



西北工业大学
NORTHWESTERN POLYTECHNICAL UNIVERSITY

《大数据技术原理与应用》

理论报告

2023-2024 学年第 2 学期

姓 名 :	贾晓阳
学 号 :	2021302855
班 级 :	14012106
学 院 :	软件学院
任课教师 :	田春伟

目 录

1	摘要
2	相关技术
3	拟提出方法
4	方法分析
5	结论

1. 摘要

随着计算机技术及互联网的飞速发展，现代社会的数据正在以不可想象的速度膨胀，对大规模数据的管理利用已然成为一种现实需求和必然趋势。云计算的大数据处理能力，使得分析和掌握大数据中蕴藏的无尽信息、知识和智慧成为可能。聚类分析作为最常用的静态数据分析方法，常被用于模式识别、机器学习、数据挖掘等领域。随着大数据时代的到来，聚类分析在大数据中的应用也变得非常普遍。

Hadoop 是云计算环境下一种分布式计算框架，它以低成本和高效率的特性赢得了市场的认可，成为目前在云计算领域最受关注和研究应用的对象之一。很多聚类算法在 Hadoop 平台已实现，比如 K-means 算法、谱聚类算法等等，在这些成果的基础上以及针对其中的一些问题，本文做了如下工作：

(1) 详细介绍了 Hadoop 生态，尤其对 HDFS 分布式文件系统、MapReduce 分布式计算框架进行了详细的研究和探讨，包括对其多作业链的作业方式，Shuffle 阶段的 Partition、Combine 等各项机制进行了深入的探讨；

(2) 在 Hadoop 平台下提出了一种基于 Hash 改进的 K-means 算法。将海量高维数据映射到一个压缩的标识空间，进而挖掘其聚类关系，选取初始聚类中心，避免了传统 K-means 算法对随机选取初始聚类中心的敏感性，降低了 K-means 算法的迭代次数。又结合 MapReduce 框架将算法整体并行化，并通过 Partition、Combine 等机制加强了并行化程度和执行效率。最后通过实验表明，该算法不仅提高了聚类的准确率和稳定性，同时具有良好的处理速度。

关键词：大数据；云计算；Hadoop；聚类；Hash；K-means

2. 相关技术

2.1 聚类算法

近年来，在大数据文本数据的查询中学者们提出了多样化类型的聚类分析的数学模型，并选择少量的纹理特征来更快速、准确的筛选出课题研究所需的关键数据。随着目前数据特征的聚类方法得到了不断完善，使得聚类分析的运用范围越来越广泛，如图像地物分类、变化检测和语音频域的分类等等。

本章接下来将对这些目前常用的数据聚类分析的相关知识进行系统的介绍和总结，侧重于研究这些聚类方法原理以及大数据文本数据聚类等应用的优缺点，本节的主要内容有：聚类的定义；聚类算法的分类；以及重点分析 K-means 算法及其在 Hadoop 平台中数据聚类技术流程。

2.1.1 聚类的定义

将物理或抽象对象的集合分成由类似的对象组成的多个类的过程被称为聚类。通俗的讲，聚类就是将具有相似性的对象划分在同一个种群，没有相似性的对象划分在不同种群的过程。目前主要有聚类类簇(clusters)和聚类分析(Cluster Analysis)等两种语义分析的分类方式，可以根据这些采集文本数据语义的某些相关特征相似性，来有效的衡量这些类别各异的文本数据特征的差异性程度。现阶段，很多学者们主要是根据不同类型文本的数据特性和需求分析，也可以在课题研究实际应用中根据数据的规模、领域、效率等条件，通过多次特征分类的对比试验来分析目前大多数聚类方法的优缺点，并设计与之相应的最佳数据特征支撑的决策方法和聚类模型，进一步快速准确的查询出海量数据所需的目标数据。其中，在试验分析中可以选择合适的评估指标来衡量参考聚类方法特征分类的优劣性，如时空开销、误码率、虚警率和准确率等具有代表性的评估指标。其中，常用的聚类分析模型如下所示。

学者们针对在 Hadoop 平台类型各异的文本数据查询提出很多改进的聚类方法中数据聚类查询的主要过程有：随机选取一组大小为 $M*N$ 的数据集存储到大数据 HDFS 系统中，在试验分析中可以通过选定文本数据的每个对象特征来设计合理的聚类模型，以及通过无监督特征分类方式来对这些 HDFS 系统中海量数据进行准确的查询。

2.1.2 传统 K-means 算法模型

数据挖掘是一种发现数据中潜在的信息、模式的过程，通过数据挖掘能够为企业提供决策支持，并且可以带来直接的经济效益，因此一直以来都受到了广泛的关注。聚类分析是将数据集合划分成有意义的组的过程，通过聚类分析可以使数据更易于理解。聚类分析技术在许多实际的问题中都扮演着重要的角色，广泛地应用于生物技术、机器学习、社会科学、数据挖掘、模式识别以及数据分析等领域。聚类可以看作是一种分类的过程，它通过类簇的编号来作为对象的标记，因此相对于带标记的监督分类技术，聚类通常又被称之为无监督分类。

K-means 算法是一种基于原型的无监督聚类技术，也是目前应用最广泛的聚类分析方法。传统 K-means 算法的主要优点是方法简单、便于在分布式平台中进行实现，缺点是算法的时间复杂度相对较大，尤其是在大数据环境下更为明显。

传统 K-means 算法的主要流程如图 2-1 所示。

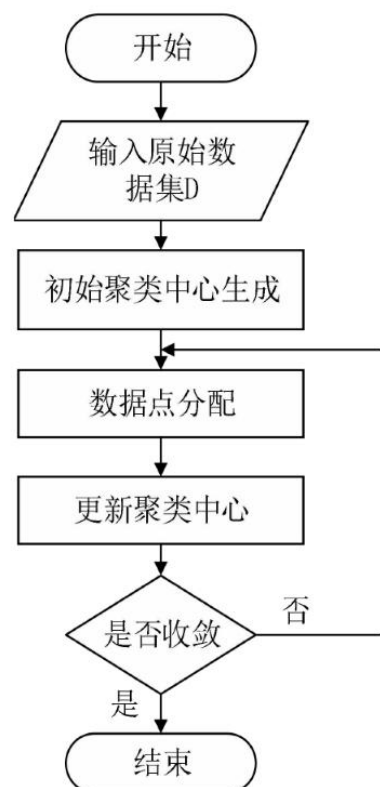


图 2-1 K-means 算法流程

首先输入原始数据集 D 、设置聚类中心个数 k ，并采用随机选点的方式生成初始聚类中心集合 $C = \{C_1, C_2, \dots, C_k\}$ ，然后开始数据点的分配过程。

对 D 中的每个数据点计算其到 k 个聚类中心的距离，并将数据点按照距离分配至距离它最近的类簇中，数据点 $A(x_1, x_2, \dots, x_m)$ 与聚类中心 $C(y_1, y_2, \dots, y_m)$ 之间的矢量距离 d 的计算方法按照公式(2-1)计算得到，并从中选择距离最小的一个

做为需要分配的聚类中心。将所有数据点分配完成后，即得到了 k 个新的类簇，每个类簇都由新的一批数据点组成，然后利用这些新的数据点更新 k 个聚类中心的矢量坐标。对于任意聚类中心 C ，假设其中包含 n 个数据点，每个数据点的矢量坐标可表示为 (x_i, y_i, \dots) 则更新聚类中心 C 的方法如公式(2-2)所示。

$$d = \sqrt{\sum_{i=1}^m (x_i - y_i)^2} \quad (2-1)$$

$$C = \left(\frac{1}{n} \sum_{i=1}^n x_i, \frac{1}{n} \sum_{i=1}^n y_i, \dots \right) \quad (2-2)$$

然后判断聚类中心的收敛情况，如果聚类中心收敛或者达到了设定的最大迭代次数，那么算法结束，否则继续进行迭代过程。按照聚类中心的变化范围是否小于设定的阈值作为是否收敛的标准。聚类中心的变化范围按照公式(2-3)计算，其中 c_i 表示更新前的聚类中心的矢量坐标， c'_i 表示更新后的聚类中心的矢量坐标。

$$\Delta x = \sum_{i=1}^k \|c'_i - c_i\|^2 \quad (2-3)$$

K-means 算法伪代码如下：

- step1 input original data set D and cluster number k ;
- step2 choose initial cluster centers;
- step3 assign all data in D by distance to the cluster center which is closest to it;
- step4 update cluster centers;
- step5 repeat step3 and step4 until cluster centers convergence or reach the maximum number of iterations which is already set;

通常采用聚类后误差的平方和(Sum of the Squared Error, SSE)作为 K-means 算法聚类质量的衡量指标。假设原始数据集为 D ，聚类中心个数为 k ，最终的聚类中心为 $C(c_i \in C)$ ，SSE 的计算方法如公式(2-4)所示。

$$SSE = \sum_{i=1}^k \sum_{d \in D} \|d - c_i\|^2 \quad (2-4)$$

K-means 算法的时间复杂度为 $O(kmn)$ ，其中 k 是聚类中心个数， m 是算法的迭代次数， n 是数据点的总数。算法的时间复杂度相对较大，尤其是在数据点的分配过程中距离计算的冗余量太大。对于 K-means 算法来说，距离计算的过程相

对于其他步骤更加耗时。特别是将一个数据点分配至最近的类簇的过程中，需要进行 k 次距离计算才能找到最近的聚类中心，对于数据规模、k 值较大的情况，传统算法的计算效率相对较低。

K-means 算法是一种实用性很强的聚类分析算法，同时也适合于进行并行化执行。在处理数据规模不大的情况下，K-means 算法优势明显。但随着数据规模的不断增加，传统 K-means 算法不能高效的完成大数据环境下的聚类任务，即便采用分布式计算框架，这种缺点也难以避免。

2.2 Hadoop 结构

作为 Apache 一个的顶级开源项目，Hadoop 这个分布式的开源云计算平台已得到广泛的研究和应用，其中 Hortonworks、Cloudera 等大型 IT 公司已对其进行商业化，将更多的业务封装到各自的 Hadoop 产品中去。但终归结底，Hadoop 的核心组件还是由:分布式文件系统 HDFS、Hadoop Common、MapReduce 框架组成，在这之上还有一些重要的应用组件: Hbase(分布式数据库)、Pig(数据分析语言)、Hive(数据仓库)、Zookeeper(协调器)等等。图 2-2 为 Hadoop 的结构图:

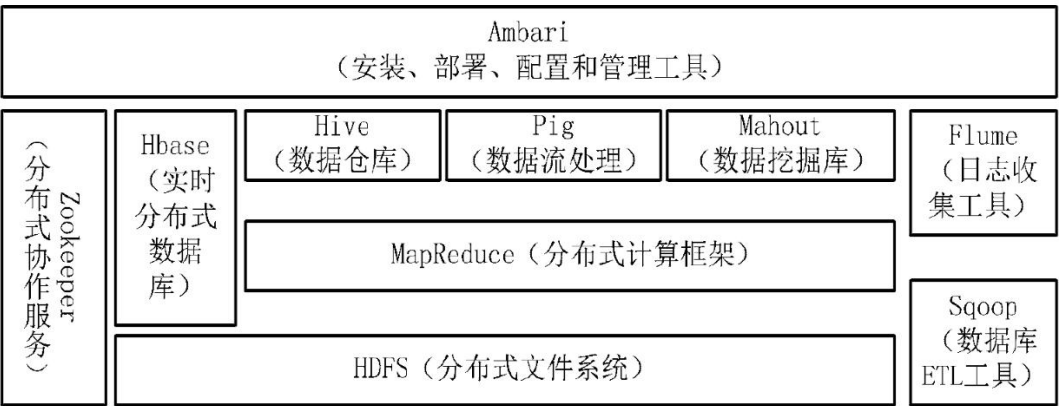


图 2-2 Hadoop 结构图

2.3 HDFS 分布式文件系统

HDFS 的全称为 Hadoop Distributed File System，中文称 Hadoop 分布式文件系统，属于 Hadoop 最核心、最基础的子项目之一，在 Hadoop 生态占据举足轻重的位置，它的开发需求来自于对超大文件的处理和流数据的访问。HDFS 的对于整个分布式计算环境的意义在于它是数据存储管理的基础，且对于运行的硬件环境要求很低，是保证 Hadoop 生态能够应用于廉价 PC 的基础。HDFS 提供了不怕故障的存储，具有高可靠性、高容错性、高可扩展性、高吞吐率、高获得

性等优点，正是由于其拥有这些特点，让 Hadoop 在处理海量数据时能够游刃有余。

从用户的角度来看，HDFS 跟普通的文件系统一样，通过文件路径就可以执行读写、删除等操作，它的数据存储在每个节点的数据块中，用户可以自行设定块的大小，默认为 64M。但由于是一个分布式文件系统，它又具有自己特性，它是 Master/Slave 结构模型，具有若干个 DataNode 和一个 NameNode。DataNode 散布在集群的各个 PC 上，用来存储数据。NameNode 是整个文件系统的管理者，通常设计在一个单独的节点上，负责管理 HDFS 的命名空间、配置副本管理、集群配置管理、处理客户端请求等事物。HDFS 还设计了一个 Secondary NameNode，它并不是 NameNode 的备份，而是负责一些辅助 NameNode 的工作，为 NameNode 分担部分工作量，它会定期将 NameNode 的修改日志和镜像文件合并，并将合并的文件进行保存，这样既可以防止日志文件过大，也可以保证当 NameNode 损坏之后，数据不会丢失。HDFS 的总体结构图如下：

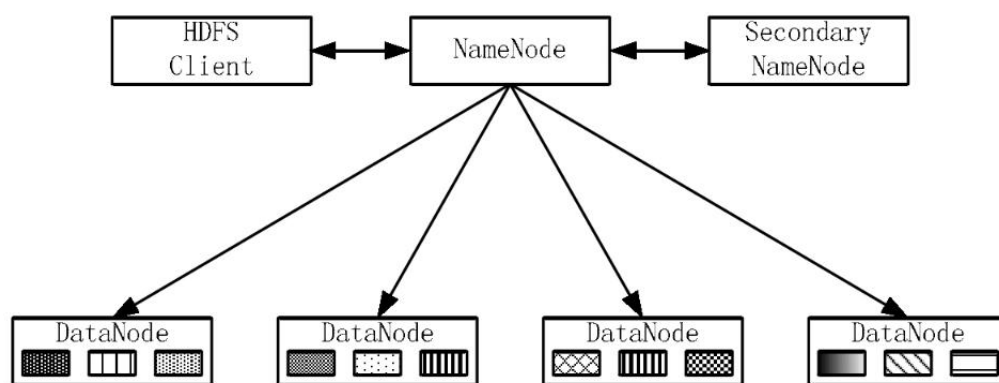


图 2-3 HDFS 结构图

2.4 MapReduce 分布式计算框架

MapReduce 来源于谷歌发表的 MapReduce 论文，Hadoop 的 MapReduce 是对它的开源实现。作为一种高效的云计算环境下的分布式计算框架，MapReduce 在大规模数据处理方面具有两个得天独厚的优点：

1) 用户不需要在意数据在处理环境下的分布、存储、拷贝和负载均衡等这些细节问题；

2) 采用函数式编程理念，用户仅需依靠一个 Map 和 Reduce 函数就能实现算法的功能，其中“Map”表示的是映射，完成的工作是对每条数据进行操作，并生成中间结果<Key, Value>，“Reduce”表示的是归约，完成的工作是将<Key, Value>中相同 Key 的 Value 值进行指定的计算，以得到最终的结果。

这种编程框架的设计，可以在很大程度降低开发人员的工作量。跟 HDFS 一

样,

MapReduce 采用的也是 Master/Slave 架构, 其架构图如图 2-4:

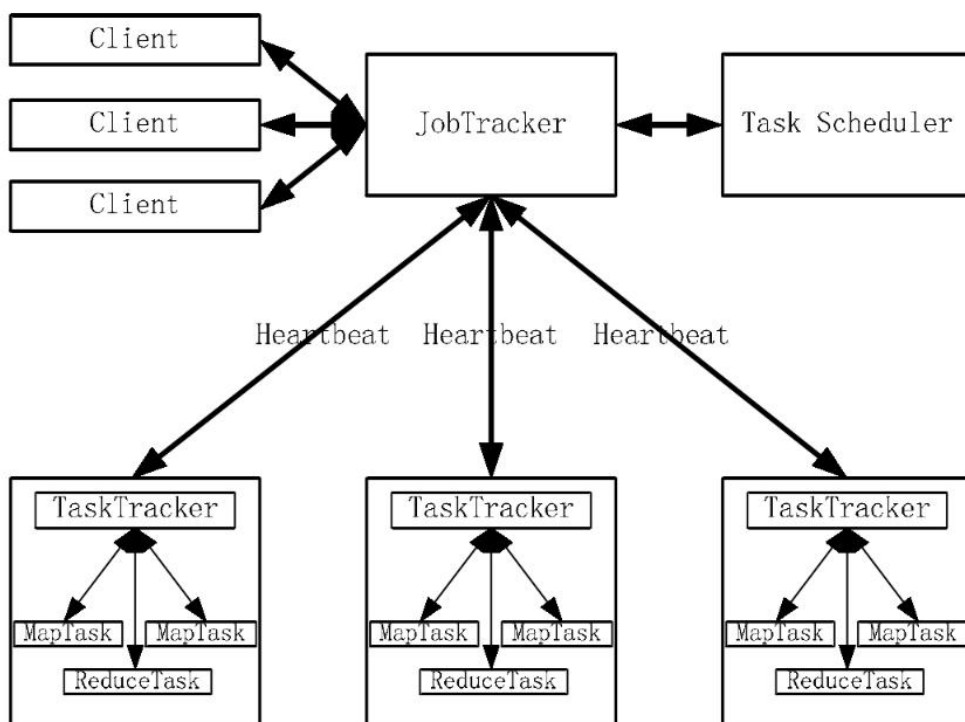


图 2-4 MapReduce 结构图

(1) JobTracker:部署在 Master 节点上, 在一个集群里只设计一个, 负责作业调度、资源调控以及错误处理等等。

(2) TaskTracker:部署在 Slave 节点上, 也就是任务处理的节点群, 执行 Map Task 和 Reduce Task, 由 JobTracker 将任务分解, 并指派给 TaskTracker, TaskTracker 还要与 JobTracker 实时通信, 汇报任务状态。

(3) Map Task:读取每一条数据, 经过 Split 后传递给用户定义的 Map 函数处理, 将处理结果以<Key, Value>形式写入本地磁盘供 Reduce 函数使用。

(4) Reducer Task:远程读取 Map 函数传输的数据, 进行合并排序等操作, 将操作后的数据交给用户定义的 Reduce 函数处理。

Hadoop 设计了一定的策略来保证 MapReduce 模型的可靠性, 所有要处理的数据都会分发给集群上的各个节点, 对每个节点有效的管理保证其可靠性的前提。从图 2-4 可以看出, JobTracker 与 TaskTracker 会进行通信, TaskTracker 会实时给 JobTracker 发送 Heartbeat, 主要是报告自己的工作状况。JobTracker 会根据 TaskTracker 发送的 Heartbeat 来判断 Slave 节点的存活状态, 如果 Slave 节点在规定时间内没有向 Master 发送 Heartbeat, 此时, 系统就会认为该 Slave 节点已经

出现故障，便不再向该节点分配任务，同时，为了保证任务的完成的进度，会将该节点处理的数据分配给其他工作的 Slave 节点。

MapReduce 是分布式的计算框架,用户不需要在意数据在处理环境下的分布、存储、拷贝、负载均衡和故障处理等这些细节问题;它采用函数式编程理念,用户仅需依靠一个 Map 和 Reduce 函数就能实现“大数据”的分布式计算。下图 2-5 表示 MapReduce 程序的具体执行过程。

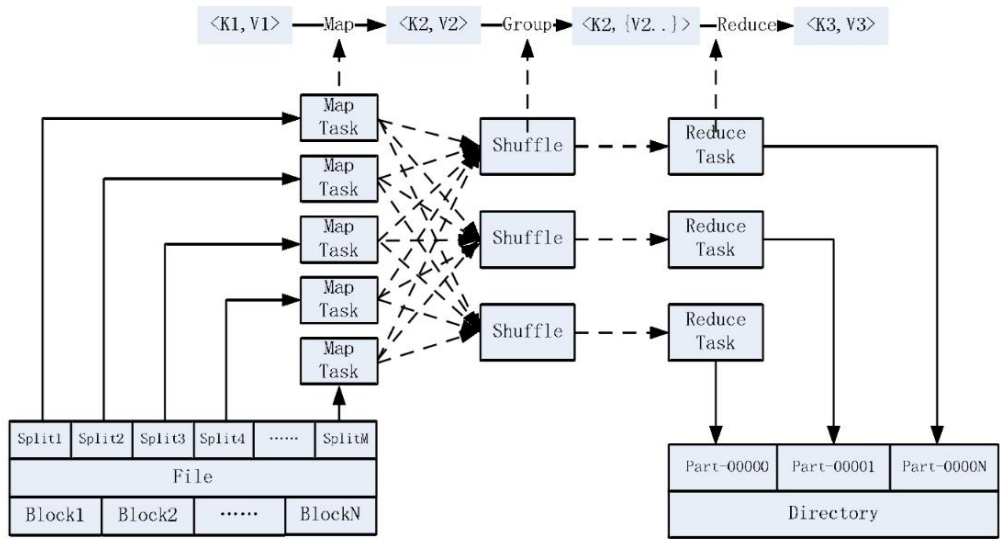


图 2-5 MapReduce 程序的具体执行过程

在实际应用中,往往需要多个 MapReduce 任务才能完成一个问题的解决,这恰是 MapReduce 的精髓思想之一,目前 Hadoop 支持的组合模式主要有下面三类:

- (1) 作业链 各个任务按顺序连接在一起,前一个任务的结果作为下一个任务的输入,该模式采用 `JobClient.runJob()`接口执行;
- (2) 作业图 各个任务之间存在多个依赖关系,而不是链条关系,比如任务 3 需要任务 1、2 的结果作为输入,该模型采用 `JobControl` 类辅助执行;
- (3) Map/Reduce 链 一个任务可能需要多个 Map 过程后执行 Reduce 过程,或者 Reduce 过程后需要再增加 Map 过程,该模式采用 `ChainMapper` 和 `ChainReducer` 两个类进行辅助操作。

3. 拟提出方法

基于 Hash 改进的 K-means 算法并行化设计

K-means 算法是经典的聚类算法之一,具有不可替代的研究价值,优点是思想

简单、收敛速度快、易于实现;缺点是对初始聚类中心敏感,易形成局部最优解,且运行复杂度高,在大数据环境下尤为明显。为了解决 K-means 算法在 Hadoop 平台下处理海量高维数据时聚类效果差,以及已有的改进算法不利于并行化等问题,本章提出了一种基于 Hash 改进的并行化方案。将海量高维的数据映射到一个压缩的标识空间,进而挖掘其聚类关系,选取初始聚类中心,避免了传统 K-means 算法对随机选取初始聚类中心的敏感性,降低了 K-means 算法的迭代次数。又结合 MapReduce 框架将算法整体并行化,并通过 Partition、Combine 等机制加强了并行化程度和执行效率。

3.1 K-means 算法并行化分析

MapReduce 框架的作业过程由 Map 和 Reduce 两个阶段执行。在 Map 阶段,每个 Map Task 读取一个 block,并调用 Map 函数进行处理,然后将结果写到本地磁盘;在 Reduce 阶段,每个 Reduce Task 远程地从 Map Task 节点上读取数据,调用 Reduce 函数进行数据处理,然后将结果写到 HDFS 上。显然,Map 和 Reduce 的作业结果均要写磁盘,虽然降低了运算性能,但提高了系统的可靠性,所以,大量的迭代计算会大大降低系统的性能。

在 Hadoop 平台下执行 K-means 算法可以大大减少聚类的时间。Chu 在集群环境下实现了基于 MapReduce 分布式框架的 Naïve Bayes、K-means、EM 等多种传统数据挖掘算法。但是他们并未考虑执行的通信代价和计算量,聚类准确率和处理速度均不理想。

虽然对 K-means 算法改进的研究成果非常之多,但是在移植到 Hadoop 平台上仍然出色的算法少之又少,原因主要有两个:一是初始聚类中心选取阶段的数据耦合性强,不适于并行;二是初始聚类中心选取仍然需要迭代计算,而 MapReduce 框架对迭代计算十分敏感。

再者,大部分算法虽然实现了并行化,但并没有充分利用 Hadoop 的各项机制,比如大部分算法没有利用好 Partition 机制,在设计 K-means 算法计算新的聚类中心时只采用一个 Reducer 去处理,这样就降低了并行化的程度,另外很多算法没有采用 Combine 机制在 shuffle 阶段进行数据处理,这样就增加带宽的传输负担,降低了系统的处理性能。

另外,在实际应用中,往往面对的都是海量、高维的数据,而这类数据在分布式环境中很难对其整体分布进行探测,这也加大了选取合理初始聚类中心的难度。于是,本章利用 Hash 算法原理,将数据的各维属性散列到一个统一的标识空间,再通过这个缩小的标识空间去挖掘数据的聚类关系,从而达到选取最优初始聚类中心的目的。本章算法在原有并行化 K-means 算法的基础上要达到的性能需求有:

- (1) 大幅提高聚类的准确率和稳定性;
- (2) 大幅提高收敛的迭代次数,且次数稳定;
- (3) 整个过程并行化充分,且设计的 job 数量不多;
- (4) 充分使用 shuffle 阶段的各项机制;
- (5) 大幅提高算法对海量高维数据的处理速度。

3.2 Hash 算法简介

Hash 算法,也叫散列算法,是将任意长度的输入通过散列函数,变换成固定长度的输出,该输出即为散列值。这种转换是一种压缩映射,散列值的空间通常小于输入的空间,不同的输入可能会散列成相同的输出。

设计合理的散列函数是 Hash 算法的关键所在,显然,散列函数会将多个输入值映射到同一个地址空间,这种情况称为“冲突”,这些发生碰撞的输入值称为“同义词”。

3.3 基于 Hash 算法的初始聚类中心选取

算法的基本思想是设计合理的散列函数将相似度大的数据散列到同一个地址空间,不同相似度的数据散列到不同的地址空间,然后在同义词最多的 k 个地址空间选择 k 个初始聚类中心,这样选择的出来的初始聚类中心能尽量满足簇内距离最小,簇间距离最大原则,该算法的依据归纳为以下几点:

- (1) 通过散列函数将海量、高维的数据映射到一个压缩的地址空间,利于对数据聚类关系的把控。
- (2) 通过映射后,相似度大的数据会发生“冲突”,成为同义词,这些最密集数据带的中心最接近聚类中心。
- (3) 通过选择不同的同义词,就使得选择的初始聚类中心尽可能不在同一个类中。
- (4) 避免了将孤立点选择为初始聚类中心。

定义 1 阶梯函数是有限的间隔指标函数的线性组合,定义阶梯函数 $y = f(x)$,定义域为 $[a, b]$,值域为 $\{y_1, y_2, \dots, y_n\}$,间隔长度为 d ,则该阶梯函数可以表示为:

$$f(x) = \begin{cases} y_1, [a, a + d) \\ y_2, [a + d, a + 2d) \\ \dots \\ y_k, [b - d, b] \end{cases}$$

定义 2 将散列函数设计为:

设数据集的个数为 n , 属性维度为 m , $key_i (0 < i < m)$ 表示数据的 i 第维属性, 其最大值 max_i , 最小值 min_i , 间隔长度为 $d_i = \frac{|max_i - min_i|}{k}$, 函数的定义域为 $[min_i, max_i]$,

值域为 $\{y_{i1}, y_{i2}, \dots, y_{ik}\}$, k 表示聚类数目, 函数图像如下图 3.1, 第 i 维属性的散列关系表达如下图 3.2:

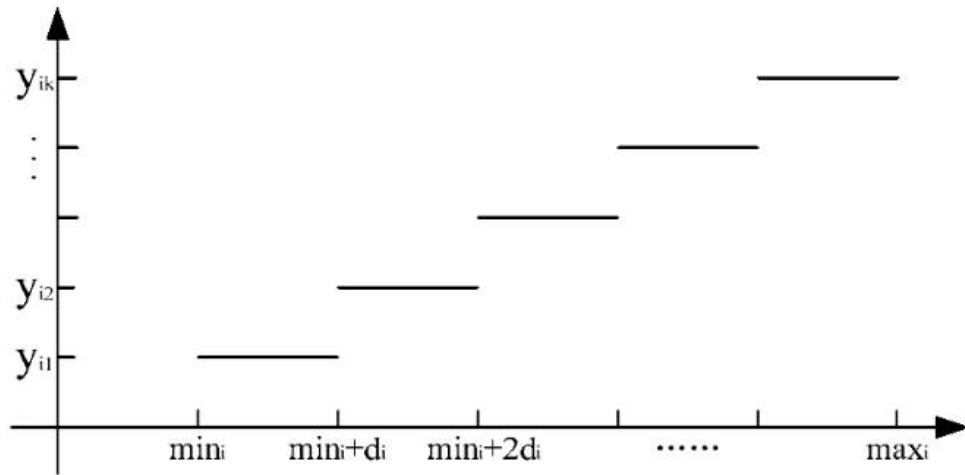


图 3.1 散列函数图

由定义 2 可知, 散列函数将输入数据每一维属性的值散列到 k 个地址空间, 数据的每一维属性都对应一个唯一的地址空间, 将每个地址空间编号, 则每个数据就可以用 m 个地址空间编号组成的行向量表示, 数据集可以用一个的矩阵表示, 对矩阵进行变换, 将相同的行向量集中在一起, 并统计其个数, 找出最多的 k 个行向量,

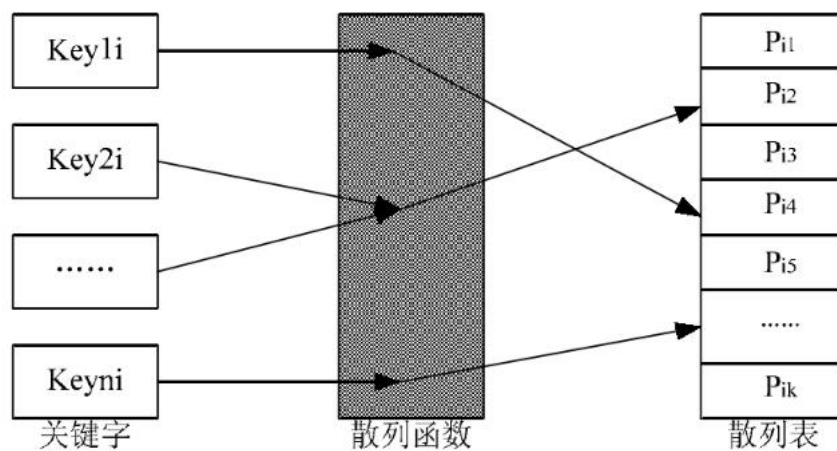


图 3.2 散列关系图

计算每个行向量对应所有数据的中心点, 得出的 k 个中心点就是选择的初始聚类中心。

3.4 基于 Hash 改进的 K-means 算法

3.4.1 算法流程

输入: dataset(数据集);k(聚类个数)。

输出: 聚类结果。

Step1: 选取 k 个初始聚类中心:

- (1) 找出数据集每一维属性的最大值 \max 与最小值 ;
- (2) 计算每一维属性的间隔长度 d ,构造每一维属性对应的地址空间,并编号;
- (3) 将所有数据散列到对应的地址空间;
- (4) 找出 k 个最密集的地址空间;
- (5) 计算 k 个地址空间中每个空间的数据的中心点,将这 k 个中心点作为 K-means 算法的初始聚类中心。

Step2:执行 K-means 聚类算法:

- (1) 把每个样本点归为离它最近的中心点的一类;
- (2) 重新设置每一类的中心点;
- (3) 重复(1)(2)步,直至收敛。

3.4.2 算法伪代码实现

输入:dataset(数据集);k(聚类个数)。

For i=1 to n do //n 表示数据的个数

{

For j=1 to m do //m 表示数据的维数

{

Maxj=GetMax();

Minj=GetMin();

$d = (\text{Maxj} - \text{Minj}) / k$;

if($\text{Minj} \leq x_j < \text{Minj} + d$) return Pj1;

if($\text{Minj} + d \leq x_j < \text{Minj} + 2d$) return Pj2;

.....

//根据 k 的取值决定

if($\text{Minj} + (k-1)d \leq x_j < \text{Minj} + kd$) return Pjk;

}//每一维属性都对应一个地址编号

将各维数据的地址编号组成一个字符串;

}//每个数据都转换成一个地址编号串

将具有相同编号串的数据保存在一起,形成数据簇;
找出 k 个数据最多的数据簇;
计算这些数据簇的聚类中心,并作为 K-means 算法的初始聚类中心;
执行 K-means 算法;
输出:聚类结果。

4. 方法分析

4.1 改进算法在 MapReduce 框架上的实现

4.1.1 算法并行化实现策略

本算法的并行策略是设计 3 个 job 作业链工作,每个 job 均独立运行,每个 job 的输出作为下一个 job 的输入,直到最后作业结束。针对 Step1 的(1)(2),由于比较每个数据的各维属性的操作是相互独立的,符合并行理念,且该操作计算量庞大,故在此设计 Map 函数,Reduce 函数的作用是汇总 Map 的作业结果,构造地址空间,此时的计算量已经相当微小,故只设计一个 Reducer 来处理。针对 Step1 的(3)(4)(5),由于将每个数据散列到对应的地址空间是相互独立的,且计算量庞大,故在此设计 Map 函数,Reduce 函数汇总 Map 函数作业的结果,计算初始聚类中心,此时的计算量仍然庞大,故利用 Partition 机制将数据按照编号串的不同分发到各个节点去计算,将集群的性能发挥到最大限度。

4.1.2 算法并行化具体实现策略

Job1 的功能是构造散列表,Map 函数找出数据每一维属性的极值,Reduce 函数为每一维属性的散列构造对应的地址空间。Job2 的功能是计算初始聚类中心,Map 函数将所有数据映射到散列表内,使得每个数据具有一个唯一的地址空间编号串,Combine 的功能是将具有相同编号的数据合并,并统计这些数据的个数,减少数据传输对带宽的损耗和 Reducer 的计算量,Partition 的功能是对数据进行分区,使数据得以传输到不同的 Reducer 上工作,Reduce 函数计算初始聚类中心。Job3 完成对所有数据的 K-means 聚类。算法的实现细节如图 4,每个 job 根据各自任务的不同,在设计上有所差别,详情参考如下关键细节描述:

表 3.2 算法准确率对比

数据集	算法 1			算法 2			算法 3		
	最高	最低	平均	最高	最低	平均	最高	最低	平均
Synthetic control	67.0	60.7	64.8	70.8	65.8	68.1	76.1	74.2	75.0
Cloud	65.8	58.6	62.4	69.9	64.6	67.8	75.7	73.6	74.4
Wine quality	62.6	54.1	58.5	63.6	58.7	61.2	70.0	67.4	69.0




图 4 并行实现细节图

4.1.3 实验数据和环境

本文的实验平台由 8 台计算机搭建,其中 1 台部署 Client 节点,负责提交 Map Reduce 作业并显示处理结果,一台作为 MapReduce 运行的主控节点部署 namenode 和 jobtracker,其余 6 台部署 datanode 数据节点和 tasktracker 服务节点。计算机环境的硬件配置如下:64 位的 Ubuntu13.10 版;Hadoop1.04 版;jdk7.0 版。实验数据来自是 UCI 机器学习库,库中的 iris、cloud 等都是常用的检验聚类效果的数值型数据集,本文采集的数据集其个数、维数、类别数分别列在表 3.1。并使用相关工具将数据集进行扩展。

表 3.1 实验数据集

数据集	样本数	维度	类别数	属性类型
Synthetic control	600	60	8	数值
Could	1024	10	6	数值
Wine Quality	4895	12	6	数值

4.1.4 实验结果及分析

在测试聚类准确率时,作者将以上数据均扩展为 1GB 大小,对比文献中的算法 1、算法 2,本文算法为算法 3,运行 10 次,并采用系统抽样法抽取其中的 10000 条记录进行统计,结果如表 3.2 所示。

表 3.2 算法准确率对比

数据集	算法 1			算法 2			算法 3		
	最高	最低	平均	最高	最低	平均	最高	最低	平均
Synthetic control	67.0	60.7	64.8	70.8	65.8	68.1	76.1	74.2	75.0
Cloud	65.8	58.6	62.4	69.9	64.6	67.8	75.7	73.6	74.4
Wine quality	62.6	54.1	58.5	63.6	58.7	61.2	70.0	67.4	69.0

实验表明,本文算法在传统并行化 K-means 算法基础上提升了 11%左右的准确率,传统的并行化 K-means 算法由于初始聚类中心是随机选取,导致聚类的结果不稳定,高与最低之间相差的幅度比较大,比如在处理 Wine quality 时,最高与最低之间相差了 8.5%,聚类的准确率也较差。本文算法也比算法 2 要高出 7%的准确率,而算法 2 的相差幅度也在 5%左右,本文算法基本维持在 2%。

为了测试算法在大数据环境下的处理速度,笔者将 Synthetic control 扩展为 5 个大小不同的数据集,分别是 0.5GB、1GB、2GB、4GB 和 8GB。对比文献中算法 1, 算法 2, 表 3.3 记录了以上 3 个算法在处理 8GB 时的迭代次数,图 3.4 记录了 3 个算法处理各个数据集所花费的时间。

表 3.3 算法迭代次数对比

次数	1	2	3	4	5	6	7	8	9	10
算法 1	43	29	24	17	52	34	29	35	21	19
算法 2	29	25	16	23	19	17	27	21	26	22
本文算法	7	8	9	7	8	9	7	9	9	8

从迭代次数来看,本文算法比较稳定且与对比算法拉开了较大的差距,只需要 8 次左右便可达到收敛条件,这是因为选取的初始聚类中心已经趋近于收敛状态,大大加快了 K-means 算法的收敛速度,而传统并行化 K-means 算法由于初始聚类中心选取的随机性,导致收敛的速度缓慢,且极不稳定。

从处理时间来看,在数据集较小的情况下,由于算法 2 多了一个中心选取阶段,算法 1 略优于算法 2,随着数据集的扩大,算法 2 逐渐体现优势,这是因为算法 2 选取的初始聚类中心减少了 K-means 聚类的迭代次数。而本文算法由于迭代次数均优于以上两种算法,又增加了 Partition、Combine 等操作,且算法全过程并行化充分,随着数据集的扩大,这种优势就越明显,因此时间性能上要大大优于以上算法。

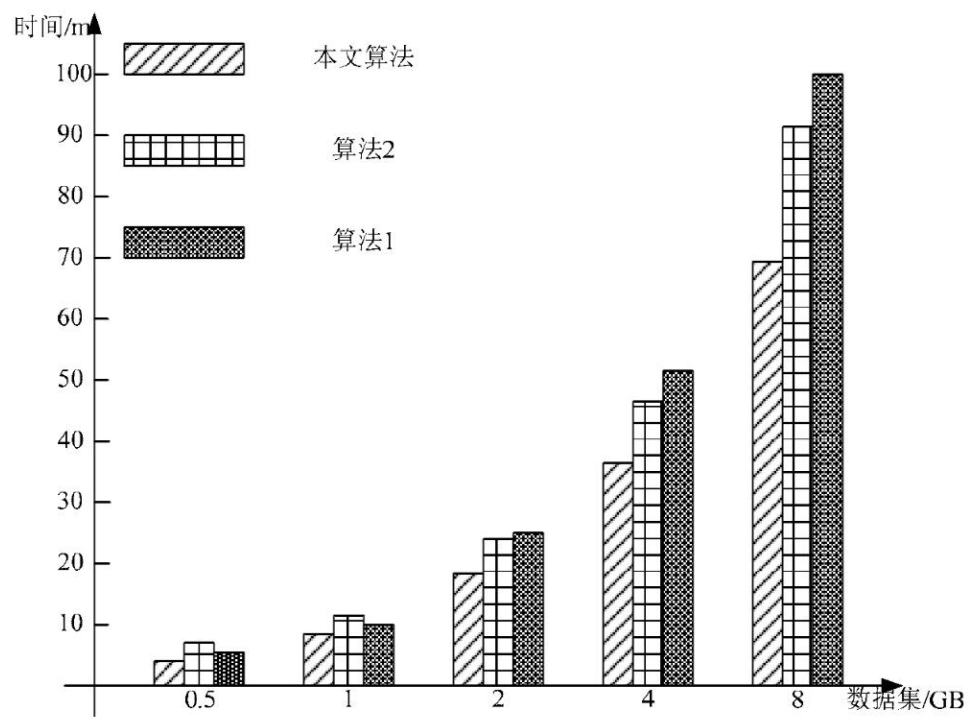


图 3.4 算法处理时间对比

5. 结论

本文对 Hadoop 平台下 K-means 聚类算法的优化进行了深入的研究,提出了一种基于 Hash 算法原理的初始聚类中心选取方案,将海量、高维的数据散列到一个压缩的空间,进而挖掘其聚类关系,使得选取的初始聚类中心尽可能趋近于收敛状态,大大降低了聚类的迭代次数、提高了聚类的准确度。同时本算法与 MapReduce 计算框架完全契合,并行十分充分,设计的 Partition 和 Combine 机制提高了算法的并行化程度和处理速度。最后通过在 UCI 不同数据集上的实验表明,该算法在准确率、稳定性、以及处理速度上等综合性能上表现良好。