象都属于一个分组开始，每次迭代将其分裂为更小的分组，直到每个对象属于一个分组或满足某一条件为止，即从顶向下。

- 基于密度的方法

从上述基于划分方法的介绍中，可以了解到其聚类之前，需先为所有数据对象计算两两之间的距离。这种算法挖掘球状簇精确度高，而对任意形状的聚簇效果较差。基于密度的聚类就是依次检测某个数据对象的给定邻居区域内的密度（数据对象个数）是否达到事先规定的阈值，若是则把它加入到临近的聚簇中。其解决了无法挖掘任意形状聚簇的缺陷。

- 基于网格的方法

样本集中的每个对象的不同属性用值来表示，数据集合的每个属性值有一个

范围。我们将每个属性划分成多个间隔，这些间隔范围组合成一个网格结构。之后，将数据对象集中的每个对象按照各自的属性值映射到网格中，以网格为单位执行所有的聚类操作。其执行速度相较于其他方法要快，因为算法执行时间仅与网格数目有关。

在实践应用中的聚类算法经常结合某些聚类分析算法的思想，因此有时较难界定某一聚类分析算法究竟属于哪类分析方法。大多时候在实际应用中，也需要将多个聚类分析算法组合在一起达到应用目标。

2.4 流数据的聚类分析

流数据聚类分析就是在流数据集上进行聚类分析的过程。然而，流数据是随时间不断变化产生的无界数据集，其隐含的聚类结果也可能随时间不断变化。鉴于流数据的动态性和无限性等特征，传统的聚类分析算法如不改进，多数已不能直接应用于流数据聚类分析。随着各领域诸多新型应用源源不断地产生大量的流数据，使得服务器的内存存储和运算能力都显得力不从心。在近些年里，流数据聚类分析的研究受到了企业界和学术界众多研究人员的广泛关注。

2.4.1 流数据聚类分析的要求

聚类分析方法是数据挖掘领域中被广泛应用的一项技术。传统的聚类分析方法是处理静态的数据对象集，可以随机多次访问存储的所有数据，并按照需求随时存取数据进行聚类。而流数据是海量的、快速的、实时动态变化的、无限的，其潜在聚类也会随时间不断变化。针对流数据的自身特征，流数据的聚类分析算法应着重权衡如下问题：

1) 使用计算机有限的内存存储空间。由于流数据源源不断地产生，具有无界性和海量性，并且计算机存储空间有限，所以不可能在计算机内存中存储所有的历史数据，必须有选择性地存储数据。如何在计算机有限的内存中尽可能多地存储数据并利用这些数据获得较好的聚类结果是流数据聚类算法应重点考虑和研究的问题。
2) 聚类分析的时效性。流数据中的数据变化很快，并且在线应用要求能实时给

出结果，这就要求流数据聚类算法的处理速度要尽可能的快。

3) 识别任意形状的簇。很多聚类算法通过数据对象之间的距离计算对象的相似度，这类算法通常只能发现大小和密度类似的球状聚簇，对非球状的聚簇效果不好。在实际应用中，随着流数据的不断输入，流数据中潜在的聚簇形状也会不断发生改变，所以要求聚类算法能实时识别出不同形状的聚簇。

4) 分析聚簇的演变。在流数据的聚类分析中，近期输入的数据应该占更大的比重，而过期的数据不应该参加聚类或者占用小的比重。人们总是对最近输入的数据更感兴趣，因此流数据的聚类分析应该能挖掘流数据的演化过程。

2.4.2 流数据聚类分析算法

目前，研究者对流数据的研究还不成熟，对流数据聚类分析算法的研究还处于起步阶段。在 2.3.2 节中介绍了五大类传统的聚类算法。由于流数据的特性，传统的聚类算法不能直接应用于流数据，但还是给流数据的聚类分析带来很多启示。事实上，大部分的流数据聚类分析算法是针对流数据的特征，通过扩展和改进传统的聚类算法来适应新应用的需求。图 2.2 描述了流数据聚类算法的发展过程。

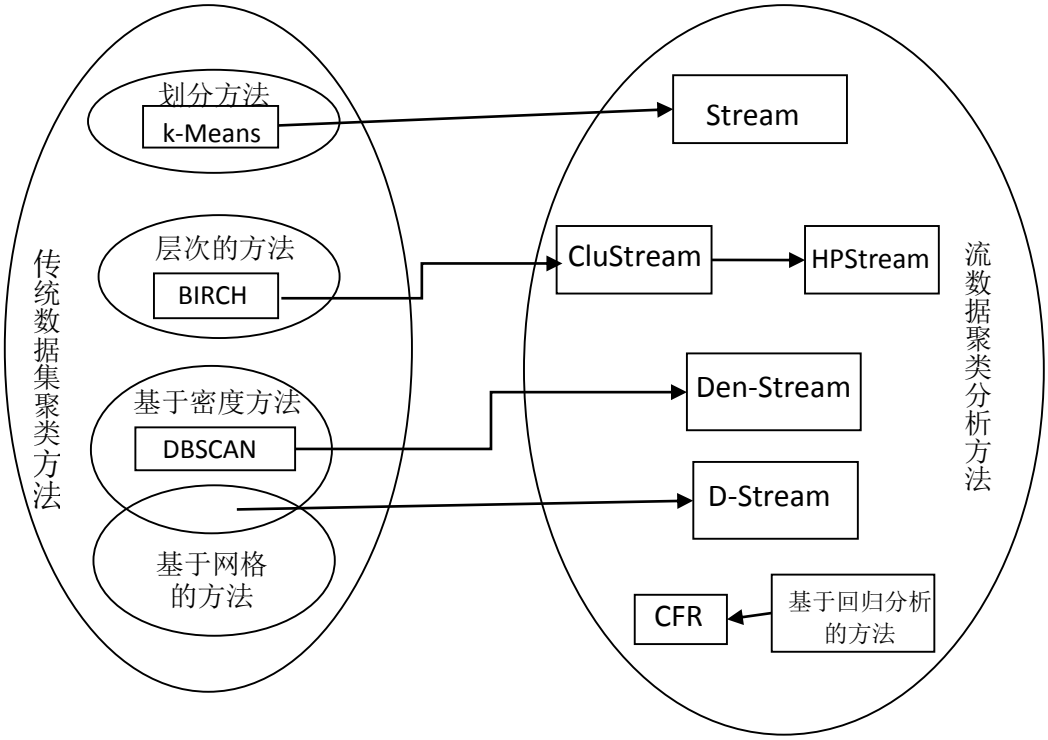


图 2.2：流数据聚类算法的发展

下面将介绍流数据聚类算法的发展历程和分类：

1) 扩展的划分方法（Extending Partitioning Methods）

基于划分的方法是将一个数据集，其包含 m 个数据元素，划分为 k（k≤m）个聚簇。已有的传统划分聚类方法包括有 k-平均算法（k-Means）和 k-中心点算法（k-Medoids）等。

Guha 等提出的聚类算法是将 k-平均算法扩展到流数据聚类上，先对流数据划分，通过一次扫描并且需要有限的存储能力。此算法的时间消耗可表示为 O(nk)，空间占用可表示为 O(n ε)，其中 n 为数据集的规模[6]。

O'callaghan 等人在 k-Medoids 算法的基础上提出了 STREAM 算法[7]，主要解决 k-中位数问题，即将数据集中的的所有数据对象分为 k 个分组，使得每个数据对象跟其所属分组的误差平方和（SSQ）最小。STREAM 算法采用分而治之的思想，将 m 个数据对象分在一个桶里，桶的大小跟内存大小相符。该算法把每个桶中的数据点都划分为 k 个聚类，并且只保留 k 个聚类中心点（丢掉其他数据）来汇总桶中的所有数据。每个中心点的权值为该聚类中数据对象的个数，如果收集到足够的中心点，加权中心点再次被聚类，不断重复这个过程直到得到最终的 k 个聚类，并且保证每个层级都最多保留 m 个数据对象点。STREAM 算法结合 LSEARCH 算法做了改进，使 k 更加灵活，在聚类的中间步骤中，聚类个数不是一个固定值，只是在算法终止时趋向于 k。此外，STREAM 算法给每个数据对象点的权重值都相等，没有考虑流数据的演化特性，因此聚类的结果受过期数据影响较大。 并且它更类似于一个批量处理，不能实时给出当前流数据的聚类结果。

2) 扩展的基于层次的划分方法

基于层次的划分方法是将数据对象集合进行层级分解，生成一棵树。典型的基于层次的划分方法有 BIRCH、ROCK 和 CURE 等。

Aggarwal 等人延伸了 BIRCH 算法中聚类特征（Clustering Feature，CF）的思想，开发了 CluStream 算法[9]。该算法将聚类过程分为如下两个步骤：1〉微聚类负责实时处理新输入的流数据集并存储生成的概要统计信息；2〉宏聚类主要任务是处理微聚类部分存储的概要统计信息并发现用户感兴趣的流数据的聚类结果。CluStream 算法的不足之处是要求客户在聚类前指定聚簇个数，不能动态改变聚簇个数。而流数据的聚簇形态是实时动态变化的，具有不可预见性，因此提前指

定聚簇个数会严重影响聚类结果的质量。同时，该算法受历史数据的影响很大，没有表现出近期数据的重要性。CluStream 算法从 k-Means 算法发展而来，因此不能发现任意形状的聚簇。最后，当流数据的噪音数据增多或者维度较高时，CluStream 算法的聚类质量往往都不高。

之后，Aggarwal 等人为了解决 CluStream 算法在数据对象点权值和处理高维流数据两方面的问题，开发了算法框架 HPStream[17]。其引入子空间聚类思想，通过将数据映射的方法降低高维流数据处理的难度。它的不足就是需要用户确定平均聚簇维数。HPStream 算法对高维流数据聚类效果很好，已经成为一个主要的流数据聚类算法。

3) 扩展的基于密度的聚类算法

大多数聚类算法都是根据空间位置的远近来判断是否类似，这类算法对任意形状的簇聚类效果不好。基于密度的聚类算法就是针对这一问题而设计的，达到某一阈值的区域连接起来构成一个聚簇。

Cao 等人受 DBSCAN 的思想的启发，提出了基于密度的流数据聚类算法 Den-Stream 算法[18]。在流数据源源不断地流入过程中，独立存在的点可能收录到某一个聚簇，一个聚簇也可能逐渐消失成单独的点，该算法可以实时挖掘当前独立存在的点和获得任意形状的聚簇。但 Den-Stream 算法反映不出流数据的动态变化。

4) 扩展的基于网格的方法

基于密度的聚类算法需要计算数据对象两两之间的距离，时间复杂度高，基于降低计算成本的目的，学者们提出了基于网格的聚类算法。STING 算法[19]是基于网格的密度聚类算法的代表，为很多算法提供了基本思路。其基本思想是：先将数据空间划分为网格结构；当流数据输入系统后，匹配这些数据点到相应的格子，并统计最底层网格中数据点的最大值、最小值、平均值和方差等信息；在有客户查询或有聚类需求时，从提前规定的某一层开始计算；如果是下层统计，可以用统计方法计算，如果是上层计算，可利用下一层的统计信息计算上一层的信息。其缺点是它挖掘出的所有聚簇都是以矩形为边界的，难以挖掘出其他形状的聚簇。

近年来，流数据挖掘方面的研究受到学术界的极大关注，许多学者将多种传

统技术结合运用到流数据聚类中，提出很多适用性强的方法。Chen 和 Tu 结合密度和网格的方法开发了一种新的流数据聚类方法--D-Stream 算法[20]。它的缺陷是流数据输入后匹配到相应网格并丢弃了自己在网格内的方位信息，对聚簇的边界描述不够精确。

5) 基于回归分析的流数据聚类分析算法

一个经典的基于回归分析的流数据聚类算法是 CFR 算法。该算法首先假定流数据集具有局部线性的特性，然后在流数据集上用数学函数的方法实现聚类挖掘，聚类过程中数据对象之间的相似度由马氏距离决定，还要考虑中心点和方差。CFR 的优点是它的初始聚类是通过数学函数创建的，不像其他算法初始聚类是近似的或者固定聚类数目的。其缺点是它的预先假定成为了很大限制。

2.5 本章小结

本章主要介绍了各种数据挖掘技术和聚类分析技术的基础知识以及相关算法。首先，介绍了数据挖掘和流数据挖掘的概念和过程，包括流数据的概念和特点以及流数据挖掘的特点。然后，描述了传统的聚类分析的相关概念和算法。最后，介绍了流数据聚类分析发展历程和存在的问题，包括一些典型的流数据聚类算法。

第3章 Spark 计算平台

二十一世纪以来，随着网络科技的迅猛发展以及网络应用的爆炸式增长，各类数据急速增多并且人们的需求也变得更加多样化，之前大量的数据分析技术面对新的应用需求越来越力不从心。这促使人们研究更多更先进的技术来解决新的问题，云计算是为了利用廉价的机器实现更高的计算性能而提出来的。随着云计算技术和产品的不断前进，近年来它已经渗透到各个领域的应用中。

3.1 云计算的概述

3.1.1 云计算的概念

云计算（Cloud Computing）是通过互联网为人类提供平台发布服务、共享服务、购买服务，通过网络可以很方便地连入云计算系统，从中获取资源（可以是存储空间、服务、应用等）[21]。它是一种通过网络实现的可扩展的由廉价计算机组成的分布式系统模型，无需投入过多的管理工作就可以快速获得所需资源[21]。

2006 年，Google 首次提出“云计算”的概念，之后 Google 和 IBM 开始在众多美国著名大学推广云计算的计划，取得了初步学术研究成果之后，Google 开始在世界范围内和一些大型企业合作，在更广泛的数据平台上进行试验研究，云计算技术和产品方面的研究取得了一定成果。

3.1.2 云计算的核心技术

在云计算系统中应用了诸多技术，包括虚拟化技术、分布式并行技术、数据管理技术、编程模型等，其中资源虚拟和分布式并行架构是云计算的两大核心技术。云计算技术使得用户可利用互联网中大量的开源软件，为云服务商有效地降低了硬件成本、软件开发成本和管理成本。

1) 虚拟化技术

虚拟化技术最早由 VMWare 公司提出，可以实现应用软件层与底层硬件层的隔离，可以将某一资源分裂为多个虚拟资源，也可以将多个资源虚拟整合成一个虚拟资源。虚拟化技术是将服务器、存储、网络等物理资源虚拟和池化，然后

由云平台统一管理、监控和分配资源池。按照虚拟资源的不同分为存储虚拟化、网络虚拟化和计算虚拟化等。虚拟化平台在物理服务器上创建多个性能可配置的虚拟机，可以统一管理集群系统中的所有虚拟机，并且按照应用的实际需求灵活的从资源池中分配资源。

2) 分布式并行架构

分布式并行架构是云计算的另一核心技术，将大量低廉的机器联系在一起成为一台超级计算机，通过分布式文件系统、分布式数据库和 MapReduce 编程技术，实现海量文件和结构化数据的存储，为处理海量数据提供了统一的编程模式和运行环境。

Google 的主营业务是为全世界用户提供搜索查询服务，因此面临着海量数据的存储和快速分析处理的挑战，为解决这一问题 Google 提出了分布式技术。分布式架构支持上百万台的廉价计算机协同工作，它主要包括分布式文件系统、分布式数据库系统、分布式计算编程模型： 分布式文件系统（如 Google 的 GFS、Hadoop 的 HDFS）主要负责海量数据文件的存储，分布式数据库负责海量的结构化数据的存储，分布式计算编程模型 MapReduce 负责任务的分解工作和并行化处理。

3.1.3 MapReduce 编程模型

MapReduce，它是由包含在 Map（映射）和 Reduce（规约）中的两个函数名字组合而成的，是在大量计算机上高度并行化地处理海量数据集的分布式框架[25]。其中映射函数将读入的原始数据按照函数内定义的方式转换成以键值对形式存储的数据，而规约函数则是处理映射函数输出的数据[11]。

MapReduce 的引入就是为了解决海量数据的分析处理问题。它基于分而治之的方法将数据分为小块，由 Map 函数并行化处理这些数据块，提高海量数据的处理速度。MapReduce 模型为编程人员提供了便利，即使对分布式并行编程还不熟悉，也可以开发出适合在分布式系统上运行的应用程序。

图 3.1 中描述 MapReduce 编程模型的运行机制：

1) MapReduce 先将输入文件按照用户指定的大小分成多个分片，然后在集群中复制这些分片；

2) 主节点 Master 监控和管理集群资源，自动选择相对空闲的 worker 节点分配 map 任务；
3) 接收到 map 任务的 worker 节点读取相应的输入分片并解析成键值对形式，之后运行用户自定义的 map 函数，并将输出的数据缓存到内存中；
4) 缓存在内存中的数据被分成多个分区写入本地磁盘，并告知主节点，方便 reduce 函数使用这些数据；
5) Reduce worker 在得到主节点通知后，读取所有 map 函数缓存在本地磁盘中的数据，并按照键值排序重组数据（相同键值在同一组）。
6) Reduce worker 迭代读取键值对数据，传递给用户定义的 reduce 函数处理，将 reduce 函数的输出保存在此 reduce 分区的输出文件中。
7) 当所有的 map 和 reduce 的工作都结束后，重新切换到用户程序。

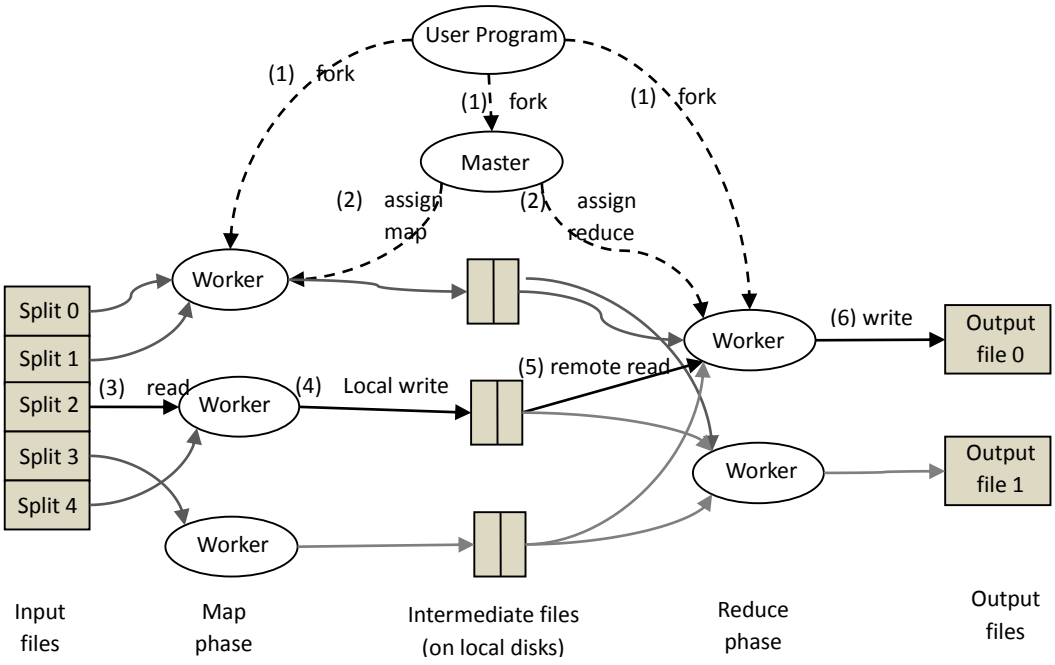


图 3.1　MapReduce 运行机制

3.2 Spark 分布式计算平台

Spark 是当今大数据业务范畴内最热门的大数据并行计算框架，基于弹性分布式数据集构建了一体化、多元化的大数据处理体系，可以通过统一的编程接口无缝连接 Spark SQL、Spark Streaming、MLLib、GraphX 四大框架库，它的最大的

优势是实现了分布式内存计算。在实际应用环境中，有很多大公司都实现了一千个节点以上的 Spark 集群，包括 eBay、Yahoo!，国内的淘宝、腾讯、百度、京东、网易等也非常强有力的研究并使用 Spark。

3.2.1 Spark 框架概述

目前，在大数据计算领域内，应用较为广泛的是 Hadoop MapReduce，它是整个 Hadoop 生态系统的核心，将数据处理过程分为 Map 和 Reduce 两个阶段，比较适合离线处理海量数据。但是由于它的中间结果写入磁盘导致调度开销大、运行慢，不适合做实时在线分析。为此，UC Berkeley AMP lab 开发了一个新的大数据处理框架 Spark。Spark 是一个分布式内存计算框架，包含 Hadoop MapReduce 的所有优点，提出了新的数据模型弹性分布式数据集(Resilient Distributed Datasets, RDD)，并且可以将中间结果以 RDD 的数据形式存储于分布式内存中，需要使用数据集时从内存中直接获取，减少访问 Spark 集群磁盘的次数，迭代运算效率高，数据处理速度快，更适合实时响应和交互式计算等应用场景，同时，还具有高容错性和高可伸缩性的优点。Spark 实现了 Hadoop 的 Map 和 Reduce 函数以及计算模型，还提供了更丰富的算子，如 groupByKey、join、filter、union 等，其底层框架采用 Scala 函数式编程语言开发，并且提供了类似于 Scala 编程思想的 API 方便开发人员调用，还支持 Python 和 Java 编写程序。Spark 可以兼容 HDFS、Hive 等分布式存储系统，还可以兼容本地文件系统、HABSE、S3 等[26]，可以整合到 Hadoop 的生态系统中，弥补 MapReduce 的缺陷。

Spark 的生态系统中主要包括有支持结构化数据查询和分析的查询引擎 Spark SQL、流数据计算框架 Spark Streaming、提供分布式机器学习算法的库 MLlib、并行图计算库 GraphX。Spark SQL 底层使用 Spark 作为执行引擎实现大数据上的 SQL 操作，用户可以直接在 Spark 上编写 SQL 语句，其定义了 SchemaRDD 数据集，可以兼容不同的持久化存储和处理不同的数据源，包括 Hive、HDFS、JSON 数据、JDBC 数据源等。Spark Streaming 是处理流数据的子库，将流数据按照指定时间片划分，生成 RDD 然后对每个 RDD 实现批处理，进而实现大批量的流数据处理。由于 RDD 可存于内存，减少了磁盘访问次数，所以其吞吐量超越现在主流的流数据处理框架。MLlib 是 Spark 提供给开发者的机器学习组件，迭代运算效

率高，其中已包含了常用的机器学习方法，开发者也可以自己编写算法扩展 MLlib 库。GraphX 是 Spark 提供的图计算和图并行计算的子库，封装了 Pregel 接口，有效地提高了图计算速度，尤其是进行多次迭代运算时，基于 Spark 的内存计算有明显优势。Spark 框架如图 3.2 所示。

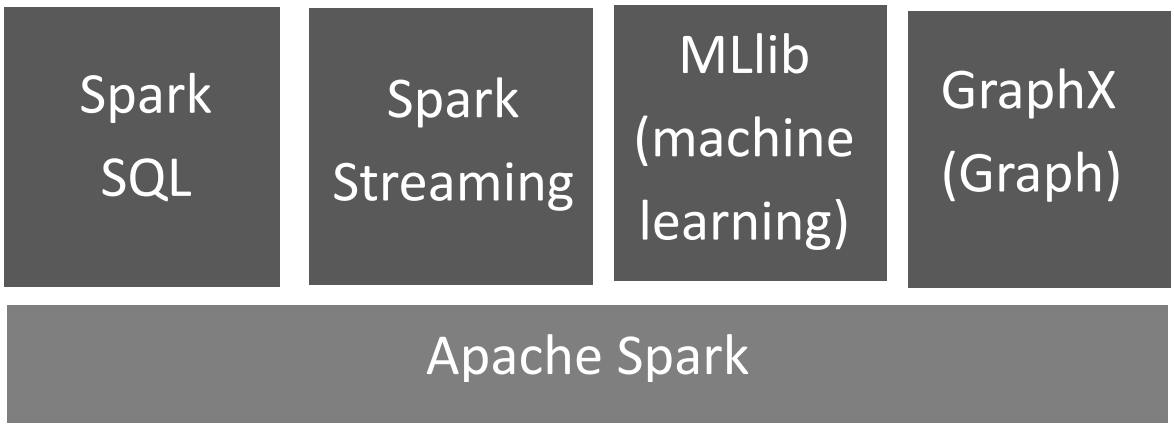


图 3.2 Spark 框架

3.2.2 弹性分布式数据集(Resilient Distributed dataset, RDD)

为了提高对数据和作业的容错能力，并优化集群的处理速度，Spark 使用了新的数据模型，弹性分布式数据集（Resilient Distributed Dataset，RDD）[27]。RDD 是分布式内存的一个抽象概念，允许开发人员在大规模集群上执行基于内存的计算。它是只读的记录分区的集合，只能通过读取 HDFS（或与 Hadoop 兼容的其他持久化存储系统）生成或由其他的 RDD 经转换操作生成，这些限制方便实现高容错性。现有的流数据处理模型对两种应用处理效率不高，一是图领域和数据挖掘领域常见的迭代式算法，另一个是交互式数据挖掘工具。RDD 可将数据存储在内存中，并极大地提高处理数据的性能。

Spark 提供的在数据集 RDD 上的操作种类很多，可以分为转换（Transformation）和动作（Action）。转换操作是惰性的，也就是说一个 RDD 到另一个新的 RDD 的转换操作不会被马上执行，而是在遇到 Action 操作时，才真正被触发。转换操作主要包括有 sample()、filter()、map()、mapPartitions()、union()、groupByKey()等接口。Action 操作会触发 Spark 提交作业给系统，并将结果输出到 Spark 系统中。Action 操作主要包含有 reduce()、collect()、count()、first()、saveAsObjectFile()、takeOrdered()等接口。通过这些操作可以从父 RDD 生成新的 RDD，而且 RDD 上

的很多操作都是批量执行，便于在内存中实现大数据的处理。

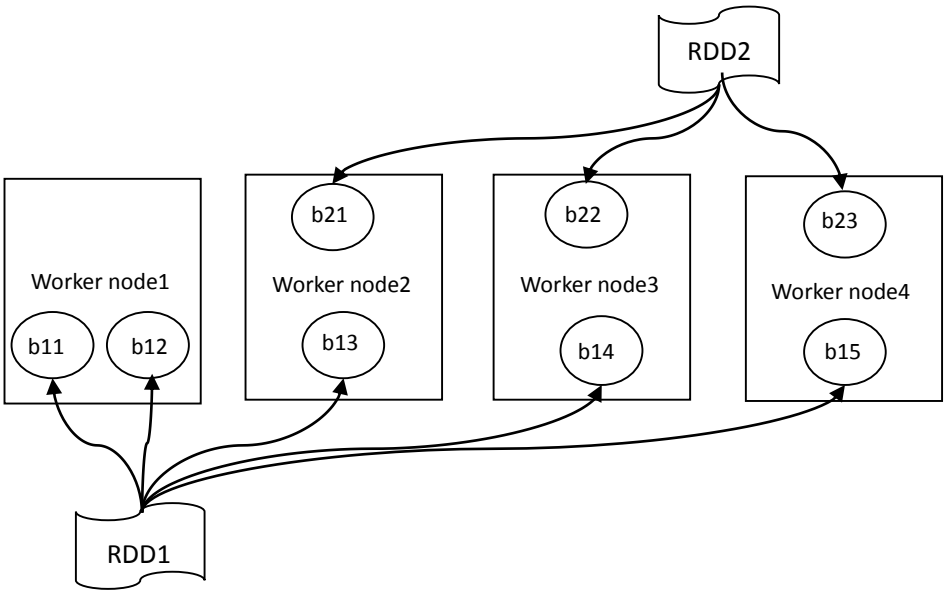


图 3.3 RDD 的数据管理模型

RDD 对象本质上来说是一个元数据结构，一个 RDD 存储着块和机器节点的信息以及其他元数据的信息。一个 RDD 可以包含多个分区，在数据物理存储上，RDD 的一个分区对应一个块，这些块可以分布地存储在不同的机器节点中，块可以存储在内存中，当内存空间不足时，也可以部分缓存于内存中，其余分区数据存储在磁盘中。RDD 的数据管理模型如图 3.3 所示。RDD1 包含有五个分区 b11、b12、b13、b14、b15，分别存储在四个机器节点 w1、w2、w3、w4 上，其中分区 b11 和分区 b12 都在机器 w1 上。RDD2 有三个分区 b21、b22、b23，分别存储在 w2、w3 和 w4 上。

RDD 的 dependencies 记录的是 RDD 由哪个或者哪些 RDD 演变而来的信息，这些依赖是 Dependencies 类或其子类的实例，在 Spark 中称之为血统，通过这些依赖，可以获得 RDD 的父母或者祖先的信息。在 Spark 中，依赖分为两大类，宽依赖和窄依赖。宽依赖（wide dependency）是指子 RDD 中的每个分区都与父 RDD 中的所有分区相关联，这是由 shuffle 类操作引起的，典型操作有 groupByKey、sortByKey 等。窄依赖（narrow dependency）指的是父 RDD 的一个分区最多被子 RDD 的一个分区所用，典型操作有 map、filter 和 union 等，Spark 通常会将窄依赖的分区划分到一个 stage 里，这样可以进行流水线计算。宽依赖和窄依赖的划分如图 3.4 所示。

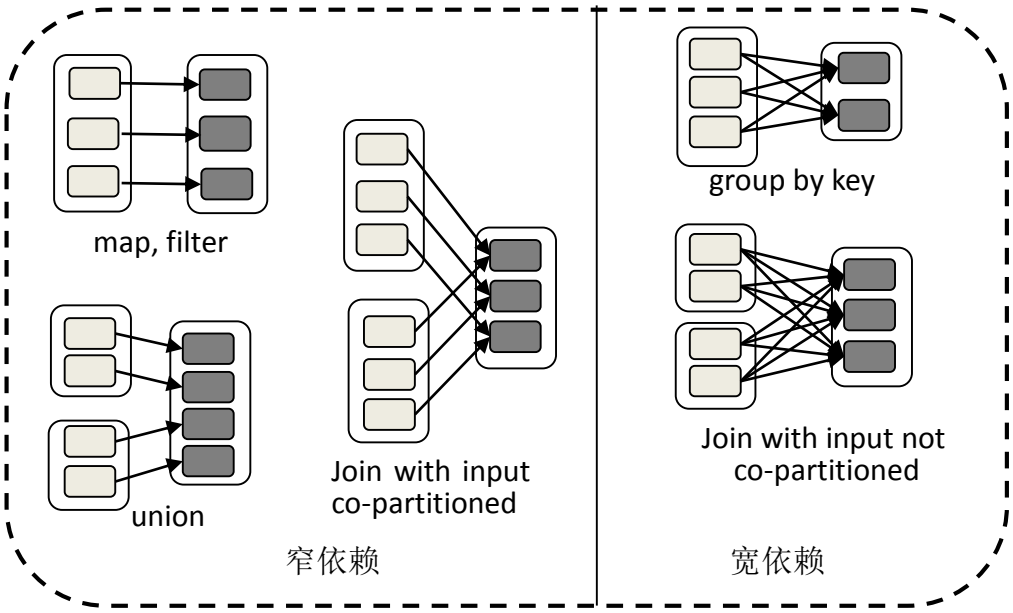


图 3.4 宽依赖和窄依赖

3.2.3 Spark 工作机制详解

本节将详细介绍 Spark 的内部运行机制，Spark 中主要包含任务与调度分配、通信控制、容错、I/O 以及 shuffle 模块。Spark 通过调度算法 FIFO 和 FAIR 分别在应用、Job、Stage 和 Task 这几个层次执行任务调度，并通过 AKKA 框架传递集群的命令和状态信息。高容错性对分布式系统非常重要，Spark 通过血统和快照保证了系统的高容错性。Spark 以块为单位管理数据，数据块可以保存在本机磁盘、内存或者集群中的其他机器内。Spark 借鉴了 Hadoop 中 MapReduce 计算模型的优势，且在此基础上对 Shuffle 机制进行了优化。

3.2.3.1 Spark 中基本组件和服务

Spark 系统部署成功后，集群中的各个节点会启动相应的各类服务，在介绍 Spark 的基本工作流程前先介绍一下基本组件和服务：

- Application：与 Hadoop MapReduce 中的 Application 概念相似，都指的是用户编写的应用程序，Spark 中的 Application 包含一段 Driver 功能的代码和分布在不同节点上运行的 Executor 代码。

- Driver：指的是 Application 代码中的主函数（main()）并创建 SparkContext，SparkContext 的作用是为 Spark 应用程序准备运行环境，负责与 ClusterManager 通信，完成分配资源、分配和监控任务等。在 Executor 执行完后，Driver 负责关闭 SparkContext。
- Executor：执行 Application 程序时，在 Worker 节点上运行的一个进程，它负责管理该 Worker 节点上的 Task 以及数据在内存或磁盘上的存储。每个 Application 程序有属于自己的一组 Executor 对象，在 Spark on Yarn 模式下，该进程名为 CoarseGrainedExecutorBackend，它与 Executor 对象一一对应，将 Task 封装成 taskRunner，并从线程池中获取一个空闲线程运行 Task。
- ClusterManager：指在集群上获取资源的服务。
- Worker：集群中可以运行应用程序代码的节点。
- Task：在某个 Executor 上运行的工作单元，是运行应用程序的基本单位，由 TaskScheduler 调度和管理。
- Job：通常由 Action 算子触发产生，包含多个 task 的并行计算。
- Stage：每个 Job 会被分成多组 Task，一组 Task 是一个 TaskSet，称其为 Stage。Stage 由 DAGScheduler 划分和调度。Stage 以发生 Shuffle 的地方为边界，分为最终的 Stage（Result Stage）和非最终的 Stage（Shuffle Map Stage）。
- DAGScheduler：按照 Job 构造基于 Stage 的有向无环图（DAG），其按照 RDD 间的依赖关系划分 Stage，并提交 Stage 给 TaskScheduler。
- TaskScheduler：提交 Taskset 给 Worker 节点运行，分配给 Executor 执行的任务。

3.2.3.2 Spark 的基本工作流程

RDD 只是一个静态模型，它所解决的只是一个规划问题，静态模型需要在真正运行起来之后才能将问题实实在在解决掉。

Spark 在接受到提交的作业后，会进行如下处理：

1) RDD 之间的依赖性分析。RDD 之间的依赖性成一个有向无环图 DAG，依赖关系的分析和判断由 DAGScheduler 负责。
2) 根据 DAG 的分析结果将一个作业分成多个 Stage。划分 Stage 的一个重要依

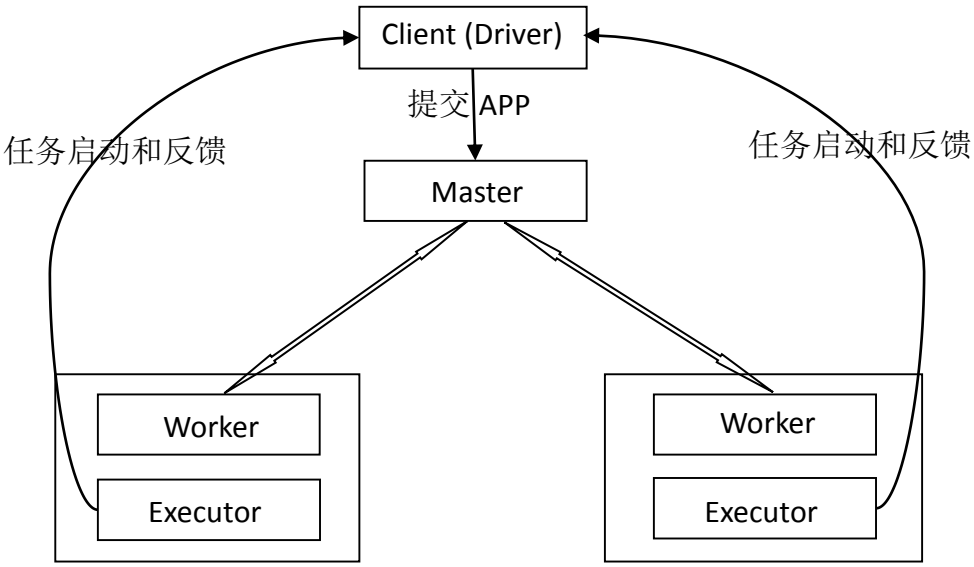
据是 RDD 之间的相互影响关系。

3) DAGScheduler 在确定完 Stage 之后，会向 TaskScheduler 提交任务集（Taskset），而 TaskScheduler 负责将应用程序的提交和执行方式。

Driver 进程是 Spark 应用程序的主控程序，主要负责 Job 的解析、划分 Stage、分配 Task 到 Executor 执行等。Spark 应用程序的提交包含两种方式：

- Driver 进程在 Client 运行，管理并监控整个应用的执行

  Spark Driver 进程运行在 Client，管理并监控整个应用的执行，作业提交后的转化过程如图 3.5 所示。



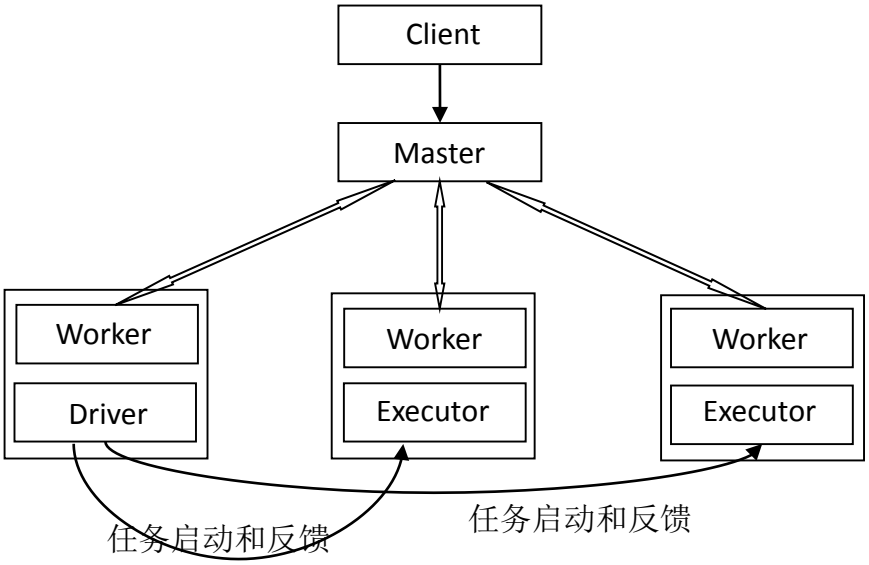
3.5　Spark Driver 在客户端启动

用户在 Client 运行应用程序 Application，Client 启动 Driver 进程。Driver 中启动或实例化 DAGScheduler 等组件，然后向 Master 注册。Worker 节点也向 Master 注册，Master 发指令给 Worker 启动 Executor。Worker 新建 ExecutorRunner 线程并在 ExecutorRunner 线程内部启动 ExecutorBankend 进程，然后 ExecutorBankend 向客户端的 Driver 注册，此后 Driver 就可以管理和分配 worker 中的计算资源，Driver 中的 DAGScheduler 组件将 RDD 有向无环图划分成 Stage，接着 TaskScheduler 分配 Stage 中的 Taskset 给 Executor，Executor 会启动线程池并行完成 Task。

- Master 指定一个 Worker 节点启动 Driver

  若 Driver 在指定的 Worker 上运行时，要使用 org.apache.spark.deploy.Client

类来运行应用程序。这种方式的应用提交和各部分的协作过程如图 3.6 所示。用户在客户端提交应用程序 Application 给 Master，Master 为 Application 指定一个 Worker 运行 Driver 进程。Master 指定集群中其他的工作节点启动 Executor，Worker 创建 ExecutorRunner 线程，ExecutorRunner 线程开启 ExecutorBankend 进程，ExecutorBankend 进程向 Driver 的 SchedulerBankend 进程注册，这样 Driver 就能调用 Worker 节点中的计算资源并将 Task 分配到计算节点执行。Driver 的 SchedulerBankend 进程按照 RDD 的有向无环图划分 Stage，每个 Stage 中的 TaskSet 都保存在 TaskScheduler 中，然后 TaskScheduler 再分派 Task 到 Executor 上并行执行。



3.6　Spark Driver 在 Worker 启动

3.2.4 Shuffle 机制

Spark 中的 Shuffle 与 Hadoop MapReduce 中 Shuffle 的基本思想一致，只是在优化和实现上有所不同。Spark 采用了分布式计算模型，不可能在一个内存空间中保存所有的处理数据，按照数据的 key 值将数据分成一块一块的小分区，然后分散到集群中不同节点的相应进程的内存空间中。Shuffle 就是在执行需要重分区的算子时对数据进行重新组合的过程。我们可以这样来理解：数据重新分区是按照数据的 key 通过映射函数实现划分，确定数据 key 值的过程称为 Map 过程，Map 过程也可以做数据处理；Shuffle 将 Map 过程的结果数据收集起来分配到指

定的 Reduce 分区，Reduce 过程根据定义的函数对相应分区的数据做处理。

Shuffle 过程包含 Shuffle Write 和 Shuffle Fetch 两个阶段。每个 Stage 对每个分区执行变换（Transformation）的流水线式的函数操作，在 Stage 的最后执行阶段进行 Shuffle Write 操作，按照下一个 Stage 的分区数目将数据划分成相应的 Bucket，然后将它们写入磁盘，这就是 Shuffle Write 阶段。下一个 Stage 从存储着上一步 Bucket 的磁盘获取需要的数据，将这些数据取到本地后执行用户自定义的聚集函数操作。这样就完成了 Shuffle 操作。

3.3 本章小结

本章阐述了云计算的重要知识体系，着重阐述了 Spark 分布式计算平台。首先，介绍了云计算的基础知识、核心技术和 MapReduce 编程模型。然后，详尽描述了 Spark 分布式计算平台的相关概念、弹性分布式数据集（RDD）、工作机制和 Shuffle 机制等内容。

第4章 RDDGD-Stream 算法的设计与实现

目前常见的流数据聚类分析算法大多基于 k-Means 算法和 k-Medoids 算法的基本思想，此类算法仅能识别球状聚簇，无法识别任意形状的聚簇。基于密度的经典聚类算法 DBSCAN 算法有效地解决了无法识别任意形状聚簇的问题，但时间复杂度高，面对流数据需要快速计算分析的需求，传统的 DBSCAN 算法就难以胜任，改进 DBSCAN 算法也就非常有必要了。

本章将详细分析 DBSCAN 算法的特性，基于网格思想对 DBSCAN 算法进行改进，提出 GDBSCAN 算法。其次，通过数据点有效时间的概念反映流数据的演化过程，结合 GDBSCAN 算法的特点和 RDD 的内存计算的优势，提出基于 Spark 的并行算法 RDDGD-Stream 算法。此外， RDDGD-Stream 还设计了基于网格数据点数目的重分区方法，平衡集群各节点的计算负载。

4.1 网格和密度

设数据对象集合 D_1，D_2，…，D_t 是分别在 t_1，t_2，…，t_n 收集的 T 时间间隔内的流数据集，其中 $D_i = \{O_1, O_2, \ldots, O_n\}$ 为时刻 t_i 收集的 T 时间间隔内的待聚类的流数据集，其所处 k 维空间区域 S 为 D_i 的计算空间。$O_j = \{x_j^1, x_j^2, \ldots, x_j^d\}$ 为 d 维空间中的数据对象（1≤i≤n），x_j^k 为 O_j 在第 k 维数轴上的投影。

定义 4.1 空间网格[13]：假设一个数据对象有 d 维，每一个数据记录用维度空间定义为 $S = S_1 \times S_2 \times \ldots \times S_d$，其中 S_i 为在第 i 维定义的空间。将每一维的空间 S_i,i=1,2,…,d 划分成 p_j 个分区，整个数据空间 S 被划分成 $N = \prod_{j=1}^{d} p_j$ 个空间网格。定义 $g_i = (a_1^i, a_2^i, \ldots, a_d^i)$ 为空间网格的一个网格单元。

定义 4.2 相邻网格[13]：两个网格单元任意第 i 维是相邻区域，而其他 d-1 维在同一区域。

定义 4-3 ε 邻域：以某一对象 O_i 为中心，在其 ε 半径内的球形 d 维空间称为 O_i 的 ε 邻域[14]。在其 ε 邻域内的任意对象 O_j，有 $dist(O_i, O_j) < \varepsilon$. O_j 称为 O_i 的邻

居。其中，dist 采用欧氏距离计算得出。

定义 4-4 核心对象：对一个对象来说，如果其 ε 半径范围内存在不少于规定数目的邻居，则称之为核心对象[14]，其中规定的数目称为对象 O_i 的密度阈值，记为 MinPts，由用户自己定义。在图 4.1 中，假如 MinPts=6，则 p 是一个核心对象。

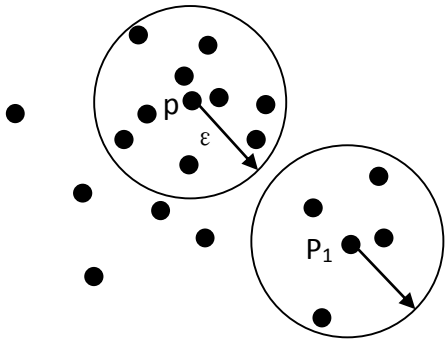
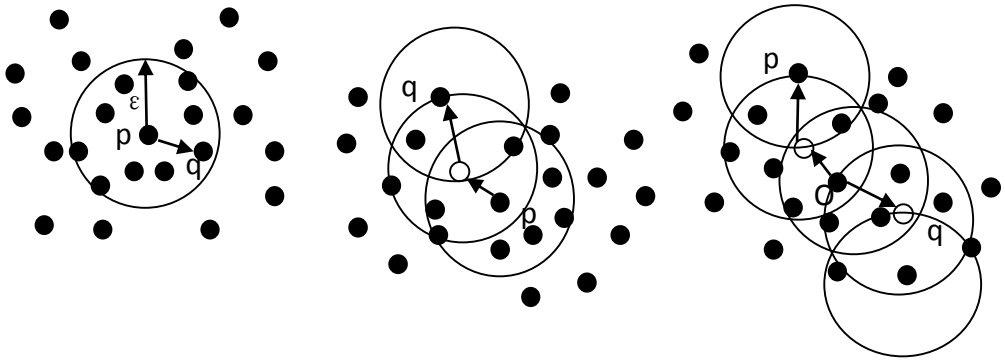


图 4.1 ε 邻域、核心对象

定义 4-5 直接密度可达：在给定的一个数据样本集中，存在两个样本 m 和 n，如果 m 和 n 相距不超过 ε，并且 n 是一个核心对象，那么可以说 n 可以直接密度可达到 m[14]。

定义 4-6 密度可达：若在一个数据样本集合中存在一条对象样本链 $p_1,p_2,\ldots,p_n$，$p_1=p,p_n=q$，并且样本 p_{i+1} 从样本 p_i 是直接密度可达的，那么样本 p 从样本 q 是密度可达的[14]。

定义 4-7 密度相连：若在一个数据样本集合中，有一个样本 h, 样本 p 和样



直接密度可达　　密度可达　　密度相连

图 4.2 直接密度可达、密度可达、密度相连

本 q 都从 h 密度可达，那么样本 p 和样本 q 是密度相连的[14]。

定义 4-8 噪音点：在基于密度的聚类分析算法中，我们把由密度相连的数据样本组成在一起的最大集合称之为一个聚簇。不包含在任何一个聚簇中的数据对象被称为噪音点[14]。

4.2 RDDGD-Stream 算法的总体框架

RDDGD-Stream 算法是受到 DBSCAN 算法和网格方法的启发而设计的，结合 Spark 的 RDD 的内存计算的优势，实现快速有效地流数据聚类，并发现任意形状的聚簇。其流程图如图 4.3 所示，具体流程描述如下：

(1) 在初始化阶段，用传统的空间分割算法[29]将整个可能的数据空间等分成边长不小于 φ*(2* ε)的网格（详情见 4.3 节）；

(2) 初始化结束后，系统可以不断地接收流数据的输入。当流数据输入后，将流数据集中的数据对象根据维度值匹配到相应的网格；

(3) 聚类分析时，对所有网格和保留的数据对象执行基于有效时间的淘汰算法（详情见 4.4 节），筛选出要聚类的网格和其中的数据点，去除掉大量过期数据，可以减少过期数据对近期数据聚类分析结果的影响，也可释放内存空间；

(4) 执行基于网格数据点数目的重分区算法（详情见 4.5 节）：根据集群并行度确定最终分区数目，对初始网格进行合并，使合并后的最终网格内的数据对象尽量均衡，达到平衡集群各节点的计算负载的目的。按照合并后的网格进行分区，一个网格对应一个分区，将各分区分散到集群的不同节点执行计算，同一分区内的数据对象分配在相同的计算节点。

(5) 受到基于网格的聚类方法的启迪，对 DBSCAN 算法进行优化，提出 GDBSCAN 算法（4.6 节详细介绍）并在 Spark 上并行实现该算法，为 RDDGD-Stream 算法实现聚类。首先在各网格（即，最终分区）内执行 DBSCAN 算法，然后记录每个网格中的边界点（即，距离网格边界 ε 范围内的数据对象），之后在分区间即记录下来的边界点上执行 DBSCAN 算法，最后将分区内的结果和分区间的结果合并获得最终聚类结果。采用“区内并行 DBSCAN--分区边界点 DBSCAN--合并结果”的基本思想，生成聚类结果。

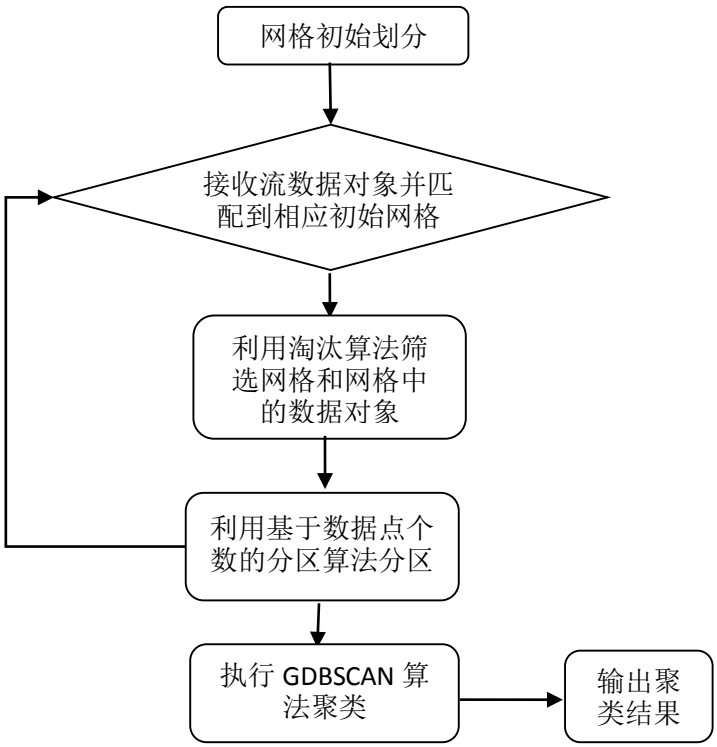


图 4.3 RDDGD-Stream 算法流程图

4.3 数据空间的初始划分

4.3.1 初始划分的基本思路

原 DBSCAN 算法在进行聚类时，需要确定每个数据对象在距其 ε 远的范围内的对象数目是否大于 MinPts（密度阈值）。虽然这样可以获得正确的聚类结果，但是效率会比较低，因为对任意一个数据对象来说，距离它超过了 ε 范围的数据对象没必要进行考察。而且每个数据对象在数据空间中的位置是固定的，很多相距远的数据对象没必要计算，RDDGD-Stream 算法中将整个数据空间先进行初始划分，然后将所有数据对象匹配到相应的网格中。之后进行聚类时，只需对各个网格操作，再对少量的网格的边界点操作就可以，大大降低了算法的时间复杂度。那么如何对数据空间进行划分呢？

整个数据空间由数据集中的每一维度的值的范围来确定。假如数据空间有 d 维，则其可以表示为（[lw_1，hg_1]，[lw_2，hg_2]，…，[lw_d，hg_d]）。在数据空间划分时，每一维度按照一定的间隔进行分割，形成互不相交的网格。我们用树来存储

划分的网格，具体划分步骤如下：

(1) 首先将整个数据空间的维度范围作为根节点 root；

(2) 采用深度遍历的方式，选择树的一个节点进行划分，在此节点的维度范围大于 φ*（2* ε ）的维度中选择最长的维度 i 等分成 k 段，等分间隔为 φ*（2* ε ），其中 $k=|(hg_i-lw_i)/(\varphi*2*\varepsilon)|$（φ>=1，我们称其为等分倍数）。即，第 i 维被分割为[lw_i，$lw_i+\varphi*2*\varepsilon$），[$lw_i+\varphi*2*\varepsilon$，$lw_i+\varphi*4*\varepsilon$），…；对一个核心点 p 来说，其 ε 邻居都在以其为中心， ε 为半径的球形范围内，因此网格大小不能小于 2* ε ，否则即使位于网格中心的核心点，也不能在所属网格中计算出所有的邻居，必须跨网格找邻居点，会增加计算量。

(3) 将(2)中等分的片段分别和剩余维度范围组合，作为此节点的 k 个孩子节点；

(4) 直到此节点的所有孩子节点的维度范围都不大于 φ*（2* ε ），然后依次对其兄弟节点做(2)和(3)操作；

以一个 2 维的数据空间为例，数据空间表示为([10,20],[0,10]), ε =5，φ=1 其初始化阶段生成的网格树如图 4.4 所示：

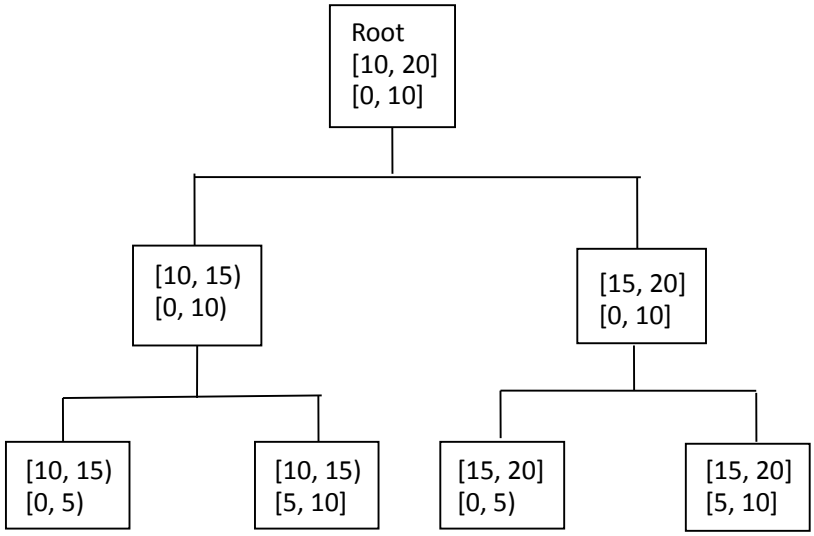


图 4.4　2 维数据空间的网格树示例

4.3.2 初始划分的实现

GridCalculator 类的作用是对数据空间进行初始划分、基于数据点数目合并网格、分配邻居网格等。GridCalculator 类中定义函数 generateDensityBasedGrids()来

实现数据空间初始划分的功能，先定义网格树的根节点 rootGrid，再调用函数 generateTreeOfGrids()生成叶子节点不大于 φ*（2* ε ）的网格树 gridTree，然后调用函数 assignAdjacentGrids()为网格分配邻居网格，最后用 Spark 中的广播函数 sparkContext.broadcast(gridTree)将网格树结构广播给 Spark 集群中的各节点。

在上述过程中最重要的是生成网格树的函数 generateTreeOfGrids()，它是一个递归调用的过程，每次调用函数 splitAlongLongestDimension()选择一个最长的维度等分，通过 filter()算子选择足够大的网格，之后调用 map()算子进行 K-V 分发。map()算子按照 filter()算子选择出的网格为 K 分发任务，每个任务再调用函数 generateTreeOfGrids()继续网格的划分，直到叶子节点的网格都达到规定大小。其实现的伪代码如下：

```
函数：generateTreeOfGrids(rootGrid, partitioningSettings, dbscanSettings)
输入：根网格、分区设置、dbscan 参数设置
输出：网格树结构

val result = new Grid()
result.child = {
        root.splitAlongLongestDimension()
            .filter(_.isBigEnough(dbscanSettings))
            .map(x => generateTreeOfGrids(x,        //递归过程
                        newPartitioningSettings,
                        dbscanSettings,
                        idGenerator
                    ))
}
```

4.4 基于有效时间的数据淘汰算法

在流数据的聚类过程中，数据的时效性对聚类结果的准确性影响较大。流数据集中的数据对象在一段时间内存在是有意义的，过期的数据占用内存且无保留意义。如果仅对当前收集的流数据集聚类，不能反映出流数据的动态演化过程；

如果对所有历史数据一同聚类，会干扰流数据的趋向判断。因此 RDDGD-Stream 算法中提出了数据点有效时间的概念。在有效时间内的数据对象参加聚类，弱化历史数据的影响，反映流数据的演化过程。

定义 4.9 数据点的有效时间 $T_{effective}$：是指数据点对聚类结果有影响的一段时间，也即数据点在内存中可存活的最长时间。

4.4.1 算法思路

为了进一步优化算法，RDDGD-Stream 设计了基于有效时间的淘汰算法，及时删除过期数据，释放内存。淘汰规则如下：

- 网格淘汰规则：记录网格更新时间，如果网格的更新时间与当前时间之差超过有效时间，将此网格中的历史数据清空，此网格不参加聚类计算。一段时间内的高维流数据集对应到相应网格后，大部分网格中数据为空。根据这一特性，可大量剔除无需参加聚类的网格，提高执行速度。
- 数据对象淘汰规则：检测未超过有效时间的网格中的每一个数据点，删除超时数据。

该算法流程图如图 4.5 所示：

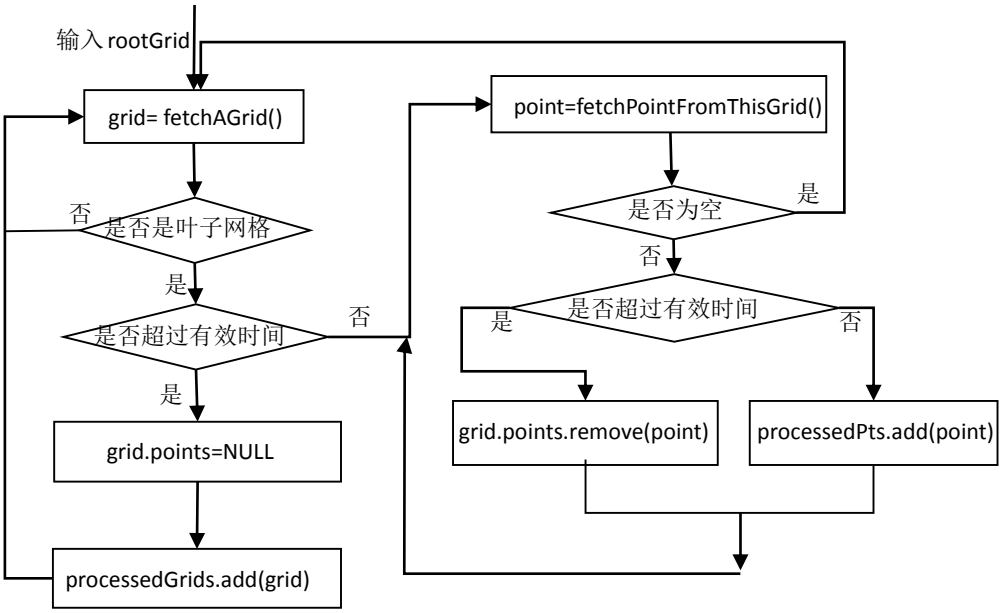


图 4.5 基于有效时间的淘汰算法流程图

4.4.2 算法实现

在 GridCalculator 类中，定义了函数 elimination()。其主要功能是基于有效时间筛选网格和网格中的数据点，删除网格中过期的数据点。

在 elimination()中还调用了函数 isLeaf()和 isExceedEffectiveTime()。函数 isLeaf()的功能是判断网格是否是叶子网格。函数 isExceedEffectiveTime()的功能是判断网格是否超过了有效时间。先通过 RDD 的算子 filter(_.isLeaf()&&_.isExceedEffectiveTime())筛选出是叶子节点并且超过有效时间的网格，将这些网格中的数据点清空。然后通过 RDD 的算子 filter(_.isLeaf()&&!_.isExceedEffectiveTime())筛选出是叶子节点并且还没超过有效时间的网格。然后，通过算子 map()处理前面筛选出的网格中的数据点，如果网格中的数据点已过期，则删除此点，否则将其记录下来。该函数的伪代码如下所示：

函数 elimination()
输入: rootGrid: 初始化阶段生成的均匀网格，包含映射到相应网格的流数据对象 输出: processedGrids: 包含有效时间内的数据点的网格集合 processedPts: 有效时间内的数据点的集合

```
1:    rootGrid.filter(_.isLeaf()&&_.isExceedEffectiveTime())
2:          .map(x=>{_.points = NULL}) //将超过有效期的叶子网格中的数据点清空
3:    rootGrid.filter(_.isLeaf()&&!_.isExceedEffectiveTime())
4:          .map(x=>{ //筛选未超过有效时间的叶子网格中的数据点
5:                _.points .foreach{pt => {
6:                      If currentTime – pt.arrivedTime > T_effective:
7:                            _.points.remove(pt)
8:                       else
9:                            processedPts.add(pt)
10:                     }//end for
```

11: })//end map

4.5 基于网格数据点数目的重分区算法

4.5.1 重分区算法的基本思路

淘汰算法筛选出的网格，大小相等，各网格内的数据分布不均匀。如直接按照筛选出的网格分区，容易使各分区内的数据点分布不均匀。RDDGD-Stream 算法采用了基于网格内数据点数目的分区方法，通过合并相邻网格确保各网格中数据点相对平均，从而使各计算节点负载均衡。

一，由公式(1)得出每个分区的最少处理数目。其中，processedPts 是淘汰算法筛选出的数据点，defaultParallelism 为 Spark 的并行度，在 Spark 中配置。由于分区中的数目不可能绝对平均，在合并网格时，为使各分区数目在平均值附近浮动，采用 defaultParallelism*2。

$$MinNum = \frac{count(processedPts)}{defaultParallelism \times 2} \tag{1}$$

二，根据分区最少处理数目合并相邻网格。迭代处理筛选出的网格，从中选出一个未合并过的网格，如果未达到 MinNum，则从邻居网格中选择一个未达到 MinNum 的网格与之合并；如果网格中的数据点达到 MinNum，则停止此次合并；直到所有的网格都被处理过。

4.5.2 重分区算法的实现

通过 sc.defaultParallelism 可以获得 Spark 环境中配置的集群并行数。然后，计算筛选出的将参加聚类的数据点 processedPoints 的总个数。接下来可以按照公式(1)计算出每个分区最少的处理对象数目。最后，按照分区最少处理数目合并网格。筛选出的网格通过弹性分布式数据集 RDD 的 map()算子分配到各计算节点，每个网格中并行执行相同的处理操作：对每个网格的邻居网格 adjacentGrids 检测是否数据点为空，如果为空从邻居网格中删除，不再参加后续的操作；遍历邻居网格，选择一个未处理过的且数据点数目小于 MinNum 的邻居网格，与当前网格合并，如果合并后网格中的数据点数目大于等于 MinNum，则停止合并，否则

迭代执行继续合并，直到所有的网格都处理完。为合并后的网格列表 partitionedGrids 中的所有网格重新分配网格编号。

Spark 平台提供自定义分区接口 Partitioner，GridPartitioner 类继承自接口 Partitioner。在 GridPartitioner 类中，numPartitions=partitionedGrids.size()定义分区数目，getPartition()定义按照网格编号进行分区。partitionedGrids 实现 GridPartitioner 类，然后通过 mapPartition 接口将网格划分成不同的分区，分配到集群各节点中。

基于数据点数目的重分区实现伪代码如下：

算法 2　基于网格数据点数目的 RDD 分区算法

输入: processedGrids: 通过淘汰算法筛选出的网格集合

processedPts: 通过淘汰算法筛选出的数据点集合

输出: partitionedGrids：根据网格编号划分的分区（一个网格是一个分区）

```
1:    ptsNum = processedPoints.count()   //计算筛选出的数据点的总数目
2:    parallNum = sc.defaultParallelism    //获取 spark 环境的并行数
3:    MinNum = ptsNum/(2*parallNum)   //计算每个分区最少处理对象数目
4:    processedGrids.map(x=> {              ///*  合并网格*/
5:              _.adjacentGrids.foreach{
8:                  x=> {if(_.points==NULL)      //如果邻居网格中的数据点为空
9:                     adjacentGrids.remove(_) //  从邻居网格中删除此邻居网格
10:                  }
11:          }
12:       tempGrid = _      //定义一个临时网格等于当前遍历到的网格
13:    WHILE (临时网格的数据点数小于 MinNum) && (还有未被处理的邻居网格):
14:             找出一个未被处理过的并且数据点数目小于 ptsNum 的邻居网格
15:              将此邻居网格合并入此网格
16:              更新临时网格的数据点数目
17:    END WHILE
18:          添加此临时网格到 partitionedGrids
19:   END FOR
20:   /*  按照合并后的网格分区*/
21:   为 partitionedGrids 中的每个网格重新分配网格编号，并生成继承自 Partitioner
      接口的对象
22:   利用 Spark 中的 MapPartitionWithIndex 接口，生成基于网格编号的 RDD 分区
```

4.6 DBSCAN 算法的优化

如何实时有效地处理流数据，并在流数据中挖掘任意形状的聚簇，是目前数

据挖掘领域的热点研究方向。DBSCAN 算法可以挖掘任意形状的聚簇，有较大的实际应用价值。但其时间复杂度较高，如何改进 DBSCAN 适用于流数据，是本节研究的重点。本节主要介绍对 DBSCAN 算法的改进和并行化的实现，以适应流数据实时处理的要求。

4.6.1 DBSCAN 算法分析

Martin 等人提出的 DBSCAN 算法 (Density-based Spatial Clustering of Application with Noise)是一个经典的基于密度的聚类算法。该算法通过低密度区域将高密度区域隔离开，其中由密度相连组成的数据对象的最大集合（即高密度区域）组成一个聚簇[15]。它的最大优势是可以发现任意形状的聚簇，解决了基于距离计算的聚类算法仅能挖掘圆形或球状聚簇的问题。另外，它还能够在含有噪音点的数据集中有效地进行聚类。DBSCAN 算法的显著优点使得它在实际生活中有广泛的应用前景。然而，DBSCAN 算法也有诸多不足，如时间复杂度高、频繁的 I/O 操作、不适于海量数据的处理等。

DBSCAN 算法通过密度来度量对象间的相似度，其中密度是指单位区域内数据对象的数目，需要计算对象间的距离确定哪些数据对象在其 ε 邻域内。常用的距离度量公式有欧式距离和曼哈顿距离：

- 欧式距离[28]：

$$dist_{(x,y)} = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \ldots + (x_m - y_m)^2} \quad (2)$$

- 曼哈顿距离公式：

$$dist_{(x,y)} = |x_1 - y_1| + |x_2 - y_2| + \ldots + |x_n - y_n| \quad (3)$$

其中：x=（$x_1,x_2,\ldots,x_m$）和 y=（$y_1,y_2,\ldots,y_m$）分别表示数据空间中的两个数据对象。

4.6.1.1 算法描述

DBSCAN 算法的核心思想是：先选择一个未处理过的数据对象 p，计算 p 的 ε 邻域内的对象数目，并确定该数目是否大于规定阈值（MinPts），如是则标记为核心点并新建一个簇 c；选择其 ε 邻域中的其他核心点，继续聚类加入簇 c；迭代执行，直到所有的数据对象点都被访问过。DBSCAN 算法执行流程如下[14]：

输入：	D：一个包含 n 个数据对象的样本集 ε：邻域半径参数值 MinPts：邻域密度阈值
输出：	生成的所有达到密度的聚簇集合

步骤：

1. REPEAT
2. 从数据对象集中抽取一个未被处理过的数据点；
3. IF 这个数据点是核心点 THEN 创建包含此数据点的新聚簇，找出所有从此数据点密度可达的数据对象点，并加入到此聚簇中；
4. ELSE 此数据点是非核心点，跳出此次循环，查找下一个未处理的数据点
5. UNTIL 所有的数据对象点都被处理过

4.6.1.2 DBSCAN 算法存在的问题

DBSCAN 算法是一个典型的基于密度的聚类算法，可以发现任意形状的聚簇并能在带有噪音点的数据集中挖掘出聚簇。在实际应用中，表现出了其他算法无法比拟的优势。但其自身也有很多缺陷，制约着它在商业应用中的长足发展。主要缺陷包括有时间复杂度高和不适于处理海量数据集等。

1) 时间复杂度高。DBSCAN 算法通过判断某一对象邻域内的对象数目是否达到 MinPts 来确定核心对象，并扩展成聚簇。算法过程中需要为所有数据对象计算两两之间的距离，因此整个算法的时间复杂度是 $O(n^2)$，其中 n 是数据集中数据对象的个数。DBSCAN 算法的时间复杂度高的缺陷，制约着 DBSCAN 算法的实际应用前景，其自身优化值得人们进一步研究。
2) 海量数据的聚类效率差。在互联网技术高速发展的时代，网络应用越来越丰富，产生的数据量也越来越大，单机处理能力已无法适应海量数据的聚类分析。DBSCAN 算法面临着同样的问题，处理海量数据时，其内存消耗和频繁的 I/O 操作是无法忍受的。

4.6.2 基于网格的 DBSCAN 算法—GDBSCAN

- **DBSCAN 算法的优化分析**

从 DBSCAN 算法的流程中，可以看出 2，3，4 步耗时最长，要想提升算法的执行速度，只能从 2，3，4 步入手改进。DBSCAN 算法需要先为数据集中所有的数据对象计算两两之间的距离，确定每个数据对象的 ε 邻居数目，选择邻居数目大于 MinPts 的数据对象为核心对象，并扩展此核心对象为聚簇。但是根据核心对象的定义可知，只需为每个对象计算在其 ε 邻域内的数据对象之间的距离即可，无需计算数据集中所有对象之间的距离。我们可以根据这一特点借鉴网格方法的思想对 DBSCAN 算法进行优化，提出了 GDBSCAN 算法。

按照数据空间的定义，每个数据对象在空间中的位置是确定的，所以我们可以按照 4.3 中介绍的方法将数据空间切分成网格，将数据集匹配到对应网格中，这样单个网格中的数据对象可以单独进行聚类，然后再将网格边界点（邻近网格边界 ε 范围内的点）抽取出来做聚类，最后将网格内的的聚类结果和网格间聚类结果合并得到最终聚类结果。如图 4.6 中，如果整个数据空间不划分网格，所有的数据对象间都要计算距离；如果将 2 维的数据空间划分成了 A、B、C、D 四个大于 φ*（2* ε ）的网格，那么在各个网格内的大部分数据只需在网格内计算对象间的距离。从图中我们可以看出，由于一个聚簇中的对象可能处于不同的网格，所以只在网格内计算邻居数目是不精确的。而对一个对象来说，如果它的邻居在相邻网格，那个这个对象必定在距离网格边界 ε 的范围内。所以我们需要将这些边界点（在距离网格边界 ε 范围内的数据点）记录下来，为这些点计算不属于同一网格中的数据点之间的距离，记录不在同一网格内的邻居数目。值得注意的是，在边界点中计算邻居数时，属于同一网格中的边界点即使两者间的距离小于 ε 也不增加邻居数目，因为它们在网格内已计算过一次。然后为每个数据点将网格内的邻居数和网格间的邻居数合并得到全部的邻居数。之后，通过寻找核心点扩展簇时，也是先在网格内查找核心点扩展簇，然后对边界点中不属于同一网格的数据点扩展簇，最后将两者结果合并得到最终的聚类结果。

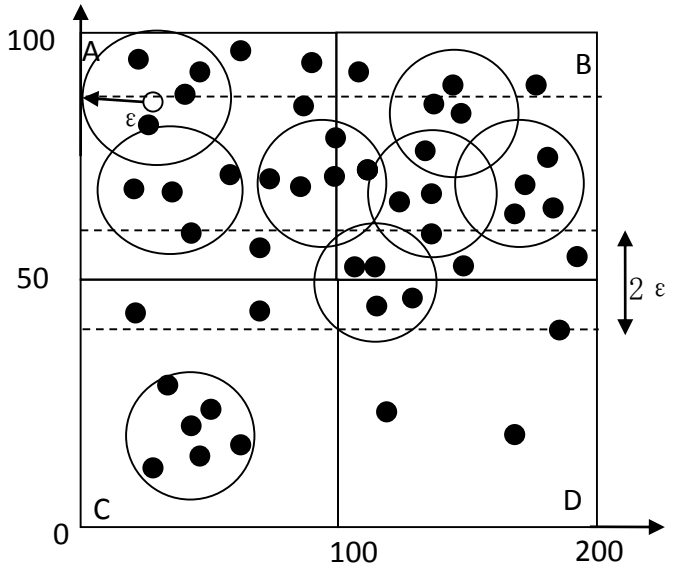


图 4.6 DBSCAN 算法划分网格示意图

- **GBDSCAN 算法的执行步骤**

下面详细介绍优化过的 DBSCAN 算法执行步骤：

第一步，按照 4.3 中介绍的方法将数据空间切分成网格。

第二步，为每个数据点计算邻居点数目：

首先为各网格内的每个数据点计算在网格内的邻居数目，并将每个网格中距离边界 ε 范围内的点（边界点）记录下来。然后，计算边界点中不在相同网格内的邻居点数目。最后将网格内的邻居数目和不同网格内的邻居数目进行合并，获得各个数据点的完整邻居点数目。借助图 4.7 更好的说明此方法，先并行计算 B1、B2 中每个点在分区内的邻居数目，如 4 在 B1 中邻居是点 3 和 9，邻居数为 2；接着，看边界点 4、6、7 是否是邻居，其中 6 是 4 的邻居，所以 4 的分区间的邻居数是 1；合并区内和区间邻居数后，4 的邻居总共是 3 个。

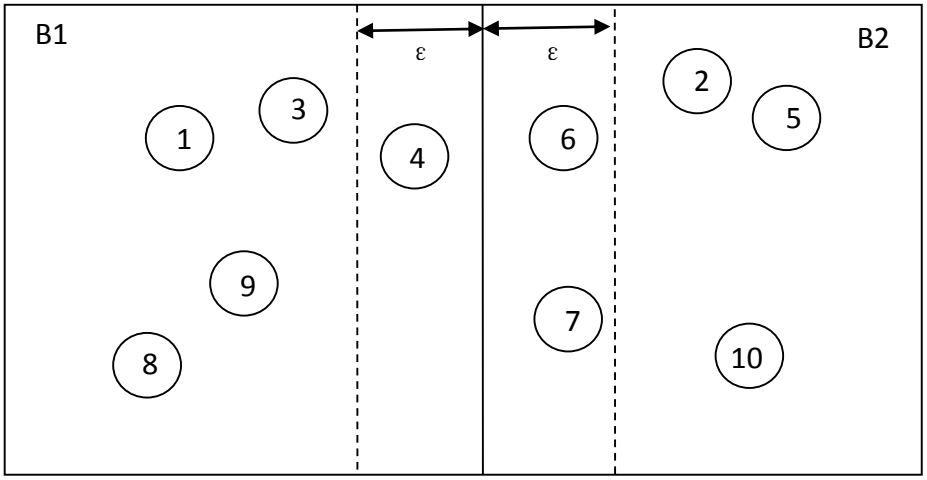


图 4.7 改进的邻居数目计算方法示意图

第三步，在网格内部发现聚簇：

在每个网格内执行此步骤：寻找一个未被处理过的数据点，如果这个数据点是核心点，则创建包含此数据点的新聚簇 C，并在该网格内找出所有从此数据点密度可达的数据对象点，并加入到此聚簇中，直到网格内所有的数据点都被处理过。

第四步，在边界点中（网格间）发现聚簇：

边界点在各个网格中已经被处理过，所以它在网格内的聚类结果是分配给某一个聚簇或者为噪音点。但是每个网格都是单独处理的，位于不同网格中的数据对象可能同属于一个聚簇，因此要对边界点进行两两处理。如果两个数据点不在同一网格中，也不属于同一个聚簇，但距离不大于 ε，则它们可以合并到一个聚簇中，将这两个点的簇标识号组成二元关系（clusterId1, clusterId2）；如果数据对象 A、B 相距不超过 ε，并且点 A 不属于任意一个聚簇，则将点 B 的聚簇标识号分配给 A。

第五步，合并网格内的聚类结果和边界点中的聚类结果：

合并所有的（clusterId1，clusterId2）并且重新为数据点分配簇标识号。如果两个二元关系中有一个相同的簇标识号，将这两个合并，直到所有的对应关系中都不存在相同的簇标识号。例如，（1，3），（2，3），（7，9），（1，5），（7，8），合并后为（1，2，3，5），（7，8，9）。在一个对应关系中的标识号代表的聚簇可以合并为一个大的聚簇。为同一组中的所有数据点重新分配标识号：将对应关系

中的第一个簇标识号，作为同一对应关系中所有数据点的新的簇标识号。

4.6.3 GDBSCAN 算法的并行化思路

在上一节中，我们分析对 DBSCAN 算法流程优化，从 2，3，4 步入手，同样作并行化处理也是如此。在 GDBSCAN 算法的基础上做并行化会方便很多，因为它将数据空间划分成互不相交的网状结构，我们可以按照网格进行分区，将每个分区的数据集分发到集群各个节点，在各分区并行计算邻居数目和聚类，然后再将结果合并。

为了进一步优化并行方案，我们可以按照 4.5 中基于网格数据点数目的重分区方法先将网格进行合并，尽量使最终分区中的数据对象数目相对平均。这样可以平衡集群各节点的计算负载，达到更优的执行效率。

如果数据点个数为 n，传统的 DBSCAN 时间复杂度就是 $O(n^2)$。本算法通过划分区域实现并行化，如果分区数为 m，则时间复杂度为 $O((n/m)^2)$，效率提升了近 m^2 倍。

4.6.4 GDBSCAN 算法的并行化实现

程序利用 Spark 中弹性分布式数据集（RDD）实现内存计算，减少频繁的 I/O 操作，更适于快速高效地处理海量数据和流数据。按照 4.5.2 节中的方法重分区后，可以在分区上并行执行 GDBSCAN 算法。GDBSCAN 算法的主程序 apply()的调用流程如下：调用函数 countNeighborsForEachPoint()，为每个数据点计算 ε 邻居数目；调用函数 findClustersInOnePartition()，在分区内进行聚类；最后，调用函数 mergeClustersFromDifferentPartitions()，对边界点聚类并合并不同分区的聚类结果。此函数中的函数调用流程图如图 4.8 所示。

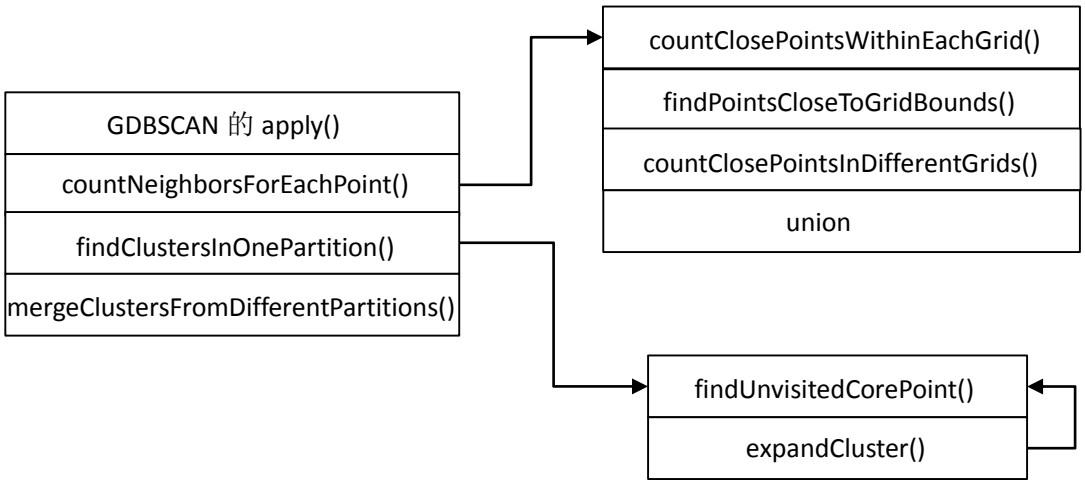


图 4.8 GDBSCAN 的主程序 apply()的函数调用流程图

函数 countNeighborsForEachPoint()的执行过程分为三步：先调用函数 countClosePointsWithinEachGrid()并行地在分区内计算邻居数；然后调用函数 findPointsCloseToGridBounds()找到分区中的边界点；接着调用函数 countClosePointsInDifferentGrids()在边界点中，为不属于同一网格的边界点计算邻居数；最后用 union()算子将前面两步计算的邻居数合并。

函数 findClustersInOnePartition()的功能是在单个分区内进行聚类，通过 mapPartitionsWithIndex 接口将网格分区并分配到集群的计算节点，每个分区并行执行如下操作：调用函数 findUnvisitedCorePoint()从未访问过的数据点中找到一个核心点（ε 邻居数目大于等于 MinPts 的数据点），该核心点为新聚簇的第一个点并将它的 ID 分配为新聚簇的 ID；接着调用函数 expandCluster()扩展这个新聚簇，不断地从邻居中找到 ε 邻居数目大于等于 MinPts 的数据点添加到该聚簇；直到所有数据点都被访问过。

函数 mergeClustersFromDifferentPartitions()的主要功能是将邻居网格中的边界点聚类，组成聚簇 ID 的二元对应关系[(clusterID1,clusterID2)]，然后与分区内的聚类结果合并，并重新为每个数据对象分配聚簇标识号。函数实现的伪代码如下表所示：

函数：mergeClustersFromDifferentPartitions(partiallyClusteredData)
输入：已分区的数据集 partiallyClusteredData

输出：最终的聚类结果

```
val pointsCloseToGridBounds = findPointsCloseToGridBounds(partiallyClusteredData)
val pointsInAdjacentGrids = PointsInAdjacentGridsRDD(pointsCloseToGridBounds)
pairwiseMappings:RDD[(clusterID,clusterID)]= pointsInAdjacentGrids.mapPartitionWithIndex{
      val pairs = HashSet[(clusterID,clusterID)]() //
      for(i <- 1 until pointsInPartition.length){
           val j = i-1
        val pi = pointsInPartition(i)
        while(j>=0 && pi.distanceFromOrigin – pointsInPartion(j). distanceFromOrigin<= ε ){
             val pj = pointsInPartition(j)
             //两个数据点不在同一分区，不属于同一个聚簇，并且距离小于等于 ε
             If(pi.gridId != pj.gridId &&pi.clusterId!=pj.clusterId && calculateDistance(pi,pj)<=
ε ){
                    if(isCorePoint(pi) && isCorePoint(pj)){
                          val (c1,c2) = addBorderPointToCluster(pi, pj)
                          pairs += (pi.clusterId, pj.clusterId)
                     }}
               j-=1
        }
}
      pairs.iterator
   }
   //接下来为重新分配聚簇标识号
   partiallyClusteredData.mapPartitions{
   it => {
        it.map (x => reassignClusterId(x._2, pairs))
}}}
```

4.7 本章小结

本章主要分析 DBSCAN 算法的特点，通过网格方法对其进行改进，提出 GDBSCAN 算法，并分析 GDBSCAN 算法的并行化思路和在 Spark 上的并行化实现。结合 GDBSCAN 算法的特点和 RDD 的内存计算的优势，提出 RDDGD-Stream 算法。该算法提出数据点的有效时间的概念，来反映流数据的演化过程，并提出基于有效时间的淘汰算法，及时释放内存。另外，还提出基于数据点数目的分区算法，均衡集群中各节点的计算负载，进一步提高算法的执行效率。

第5章 实验与实验结果分析

5.1 GDBSCAN 算法的实验设计与结果分析

5.1.1 实验环境与数据准备

本实验在一台 CPU 为 Pentium(R) Dual-Core E5300@2.60GHz、内存为 2G、操作系统为 CentOS 6.5 的普通 PC 机上，运行 C++开发的 GDBSCAN 算法。

实验数据集采用如表 5.1 所示的经典权威数据。数据集 CreditCardClients、BlogFeedBack、ReactionNetwork、HyperPlane、Skin_NonSkin 和 3D_spatial_network 均来自 UCI repository[31]。数据集 KDD-CUP-99[32]是下载的网络攻击类型的经典权威数据。数据集 Sea 来自 KDUS[33]. 本实验在表 5.1 中的 8 个数据集上，通过设置不同的等分倍数 φ 值（分别为 0.5、1、1.5、2、2.5、3、4、6）来验证 GDBSCAN 算法的有效性。

表 5.1 实验数据集信息

数据集名称	记录条数	数据集维度
CreditCardClients	30, 000	24
BlogFeedBack	52, 397	281
Sea	60, 000	3
KDD-CUP-99	60, 000	41
ReactionNetwork	65, 554	29
HyperPlane	100, 000	10
Skin_NonSkin	245,057	4
3D_spatial_network	434, 874	4

5.1.2 等分倍数分析

本实验在 KDD-CUP-99 数据集上，对 GDBSCAN 的等分倍数 φ 设置 8 个值（分别为 0.5、1、1.5、2、2.5、3、4、6）并与原 DBSCAN 算法比较，来测试 GDBSCAN 算法的有效性，10 次实验取平均值结果如图 5.1 所示。从图中可以看到，φ 取值为 0.5 时，GDBSCAN 算法的运行时间比原 DBSCAN 算法的运行时间要长，这是

因为 φ 取值为 0.5 时网格会小于 2* ε，这时即使处于网格中心的点也无法在同一网格内找出所有的 ε 邻居，所有的数据点都会归属于边界点，边界点聚类部分耗时大量增加，再加上所有网格内的聚类耗时，整个算法的运行时间增长；φ 取值为 2、2.5、3 时，GDBSCAN 算法的执行效率是 DBSCAN 算法的近 1.8 倍，φ 取值为 1、1.5、4、6 时，GDBSCAN 算法的执行效率虽不是最优但仍优于 DBSCAN 算法，这和数据的空间分布以及网格划分的大小有关。因为单机运行 GDBSCAN 算法，时间消耗主要由串行的网格内聚类耗时之和以及邻居点聚类耗时组成，而 φ 的取值直接影响网格的划分和邻居点的数量，继而影响算法的运行时间。φ 值大于等于 1 时，算法的执行效率总体上都有了提升，并且选择合适的 φ 值可以使算法的执行效率达到最优。

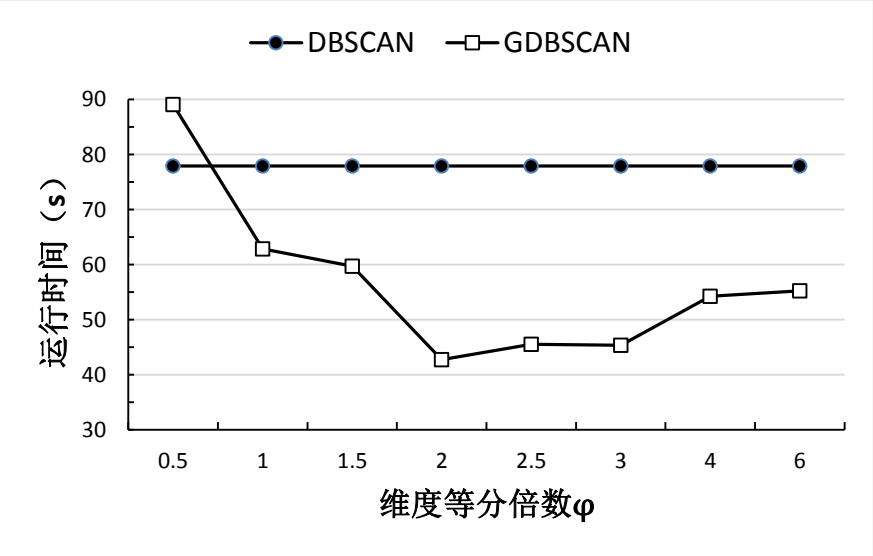


图 5.1 维度等分倍数 φ 对 GDBSCAN 算法的影响

5.1.3 加速比分析

加速比(Speedup)，指的是处理相同任务的两个算法运行时间的比率，用来衡量算法优化的效果。使用 $S_p=T_D/T_G$，其中 S_p 为加速比，T_D 为原 DBSCAN 算法的运行时间，T_G 为优化后的算法 GDBSCAN 的运行时间[34]。

本小节对 GDBSCAN 算法在 8 个数据集上进行加速比分析，实验分别设计等分倍数取 5 个不同值时算法的运行时间，每个时间都是 10 次计算的平均值，由此计算出 GDBSCAN 算法相对于 DBSCAN 算法的加速比，如表 5.2 所示：

表 5.2　GDBSCAN 算法在 8 个数据集上取不同等分倍数的加速比

数据集\等分倍数	1	1.5	2	2.5	3
CreditCardClients	1.33	1.30	1.47	1.41	1.25
BlogFeedBack	1.19	1.70	1.82	1.77	1.63
sea	1.94	2.42	2.23	1.56	1.56
KDD-CUP-99	1.24	1.30	1.83	1.77	1.72
ReactionNetwork	1.31	1.99	1.83	1.96	1.72
HyperPlane	1.59	1.42	2.14	1.76	1.42
Skin_NonSkin	1.76	2.93	3.24	1.98	1.81
3D_spatial_network	2.86	2.77	8.24	6.95	3.82
平均加速比	1.65	1.98	2.85	2.40	1.87

从表 5.2 中，我们可以分析出在等分倍数取 2 时，平均加速比最大，GDBSCAN 算法的执行效率最优。

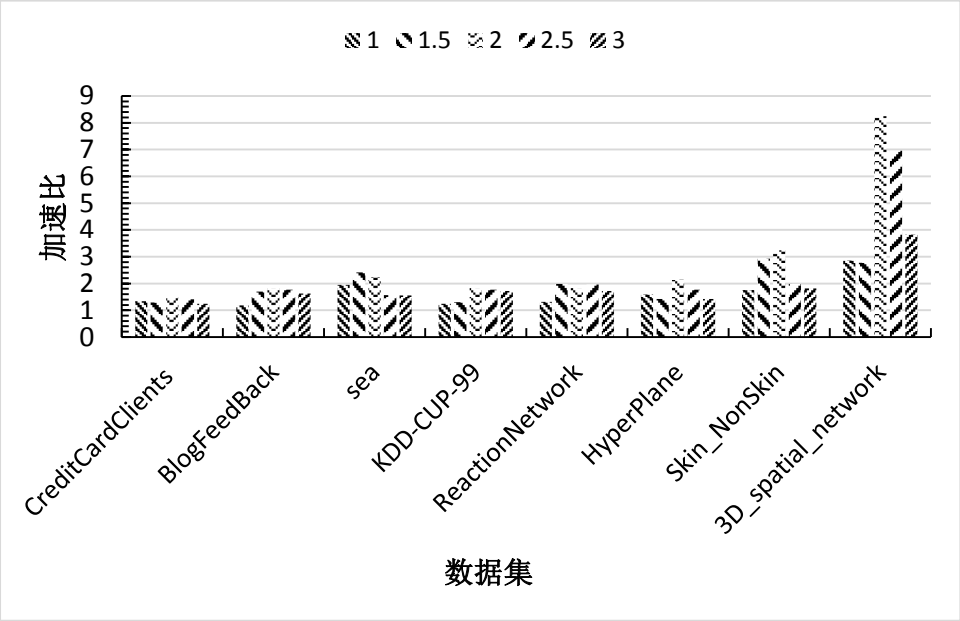


图 5.2 不同数据集上对不同维度等分倍数 φ 的加速比实验结果

为了更清晰直观地获得加速比，图 5.2 为 GDBSCAN 算法的加速比分析图，从图中可以看出：

1) 在等分倍数 φ 大于 1 时，GDBSCAN 算法较 DBSCAN 算法的运行效率都有一定提高；

2) 在处理大数据量时，GDBSCAN 算法的运行效率提高显著。

5.2 RDDGD-Stream 算法的实验设计与结果分析

5.2.1 实验环境与数据准备

RDDGD-Stream 流数据聚类算法是借用分布式计算框架 Spark 来实现的，所以需要准备几台 Linux 机器用以构建实验方法需要的 Spark 集群，安装配置 JDK、Scala、HDFS、Spark 等环境。

5.2.1.1 软硬件环境

本实验中使用的硬件环境是由实验室提供的 4 台普通 PC 机。

1. 硬件配置

- CPU： Intel 2.20GHz
- 内存： 1G DDR2 800
- 网络： 100Mbps/以太网
- 硬盘： 160GB/7200rpm/8MB/SATA

2. 软件环境

- 操作系统：CentOS 6.5
- Java：Java 1.7.0_79
- Scala：Scala 2.10.4
- 文件系统：Hadoop 2.6.0（HDFS）
- Spark 集群：Spark-1.3.1-bin-hadoop2.6

5.2.1.2 Spark 平台搭建

Spark 集群由 4 台计算机组成，一台主节点 hm，其他从节点分别命名为 slave1、slave2、slave3。首先为每台机器安装 Java 和 Scala 环境。

1. 安装配置 SSH 服务

SSH（Secure Shell）为一些网络服务提供安全性的协议，适用于多种平台，可以将数据进行加密，确保数据传输的安全[30]。Spark 集群需要在 master 和 worker 之间通信，如果在传输过程中不断地要求操作人员输入密码显然是不合理的，所以在安装 Spark 平台时，借助 SSH 服务，配置 SSH 的无密码登录，可以加快整个集群中数据传输的速度。本实验中都是使用 root 账号登陆集群各节点执行操作的。SSH 服务的安装配置如下：

- 用 root 账号登陆主节点 hm，并检验集群各节点是否已安装 SSH 服务；
- 如果未安装，可使用命令“yum install openssh-server”安装 SSH；
- 安装成功后，启动 SSH 服务：service sshd start；
- 设置免密码登陆，获得公钥和私钥：

  ssh-keygan –t rsa –P “” ~/.ssh/id_rsa

  在~/.ssh 中生成两个密钥文件，其中 id_rsa 为私钥，id_rsa.pub 为公钥；
- 将公钥 id_rsa.pub 追加到 authorized_keys 文件中：

  cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
- 在集群中的其他节点上重复上述操作。

2. 安装配置 HDFS

Spark 的数据可以存储于 HDFS 中，所以在安装 Spark 前需先配置 Hadoop。在 Hadoop 官网下载 Hadoop 的稳定版本 hadoop-2.6.0.tar.gz，解压并复制到集群的各节点，安装过程比较简洁方便。为了使 Hadoop 正常运行，需要进行相关系统配置，配置主要步骤如下：

- 将 Hadoop 的解压路径配置到“~/.bashrc/”文件中，并使用命令“source ~/.bashrc/”使 Hadoop 命令可随时使用。除此之外，还要设置 Java 和 Scala 的目录；
- 在 hadoop 目录下的 hadoop-env.sh 文件中，配置 JAVA 的可执行文件路径；
- 修改 slaves 文件，添加所有从节点的 IP 地址或机器名；
- 配置 core-site.xml 文件

  配置文件中，fs.default.name 表示访问 HDFS 的地址和端口号，hadoop. tmp.dir

表示存储 HDFS 数据的路径。

- 配置 mapred-site.xml 文件

  此文件用来配置 Hadoop 的 JobTracker 和 TaskTracker 的参数值，包含了主机名称和端口号。

- 格式化 HDFS

  在完成 Hadoop 的相关配置后，使用命令“hadoop namenode -format”对 HDFS 进行格式化。格式化完成后，使用命令“start-dfs.sh“启动 HDFS 文件系统。

3. 安装配置 Spark

Spark 是基于 Scala 语言开发的，因此要先配置 Scala 语言环境。然后再配置 Spark 集群。

- 从官方网站下载 scala-2.10.4.tgz，解压至目录/usr/local/scala。然后在“~/.bashrc/”文件中添加 Scala 的可执行文件路径，通过“source ~/.bashrc/”使之生效；
- 从 Spark 官方网站下载 spark-1.3.1-bin-hadoop2.6.tgz，解压至路径/usr/local/spark/，同样配置 Spark 的执行文件路径和 Spark 的 bin 目录；
- 修改配置文件 conf/spark-env.sh，添加 Java 环境、Scala 环境、Spark 主节点 IP 地址和端口号、Hadoop 环境、Spark 可工作节点的内存大小和 Spark 工作节点的内核等信息；
- 修改 conf/slaves 文件，设置 Spark 系统的 Worker 节点，Master 节点既可以作为主节点也可以作为 worker 节点；
- Spark 中的从节点的配置与主节点配置完全相同，可以将前面的配置文件直接复制到从节点中；
- 在主节点中，运行文件“sbin/.start-all.sh”启动 Spark 系统。

5.2.1.3 实验数据

本实验采用了两组数据，如下：

- DS1 是人工合成的 2 维数据，数据量为 80000（原始分布见图 5.3），此数据集主要用来测试 RDDGD-Stream 算法的演化性。

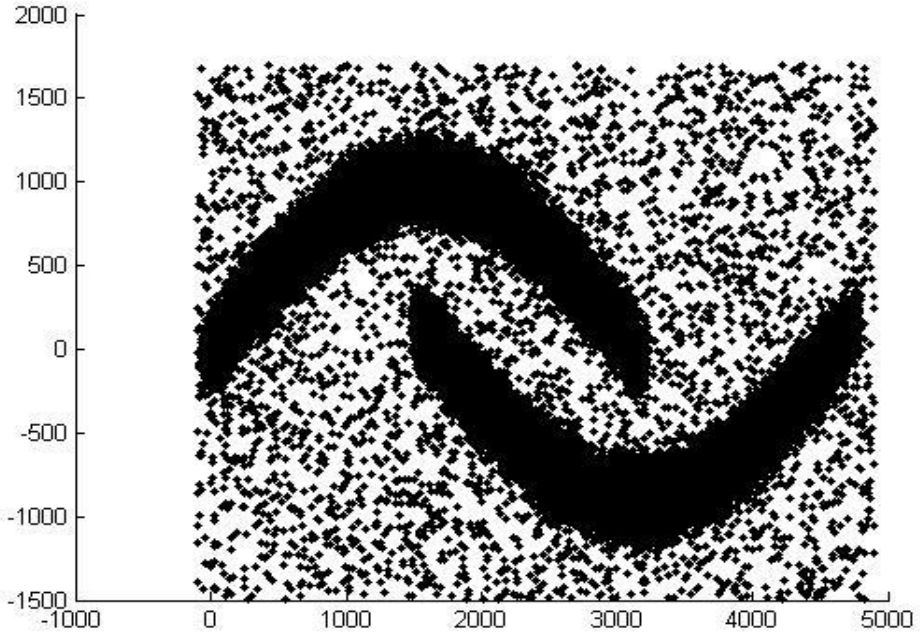


图 5.3　DS1 原始分布图

- KDD-CUP-99 是下载的网络攻击类型的经典权威数据，分为 5 个大类，41 个属性。在做聚类效率测试时，对数据集进行了维度和长度的扩展。首先，数据集中记录条数固定为 60000 条，对数据集的维度分别取 2 维、10 维、20 维、30 维和 40 维做聚类效率实验，测试数据维度扩展对聚类效率的影响。其次，数据集的维度固定为 42 维，对数据集的长度分别取 5000 条、20000 条、40000 条、60000 条和 80000 条做聚类效率的实验，测试数据长度扩展对聚类效率的影响。

5.2.1.4 流数据模拟

在本实验中验证流数据聚类算法 RDDGD-Stream 的演化性、聚类质量和聚类效率，需要模拟实际应用环境，源源不断地产生流数据并将其输入流数据处理中心。为了更接近真实环境开发了一个流数据模拟器。

该模拟器主要是通过 socket 方式监听某一指定的端口号，当访问程序通过 socket 连接该端口并请求数据时，模拟器将定时发送数据给访问程序。该模拟器的具体执行步骤：1) 先指定实验数据文件的物理路径，数据文件中每一行代表一个数据对象点；2)通过 Source.fromFile()读取文件内容，getLines.toList 计算文件

行数，并定义一个变量 lineindex 记录当前读取的数据对象点所在行数；3)使用 listener = new ServerSocket(6600) 新建一个 ServerSocket，并指定访问程序连接端口号为 6600；4) 之后，模拟器保持监听状态，listener.accept()用来接收外部程序连接；5)用 new PrintWriter((socket.getOutputStream(),true)创建一个向外部程序传输流数据的通道；6)通过 Thread.sleep(1)定义每隔 1ms 输出一个数据点，模拟每秒 1000 个数据点的传输速率；7)传输结束后，结束此次传输过程 socket.close()。

5.2.2 演化性测试

本文基于 DS1 以 1000points/s 的速度模拟流数据，验证 RDDGD-Stream 算法的演化性。分析了在 20s、40s、60s、80s 四个时刻 RDDGD-Stream 算法的聚类情况，详见图 5.4~图 5.7。由于算法采用了数据点的有效时间，且基于 Spark 中 RDD 的并行内存计算，所以能实时动态挖掘流数据的演化过程。

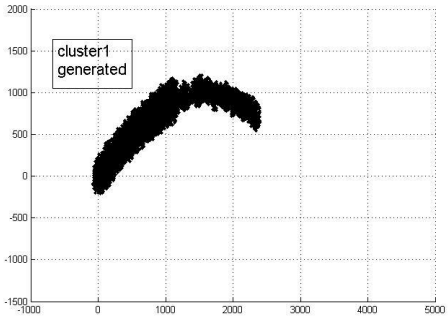


图 5.4 t=20s 时的聚类结果

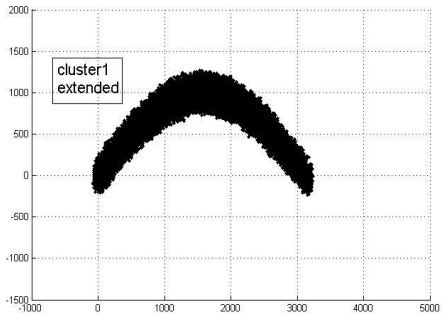


图 5.5 t=40s 时的聚类结果

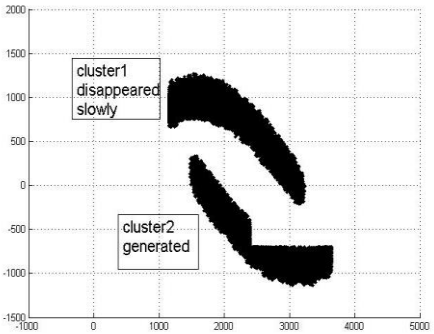


图 5.6 t=60s 时的聚类结果

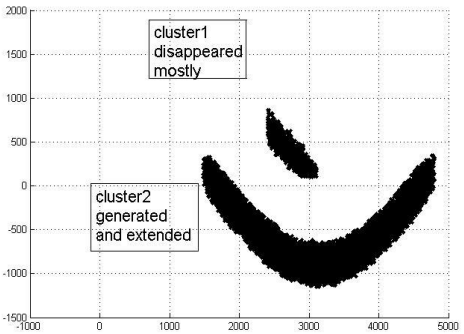


图 5.7 t=80s 时的聚类结果

5.2.3 聚类质量对比测试

采用聚类纯度作为衡量算法聚类质量的一个标准。聚类纯度 $purity = \frac{1}{k}\sum_{i=1}^{k}\frac{|C_i^r|}{|C_i|}$，其中 k 表示簇的个数，$|C_i|$表示第 i 个簇中数据点数目，$|C_i^r|$表示正确划分到第 i 个簇中的数据点数目。RDDGD-Stream 算法和 CluStream 算法在真实数据集 KDD CUP-99 上进行了聚类纯度比较，结果如图 5.8 所示。在 40s、60s、80s 时，RDDGD-Stream 的聚类纯度明显高于 CluStream，这是因为随着新数据不断流入，产生了任意形状且数目未知的聚簇， RDDGD-Stream 不需要预先指定簇的数目，且可以挖掘流数据中任意形状的聚簇，而 CluStream 需事先指定聚簇的数目，且仅能发现流数据中的球状聚簇，RDDGD-Stream 可以将网络攻击类型划分到正确的聚簇中，而 CluStream 误将不同的攻击类型归为一类。因此在数据集 KDD CUP-99 上的对比实验结果表明，本文提出的 RDDGD-Stream 算法比 CluStream 聚类纯度更高。

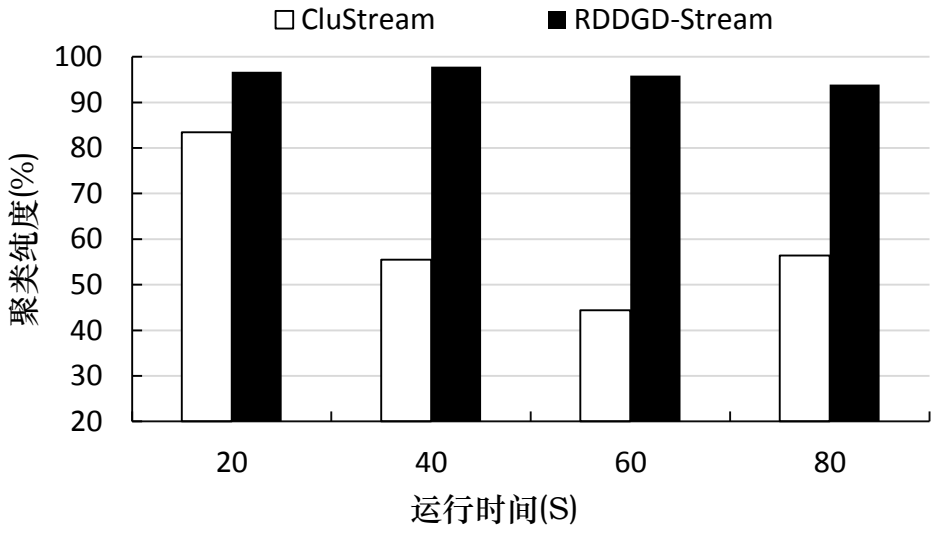


图 5.8 两个算法在 KDD CUP-99 数据集上的聚类质量比较（1000points/s）

5.2.4 聚类效率对比测试

在真实数据集 KDD CUP-99 上，对数据规模和数据维度做了扩展，测试了 RDDGD-Stream 算法的聚类效率。在流数据维度为 42，规模不断增长的情况下，

两算法的效率如图 5.9 所示。CluStream 会随着流数据规模的增大，效率明显降低，RDDGD-Stream 算法的处理时间不会发生大幅增长。随着流数据维度从 2 增长到 40 维，RDDGD-Stream 和 CluStream 的效率比较结果如图 5.10 所示，RDDGD-Stream 算法的效率高于 CluStream 算法。这是因为，CluStream 算法基于复杂的距离计算，随着流数据规模和维度的增长，增加了 CluStream 的计算负载，而 RDDGD-Stream 基于网格进行分区，每个分区独立聚类，然后为少量的边界点聚类，降低了计算复杂度，并且分区间实现了并行计算以及借助弹性分布式数据集实现内存计算，提高了执行效率。

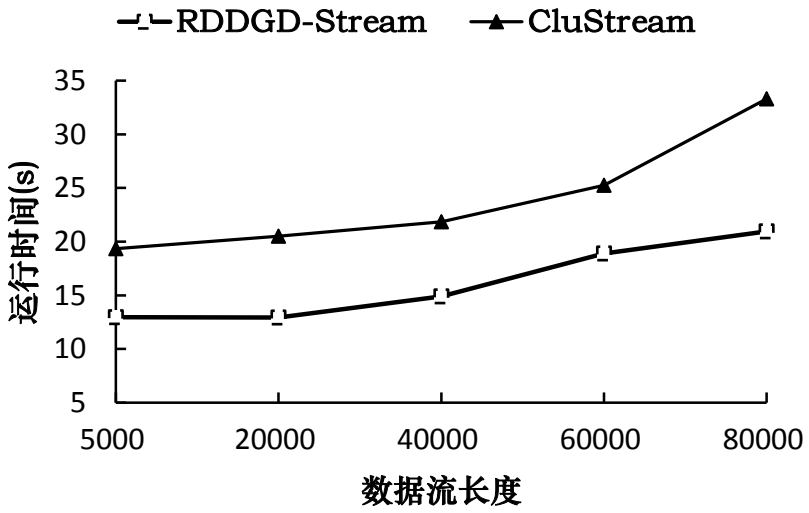


图 5.9 处理时间随流数据长度变化（维度为 42）

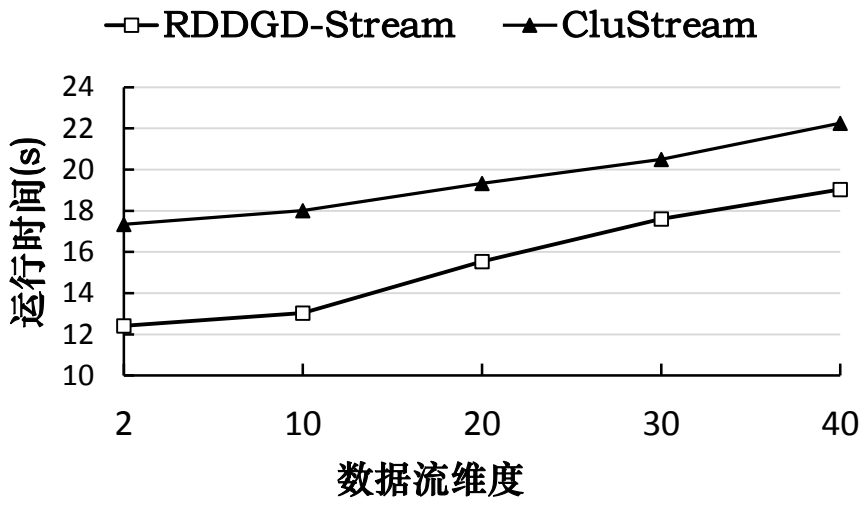


图 5.10 处理时间随流数据维度变化(数据长度为 6 万)

5.3 本章小结

本章设计多组实验，从聚类效率（运行时间和加速比）、演化性、聚类质量等多个方面加以考察，通过实验结果对比分析，检验 GDBSCAN 和 RDDGD-Stream 算法的有效性。实验结果表明 GDBSCAN 和 RDDGD-Stream 算法的执行效率有明显的提高，聚类质量也有一定程度的提高。

第6章 总结与展望

6.1 总结

本文借鉴网格方法的思想改进经典的基于密度的聚类算法 DBSCAN 算法，提出了聚类算法 GDBSCAN，在保留 DBSCAN 算法可以挖掘任意形状的簇的特性的前提下，降低了其时间复杂度。同时，对 GDBSCAN 算法进行了并行化实现的分析。其次，通过数据点有效时间的概念反映流数据的演化过程，结合 GDBSCAN 算法的特点和 Spark 的 RDD 内存计算的优势，本文提出了基于弹性分布式数据集的并行 RDDGD-Stream 算法，用于实时高效地对流数据进行聚类分析。此外，为了进一步提高算法的效率，RDDGD-Stream 还设计了基于网格数据点数目的重分区方法，平衡集群各节点的计算负载。

为了检验 GDBSCAN 和 RDDGD-Stream 算法的有效性，我们设计了多组实验，从聚类效率（运行时间和加速比）、演化性、聚类质量等多个方面加以考察。GDBSCAN 算法的空间划分数量对聚类性能有较大影响，划分数量通过等分倍数来确定，实验中在多个数据集上都设置多个等分倍数来验证 GDBSCAN 算法的有效性，实验结果表明：适当的划分空间有利于提高算法的执行效率，在等分倍数取 2 时，算法效率平均提高了近 3 倍。对 RDDGD-Stream 算法在演化性、聚类质量和聚类效率方面进行了验证，实验结果表明：该算法不仅执行效率高，聚类质量也优于经典流数据聚类算法 CluStream 算法。

本文的主要创新点如下：

(1) 借鉴网格方法的思想改进经典的基于密度的聚类算法 DBSCAN 算法，提出了聚类算法 GDBSCAN 算法，保留了 DBSCAN 算法可以挖掘任意形状的簇的特性，并通过网格方法划分数据空间降低了其时间复杂度。

(2) 研究了 GDBSCAN 算法的并行化实现方案，结合 Spark 平台的内存计算优势，提出了基于弹性分布式数据集的聚类算法 RDDGD-Stream 算法，实现了流数据的实时高效聚类。

(3) 提出了基于数据点数目的重分区方法。在 RDDGD-Stream 算法中，重分区方法基于数据点数目对数据重新分区，均衡集群中各节点的计算负载，从总体上缩短计算时间。

(4) 提出了数据点的有效时间的概念，并提出了基于有效时间的淘汰算法，一方面提高了计算效率，另一方面反映了流数据的动态演化过程。

6.2 展望

如上所述，RDDGD-Stream 算法在多个方面都优于经典流数据聚类算法 CluStream 算法，但该算法仍在多个方面有需要更深入研究的内容，比如：

(1) 智能地动态确定数据点的有效时间，从而更准确及时地反映流数据的动态演化过程。

(2) 等分倍数 φ 的值直接影响算法的聚类效率。目前，这个算法采用的是通过经验确定 φ 的值，更理想的方法是根据输入数据动态地自动确定。

参考文献

[1] 王珊[等]. 数据仓库技术与联机分析处理[M]. 北京：科学出版社，1998：154-156.

[2] 李敏，李英梅. 数据流聚类算法研究[J]. 智能计算机与应用，2014，4(1):13-16.

[3] Jonathan A. S., Elaine R. F., Rodrigo C. Data Stream Clustering: A Survey. ACM Computing Surveys, 2013, 46(1): Article No.13.

[4] Monika R. H., Prabhakar R. and Sridhar R. Computing on data streams. External Memory Algorithms, 1998, 50:107-118.

[5] Shifei D, Fulin W, Jun Q, Hongjie J, Fengxiang J. Research on data stream clustering algorithms. Artificial Intelligence Review, 2015:593-600.

[6] Guha S, Mishra N, Motwani R. Clustering data streams[C]. //Proceeding(s) of 41st Annual Symposium on Foundations of Computer Science. 2000:359-366.

[7] O'callaghan L, Meyerson A, Motwani R, Mishar N, Gha S. Streaming data algorithms for high-quality clustering[C]. //Proceeding(s) of 18th International Conference, Data Engineering. 2002:685-704.

[8] Kiri W, Claire C. Constrained K-means Clustering with Background Knowledge[C]. Proceedings of the Eighteenth International Conference on Machine Learning, 2001:577–584.

[9] Aggarwal C, Han J, Wang J, Yu P. A Framework for Clustering Evolving Data Streams[C]. //Proceedings of the 29th international conference on Very large data bases. 2003:81-92.

[10] Chen Y, Tu L. Density-Based Clustering for Real-Time Stream Data[C]/ /Proceeding of the ACM KDD'07 Conference. 2007:133-142.

[11] Jeffrey D, Sanjay G. MapReduce: simplified data processing on large clusters.//Communications of the ACM, 2008, 51(1):107-113.

[12] Matei Z, Mosharaf C, Tathagata D, Ankur D, Justin M, Murphy M, Michael J, Scott S, Ion S. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing.//Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. 2012:2-2.

[13] Spark, mllib, http://spark.apache.org/docs/latest/mllib-clustering.html#streaming-k-means, 2013.

[14] 毛国君. 数据挖掘原理与算法[M]. 北京：清华大学出版社, 2005.164-166, 185-186.

[15] 邵峰晶，于忠清. 数据挖掘原理与算法[M]. 北京：科学出版社，2009. 175-176.

[16] 朱明. 数据挖掘导论[M]. 安徽：中国科学技术大学出版社, 2012. 138-139，150-151.

[17] Aggarwal C, Han J, Wang J, et al. A Framework for Projected Clustering of High Dimensional Data Streams[C].//Proceedings of the 30th international conference on Very large data bases. 2004:81-92.

[18] Feng C, Martin E, Weining Q, et al.Density-Based Clustering over an Evolving Data Stream with Noise[C]. // Proceedings of the 2006 SIAM International Conference on Data Mining.2006:328-339.

[19] Wei W, Jiong Y, Richard M. STING: A Statistical Information Grid Approach to Spatial Data Mining[C].//Proceedings of International Conference on Data Mining. 1999:116-125.

[20] Yixin C, Li T. Density-Based Clustering for Real-Time Stream Data[C]. //Proceedings of the 13th ACM SIGKDD international conference on Knowledge Discovery and Data Mining, ACM, 2007:133-142.

[21] Armbrust M, Griffith R, et al. A view of cloud computing [J]. Communications of the ACM, 2010, 53(4):50-58.

[22] Tianfield H. Cloud computing architectures[C]//Systems, Man, and Cybernetics (SMC) , 2011 IEEE International Conference on. IEEE, 2011:1394-1399.

[23] Bhardwaj S, Jain L, Jain S. Cloud computing: A study of infrastructure as a service (IAAS)[J]. International Journal of enginerringand information Technology, 2010.

[24] Lawton G. Developing software online with platform-as –a-service technology[J]. Computer, 2008, 41(6):13-15.

[25] 穆玉伟,靳晓辉. Hadoop 高级编程[M]. 清华大学出版社, 2014:55-60.

[26] Carlini E, Dazzi P, Esposito A, et al. Balanced Graph Partitioning with Apache Spark [J]. Lecture Notes in Computer Science, 2014:129-140.

[27] 夏俊鸾等. Spark 大数据处理技术[M]. 电子工业出版社，2015:1-30.

[28] Chowdary N S, Prasanna D S L, Sudhakar P. Evaluating and Analyzing Clusters in Data Mining using Different Algorithms[J]. 2014.

[29] Marsha J and Shahidh B. A partitioning strategy for nonuniform problems on multiprocessors [J]. IEEE Transactions on Computers, 1987, 100(5):570–580.

[30] Barrett D J, Silverman R E. SSH, the Secure Shell: the definitive guide[M]. O'Reilly Media, Inc. , 2001.

[31] K. Bache, M. Lichman, UCI machine learning repository (2013).

[32] KDD dataset. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[33] KDUS. http://www.liaad.up.pt/kdus/products/datasets-for-concept-drift.

[34] Yipu Wu, Jinjiang Guo and Xuejie Zhang. A Linear DBSCAN Algorithm Based on LSH[C]. //Proceedings of the Sixth International Conference on Machine Learning and Cybernetics, 2007: 19-22.

攻读硕士学位期间发表的论文

论文名称	刊物名称	会议类型/检索类型	录用时间	作者排名
A Density-Grid based Clustering Algorithm on Data Stream using Resilient Distributed Datasets.	The 29th Canadian Conference on Artificial Intelligence (CAAI 2016)	Rank3/ EI 检索	2016.03	第一作者

致谢

岁月如歌，光阴似箭，看似漫长的三年研究生学习生涯，转瞬即逝。值此毕业之际，对三年来无私帮助过我的老师、同学和一直支持我的父母献上最诚挚的感谢与最美好的祝愿。

首先，最深的谢意献给我的导师章炯民副教授，感谢您在我读研期间的关心、爱护和悉心指导。章老师渊博的专业知识、活跃的学术思想、严谨的治学态度，精益求精的工作作风、诲人不倦的高尚师德，平易近人的待人方式，使我终生受益。本论文从选题到定稿，每一步都在章老师的悉心指导下完成，倾注了您大量的心血。在算法设计和实验过程中，您不仅帮我指点迷津，开拓思路，寻找更优的解决方案，更帮助我提高了科研水平和实践能力。

其次，衷心地感谢计算机系和章老师为我提供良好的实验室环境和搭建 Spark 环境的机器设备，让我能够潜心研究和实验。

还要感谢实验室的师兄、师姐、同学、学弟和学妹在学业和生活上对我的帮助与关怀。感谢你们对我工作的支持与帮助，感谢你们创造的和谐欢乐氛围，感谢你们和我共同学习，解决技术难题，感谢你们与我一起锻炼身体。感谢师兄蒋耀斌、张诸俊、满毅，师姐卞咏梅，同学周志民、李超，师弟亓麟，师妹贾柯、蒋超群、任南南，和你们共同学习生活的日子是我最珍贵的回忆。

最后，感谢我的父母，感谢你们一直以来的理解、支持和鼓励，感谢你们对我学业和生活无私的关怀，你们的理解和鼓励一直是我前进的动力。