云环境下基于LSH的分布式数据流聚类算法

曲 武[1,2] 王莉军[3] 韩晓光[4]
(清华大学计算机科学与技术系 北京 100084)[1]
(北京启明星辰信息安全技术有限公司核心研究院 北京 100193)[2]
(中国科学技术信息研究所 北京 100038)[3] (北京科技大学计算机与通信工程学院 北京 100083)[4]

摘 要 近年来,随着计算机技术、信息处理技术在工业生产、信息处理等领域的广泛应用,会连续不断地产生大量随时间演变的序列型数据,构成时间序列数据流,如互联网新闻语料分析、网络入侵检测、股市行情分析和传感器网络数据分析等。实时数据流聚类分析是当前数据流挖掘研究的热点问题。单遍扫描算法虽然满足数据流高速、数据规模较大和实时分析的需求,但因缺乏有效的聚类算法来识别和区分模式而限制了其有效性和可扩展性。为了解决以上问题,提出云环境下基于LSH的分布式数据流聚类算法DLCStream,通过引入Map-Reduce框架和位置敏感哈希机制,DLCStream算法能够快速找到数据流中的聚类模式。通过详细的理论分析和实验验证表明,与传统的数据流聚类框架CluStream算法相比,DLCStream算法在高效并行处理、可扩展性和聚类结果质量方面更有优势。

关键词 数据流聚类,位置敏感哈希方法,Map-Reduce框架,DLCStream算法
中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.11.039

Distributed Data Stream Clustering Based on LSH on Cloud Environments

QU Wu[1,2] WANG Li-jun[3] HAN Xiao-guang[4]
(Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)[1]
(Core Research Institute, Beijing Venustech Cybervision Co. Ltd., Beijing 100193, China)[2]
(Institute of Scientific and Technical Information of China, Beijing 100038, China)[3]
(School of Computer&Communication Engineering, University of Science&Technology Beijing, Beijing 100083, China)[4]

Abstract In recent years, with the wide application of computer technology and internet technology in the field of industrial production and information processing, these applications will continuously produce large amounts of sequence data evolved over time and constitute time series data stream, such as internet news feed analysis, network intrusion detection system, stock markets analysis and sensor networks data analysis. The real-time clustering analysis of data stream is a hot issue of the current data stream mining. However, due to the high speed, large-scale data and real-time analysis, data must often be analyzed on the fly. Although the one-pass-through scanning algorithm is able to meet the needs, the lack of efficient clustering algorithms to identify and distinguish patterns limits the effectivity and scalability of this method. In order to solve the above problems, we proposed a novel stream clustering algorithm called DLCStream, which is based on LSH on cloud environments. It is a distributed data stream clustering approach that uses the Map-Reduce framework and LSH mechanism to quickly find the clustering pattern in the data stream. Finally, the theoretical analysis and experiment results illustrate that the DLCStream algorithm results is significantly more efficient in efficient parallel processing, scalablity, and quality of the clustering results compared with traditional data stream clustering framework CluStream algorithm.

Keywords Data stream clustering, Locality sensitive hashing, Map-Reduce frame, DLCStream

1 引言

近年来,随着计算机技术、网络技术在工业生产、信息处理等领域的广泛应用,数据已不仅仅拘泥于文件、数据库等传统的静态形式,一种连续、无界、不定速度的流式数据已经出现在越来越多的应用领域中,而且这些应用领域通常是多数据源的系统,例如入侵监测系统、电子商务、电信、分布式传感器网络、气象监测、科学数据实时分析以及P2P计算等应用场景。在这些应用中,大量的高维数据以较高的速度流向数据收集中心,实时高效地聚类这类数据是一项非常有挑战性

到稿日期:2013-12-19 返修日期:2014-03-21 本文受国家"九七三"重点基础研究发展规划项目基金(2007CB310803),国家自然科学基金重点项目(61035004),国家自然科学基金(60875029),国家科技部博士后基金(2013M541005)资助。
曲 武(1981—),男,博士后,主要研究方向为自然语言理解、云计算、数据挖掘和大数据;王莉军(1978—),女,博士,主要研究方向为自然语言处理、数据挖掘和云计算,E-mail:wanglj@istic.ac.cn;韩晓光(1981—),男,博士,主要研究方向为网络安全、自然语言理解、数据挖掘。

的任务。例如，网络入侵检测系统每秒以 Gb 的速度接收网络流量和系统日志。通过有效的数据聚类算法可以实时地从海量数据中挖掘异常模式。随着数据流速度和规模的提高，分布式数据流聚类算法由于其可扩展性和实时特性能够满足需求。目前主流的分布式数据流聚类的基本框架是每个站点对各自的数据流进行聚类分析，及时得到局部聚类模型；然后传送各个站点间的聚类模型；最后通过分析处理得到全局聚类结果。高效分布式数据流聚类算法，必须解决以下几个问题：

1）数据流可聚类成的簇的个数是不断变化的，因此不能给算法一个固定的值作为参数。

2）算法使用单遍扫描高维数据点形成聚类，无需遍历整个数据集。随着新的数据点到来，算法必须能够维持增量更新。

3）聚类高速、大量的数据流，需要高效的分布式聚类算法来减小通讯负担、存储和计算时间。

基于以上考虑，本文提出云环境下基于 LSH 的分布式数据流聚类算法——DLCStream。通过引入 Map-Reduce 框架和位置敏感哈希（Locality sensitive hashing，LSH）机制，DLCStream 算法可以近实时地高效聚类高速、大量数据流。本文的主要贡献如下：

1）DLCStream 算法不需要事先设定簇的个数，通过 LSH 机制能获得性能较优的聚类簇的个数 X；

2）求解聚类质心过程中，使用 X-中心点方法以避免出现文献[1]中离群点干扰聚类质心的问题；

3）通过引入 Map-Reduce 框架进行数据流聚类，DLCStream 算法能够实现近实时聚类高速、大量的动态数据流；

4）实验验证了本文所提方法的有效性和可扩展性。

本文第 2 节介绍背景知识和相关工作；第 3 节介绍 DLCStream 算法的基本思想；第 4 节对 DLCStream 算法进行性能评估和有效性验证；最后进行总结和展望。

2 背景知识和相关工作

本节首先综述了一些现有的数据流聚类算法，接着对 LSH 方法进行详细的描述，最后介绍了 Map-Reduce 框架及其 Java 实现 Hadoop。

2.1 数据流聚类

目前文献中存在大量的聚类算法，这些算法大体上可分为两类：划分聚类算法和层次聚类算法。划分聚类算法又可以进一步分为基于距离的和基于密度的方法。基于距离的方法主要是在聚类之前计算数据点之间的距离或数据点到簇质心的距离，主要有 K-means 算法[2-4]、直方图相似算法（Similarity histogram-based）[5-7]、EM（Expectation maximization）算法[7]。基于密度的方法主要有基于网格的聚类算法（Grid-based）[8,9]、基于微簇的聚类算法（Micro-cluster-based）[2,3,10]、核密度评估（Kernel density estimation）[11-14]、小波密度评估算法（Wavelet density estimation）[15]、模糊聚类算法（Fuzzy clustering）[16]。以上这些算法主要应用于静态数据集的聚类分析。

与静态数据不同，数据流作为一种数据处理模型由 Henzinger 等人[17]于 1998 年首次提出，他将数据流定义为只能以事先规定好的顺序读取一次的数据序列。随着数据流应用的产生和发展，Golab 等人[18]对 Henzinger 提出的数据流定义进行了修改，数据流是大量的、实时的、连续到达的、潜在无限的有序（到达时间有序或隐含时间戳）数据序列，这些数据或其摘要信息只能按照顺序存取并被读取一次或有限次。与传统的静态数据相比，数据流具有以下特点：

1）高速无限性。数据流通常是源源不断地快速产生，理论上其长度是无限的，在实际应用中远超过系统所能存储的范围，而传统数据库中的数据主要用于持久存储，其存储量和数据更新次数都相对有限。

2）不确定性。数据流产生的速度和间隔时间等统计特性事先难以确定，其产生顺序不受外界控制，数据流的产生速度很有可能超出系统所能接受并处理的限度，而传统数据库中的数据规模和处理能力等性能指标通常是已知的。

3）时变性。数据流随时间而变化，这将引起数据的统计特征也随时间而改变，如数据的方差、分位数、概率分布等，而传统数据库中的数据通常是静态的，一旦存储则很少随时间发生改变。

4）单遍扫描性。由于数据规模大、增长迅速，对数据流仅限于单遍扫描，即除非特意或显式存储外，每个数据只被处理一次。而传统数据库对数据进行持久存储，便于多遍扫描，并建立相应的索引机制以利于高效的查询。

5）并发性。通常应用场合都是多数据源的系统，这对于算法的并发特性要求较高。而传统数据库模式通常是将分布式数据源收集到主数据库中处理，或是使用分布式处理算法处理分布式节点中的静态数据库。

6）结果近似性。大量的数据流分析处理中并非一定需要精确的查询结果，满足精度误差要求的近似结果即可。而传统数据库建立在严格的数学基础之上，其查询语义明确，查询结果一般是精确的。

其中，高速无限性和单遍扫描性是两个最为重要的特点，也是数据流区别于传统数据库中数据的关键。

聚类数据流的算法主要分为两大类：单阶段机制（Single-phase schemes）和两阶段机制（Two-phase schemes）。单阶段机制可以视为一个时间窗口范围内对静态数据进行聚类[19,20]。该方法首先按照数据到来的次序把数据流分块，构造这些块的聚类簇，最后合并这些聚类簇。换言之，单阶段机制遵循分而治之的策略。单阶段机制可以实现对数据流进行聚类，但并不是真正的实时聚类方法；而且，由于该机制对于当前的数据和过期的数据赋予相同的权重，因此不能够获得数据流的演变特征[8]。如果数据流随着时间逐渐演变，这类算法将数据流视为若干段静态数据，不能够发现时序模式。两阶段机制包含一个在线处理组件和一个离线处理组件[2,3,13,21,22]。在线组件处理原始数据流获得数据流的概要信息。离线组件被定期触发，使用在线组件获取的概要信息生成聚类簇。由于最耗时的聚类过程仅仅周期执行，两阶段机制比单阶段机制具有更优的时间效率，因此，两阶段机制被广泛使用在当前的数据聚类算法中。

本文提出的 DLCStream 算法遵循两阶段机制：随着数据的到来，在线组件执行概要获取算法（该算法不作为本文研究

重点),获取特征向量,利用 LSH 算法映射特征向量到位置敏感哈希表结构。离线组件周期运行,用于聚类由在线组件获取的特征向量。

2.2 位置敏感哈希原理

2.2.1 位置敏感哈希算法

LSH 算法首先由 Indyk 等人[23-25]提出,用来解决主存储器中的近邻相似性检索问题,能够证明其对数据规模 n 具有线性时间复杂度。它的关键思想是使用一些哈希函数,确保距离近的点比距离远的点冲突的概率大,当要检索的时候,只需要检索与检索点 q 冲突的点,从而减少了距离计算,加快了检索时间。在文献[23,25]中,作者提出一个以二进制海明距离为度量方式的位置敏感哈希函数,其已经在很多领域中应用,但它有一个明显的缺点:通常距离度量函数都针对欧拉距离,要应用此算法,必须将欧拉距离转换为二进制海明距离,这将增加算法的检索时间和复杂性。为了提高算法的效率和通用性,在文献[24]中,作者提出了基于 P-Stable 分布的位置敏感哈希算法,该算法可以直接处理二次欧拉距离,并解决了 (R,c)-NN 问题;另外,它对高维稀疏数据处理效果很好,特别是当高维向量中非零数据数目一定时,算法的检索时间不变,这个性质是其它算法所没有的,因此用其处理高维稀疏数据时比线性扫描有更大的优势。

对于一个基于距离函数为 D(如欧氏距离、曼哈顿距离等)的点集域 S,一个位置敏感哈希函数族形式化定义如下:

定义 1(位置敏感哈希,LSH) 对于任意数据点 $p,q\in R^d$,函数族 $H=\{h:S\to U\}$ 被称为 (r_1,r_2,p_1,p_2) 对距离函数 $D(\|p-q\|)$ 敏感,满足条件:

$$\begin{cases}\text{if } v\in B(q,r_1) \text{ then } Pr_H[h(q)=h(q)]\geqslant p_1\\ \text{if } v\notin B(q,r_2) \text{ then } Pr_H[h(q)=h(p)]\leqslant p_2\end{cases}$$

一个位置敏感哈希函数族可用,必须满足条件 $p_1>p_2$ 和 $r_1<r_2$。

位置敏感哈希示例如图 1 所示,空间上的点经过位置敏感哈希函数散列之后,对于检索点 q,其 (R,c)-NN 有可能散列到同一个桶(如第一个桶),即散列到第一个桶的概率较大,大于某一个概率阈值 p_1;而其 $(1+\varepsilon)\gamma$ 之外的对象则不太可能散列到第一个桶,即散列到第一个桶的概率很小,会小于某个阈值 p_2。接下来,介绍 L_p 范数下的位置敏感哈希机制。

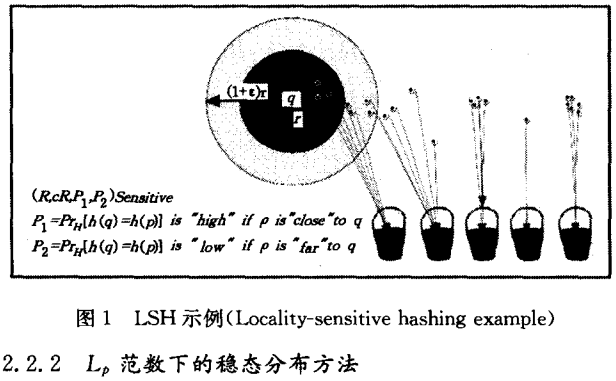


图 1 LSH 示例(Locality-sensitive hashing example)

2.2.2 L_p 范数下的稳态分布方法

本文构造 MLSH 使用的哈希函数族 H 是基于 P-Stable 分布的,$p\in(0,2]$。在概率论中,稳态分布(Stable distribution)又称为雷维偏阿尔法-稳定分布(Levy skew alpha-stable distribution),是一种连续概率分布,它是由保罗·皮埃尔·雷维发展起来的[26]。在稳态分布中,独立同分布的随机变量之和及它们本身具有相同的分布。如果 X_1,X_2 是稳定的并且独立同分布,而且 $Y=aX_1+bX_2+c$ 是两个随机变量 X_1,X_2 的线性组合,那么 $Y=dX+e$。如果对于所有的 a,b 和 $c,e=0$,这称为严格稳态。稳态分布的种类比较多,最常用的稳态分布是高斯分布。P-Stable 分布形式化如下:

定义 2(P-Stable 分布) 若存在 $p\geqslant 0$,对于任意 n 个实数 $v_1,v_2,v_3,\cdots,v_n$,以及服从 D 分布的独立同分布变量 X_1,$X_2,X_3,\cdots,X_n$,随机变量 $\sum_i v_iX_i$ 与随机变量 $(\sum_i|v_i|^p)^{1/p}X$ 同分布,其中 X 是一个服从分布 D 的随机变量,则称 D 为 R 上的 P-Stable 分布。

对于任意的 $p\in(0,2]$,稳态分布是存在的,特别是:

1)柯西分布 D_c,密度函数为 $c(x)=\frac{1}{\pi}\frac{1}{1+x^2}$,为 1 稳态分布;

2)高斯分布 D_G,密度函数为 $g(x)=\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$,为 2 稳态分布。

从实用的角度讲,尽管缺少闭合形式的密度分布函数,在 $[0,1]$ 上,可以通过两个相互独立、服从均匀分布的变量来生成 P-Stable 随机变量。利用 P-Stable 分布可以有效地近似高维特征向量,并在保证度量范数的同时,对高维特征向量进行降维,其关键思想是,产生一个 d 维的随机向量 a,随机变量 a 中的每一维随机、独立地从 P-Stable 分布中产生。对于一个 d 维的特征向量 v,如在定义 2 中一样,随机变量 $a.v$ 具有与 $(\sum_i|v_i|^p)^{1/p}X$(其中 X 是满足 P-Stable 分布的随机变量)一样的分布,因此可以用 $a.v$ 表示向量 v 来估算 $\|v\|_p$,很容易得出 $a(v_1-v_2)=a.v_1-a.v_2$。

2.3 Map-Reduce 框架

Map-Reduce[27]是一种编程模式,它与处理或产生海量数据集的实现相关。用户指定一个 Map 函数,通过这个 Map 函数处理 key/value(键/值)对,并且产生一系列的中间 key/value 对,使用 Reduce 函数来合并所有的具有相同 key 值中间键值对中的值部分。使用这样的函数形式实现的程序可以自动分布到一个由普通机器组成的超大集群上并发执行。Run-time 系统会解决输入数据的分布细节,跨越机器集群的程序执行调度,处理机器的失效,并且管理机器之间的通讯请求。这样的模式允许程序员不需要并发处理或者分布式系统的经验,就可以处理超大的分布式系统的资源。Map-Reduce 系统的实现运行在一个由普通机器组成的大型集群上,并且有着很高的扩展性:一个典型的 Map-Reduce 计算处理通常分布到上千台机器上来处理 TB 上的数据。使用者提供的 Map 和 Reduce 函数有着如下相关类型:

$$\begin{cases}Map(k_1,v_1)\to list(k_2,v_2)\\ Reduce(k_2,list(v_2))\to list(v_2)\end{cases}$$

也就是说,输入的键值和输出的键值是属于不同的域。进一步地讲,中间的键值和输出的键值是属于相同的域。

Hadoop 是 Map-Reduce 框架的 Java 实现。它将应用切分成许多子任务集合。每个子任务可以在任何集群节点上执行。同时,Hadoop 还提供一个容错的分布式文件系统 HDFS,它能够自动处理节点故障。本文是基于 Hadoop 环境实施 DLCStream 算法,下文将会详细讨论 DLCStream 算法的细节。

3 DLCStream 算法

3.1 符号定义说明

DLCStream 算法，即云环境下基于 LSH 的分布式数据流聚类算法，该算法为分布式的、两阶段的数据流聚类算法。首先将调用数据流概要算法获取特征向量数据点，然后使用 LSH 算法进行聚类。表 1 列出文中 DLCStream 算法涉及的重要符号和定义。

表 1 文中涉及的重要符号和定义

d	特征向量的维度
p_i	第 i 个特征向量，$p_i=\langle x_1, x_2, \cdots, x_{d-1}, x_d\rangle$，通常也表示为 o_i
$F_{syn}(x)$	获取数据流的概要描述函数，其输出为 d 维的向量，x 为数据流
b	LSH 桶，聚类簇单元
X_t	LSH 返回可变的聚类簇数，主要依赖于 LSH 表中簇的总数、相似性和构建 LSH 的参数
X_{max}	从 X_t 中选出包含数据点最多的前 X_{max} 个聚类簇作为候选聚类簇 C_{can}
C_{can}	表示候选聚类簇
C_{cen}	聚类簇质心
d_{co}	距离范数相关系数，$d_{co}=\lambda^{(t_c-t_a)}$，λ 为衰减因子，$t_c$ 为当前时间，t_a 为数据点到达时间
d_f	新到的数据流特征向量 p_i 到聚类簇质心 C_{cen} 的距离范数
d_t	随着时间衰减的距离范数，$d_t=d_f\times d_{co}$
d_{min}	新到的数据流特征向量 p_i 到聚类簇质心 C_{cen} 的最小 d_t
d_{th}	数据点到聚类簇质心的距离阈值

3.2 DLCStream 算法流程

这部分将讨论算法流程，如图 2 所示。DLCStream 算法包含一个在线组件和一个离线组件。当一个新的数据点到来时，在线组件被执行。首先调用概要获取函数 $F_{syn}(x)$ 来获取 d 维数据流特征向量 p_i，利用 LSH 算法将 p_i 投影到位置敏感哈希表相应的桶 b 中，最后返回 X_t 个聚类簇。从 X_t 中选出包含数据点最多的前 X_{max} 个聚类簇作为候选聚类簇。而离线组件以一定的时间间隔周期并发执行。调用 X_{max}-median方法获得聚类簇质心 C_{cen}，分别计算新到的数据流特征向量 p_i 到 X_{max} 个质心之间的距离范数，获取最小距离范数 d_{min} 的聚类簇，若 $d_{min}\leqslant d_{th}$，则将 p_i 归为 d_{min} 对应聚类簇，否则建立新的聚类簇。离线组件通过使用 Map-Reduce 框架实现分布式计算。聚类过程的并发执行很大程度上减少了计算时间，特别是对于海量高维数据具有较好的可扩展性。

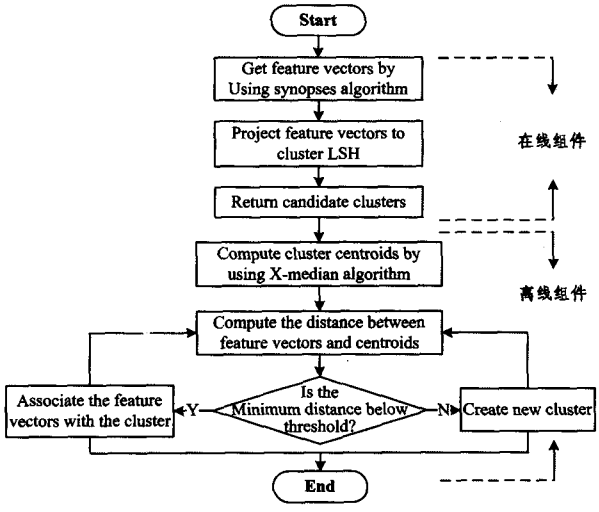


图 2 DLCStream 算法流程

3.2.1 在线组件

在线组件包括 3 个阶段（见图 2）。

1）为获取对数据流的概要描述，本文使用文献[18]中提出的滑动窗口模型对数据流进行分析。其基本思想是仅仅基于最近的数据做出聚类决策。即，在每个时刻 t，一个新的数据点到来时间为 t_a，t_c 为当前时间，λ 为衰减因子，使用距离范数相关系数 $d_{co}=\lambda^{(t_c-t_a)}$ 来控制数据点聚类过程。

2）LSH 映射阶段，选择 l 个函数，$g_j(p_i)=(h_{1,j}(p_i),\cdots,h_{k,j}(p_i))$，从函数族 H 中以独立、一致随机的方式来选择 k 个 $h_{t,j}(1\leqslant t\leqslant k, 1\leqslant j\leqslant l)$，然后使用这些函数对数据点进行哈希。将特征向量集合中每一个数据点 p_i 映射到桶 $g_j(p_i)$ 中，$1\leqslant j\leqslant l$，构建位置敏感哈希数据结构。由于桶的总数过大，不可能精确存储所有可能的桶，因此仅存储非空桶。通过该阶段的 LSH 映射（一共 $l\times k$ 个 LSH 函数），可以得到 l 个 k 维映射值。

3）二次哈希阶段，对 k 维的映射值进行二次哈希，得到一维索引值。二次哈希采用 MD5 算法，一方面可以把一个任意维的数据映射成较短的一维数据，降低了存储代价；另一方面 MD5 算法的冲突概率较低，可以保证第一阶段产生的不同的 k 维映射值仍然被映射到不同索引值上。每个索引值构成一个桶，将特征向量 p_i 的 HDFS 物理地址链接到桶上，形成一个倒排表。

通过阶段 2）和 3），LSH 方法为特征向量集合提供了一个索引结构来确定近似最近邻。对于特征向量 p_i，LSH 方法利用哈希表返回相对较小的候选聚类簇，这些聚类簇表示与目标特征向量（例如，聚类簇质心）以较高概率相似的特征向量集合。LSH 返回可变的聚类簇数 X_t，这个参数主要依赖于 LSH 表中簇的总数、相似性和构建 LSH 的参数。为确保有限的处理时间，从 X_t 中选出包含特征向量最多的前 X_{max} 个聚类簇作为候选聚类簇 C_{can}，C_{can} 的选择使用 Top-k 算法，最后返回候选聚类簇 C_{can}。

3.2.2 离线组件

离线组件采用 Map-Reduce 框架进行分布式聚类，包含两个阶段（见图 2）。

1）获取在线组件生成的聚类簇，计算各聚类簇的质心。由于 k 均值方法对于离群点是敏感的，而且 LSH 方法仅仅是以一定概率保证的近似相似性方法，因此，一个具有很大极端值的特征向量很可能扭曲数据的分布，k 均值方法中平方误差函数 $E=\sum_{i=1}^{k}\sum_{p\in C_i}|p-m_i|^2$，更加恶化了这一影响。本文采用 X-median 算法[28]，不采用簇中对象的均值作为参考点，而是在每个簇中选出一个实际对象来代表该簇。其余的每个对象聚类到与其最相似的代表性对象所在的簇中，划分方法使用绝对误差标准（Absolute-error criterion，AEC），$E=\sum_{j=1}^{k}\sum_{p\in C_j}|p-o_j|$。其中，$E$ 是特征向量集合中所有对象的绝对误差之和，p 是代表簇 C_j 中的一个给定对象，o_j 为簇 C_j 中的代表对象。

2）计算新到的特征向量到质心的距离范数，可以使用距离范数公式。例如，令一个 n 维特征向量分别为 $o_i(x_1, x_2, \cdots, x_n)$，质心为 $c_{cen}(y_1, y_2, \cdots, y_n)$，1 范数定义为 $\sum_{i=1}^{n}|x_i-y_j|$，

2 范数定义为$(\sum_{i=1}^{n}|x_i-y_j|^2)^{1/2}$，夹角余弦 $\cos(\theta)=\sum_{i=1}^{n}x_k y_k/\sqrt{\sum_{i=1}^{n}x_k^2}\sqrt{\sum_{i=1}^{n}y_k^2}$ 等。使用公式 $d_t=d_f\times d_\omega$ 计算随着时间衰减的距离范数，求解最小的 d_t，即 $d_{\min}=\min(d_t)$。若 $d_{\min}<d_{\text{th}}$，将新到的特征向量归为该质心所在的簇，否则为该特征向量建立新的簇。

3.3 基于 Map-Reduce 的 LSH 创建和聚类算法

当数据流的流速越来越快，数据量越来越大时，集中式的 LSH 创建和聚类都会因内存限制而变得越来越慢，甚至不可行。本文利用 Hadoop 平台提供的 Map-Reduce 计算框架，将 LSH 结构创建和聚类过程分布化和并行化，以适应海量高维、高速数据的聚类需求，基于 DLCStream 算法的聚类系统架构如图 3 所示。DLCStream 算法主要由两个子算法组成，分别为基于 Map-Reduce 的 LSH 结构创建算法 MRLSH-Create 和基于 Map-Reduce 的聚类算法 MRClu。

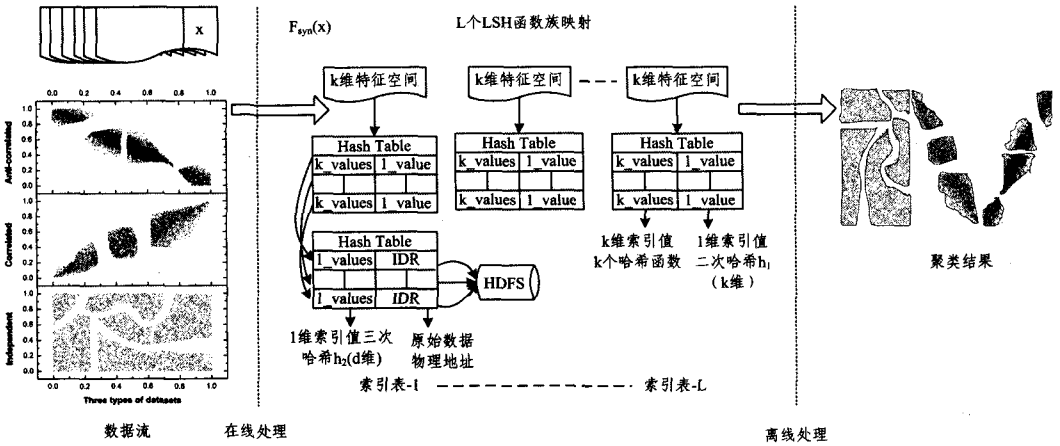


图 3 基于 DLCStream 算法的聚类系统架构

3.3.1 基于 Map-Reduce 的 LSH 结构创建算法

当数据流的流速较快，数据量变得较大时，可利用 Map-Reduce 框架分布式创建 LSH 结构。下面的 MRLSHCreate 算法中每个 Map 任务的计算时间相对于 Map 的启动时间来说不大。基于 Map-Reduce 的 LSH 结构创建算法 MRLSH-Create 如算法 1 所示。

算法 1 MRLSHCreate 算法

输入：参数文件 lsh. conf，数据流时间片段集合 DS，任务参数 jobConf；

输出：LSH 结构和 Top-X_{max} 个候选聚类簇 C_{can}；

```
BEGIN
//初始化 MapReduce 任务
InitMapReduce(DS,lsh. conf,jobConf);
//对 DS 中的每个数据流片段启动一个 Map 任务，设置 Reduce 任务的个数为 L，输出为空；
Map(key1,value1,key2,value2)
  {
//从 jobconf 中获得算法参数，初始化 LSH 函数族；
   InitLSHFamily(jobConf,LSHFamily[L][k]);
//解析数据流片段，即 value1，获取特征向量 pi；
  ComputeFsyn(value1);
   ReadFromHDFS(key1,pi);
//利用 LSHFamily 计算出 L 个索引值；
   ComputeIndexValue(LSHFamily[L][k],IndexValue[L]);
//输出 L 个(key2,value2)=〈IndexTable_i,Indexvalue[i]〉
  Output(key2,value2,IndexTable_i,IndexValue[i]);
  }
Reduce(key3,value3,key4,value4)
  {
  //将 Map 的输出按照 IndexTable_i 进行合并，即(key3,value3)=(IndexTable_i,List[IndexValue[i]]);
   Combine(dist,List〈HDFSAddr〉);
  //将 List[IndexValue]按照不同的 IndexValue 组织成哈希表；
   OrganizeHashTable (List [IndexValue], Hashtable (IndexValue));
  //将哈希表写入对应的索引表 IndexTable_i 中；
  }
END
```

3.3.2 基于 Map-Reduce 的聚类算法

针对一个较长时间段数据流的聚类，由于数据量变得很大，聚类特征向量需要迭代获取聚类质心和距离计算，时间复杂度较高，因此可利用 Map-Reduce 机制启动 Map 任务并行化获取聚类簇质心，在 Reduce 阶段将特征向量分配到相应的候选聚类簇中，利用新到的特征向量到聚类簇质心的精确距离进行筛选。MRClu 算法流程如算法 2 所示。

算法 2 MRClu 算法

输入：新到数据流的特征向量 p_i，从 LSH 表中获得的 X_{max} 个聚类簇，任务参数 jobConf；

输出：将 p_i 分配到相应的聚类簇中形成新的聚类簇，并更新该聚类簇的质心，输出当前所有聚类簇；

```
BEGIN
//从 jobConf 中获取参数，初始化 LSHFamily；
  InitLSHFamily(jobConf,LSHFamily[L][k]);
//对聚类簇中的每个特征向量 pi 启动一个 Map 任务，设置 Reduce 任务的个数为 Xmax；
Map(key1,value1,key2,value2)
{
  //获取聚类簇的质心，key1 和 key2 分别为 CluId. pi,CluId. cen，表示聚类簇 Id 为 CluId 中的特征向量和质心标识，value1 和 value2 为特征向量和质心值；
   ReadFromHDFS(key1,value1,key2,value2);
  //利用绝对误差标准(Absolute-error criterion, AEC)函数计算聚
```

```
    类簇质心;
    AECFun(key1,value1,key2,value2);
  //输出 Xmax个(key2,value2)=<CluId. pi,Evalue>
    Output(key2,value2,CluId. pi,Evalue);
}
Reduce(key3,value3,key4,value4)//Reduce 任务个数为 1;
{
  //将 Map 的输出按照 CluId 进行合并,对于每个 CluId 中的最小 E
    值对应的质心为最优质心,更新该 CluId 的质心;
    Combine(CluId,List<E>);
  //排序得到簇中最小的质心,输出簇标识 CluId 和最优质心 Opti-
    malCen;
    SortAndOutput(CluId,OptimalCen);
}
  //计算新到数据流的特征向量 pi 到簇质心集合的距离,将 pi 归到
    最小的簇中,更新质心;
    Distance(pi,OptimalCenSet,dco);
    Integrate(pi,CluId);
    UpdateCen(CluId);
END
```

4 实验验证与性能分析

4.1 实验环境

本文通过详细的理论分析和实验验证来评估DLCStream算法的性能。实验过程中,使用2-norm进行距离计算。接下来,主要介绍DLCStream算法的实验评估细节,然后基于实验结果,评估DLCStream算法的性能和局限性。在实验过程中,由于分布式环境运行了其他计算服务,本次实验是与其他程序共享分布式资源,因此,在实验结果上可能会出现小幅波动。分布式环境构成:一个Master管理节点服务器,8个刀片服务器(HS21)计算节点,一套8T磁盘阵列(IBM DS3400)存储。具体软硬件配置如下:

1)Master管理节点服务器:两颗英特尔四核至强E5420(2.5GHz/EM64T,12MB L2缓存),配置8G PC2-5300 CL5 ECC DDR2 667MHz内存;3块146G硬盘。

2)刀片服务器(HS21)计算节点:两颗四核Intel Xeon E5430(2.66GHz,12MB L2,1333MHz FSB,80W),8GB(4*2GB)PC2-5300 FB-DIMM内存,1块146GB 10k SFF SAS Fixed HDD,双千兆以太网。

3)磁盘阵列:DS3400磁盘柜Single Controller,8*1T。

4)软件系统:操作系统,RadHat Linux 5.3,64位;开发环境为NetBeans IDE 6.5和Karmasphere Studio;分布式编程工具为Apache Hadoop;Hadoop-0.20.203.X。

4.2 DLCStream算法参数、性能权衡及评价标准

令数据集规模为n,维度为d,近似因子为$c=(1+\varepsilon)$,计算节点的个数为R,投影个数为k,哈希函数个数为L,投影区域宽度为w。除非特别指出,本文使用的数据流速v设定为每秒300个数据点,衰减因子$\lambda=0.97$,$L=30$,$k=10$,$\varepsilon=1$,$w=4$,$R=8$,$n=2.1\times10^4$和$d=34$。

在数据流聚类中,距离平方和(Sum of Square Distance,SSQ)和聚类纯度是常用的评估指标。此外,本文还选用聚类簇的个数与基准数据的差异来评估算法的性能[29]。距离平方和定义如下,若当前时间为t_c,对于确定的时间基准t,时间间隔$t-t_c$内,有N个数据点到来,这些数据点中,有N'个数据点被分配给一些聚类簇,$N-N'$个数据点为离群点。对于每个数据点p_i,可以找到距离它最近的聚类簇的质心C_{p_i},计算点p_i和质心C_{p_i}之间的距离$d(p_i,C_{p_i})$,对于N'个数据点,在时间点t_c的$SSQ=d^2(p_i,C_{p_i})$。即通过计算所有点到各自的聚类中心的距离的总和来衡量算法所给出的聚类质量,SSQ值越小,说明算法聚类质量越好。聚类纯度定义如下,若聚类簇C_i的大小为n_i,对该簇的纯度定义为$S(C_i)=\frac{1}{n_i}\max(n_i')$。其中,$n_i'$表示聚类簇$C_i$与基准第$i$类的交集大小,整个聚类的纯度定义为:$Purity=\sum_{i=1}^{k}\frac{n_i}{n}S(C_i)$。其中,$k$为聚类最终形成的聚类簇的数据。聚类纯度刻画了聚类算法分类的准确性,一般纯度越高,聚类算法越有效。聚类簇数目也是评估数据流聚类算法性能的一个指标,定义为一定的时间周期$t-t_c$内,聚类结果中聚类簇的数目。

4.3 实验数据集

实验中选用的数据集来自MIT林肯实验室入侵检测数据集[30]。原始训练数据集为4GB压缩的tcpdump(Unix系统中捕获数据包工具)采集的7周网络通信数据,处理后包括500万条链接记录。同样,使用两周的测试数据集产生200万条链接记录。在本文的实验中,使用该数据集的子集,18000个数据点,每个数据点表示一个时间段数据流的特征向量,即表示一个链接。每个数据点包括34个属性,具体细节参照文献[3]。链接类型主要有5类,包括正常链接和4类攻击,攻击分为DOS攻击、R2L攻击、U2R攻击和针对扫描攻击(Probing)。每种攻击类型进一步又分为若干类。整个数据集包含22个子类。本文使用的子集包含8个子类。对于选定的数据集,首先对各属性进行归一化处理,归一化因子为$\theta_k=\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_{i,k}^2}$。使用$\theta_k$将数据点$p_i$的第$k$维归一化,若归一化后的维度值大于1,默认设置为1。这将会很大程度上减小离群点对于DLCStream算法的影响。

4.4 实验结果

本文通过使用3种性能评估方法:SSQ、聚类纯度和聚类簇数目来比较CluStream算法和DLCStream算法的性能。在图4和图5中,Elapsed time轴表示从数据流起始位置,有多少个时间单元已经流逝。如图4(a)所示,对于DLCStream和Clu Stream算法的平均SSQ性能评估,这两个算法有近似的SSQ均值。由于SSQ均值坐标使用对数评估,因此一个数量级的差异性可以忽略。如图4(b)所示,对于DLCStream和CluStream算法的聚类纯度评估,这两个算法都获得了相对较高的聚类纯度(>85%),总体上,DLCStream算法的聚类纯度稍微低于CluStream算法,但二者的最大差异不到4%。值得注意的是,在每个时间单元生成聚类簇的个数很大程度上影响着聚类纯度和平均SSQ的值。显然,较大的聚类簇个数将会导致较小的平均SSQ和较高的聚类纯度。因此,本文设计了一系列实验探索DLCStream和CluStream算法生成的聚类簇的个数,并且与基准聚类簇个数进行了比较,如图4(c)所示。基准聚类簇个数是由人工标注的数据集真实聚类簇的数目,DLCStream算法生成聚类簇的数据更接近基准值,这也可以解释DLCStream算法在聚类纯度指标上略低

于 CluStream 算法。如图 5(a)所示，对于 DLCStream 和 CluStream 算法，图 5(a)示出数据流速与算法运行时间的关系，图 5(b)比较了算法对数据流的处理效率。由于 Hadoop 环境下，Map-Reduce 框架的初始化时间较长，因此在数据流初始 5 秒时间内，DLCStream 算法的初始化时间占了总运行时间的 80%。因此，对于图 5(a)，当数据流速较慢($v \leqslant 200$ points/s)时，分布式框架初始化时间占了总运行时间一半以上，CluStream算法的运行时间优于 DLCStream 算法，当 $v \geqslant 300$ points/s)时，DLCStream 算法优于 CluStream 算法。而且，随着数据流速的继续增加，DLCStream 算法的运行时间几乎不变，CluStream 算法的运行时间与数据流速成正比例上升趋势。显然，DLCStream 算法具有良好的可扩展性。图 5(b)比较了两种算法对数据流的处理效率(每秒钟处理高维数据点的个数)，由于 DLCStream 算法需要对分布式框架进行初始化，在数据流开始阶段处理效率相对较低，一旦到达稳态($t \geqslant 20$s)，DLCStream 算法的处理效率优于 CluStream 算法。

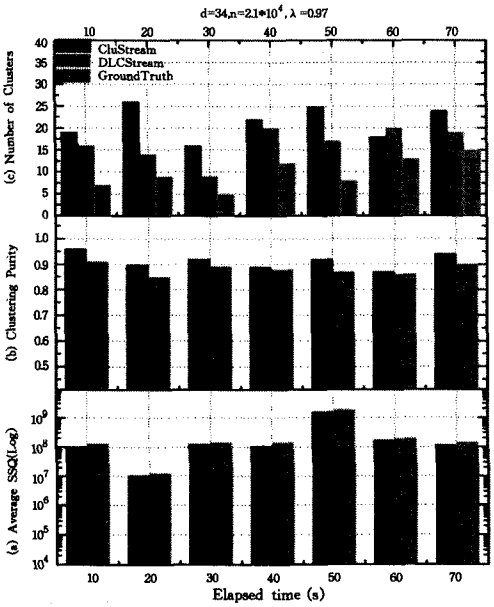
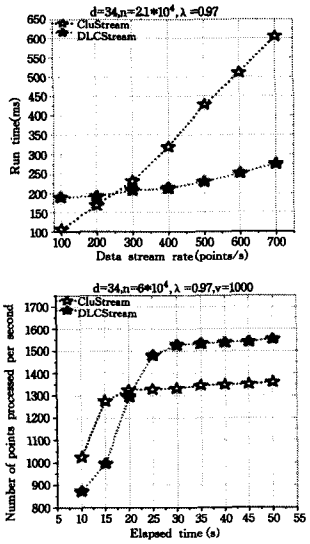


图 4　DLCStream 和 CluStream 算法之间聚类性能的比较(平均 SSQ、聚类纯度和聚类簇个数)



(b)数据流处理效率

图 5　DLCStream 和 CluStream 算法之间运行时间和处理效率比较

结束语　本文提出了适合分布式环境下的数据流聚类算法——DLCStream 算法。该算法面向云环境中高速、大量的动态数据流实时聚类要求，设计了一个基于 LSH 的分布式数据流聚类算法。结合 MIT 林肯实验室发布的入侵检测数据集，实现了基于 Map-Reduce 的分布式数据流聚类算法，包括 LSH 分布式建立算法和分布式高维聚类算法。详细的理论分析和实验验证表明，与传统的数据流聚类框架 CluStream 算法相比，在高效并行处理、可扩展性和聚类结果质量方面，DLCStream 算法具有一定优势。本文所提出的分布式数据流聚类算法仅仅是一个初步尝试，尚有不少研究工作需进一步开展，如通过调节 LSH 的参数来进一步提高聚类性能，提出更有效的聚类性能评价方法对 DLCStream 算法进行更全面的测试评估。

参 考 文 献

[1] Guerrieri A, Montresor A. DS-Means: Distributed data stream clustering [C]// Euro-Par'12 Proceedings of the 18th International Conference on Parallel Processing. 2012: 260-271

[2] Aggarwal C, Han J, Wang J, et al. A Framework for Clustering Evolving Data Streams [C]// Proceedings of the 29th International Conference on Very Large Databases. 2003, 29: 81-92

[3] Aggarwal C, Han J, Wang J, et al. A Framework for Projected Clustering of High Dimensional Data Streams [C]// Proceedings of the international conference on Very Large Databases. 2004: 852-863

[4] Shahnewaz S M, Rahman M A, Mahmud H. A Self Acting Initial Seed Selection Algorithm for K-means Clustering Based on Convex-Hull [J]. Informatics Engineering and Information Science, Communications in Computer and Information Science, 2011, 252(5): 641-650

[5] Krstinićć D, Skelin A K, Slapnicar I. Fast two-step histogram-based image segmentation [J]. Image Processing, IET, 2011, 5 (1): 63-72

[6] Gao Song, Zhang Cheng-cui, Chen Wei-bang. Identifying image spam authorship with variable bin-width histogram-based projective clustering [C]// 2011 IEEE International Conference on Multimedia and Expo (ICME). 2011: 1-6

[7] Zhou A, Cao F, Yan Y, et al. He Distributed Data Stream Clustering: A Fast EM-based Approach [C]// IEEE 23rd International Conference on Data Engineering. 2007: 736-745

[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. 2007: 133-142

[9] Kriegel H-P, Kröger P, Sander J, et al. Density-based clustering [J]. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2011, 1(3): 231-240

[10] Mao Guo-jun, Yang Yi. A Micro-cluster Based Ensemble Approach for Classifying Distributed Data Streams [C]// 2011 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI). 2011, 753-759

[11] Xie Cong-hua, Song Yu-qing, Chen Jian-mei. Fast medical image mixture density clustering segmentation using stratification

sampling and kernel density estimation [J]. Signal, Image and Video Processing, 2011, 5(2): 257-267

[12] Cai Qiang, Rushton G, Bhaduri B. Validation tests of an improved kernel density estimation method for identifying disease clusters [J]. Journal of Geographical Systems, 2012, 14(3): 243-264

[13] Subramaniam S, Palpanas T, Papadopoulos D, et al. Online Outlier Detection in Sensor Data Using Non-Parametric Models [C]// Proceedings of the 32nd International Conference on Very Large Databases. 2006: 187-198

[14] He Hai-bo, Cao Yuan. Kernel density estimation with stream data based on self-organizing map [C]// 2011 IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS). 2011: 24-30

[15] Heinz C, Seeger B. Wavelet Density Estimators over Data Streams [C]// Proceedings of the 2005 ACM Symposium on Applied Computing. 2005: 578-579

[16] Tang Cheng-long, Wang Shi-gang, Xu Wei. Improved Fuzzy Clustering Algorithm Based on Data Weighted Approach [J]. 2010, 32(6): 1277-1283

[17] Henzinger M R, Raghavan P, Rajagopalan S. Computing on data streams, External memory algorithms [M]// American Mathematical Society. Boston, MA, 1999

[18] Golab L, Özsu M T. Issues in data stream management [J]. SIGMOD Records, 2003, 32(2): 5-14

[19] Guha S, Mishra N, Motwani R, et al. Clustering data streams [C]// Annual IEEE Symposium on Foundations of Computational Science. 2000: 359-366

[20] O'Callaghan L, Mishra N, Meyerson A, et al. Streaming-data algorithms for high-quality clustering [C]// Proc. of 18th International Conference on Data Engineering. 2002: 685-694

[21] Bandyopadhyay S, Gianella C, Maulik U, et al. Clustering Distributed Data Streams in Peer-to-Peer Environments [J]. Information Science Journal, 2006, 176(14): 1952-1985

[22] Beringer J, Hullermeier E. Online Clustering of Parallel Data Streams [J]. Data and Knowledge Engineering, 2006, 58(2): 180-204

[23] Indyk P, Motwani R. Approximate nearest neighbors: towards removing the curse of dimensionality [C]// Proceedings of the thirtieth annual ACM symposium on Theory of computing (STOC' 98). New York, NY, USA: ACM, 1998: 604-613

[24] Datar M, Immorlica N, Indyk P, et al. Locality-sensitive hashing scheme based on p-stable distributions [C]// Proceedings of the twentieth annual symposium on Computational geometry (SCG' 04). New York, NY, USA: ACM, 2004: 253-262

[25] Gionis A, Indyk P, Motwani R. Similarity Search in High Dimensions via Hashing [C]// Proceedings of the 25th International Conference on Very Large Data Bases (VLDB' 99). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999: 518-529

[26] Zolotarev V M. One-dimensional stable distributions [M]// Translations of Mathematical Monographs, American Mathematical Society, 1986

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

[28] Kaufman L, Rousseeuw P J. Finding Groups in Data: an Introduction to Cluster Analysis [M]. John Wiley & Sons, 1990

[29] Sun Xin, Jiao Yu. pGrid: Parallel Grid-Based Data Stream Clustering with MapReduce [R]. Reports, OAK Ridge National Laboratory Oak Ridge, Applied Software Engineering Research Group, 2009

[30] MIT Lincoln Lab. DARPA Intrusion Detection Data Sets [OL]. [2009-02]. http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/index.html

(上接第 177 页)

计算资源受限、噪声来源复杂、对速度要求较高等新的要求。针对上述不足之处，本文利用提升算法简单、快速的特点，通过改进阈值选取方式和阈值函数提出面向体域网 ECG 信号的新的去噪方法。实验论证表明，本文提出的新的小波阈值 ECG 信号去噪法能够更好地去除 ECG 信号中的噪声，极大地提高了去噪的效率，适合用于体域网环境下的 ECG 信号处理。

参 考 文 献

[1] 刘毅，宋余庆. 无线体域网技术研究[J]. 小型微型计算机系统，2013, 34(8): 1757-1762

[2] 殷俊鹏，田应洪，赖宗声，等. 基于小波域数字滤波的心电信号 BW 去噪算法[J]. 计算机工程，2013, 39(3): 267-271

[3] 杨向林，严洪，许志，等. 基于 Hilbert-Huang 变换的 ECG 消噪[J]. 电子学报，2011, 39(4): 819-824

[4] 姚成，司玉娟，郎六琪，等. 基于小波提升的 ECG 去噪和 QRS 波识别快速算法[J]. 吉林大学学报：工学版，2012, 42(4): 1037-1043

[5] 王小飞，李鸿强，陈磊，等. 基于提升小波和改进包络的心电特征检测算法[J]. 系统仿真学报，2013, 25(12): 2893-2899

[6] 田絮资，杨建，黄力宇. 心电信号去噪的数学形态学滤波器[J]. 计算机工程与应用，2012, 48(2): 124-126

[7] 蔡兆文，陶进绪，杨振森. 自适应提升小波用于心电信号除噪[J]. 中国生物医学工程学报，2007, 26(4): 633-636

[8] Donoho D L. De-noising by soft-thresholding[J]. IEEE Transactions on Information Theory, 1995, 41(3): 613-627

[9] Moody G B, Mark R G. The impact of the MIT-BIH Arrhythmia Database[J]. Engineering in Medicine and Biology Magazine, IEEE, 2001, 20(3): 45-50