分类号_____ 学号 M201372801

学校代码 10487 密级 _____

華中科技大學

硕士学位论文

基于 Spark 的 K-means 算法的并行化实现与优化

学位申请人： 张波

学 科 专 业： 计算机技术

指 导 教 师： 韩建军 副教授

答 辩 日 期： 2015 年 5 月 27 日

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science

The Parallelization and Optimization of K-means Algorithm Based on Spark

Candidate : Zhang Bo

Major : Computer Technology

Supervisor : Prof.Han Jianjun

Huazhong University of Science & Technology

Wuhan 430074, P. R. China

May,2015

独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除文中已经标明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对本文的研究做出贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到，本声明的法律结果由本人承担。

学位论文作者签名：

日期：　　年　月　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，即：学校有权保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权华中科技大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

保密□，在_____年解密后适用本授权书。

本论文属于　　不保密□。

（请在以上方框内打“√”）

学位论文作者签名：　　指导教师签名：

日期：　年　月　日　　日期：　年　月　日

摘 要

移动互联网浪潮衍生出海量的数据，这些数据中蕴含着不可估量的商业价值和指导价值，而如何从这些杂乱无章的海量数据中挖掘出有用的信息已经成为一个相当重要的研究课题。为了快速提升聚类算法处理海量数据集的效率，可以利用集群资源来高效地执行挖掘任务，而基于分布式计算平台 Spark 的并行化改进聚类算法能有效地解决此类难题。

本课题研究了 K-means 算法及其优化算法在 Spark 平台上的并行化实现。一方面，基于 K-means 算法存在初始 k 值不确定性、初始聚类中心点随机选取的不稳定性问题，本研究课题提出了基于预聚类算法 Canopy 来初始化 K-means 算法中的 k 值和初始聚类中心用以提高算法的收敛速度和聚类结果的稳定性；另一方面，为了充分利用 Spark 的 RDD 特性优势，可以从内存优化、数据压缩、数据序列化、运行内存占比、运行堆占比、缓存大小等系统配置方面进行 Spark 调优，从而利用 Spark 相比于 Hadoop 的平台优势进一步加快改进 Canopy_K-means（CKM）算法的并行计算效率和分布式计算环境下的应用能力。

基于 Spark 集群环境下的 K-means 算法及其改进 CKM 算法的对比研究结果可得以下结论：（1）Spark 分布式集群在迭代计算方面相对于 Hadoop 具有不可比拟的效率（收敛速率、聚类准确度）优势；（2）基于 Spark 的改进 CKM 并行算法比 K-means 并行算法的聚类结果更加准确可信、收敛速度更快；（3）基于 Spark 的改进 CKM 并行算法比 K-means 并行算法加速比增幅更快、扩展比更快地收敛于某一稳定值。总体而言，基于 Spark 集群实现的 CKM 并行聚类算法比传统 K-means 算法效率（准确性、迭代收敛速率、并行算法健硕性能）更高。

关键词：K-mean，分布式计算，并行化，Spark，聚类分析

ABSTRACT

Mobile Internet wave has derived massive data, these data contains immeasurable business value and practical significance, and how to mine useful information from these chaotic massive data has become a quite important research topic. In order to quickly promote it's capability of massive data processing, we can use cluster with the integration of resources to effectively perform the task of data mining, and the parallel improved mining algorithm combined with the distributed computing platform can effectively solve the problem of data mining tasks.

This research studies the parallel implementation of k-means in Spark. On the one hand, based on uncertain initialized K value and randomly selected initial cluster centers of K-means, this research proposed the Canopy algorithm of pre clustering to initialize the K-means algorithm of K value and initial clustering centers to improve the stability of the convergence speed and clustering results. On the other hand, in order to make full use of the RDD features of Spark, this research can do the Spark tuning form the aspects of memory optimization, data compression, data serilization, executor-memory ratio, executor-shiffle ratio, Cache size, et al. Thus, the parallel computing efficiency and application ability in distributed computing environment of improved Canopy_K-means(CKM) algorithm will further improved.

Based on the comparative experiment results of improved CKM and K-means on Spark cluster environment, it shows the following conclusions: (1)Spark has incomparable efficiency (convergence rate, clustering accuracy) in the iterative calculation relative to Hadoop; (2) The clustering result of CKM parallel algorithm is more accurate and reliable than K-means parallel algorithm on Spark, and the convergence speed of CKM parallel algorithm is faster than K-means parallel algorithm; (3) The speed-up ratio amplitude of improved CKM parallel algorithm is faster than K-means parallel algorithm on Spark, and the expansion ratio of the former is more quickly converged to a stable value. Overall, the improved CKM parallel clustering algorithm is more efficient (accuracy, convergence rate, parallel performance) than the traditional K-means algorithm on Spark.

Key words: K-means, Distributed computing, Parallelization, Spark, Clustering

目 录

1 绪论

1.1 论文的研究背景和意义

随着人类社会逐渐由 PC 互联网时代过渡到移动互联网时代，同时移动互联网浪潮带来的海量数据也正以井喷式的速度在急速膨胀[1]。随着商业化经济和信息产业不断地发展，商业竞争日渐激烈，尤其是移动互联网应用的飞速发展产生了海量的数据（如企业日常运营积累的商业数据、科研工作中检测得到的科研数据），这些都迫切需要从海量的复杂数据中挖掘出有价值的潜在信息知识来科学地指导下一步的商业决策和科研方向。

在这样的大数据背景下，数据挖掘作为一门新兴学科应运而生，它本质上是一门融合了计算机科学、信息科学、应用数学、统计学等学科的集成学科[2]。随着云计算和大数据的概念愈演愈热，搜索引擎、电子商务、社交网络、即时通信等互联网应用都聚合了大量的资讯，比如 Baidu 的搜索排名策略、Tianmao 和 Amazon 的商品推荐广告、Tencent 的好友推荐信息，这些内藏的海量数据都成为企业竞争力和社会发展的重要资源，因此海量数据挖掘领域的的技术革新速度在企业创新中变得愈发耀眼[3]。

数据挖掘技术主要分成分类、聚类、推荐三类算法，其中聚类分析主要用于将数据集划分成多个类簇，让同一类簇中的数据之间高度相似，相异类簇中的数据之间高度相异[4]。聚类分析在现实中有着广泛的应用场景，如网络主动防御、广告高效推荐、搜索精准定位等领域[5]。在网络主动防御领域，聚类分析能依据曾经遭受过的网络攻击、木马病毒等等提炼出网络入侵的固有特征，从而推测出未知的攻击新模式来主动部署网络安全防御策略；在广告高效推荐领域，聚类分析可以辅助企业更好地挖掘出潜在的目标客户，从而精准地把广告投放给目标客户，这样不仅可以降低销售营销成本，同时也可以提高产品销量和知名度。在搜索精准定位领域，聚类分析广泛用于 web 文档搜索聚类、地理信息系统目标方位聚类等应用方向。

聚类分析算法集中的 K-means 以快速简单、对大数据集有较高的效率和可伸缩性、时间复杂度近于线性、适合挖掘大规模数据集等优点而被广泛应用，但是 K-means 算法也有其特定的性能瓶颈。一方面 ，K-means 算法初始化过程中预先设定的 K 值很难估计，大多情况下全凭经验决定，所以在很大程度上存在一定的主观性[6]；一方

面，算法开始前随机选定的初始类簇中心也会在很大程度上影响聚类结果，如果初始中心选取不当，可能会得到与标准聚类相差甚远的聚类结果[7]。

随着智能终端的飞速普及，移动互联网浪潮衍生出了海量数据，这些海量数据的数据类型各异、体量巨大、价值密度低、挖掘难度大[8]。传统的数据挖掘模型及其优化算法大多在单机上进行串行运算，当面对如此复杂多样的大规模数据集和多维数据类型时，由于单机的计算资源（如处理器数量、内存容量）有限而造成挖掘算法不能快速准确地完成数据挖掘任务[9]。可以利用多台机器的资源来高效地并行执行数据挖掘任务，而结合分布式计算平台的并行化改进挖掘算法能有效地解决这一难题。

传统的 Message Passing Interface 是并行计算的一种通用低层次编程接口，虽然具有良好的复杂均衡和容错性能，但其抽象度不高不易编写，需要考虑到底层的节点通信、任务调度等问题[10]。Google 提出的 MapReduce 是处理大数据的并行编程范式，MapReduce 高度抽象容易编写，开发者只需要考虑应用逻辑。基于 MapReduce 并行编程模型实现的并行计算框架有 Hadoop、Spark、Sphere 等，Hadoop1.x 适合处理离线批处理文件，对迭代计算和实时处理却捉襟见肘[11]。Spark 是一种基于弹性分布式数据集（Resilient Distributed Datasets）的并行计算框架，它成功地构建了一体化、多元化的大数据体系可以大幅度提高海量数据处理的高效性[12]。作为 Spark 底层的资源调度管理平台，Yarn 是 Hadoop1.x 的全新优化版本 Hadoop2.x 的资源管理器，可以良好地与 Spark 上层应用相结合来解决 Spark 自身没有任务调度能力的缺憾，同时也让开发者不用过多地考虑底层资源调度策略。

1.2 课题的国内外研究现状

聚类分析可以将数据集划分为若干个类簇，通过聚类分析可以让同一类簇内的数据对象具有比较高的相似度 ，而让不同类簇中的数据对象具有比较低的相似度[13]。由于 K-means 算法简单高效，适用于数据量大、特征维度高的数据集，而且它对数据的依赖度较低，所以 K-means 成为目前应用比较广泛的一种聚类方法[14]。但传统的 K-means 算法也存在诸多不足：初始化时需预先确定的 K 值仅凭开发人员的经验决定，这样的主观性会影响聚类效率和结果的可信度[6]；初始聚类中心的随机选择会造成聚类结果的不稳定性 [15][16]。

为了改进传统 K-means 算法的聚类效率，研究人员对算法的不足之处提出了相

应的优化措施[17] [18]。针对初始类簇中心选择的随机性缺陷，很多学者发表了许多针对各领域的优化初始中心的粗糙聚类算法[13] [15] [16]。针对 k 值初始化的主观性问题，许多学者提出利用支持向量机、遗传算法等已有算法来确定最佳 k 值[7] [19] [20]，ZhuJian 等人利用遗传算法对 K 值使用二进制编码的方式并且设置了新的适应度函数来优化 k 值的选取[21]。针对算法的时间开销问题，可以利用聚类算法的并行策略。梁红将 MPI（Message Passing Interface）并行计算框架应用到小波聚类算法，提出了 MPI-WaveCluster 算法 [22]；Sanpawat Kantabutra 提出了基于密度的 K-means 算法的并行化实现，可以有效地消除孤点[23]；贾瑞玉等人提出基于 MapReduce 模型的并行遗传 K-means 算法，使 MapReduce 分布式编程思想融入 K-means 算法来提高整体运行的高效性[24]。

1.3 论文的主要工作

本课题是在 Spark 分布式集群环境下对数据挖掘聚类算法进行并行化研究。论文的工作主要分成以下几个部分：

(1) 大量查阅前人的研究成果，对数据挖掘、聚类分析及其并行化、分布式计算平台 Hadoop、Spark 及其资源管理器 Yarn 等技术进行了理论准备。

(2) 分析传统 K-means 算法的优缺点，提出可以优化算法效率的三个改进之处。

(3) 研究 MapReduce 编程模型，在熟悉 Hadoop 和 Spark 分布式数据处理平台的基础上设计 K-means 算法的并行化实现。

(4) 针对传统 K-means 算法在初始 K 值和聚类中心方面的缺陷，本研究课题在融合预聚类算法 Canopy 和精确聚类算法 K-means 的基础上，设计并实现基于 Spark 分布式数据处理平台的 Canopy_K-means（CKM）并行化算法的实现。

(5) 通过实验，对相同数据量下 K-means 并行算法在 Hadoop、Spark 集群环境下的性能进行比较；对相同数据量下 K-means、Canopy_K-means 分别在 Spark 分布式集群环境下的并行计算的性能进行比较；对不同数据量下传统 K-means 算法在 Hadoop 集群环境下和 Spark 集群环境下并行计算的性能比较；对不同数据量下 K-means 算法、Canopy_K-means 算法在 Spark 集群环境下并行计算的性能比较。

1.4 论文的组织架构

第一章概要地介绍了本研究课题的研究背景与意义、课题的国内外研究现状，由传统 K-means 算法应用广泛的同时所遇到的计算瓶颈抛砖引玉，阐述本研究课题的研究意义。

第二章概要地介绍了海量数据处理中的聚类挖掘技术的基础知识，然后结合海量数据聚类的计算瓶颈提出了并行化的可行性分析，接着介绍了基于 MapReduce 编程模型的分布式计算框架 Hadoop、基于 RDD 内存计算及 Scala 函数式编程的分布式计算框架 Spark，并对比了两者的性能优势。

第三章在详述了传统 K-means 算法的流程、应用及其优缺点，然后针对其在单机环境下串行计算效率低的瓶颈提出 K-means 算法在 Hadoop 和 Spark 分布式集群环境下的并行化实现。

第四章针对传统 K-means 算法在 K 值选择的主观性和类簇中心点选取的随机性上的缺陷，提出了改进的 Canopy_K-means 算法来更精确地选取 K 值和初始中心点，最后给出改进算法在 Hadoop 和 Spark 分布式集群环境下的并行化实现。

第五章详述两个分布式计算平台 Hadoop、Spark 在集群资源管理器 Yarn 下软硬件的环境配置，然后实现 K-means 算法和优化后的 Canopy_K-means 算法在 Hadoop、Spark 平台上的并行化，通过实验结果进行聚类准确性、加速比、可扩展性、与其他平台比较以及资源调度稳定性的测试。

第六章根据实验对比结果给出改进算法在聚类分析性能上的提高程度，并针对试验中所存在的不足之处展望进一步优化的工作。

2 聚类分析与分布式计算框架研究现状

数据挖掘（Knowledge Discovery in Database）的对象一般都是海量的、不完全的、不规则的、带噪音的、随机的数据 [13][25]。从这些数据中被挖掘出来的信息可以广泛的应用于金融、管理、教育、新闻、电子商务、市场营销、广告等领域，具有非常高的实际应用价值 [26] [27] [28]。

在面对海量数据的信息挖掘任务时，经常会发现数据集普遍存在数据量大、数据格式参差不齐、数据信息完整性缺失等问题，所以这时最好能减少分析的数据量来方便进一步地快速准确地挖掘出有用信息，而比较好的方法就是将需要挖掘的数据集划分成多个子类，然后再对每个子类数据单独进行数据分析。人们通常凭借历史累积的经验和系统的专业知识体系来对数据对象集进行分类，但这样不仅工作量繁重而且很可能因为人为误差导致分类结果与预期相悖，此时聚类挖掘作为数据分析的有效捷径则可提供更高的准确性[29]。

2.1 聚类分析技术概述

2.1.1 聚类分析的定义

聚类分析是由数据对象本身的特有属性进行聚类在一起，这种学习方式属于无指导性学习。聚类分析严格的数学公式定义如下：有数据对象集 $S=\{s_1,s_2,...s_k,...s_n\};s_k(k=1,2,...,n)$，其中 s_k 为数据对象个体。聚类分析就是依据相似性度量将整个数据集分割为 K 个子集：$S=\{C_1,C_2,...C_i,...C_k\}$，其中$(i=1,2,...,k)$，而且这些子集必须满足以下条件才能使之成立：

$$C_i \subseteq S, C_1 \cup C_2 \cup ... \cup C_k = S, k \in [1,n]; C_i \cap C_j = \phi, i \in [1,k], j \in [1,k] \qquad \text{(式 2-1)}$$

由公式（2-1）定义可知：数据对象全集中的数据对象个体 s_k 一定属于某一子集 C_k，同时数据对象全集中的数据对象个体 s_k 一定只属于某一子集 C_k，不会同时属于两个或以上的子集。

2.1.2 聚类分析的流程

通过对现有聚类算法的进一步深入研究，可以明白这些聚类算法能够在特定的环境下取得良好的聚类效果。由于聚类分析的应用环境千差万别，所以进行聚类分析时的方式和步骤也可能会有相异之处，但其分析流程大致相同：

（1）先将需要聚类分析的原始数据对象集进行预处理，从复杂的数据集中提取其要分析的特征属性。

（2）其次按照提取出的特征属性来选择合适的相似性测量公式，以便更恰当地将数据对象划分到合适的子集。

（3）然后结合数据集的特征属性和选取的相似性度量公式来选择适合的聚类分析算法，进而更好更快地将数据对象集准确地划分为多个子集。

（4）最后分析算法计算结果，由已经训练过的测试分类来判别聚类结果与标准聚类结果的差异。

2.2 聚类算法的并行化概述

海量数据的数据类型各异、体量巨大、价值密度低、挖掘难度大，而传统的数据聚类模型及其优化算法大多在单机上进行串行运算，由于单机的计算资源（如处理器数量、内存容量）有限而造成挖掘算法不能快速准确地完成数据挖掘任务，可以利用多台主机的计算资源来更加高效地执行数据分析任务，所以在处理大数据集时，单机上有限的内存容量和 CPU 处理速率都会成为制约聚类算法分析性能的瓶颈，如果将聚类分析算法并行化也许是解决大数据处理瓶颈的有效手段。

通常使用消耗的运行时间长短、占用的存储空间大小作为衡量串行算法性能的标准。但是这些串行算法的衡量标准却不一定适用于并行算法，因为数据通信、任务调度、数据存储、数据共享、数据划分等方面都与并行算法的实际运行紧密相连，所以通常使用算法的计算成本、加速比、可扩展性来衡量一个并行算法的性能好坏。以下是这些衡量标准的一般化定义：

（1）计算成本 $E_n = T_n * n$：并行算法的计算成本 E_n 是指计算时间 T_n 与并行执行的节点数 n 的乘积，即 $E_n = T_n * n$。

（2）加速比 $E_r = T_s / T_r$：加速比是测试算法并行化性能的重要指标之一，它体现了通过并行化使算法运行时间缩短而得到的整体性能提升。加速比的计算公式中 T_s 表示算法在单机环境下运行所需要的时间，T_r 表示算法在由 r 个计算节点所组成的集群环境下运行所需要的时间。加速比 E_r 愈大，说明该算法并行计算所需要的相对时间愈少，进一步说明算法的并行化效率愈高。

（3）可扩展比 $E_d = E_r / r$：扩展比也是检测并行算法性能的重要指标之一，它描

述了通过不断提高集群中计算节点数目时集群的利用率情况。随着集群中计算节点的增加，如果算法的扩展比曲线趋于稳定，则可以认为算法并行化时的可扩展性出色；如果算法的扩展比曲线下降幅度较快，则可以认为该算法并行化时的可扩展性逊色。

2.3 分布式计算框架之 Hadoop 研究现状

2.3.1 MapReduce 并行编程模型详述

Hadoop 主要由 MapReduce 并行计算框架和分布式文件系统 HDFS（Hadoop Distributed File System）两部分组成，Hadoop 是搭建于廉价机器集群之上的分布式文件系统，它可以最大化的利用集群资源（CPU、Memory），而且 Hadoop 也还具有比较优良的扩展性能和海量数据集的存储性能。Hadoop 作为一个海量数据处理的分布式并行计算框架，具有高可信度、高可扩展性、高效率和强容错机制等优势。

Hadoop 体系中的 Mapreduce 是一个主从结构的并行计算模型，其中主控节点下的 Jobtracker 进程负责将任务分派给计算节点中 Tasktracer 进程。作为 MapReduce 作业的主控程序，Jobtracker 运行于单个主控节点上，它的主要任务是负责管理作业的创建与调度、任务划分与分发和任务执行进展的监控等[14]。作为 MapReduce 作业的计算节点，Tasktracker 运行于所有的从节点上，它的主要任务是负责接收并启动 MapReduce 子任务、定期将从节点的任务执行情况上报给主控节点中的 Tasktracker 执行。Mapreduce Task 的主要原理如下图 2-1 所示：

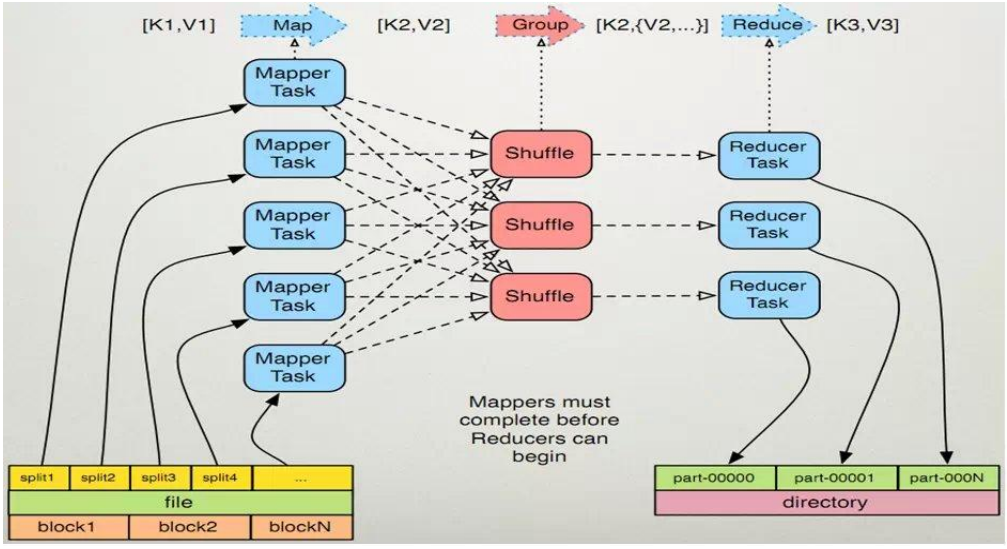


图 2-1：MapReduce Task 执行流程图

MapReduce 作业的执行流程如下描述所示：

(1) 提交 MapReduce 作业： 提交 MapReduce 作业之后，程序会自动运行，用户只能够由监视系统查看 Task 的运行情况。

(2) 初始化 MapReduce 作业 ：从任务提交开始，Jobtracker 会依据已经配置好的 TaskScheduler(默认情况下为 FIFO 调度策略)去执行用户任务，同时也需要调用 InitTasks 函数来对用户任务进行初始化，主要就是将输入数据集分块成指定的小数据块，然后创建并初始化 MapReduce 任务。

(3) 任务的分配：各个从节点的 Tasktracer 通过心跳信息向主控节点的 Jobtracker 上报运行情况，然后对 Tasktracer 的心跳信息进行处理，一旦接收到 Task 的任务请求，Tasktracer 则会将任务划分后分配给各个从节点的 Jobtracker。

(4) 任务的执行： 主节点的 Tasktracer 接收到任务后，首先对本地任务进行初始化，即将任务所需要的初始配置信息、程序代码和原始数据集从 HDFS 中 拷贝到本地)，然后各个从节点分别调用 MapTaskRunner、ReduceTaskRunner 去执行 Map 、Reduce 任务，并将局部计算结果输出。

(5) 作业完成：主节点上的 Jobtracker 对各个从节点的 Tasktracer 任务执行情况进行汇总，一旦 Jobtracker 确认任务完成则将作业的运行状态设置为已完成，然后 JobClient 通知用户程序任务已完成，返回全局计算结果。

2.3.2 HDFS 分布式文件系统概述

HDFS 能够对超大文件进行存储，也可以满足用户对数据流形式访问的需求，而且在廉价的服务器集群上还具有很强的容错能力。HDFS 是由 NameNode 和 DataNode 两种节点组成的分布式文件系统，其中 NameNode 负责命名空间的管理以及客户端之间文件传输的监控， DataNode 存储着实际的数据集，数据集是以数据块的形式存储在各个从节点上。

HDFS 的命名空间（namespace）支持 create、delete、copy 和 rename 等操作。HDFS 中的文件都是以 blocks 的形式连续存放，而且用 block 做文件容灾备份可以更好地提高 HDFS 系统的容错能力。DataNode 会周期性地向 NameNode 上报心跳信息以及 BlcokerReport，这样可以保持 NameNode 对集群中各个 DataNode 的实时监控，如果收到心跳则表示此 DataNode 在线，其中 BlcokerReport 则记录了其他 Blcok 存储在了在哪些 DataNode 里面。

2.4 分布式计算框架之 Spark 研究现状

Spark 是基于 DAG（有向无环图）计算模型和 RDD（弹性分布式数据集）实现的分布式计算框架， Spark 的中间输出和最终结果输出都能够存储在内存中，而不需要像 Hadoop 那般反复地读写 HDFS，节省了大量磁盘与内存间 IO 的开销，所以 Spark 才可以较好地运用于数据挖掘和机器学习等需要迭代计算的算法中。

2.4.1 Spark 的体系结构

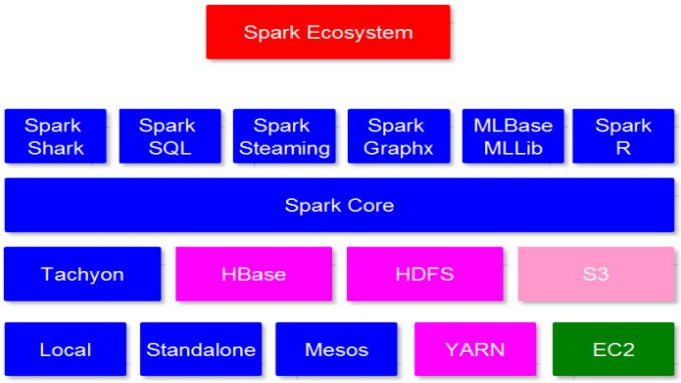


图 2-2：Spark 系统架构图

Spark 是一个基于内存计算的快速数据处理框架，其系统架构如图 2-2 所示。Spark 使用 YARN 作为其集群资源管理器以便实现资源的隔离和共享。它将数据对象集缓存在内存中来提升应用程序对数据对象的读写速率。

Spark 提出的 RDD（Resilient Distributed Datasets）是一种弹性分布式数据集，应用程序可以将中间计算结果暂时缓存到内存中方便下一次迭代计算，这样不仅能节省不必要的 IO，还可以实现数据集的重用，进而可以优化迭代计算的负载。

Spark 的并行计算模型使用了和 Hadoop 不同的 DAG（有向无环图）编程模型，同时也改进提升了 MapReduce 的并行效率。Spark 底层文件系统可以同时使用多种分布式文件系统来存放数据，它可以单机运行在 local 模式下也可以集群运行在 Yarn 或者 Memos 模式下，不过常见的计算环境一般是搭建于 YARN 等资源管理平台上。

2.4.2 RDD 内存计算概述

Spark core 包含 Spark 的基本功能，这些功能包括任务调度，内存管理，故障恢复以及存储系统的交互等，其中 RDD 更是其重中之重的核心思想，它将数据集缓存在内存中，并用 Lineage 机制来进行容错。

（1）RDD 功能特性

RDD 是 Spark 整个系统的的架构基础，RDD 具有智能容错机制、位置感知调度和可伸缩性， RDD 还支持粗粒度的转换以便恢复丢失的分区。RDD 的数据存储结构具有不可变性，还支持跨分布式集群的数据操作，还可以对数据对象按 key 值进行分区，还提供粗粒度的数据转换操作，最为重要的一点是 Spark 将数据存储在内存中形成 RDD 可以保证迭代计算的低延迟性。

（2）RDD 编程接口

Spark 提供了丰富的 API 来操作这些数据集，RDD 包含 2 类 API：Transformations 和 Action。Transformations 转换操作，返回值还是一个 RDD，如 map、filter、union；Actions 行动操作，返回结果或把 RDD 持久化起来，如 count、collect、save。

2.4.3 Spark 相比于 Hadoop 的优势分析

随着 Hadoop 的逐步普及它的缺陷也慢慢显露出来，越来越多的案例分析表明 Hadoop 自身的 MapReduce 结构比较单一，基本上只适用于数据量较大、计算简单的挖掘任务[30]。在用 MapReduce 计算模型去实现含有递归、迭代等较为复杂的计算任务时，不但编程过程比较复杂，而且算法的实际处理效率也差强人意。由于大多数数据挖掘算法是依靠迭代来不断重复迭代计算进行数据实际处理，所以下文以迭代计算模型来举例说明 MapReduce 计算模型的计算瓶颈。

（1）虽然每一次迭代计算执行的计算操作都相同，但是每次的迭代计算都是以一个独立的新作业执行数据处理操作，都要重新初始化参数设置、读写数据以及各节点间数据的传输，这样就会造成许多系统开销浪费。

（2）大部分数据在迭代计算过程中基本上是一成不变的，但是在每一次迭代计算时仍然会被系统重新读写和计算处理，这样就会导致大量 I/O、CPU 资源和网络带宽等系统资源被不停的浪费。

（3）每一次新的迭代计算都必须等待上一次迭代计算全部完成、中间输出数据全部载入 HDFS、判断是否满足迭代终止条件。

综上所述，MapReduce 并行计算模型在处理含有迭代的计算任务时效率并不是那么理想，但迭代计算在数据挖掘、模式识别、深度学习等领域都是比较常用的重要模型，而基于内存计算 Spark 并行计算框架相比于 Hadoop 却能够比较好地处理频繁迭代计算难题[31]。

Spark 和 Hadoop 都可以基于 MapReduce 进行并行计算处理，但是 Spark 与 Hadoop

在迭代计算方面的处理方式却大相径庭。Spark 会把迭代计算所需要的初始数据集定义为 RDD 并以分区的形式加载到集群中所有计算节点的内存分区中，接着由计算节点里的任务集对本地内存分区执行迭代计算。当计算节点的内存远远大于待处理的数据集时，迭代计算过程中应用程序基本无需和磁盘进行数据 IO，从而在很大程度上进一步加快了执行时间。

2.5 本章小结

本章先概述了聚类分析技术的具体思想、执行流程及其并行化思想，再简明扼要地从 MapReduce 编程思想、HDFS 分布式文件系统两个方面系统地介绍了 Hadoop 数据处理框架，最后在已有 Hadoop 知识的基础上从 Spark 体系结构、RDD 内存计算两个方面具体地介绍了 Spark 在迭代计算方面所具有的优势。

3 分布式计算环境下 K-means 算法的并行化研究现状

在数据挖掘中，聚类分析是不可缺少的重要分支，聚类算法根据数据对象的特有属性去聚合成各类“簇”（cluster），其中常用的属性聚合标准有基于划分、基于密度、基于分布、基于模型等情况。没有“大而全”普适的聚类算法,合适的算法取决于应用对数据集和其结果的预期用途。在这些聚类分析算法中，K-means 算法是一种简单高效、实用性很强的算法，它在处理少量的数据时具有比较好的聚类效果。在处于大数据背景的当下，已有的传统 K-means 聚类算法已经不能高效地完成海量数据的挖掘任务，它会受制于庞大的数据量、有限的硬件资源配置（如 CPU、内存、磁盘等资源），此时传统的 K-means 算法在单机上运行极为耗时，不能有效地执行聚类挖掘的任务，所以此时可以利用并行计算技术来进一步提高聚类算法的处理效率。在本章中主要是介绍传统 K-means 算法的基本思想，并将其应用到 Hadoop 和 Spark 的分布式计算平台上，通过利用分布式计算思想实现 K-means 的并行化来提高其运行的效率和有效性。

3.1 K-means 算法概述

作为聚类分析中最常用的算法之一， K-means 聚类算法通常可以产生比较良好的聚类效果。Mineset 作为 SGI 提供的一种有名的数据挖掘工具也将 K-means 聚类算法作为其标准聚类分析算法之一。K-means 聚类算法还被应用于科学统计软件包（SPSS），其中 SPSS 是全球三大统计分析软件包之一。另外 K-means 聚类算法还可以应用于其他领域，例如语音识别、基因数据分析、生物信息学中的生态系统数据分析、地理信息系统分析等领域[32]。

传统的 K-means 算法的执行逻辑具体如下：

（1）从原始数据集中随机选取 k 个数据对象，其中每个数据对象都代表一个初始聚类的质心；

（2）遍历数据集中所有的数据对象，把它们分配到相似度最高的一类簇中，其中相似度计算标准是以该数据对象和质心的距离长短来衡量的，即取数据对象与质心间最小欧式距离：$D(x,y)=\sqrt{\sum_{i=1}^{m}(x_i-y_i)^2}$；

（3）其后该类簇的新的质心就会被重新计算，一般用最小误差平方和对其进行

定义：$Z=\sum_{j=1}^{K}\sum_{i=1}^{M_i}(s_i-\overline{C_i})^2$；

（4）该过程继续以迭代的方式进行下去，重复步骤（2）和步骤（3）直到标准函数收敛，即当新中心点和旧中心点的距离低于某个选定的阈值时迭代结束。

K-means 算法的伪代码逻辑描述如 Algorithm 3-1 所示：

Algorithm 3-1:Pseudo-code of K-means algorithm general framework

Input: k, KmeansCenterList $C(C_1、C_2、\ldots、C_k)$

1: **Begin** initialize $n,k,C_1,C_2,...,C_k$

2: **Do** 按照最近邻C_i分类n个样本

3： 重新计算C_i得到$C_i^{'}$

4： **If**（ $C_i^{'}-C_i<=\alpha$，其中α为迭代停止阈值）

5： Return $C_1,C_2,...,C_k$

6： **Else** $C_i=C_i^{'}$

7： **End If**

8： **End Do**

9： **End**

3.2 K-means 算法的计算瓶颈及其改进策略分类

传统 K-means 算法必须重复迭代计算所有对象与质心的欧式距离，故传统 K-means 算法的时间复杂度与数据集大小正相关，它的时间复杂度是$O(kntd)$=子类簇数*数据对象数*迭代次数*数据对象的维度，其中k是子类簇的数量，n是数据全集中所有对象的数量，t是算法迭代计算的次数，d是计算的特征属性个数。虽然 K-means 目前已经成为应用比较广泛的一种聚类方法，它在许多现实应用场景中都具有良好的聚类效果，但传统的 K-means 算法也存在诸多缺陷。

（1）$O(kntd)$与 k 值成正比。K-means 的算法复杂度为$O(kntd)$，当k值选取过大时会直接影响算法整体的效率，当k值选取过小时又会丢失聚类的精准度。它要求预先初始化一个确定的k值，这个k值的确定又是非常难以估计的，一般都是依据使用者的经验得来，这就造成k值选取的主观性问题。

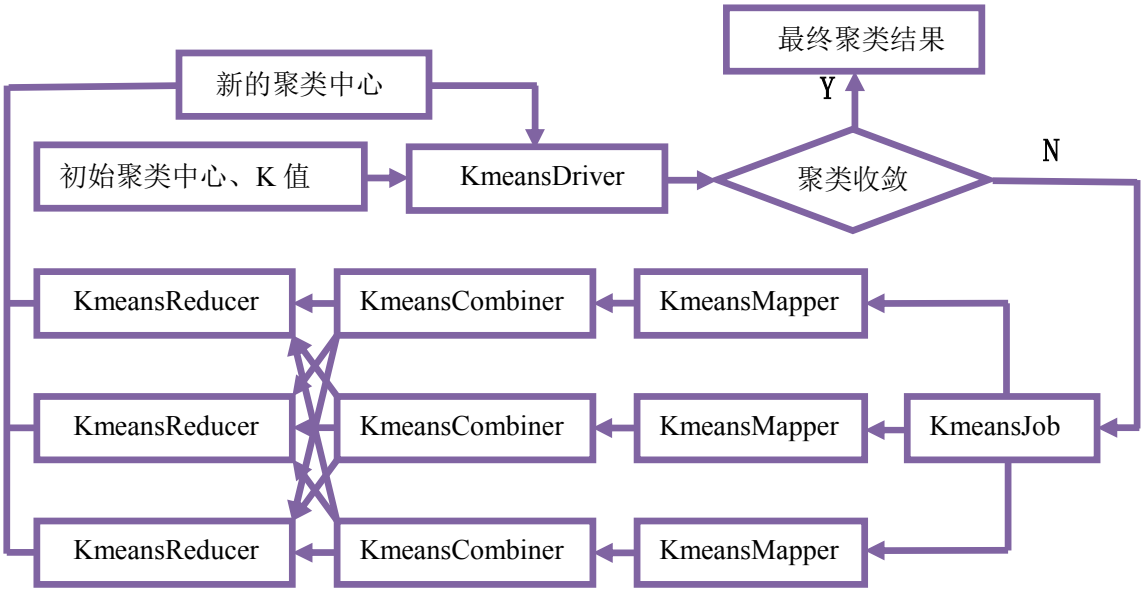
（2）$O(kntd)$与迭代次数t成正比。整个算法的时间复杂度为$O(kntd)$，其中迭代次数t很大程度上是由初始中心点的选取决定的，一旦初始值选择的不好，可能无法得到有效的结果，这就初始中心点选取造成的随机性问题。

（3）$O(kntd)$与数据量n成正比。当数据集的数据量变得十分庞大时，所有的数据对象不可能同时加载入内存进行计算，此时就需要在内存和硬盘间频繁地交换数据，而此种数据交换自然而然会降低数据挖掘的效率，这样就需要耗费大量的CPU、IO、Memory 等系统资源 [33]。

在数据量不是很多的应用中，传统的 K-means 算法还是可以发挥比较满意的数据挖掘效果，但处于信息大爆炸的今天，传统的 K-means 算法已然不能适应海量数据的分析处理，此时就需要对传统的 K-means 算法进行改进优化，而将并行计算的思想融合与传统的 K-means 算法正是一种不错的解决方案[34]。

3.3 基于 Hadoop 的 K-means 算法并行化设计与实现

聚类过程中一般需要多次进行迭代过程，这就需要在聚类主程序中多次建立 MapReduce 新任务。基于 Hadoop 的 K-means 算法实现的聚类流程如图 3-1 所示。



3-1：K-means 聚类流程图

每一次迭代过程结束，都会依据当前的聚类结果进行迭代收敛条件判断。如果未满足收敛条件，则反复进行迭代计算，直至聚类结果满足迭代收敛条件或者迭代次数达到最大迭代次数为止。在执行聚类任务时，Map、Reduce 的任务调度过程皆

是由 Hadoop 自身的系统进程自动实现，同时 Hadoop 还有容错的任务调度解决方案。

Hadoop 下 K-means 算法的实现主要包含 KmeansDriver，KmeansJob，KmeansMapper，KmeansReduce，KmeansCombine 这 5 个文件，其函数说明代码如下：

KmeansDriver 函数为程序的主入口，step2-3 从 HDFS 上的初始数据集中随机抽取 k 个初始类簇中心点，并将抽取结果存储在 InputPath 路径下 HDFS 文件中作为初始输入；step4-8 是聚类分析的主要逻辑过程，step6 实例化一个 MapReduce 作业 KmeansJob，step7 将 KmeansJob 提交给 Hadoop 的 MapReduce 框架启动执行；KmeansJob 执行结束后检查前后两次类簇中心点的距离之差与阈值 threshold 作比较；如果不满足迭代停止条件则继续执行 step4-8 直到聚类收敛，其中 step5 将上一次迭代计算的类簇中心点作为本次迭代的初始中心点，其函数伪代码 Function3-1 如下。

Function 3-1: Code of KmeansDriver function general framework

```
1. boolean    converged = false
2. CenterPoint[] = RandomSeed(DataPath,k)
3. PersistHDFS(CenterPoint[],InputPath)
4. While(!converged){
5.     OutputPath = InputPath
6.     Job KmeansJob=new          Job(DataPath,Inputpath,OutputPath,threshold)
7.     KmeansJob.Submit()
8.     Converged=IsConverged(InputPath,OutputPath,threshold)
9. }
```

KmeansJob 函数用于配置 MapReduce 作业执行前所需要配置的函数模板及文件的 IO 参数，其函数伪代码 Function3-2 如下。

Function 3-2: Code of KmeansJob function general framework

```
1. job.setJarByClass(KmeansDriver.class)
2. job.setMapperClass(KmeansMapper.class)
3. job.setMapOutputKeyClass(Text.class);
4. job.setMapOutputValueClass(Text.class);
5. job.setReducerClass(KmeansReducer.class);
6. FileInputFormat.addInputPath(KmeansJob, new Path(args[0]));
7. FileOutputFormat.setOutputPath(KmeansJob, new Path(args[1]));
```

KmeansMapper 函数主要执行局部的数据聚类，step1 将一条文本数据解析成一个符合处理格式的数据对象 point；step3-5 计算 point 与类簇中心列表中最短距离的类簇中心 centers[i]并标记 index；step7 输出数据记录（类簇索引 index，数据对象 point）。其函数代码 Function3-3 如下。

Function 3-3: Code of KmeansMapper function general framework

```
1. DoubleWritable point[]=ParseVector(TextString)
2. Double distance=Inf, minDistance=Inf ;Int index=-1
3. For(int i=1;i<=k;i++){
4.       Distance=Distance(point，Centers[i])
5.       If(Distance<minDistance)    minDistance = distance;
6. index = i
7. }
8. Context.write(index,point)
```

KmeansReducer 对 KmeansMapper 函数生成的中间值进行合并操作，它可以减少中间值的数量，从而减少 Map 阶段之后中间值在计算节点间数据传输开销。Step2-4 遍历某一类簇中的数据对象列表，统计该类簇中全部数据对象的特征和 SumPoint、数据对象数目 count；step5 生成新的键值对 value2 用于存放（SumPoint，count）；step6 将新的键值对（key2，value2）写入文件。其函数伪代码 Function3-4 如下。

Function 3-4: Code of KmeansReducer function general framework

```
1. DoubleWritable    SumPoint[] = new DoubleWritable[k]
2. for point in list[value1]{
3.       Count++
4.       SumPoint += list[point]
5. }
6. SumPointWritable value2 = new    SumPointWritable(SumPoint,count)
7. Context.write(index,value2)
```

KReducer 类主要负责全局聚类，接受来自 KMapper 合并后的输出结果，进行排序后对每个子类簇进行聚类运算。Step1-3 初始化中间变量；step4-8 用于计算同一类簇中全部数据对象的特征值之和，然后计算类簇新中心点；step9 输出中心点到文件，

其函数代码实现 Function3-5 如下。

Function 3-5: Code of KReducer function general framework

```
1. Int number = 0
2. DoubleWritable    AllPoint[] = new    DoubleWritable[]
3. DoubleWritable    CenterPoint[] = new    DoubleWritable[]
4. for value2 in list[value2]{
5.         number += value2.count
6.         AllPoint += list[value2.SumPoint]
7. }
8. CenterPoint = AllPoint/number
9. Context.Write(index,CenterPoint)
```

3.4 基于 Spark 的 K-means 算法并行化设计与实现

Spark 与 Hadoop 之间最大相异之处为 Spark 对所有类簇中心点的全部迭代计算基本都是于内存中计算完成，当数据处理量远小于节点的内存容量时，计算过程中几乎不需要再与磁盘进行 IO，而 Hadoop 则不行，基于 Spark 的 Kmeans 算法的并行实现流程如图 3-2 所示。

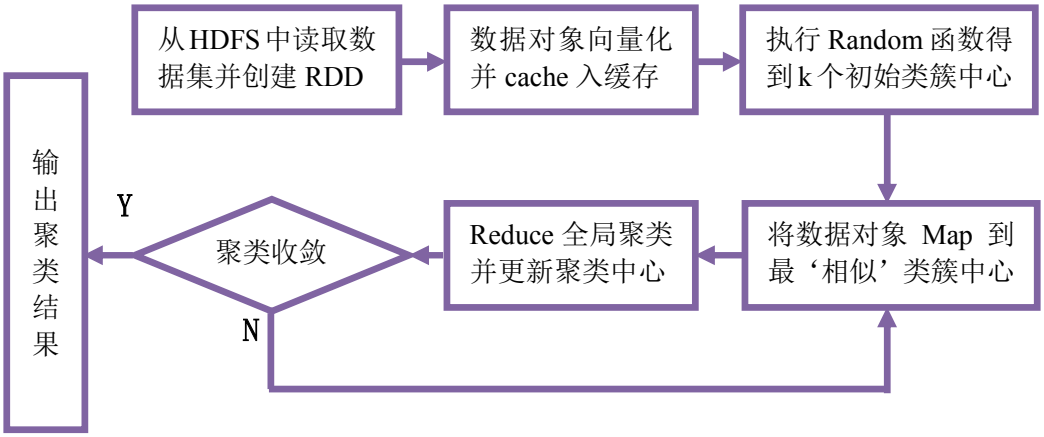


图 3-2：Kmeans 算法在 Spark 上并行化的流程图

Spark 使用 scala 语言来实现 K-means 算法的并行化代码编写，因为 scala 语言的表现力比较强大，其代码相比 java、C++、Python 等编程语言要简洁许多[35]。K-means 算法用 scala 编程的主要代码解释如下 Algorithm 3-2 所示：

Step4-5 数据预处理函数定义，Step6-18 将数据对象划分到最近的聚类中心点函数定义，Step19-45 为 K-means 的主函数，为整个程序的入口。其中 Step20-21 执行

Spark 集群初始化，Step22-24 从 HDFS 读入预处理过的数据向量、输入参数对相应变量进行初始化，Step25 将每个数据分区以 RDD 的形式缓存到内存中，Step26 随机产生 k 个初始类簇中心，Step27-29 完成相应变量的初始化操作。Step30-42 为 Kmeans 算法主要代码段，当聚类准则比较变量 tempDist 比聚类准则值 convergeDist 大，而且当前迭代次数 tempIteration 小于最大迭代次数 MaxIteration 时，运行循环内容，具体执行流程如下：Step31 求数据对象的局部数据聚类，由 closestPoint 函数求得数据向量 p 与哪一个类簇中心的相似度最高，然后标记数据对象的所属类别，每个数据对象 p 可以映射成（id，（point，1））。 Step32-33 中的 pointStats 操作是对具有相同 id 的 p 进行合并，可以表示为（数据对象特征值和，数据对象数量和）。Step34 中 newPoints 变量是指求得的新类簇中心点，键值对（id， 数据对象特征值和/ 数据对象数量和）。 Step36 将 tempDist 重新归零。Step37-38 求新旧类簇中心点的差平方和。Step39-40 更新聚类中心。Step42 对当前迭代次数进行更新。

Algorithm 3-2: Code of Canopy_K-means algorithm general framework

```
1: object SparkKmeans {
2:    val R = 1000
3:    val rand = new Random(42)
4:    def parseVector(line:String): Vector[Double] = {
5:    DenseVector(line.split(‘ ‘).map(_.toDouble))    }
6:    def closestPoint(p:Vextor[Double] , centers: Array[Vector[Double]]):Int ={
7:        var index =0
8:        var bestIndex = 0
9:        var closest = Double.positiveInfinity
10:       for( i<-0 until centers,length)    {
11:           val tempDist = squaredDistance(p,centers(i))
12:           if (tempDist < closest)    {
13:              closest = tempDist
14:              bestIndex = I }
15:       }
16:       bestIndex
17:   }
18:   def main(args:Array[String]) {
19:       val SparkConf = new SparkConf().setAppName(“SparkKmeans”)
```

续 **Algorithm** 3-2:Pseudo-code of Canopy_K-means algorithm general framework

```
20:       val sc = new SparkContext(”SparkKmeans”)
21:       val lines = sc.textFile(args(0))
22:       val K = args(1).toInt
23:       val convergeDist = args(2).toDouble
24:       val data = lines.map(parseVector _).cache()
25:       var kPoints = data.takeSample(withReplacement = false,K,42).toArray
26:       val tempDist=1.0
27:       val MaxIteration = args(3).toInt
28:       Val tempIteration = 0
29:       while(tempDist > convergeDist && tempIteration < MaxIteration){
30:            val closet = data.map (p => (closestPoint(p,kPoints),(p,1)))
31:            val pointStats = closest.reduceByKey {
32:                 case ((x1,y1),(x2,y2)) => (x1+x2 , y1+y2)    }
33:                 val newPoints = pointStats.map {pair =>
34:                 (pair._1,pair._2._1*(1.0/pair._2._2))}.collectAsMap()
35:                 tempDist = 0.0
36:                 for(i <-0 until K){
37:                      tempDist += squaredDistance(kPoints(i),newPoints(i))    }
38:                 for(newP <- newPoints) {
39:                      kPoints(newP._1) = newP._2    }
40:                 println(Finished iteration (delta = “+ tempDist +”)”)
41:                 tempIteration= tempIteration + 1      }
43:       println(“Final centers:”)
44:       kPoints.foreach(println)
45:       sc.stop()      }
47: }
```

3.5 本章小结

本章主要是实现传统 K-means 算法在分布式平台 Hadoop、Spark 上的并行化设计思路，并给出了具体实现以比较两种分布式计算平台的性能。分析表明，基于 Spark 的 K-means 算法比基于 Hadoop 的并行化实现性能更好。

4 分布式环境下的 K-means 改进算法的并行化研究

基于 Spark 的 K-means 算法的并行化实现策略虽然可以在集群计算节点增多的条件下提高数据并行挖掘的效率，但是当集群中的节点资源有限时其计算效率则达到瓶颈，此时以优化 K-means 算法内部效率来作为突破口将是一个不错的选择。为了提高 K-means 聚类算法对维度比较高的海量数据集的聚类效率，同时也解决 K-means 算法类簇初始中心点的选取和 k 值的选取问题，本章选取 Canopy 粗糙聚类算法来对基于 Spark 分布式框架的 K-means 并行算法进一步改进[36]。通过 Canopy 算法优化初始中心点和 K 值可以降低 k-menas 算法的迭代次数，从而改进 K-means 聚类算法在处理大数据集时数据挖掘的效率和实用性，也可以避免局部最优解。

4.1 粗聚类算法之 Canopy 算法思想详述

4.1.1 Canopy 算法思想概述

Canopy 算法适用于高维度的大数据集预聚类分析，它使用简单粗糙的距离测量方法可以把数据对象集高效地分割为多个不完全重叠却可以相交的子集 Canopy，接着再在各个 Canopy 中执行精细聚类算法，这样粗细聚类的结合能提高海量数据处理的效率和准确性。

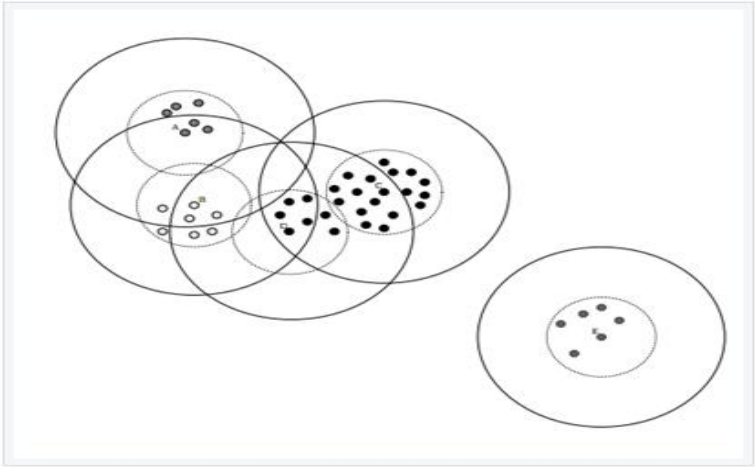


图 4-1：Canopy 算法预聚类简图

Canopy 算法聚类逻辑过程如上图所示，首先从数据对象点集中随机地选择一个数据对象点 A 作为第一个子集 E_1 的聚类中心点，然后再根据其他数据对象点与 A 的相似度（一般以欧式距离作为相似度计算准则），将一些数据对象点划分到子集 E_1 中；接着通过子集 canopies 的创建，将数据对象集中的所有数据对象划分到各个子

集 Canopy 中，使这些 canopy 子集必须完全包含全部数据对象。每个数据对象至少要属于其中的任何一个 Canopy 子集,而且它还有可能同时属于一个以上的 Canopy 子集。由上述 Canopy 算法定义可知：任意两个数据对象点如果没有归属于同一个子集 Canopy，那么这两个数据对象点就一定不可能是同一类簇的数据对象。

4.1.2 Canopy 算法流程详述

虽说 Canopy 算法可以不用如 K-means 算法那样人为设置聚类的数目，但仍需要设置两个相似性度量标准 T_1、T_2 来间接决定聚类后子集 Canopy 的数目和每个子集 Canopy 中数据对象点的数目，Canopy 算法的详细计算流程如下所示。

Step1： 将数据对象集 D 预处理后得到一个数据向量列表 D' 后写入内存，选取距离阈值：T_1、T_2，其中 $T_1 \geq T_2$ 对应上图，而且 T1 和 T2 可以通过交叉校验决定；

Step2： 从数据列表 D' 中随机选取数据对象 P 并计算 P 与全部 Canopy 子集中心点之间的距离（假如 Canopy 不存在，则可以将数据对象 P 作为一个新的 Canopy 而加入到 Canopy 列表中去，同时将 P 从 D' 中去除），如果 P 与某个 Canopy 中心的距离小于等于 T_1，则可以将点 P 加入此 Canopy 中的数据对象列表 $D_i'(i=1,2...k)$ [37]；

Step3： 只要数据对象对象 P 与某个 Canopy 中心的距离小于等于 T_2，则需要将数据对象 P 从数据列表 D' 中去除，这一步主要是判断数据对象点 P 与此 Canopy 已经比较接近，故 P 不能再当作另外 Canopy 子集的中心；

Step4： 重复步骤 2、3，当数据列表 D' 为空时迭代计算结束。

Canopy 算法的伪代码描述如 Algorithm 4-1 所示：

Algorithm 4-1:Pseudo-code of Canopy algorithm general framework

Input：T1、T2、DataList D' ；**Output**：CanopyCenterList $C(C_1、C_2、\ldots、C_k)$

```
1：While DataList is not empty do
2：     take a data p from DataList
3：     If  CanopyCenterList is null
4：         add p to CanopyCenterList
5：         delete p from DataList
6：     Else
7：         For each CanopyCenter in CanopyCenterList
8：             If  Distance(p, CanopyCenter) > T1
```

续 **Algorithm** 4-1:Pseudo-code of Canopy_K-means algorithm general framework

```
9：                    add p into CanopyCenterList
10：                   delete p from DataList
11：              Else If  Distance(p, CanopyCenter) <= T2
11：                   delete p from DataList
12：              Else
13：                   add p into Canpy Ci
14：               End If
15：        End If
16：End While
```

4.2 改进算法之 Canopy_K-means（CKM）算法性能分析

Canopy 算法不需要人为去主动指定聚类的数目，它可以依据自身的迭代来主动聚簇成类，只需要设定聚类过程中的阈值 T_1、T_2，其中 $T_1 \geq T_2$，故可以将 Canopy 预聚类算法与 K-means 算法结合起来。当 Canopy 算法的输出作为 K-means 的输入（即将 Canopy 聚类输出的中心点列表作为 K-means 聚类输入的初始中心点列表，将类簇中心点的数目作为 K-means 聚类输入的 K 值）时，可以在一定程度上合理规避 K-means 算法中 K 值选取的主观性和初始类簇中心点选取的随机性，从而有效减少 K-means 聚类算法中的迭代次数来提高效率并同时提高聚类的准确性[25]，基于以上分析可以提出基于 Canopy 的 K-means 优化算法，即 CKM 算法。其详细聚类流程如下所示：

Step1： 将数据对象集 D 预处理后得到一个数据列表 D' 后写入内存，选取合适的距离阈值：T_1、T_2（$T_1 \geq T_2$）其中实线圈表示 T1 范围、虚线圈为 T2 范围，T1 和 T2 由交叉校验来决定；

Step2：从数据列表 D' 中随机选取数据对象 P 并计算 P 与全部 Canopy 子集中心点之间的距离（假如 Canopy 不存在，则可以将数据对象 P 作为一个新的 Canopy 而加入到 Canopy 列表中去，同时将 P 从 D' 中去除），如果 P 与某个 Canopy 中心的距离小于等于 T_1，则可以将点 P 加入此 Canopy 中的数据对象列表 $D_i'(i=1,2...k)$ [37]；

Step3：只要数据对象点 P 与某个 Canopy 中心的距离小于等于 T_2，则需要将数

据对象点 P 从数据列表 D' 中删除，这一步主要是判断数据对象点 P 与此 Canopy 已经比较接近，故 P 不能再做其它子集 Canopy 的中心；

Step4：重复步骤 2、3，当数据列表 D' 为空时迭代计算结束。通过 Step1-Step4 可以将数据对象集划分为 k 个 Canopy 子集，每个数据对象可以属于多个 Canopies,k 个 Canopy 子集并集必须为数据对象全集：$Canopy_1 \cup Canopy_2 \cup ... \cup Canopy_k = D'$，没有数据孤点。

Step5： 定义 k 个 Cluster 类簇中心点，其初始类簇中心点列表为 Step1-Step4 产生的 k 个 Canopy 中心点列表。

Step6： 由于 Canopy 之间可能相互重叠，一个数据对象可能属于多个 Canopies，所以需要将这样的数据对象分配给离它最近的 Canopy 中心点所对应的 Canopy 子集中。此时就需要计算每个 Canopy 中所有数据对象和 Canopy 中的类簇 cluster 中心点间的距离：$D(x, y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$

Step7： 其后该 Canopy 新的质心就会被重新计算，一般用质心公式对其进行定义：$\overline{C_i} = \frac{1}{M_i} \sum_{i=1}^{M_i} s_i$，产生新的类簇 cluster 中心点列表。

Step8： 计算新的类簇 cluster 中心点到相对应 Canopy 中心点的距离，如果其距离小于等于 T1，则将该类簇新的中心点划分到该 Canopy 子集中。

Step9： 该过程继续以迭代的方式进行下去，重复步骤 Step6-Step8 直到标准函数收敛，即当新中心点和旧中心点的距离低于某个选定的阈值时迭代结束。

CKM 算法的伪代码描述如 Algorithm 4-2 所示。

Algorithm 4-**2**:Pseudo-code of Canopy_K-means algorithm general framework

Input：T1、T2、DataList D' 、CanopyCenterList $C(C_1、C_2、\ldots、C_k)$、MaxIteration 、ConvergeDist

OutPut：ClusterCenterList $L(L_1、L_2、\ldots、L_k)$

1：Use the CanopyCenterList and k Canopies in Algorithm 1

2：$L(L_1、L_2、\ldots、L_k) = C(C_1、C_2、\ldots、C_k)$

3：Converged = false ,iteration = 0

4：**While** Converged is false

5: **For** each $Canopy(i)$ in Canopies

续 **Algorithm** 4-**2**:Pseudo-code of Canopy_K-means algorithm general framework

6: **For** each C_i in CanopyCenterList $C(C_1、C_2、\ldots、C_k)$

7: get the min*Value of* W_i : $Dist(L_i, C_i)$ in $Canopy(i)$

8: add C_i into corresponding $Cluster(i)$

9: **End For**

10 **End For**

11: **For** each $Cluster(i)$ in Clusters

12: get the $L_i' = \frac{1}{M_i}\sum_{i=1}^{M_i} s_i$

13: add L_i' into $L'(L_1'、L_2'、\ldots、L_k')$

14: **End for**

15: **For** each L_i' in $L'(L_1'、L_2'、\ldots、L_k')$

16: **For** each C_j in CanopyCenterList $C(C_1、C_2、\ldots、C_k)$

17: **If**($Dist(L_i', C_j) <= T1$)

18: add L_i' into $Canopy(j)$

19: **End If**

20: **End For**

21: **End For**

22: **If**($L_i' - L_i < ConvergeDist$)

23: Converged = true

24: **End If**

25: **End While**

传统 K-means 的时间复杂度是$O(kntd)$=子类簇数*数据对象数*迭代次数*数据对象的维度，其中k是子类簇的数量，n是数据全集中所有对象的数量，t是算法迭代计算的次数，d是计算的特征属性个数[38]。

经过优化后的 CKM 算法主要由 Canopy 预聚类算法和 K-means 精细聚类算法组成。在 Canopy 算法中，每个数据对象都可能属于多个 Canopy 子集，假设每个数据对象平均属于e个 Canopy 子集，则 CKM 聚类算法的时间复杂度为$O(kntd)$，其中 e值一般比 K 值要小。CKM 改进算法在聚类准确性上能体现其最大优势，在传统 K-means 算法中,需要事先预估 K 值和选取 K 个初始类簇中心，这样就会造成聚

类结果有一定程度的不准确性，而 CKM 算法可以将 Canopy 预聚类输出的 Canopy 个数作为 K-means 算法的聚类 K 值，这样能在很大程度上避免该问题[39]。

4.3 基于 Spark 的 Canopy_K-means（CKM）算法的并行化设计

CKM 算法主要由 Canopy 和 K-means 两部分组成，基于 Spark 的并行化思路就是依据 CKM 算法的两部分来执行并行化设计。基于 Spark 的 CKM 并行算法与其串行逻辑大致相同，不同之处就在于 CKM 并行算法在使用 DAG 并行编程模型的同时也利用了 Spark 最具特色的 RDD。在前文基于 Spark 的 K-means 算法的并行化实现的基础上，将 Canopy 算法的预聚类思想添加进去，则可以实现改进优化后的 CKM 算法的并行化实现[40]。CKM 算法基于 Spark 平台实现其并行化的流程大致如下：

Step1：配置 Spark 和 Yarn 集群的有关参数，初始化集群计算环境；

Step2：从 HDFS 中读出数据对象集生成初始 RDD；

Step3：各节点执行 Map 操作对原始数据进行格式化，将文本数据向量化；

Step4：集群中各个并行节点执行 Map 操作计算各自分片的数据对象与 Canopy 中心点的距离，从而得到局部 Canopy 中心点；

Step5：执行 Reduce 操作合并成全局 Canopy 子集中心点；

Step6：各节点依据全局 Canopy 中心点执行 Map 操作，将数据对象全集划分到不同的 Canopy 并执行 cache 操作将其数据持久化；

Step7：将 Canopy 中心点列表赋值给 Cluster 中心点列表；

Step8：各节点在经过 cache 之后的 RDD 进行 Map 操作执行 K-means 局部聚类；

Step9：主控节点执行 Reduce 操作将各个计算节点产生的局部聚类结果归并为全局聚类结果，更新类簇中心点。

Step10：判断迭代退出条件是否满足，如果满足则输出全局聚类结果，如果不满足则重复执行 Step8-Step9。

其中 Canopy 中的数据对象是放置在内存，所以在后面 K-means 聚类时候可以多次重复使用，并不需要每次都在 HDFS 分布式文件系统上读取，这样提高了算法实现的效率。Spark 会根据算法的逻辑自动实现 CKM 聚类算法的并行化，在并行化实现中的任务分配和数据分配都是由系统自动实现。

4.4 基于 Spark 的 Canopy_K-means（CKM）算法的并行化实现

基于 Spark 的 Canopy_k-mean 算法的并行化会首先使用 Map 操作执行 Canopy 局部预聚类和 K-means 局部精细聚类，然后再使用 Redcue 操作汇总各个计算节点局部聚类的结果，继而执行全局聚类。

Canopy_k-mean 聚类算法基于 Spark 分布式计算平台的并行化实现代码细节描述如下：

Step1：Kryo 并不是向所有的数据对象都提供数据序列化支持，使用 Kryo 前需要将应用中使用 class 提前注册进 Kryo，这时候需要对 spark.KryoRegistrator 进行 inherit 和 override，同时将系统属性参数 spark.kryo.registrator 指向继承和重写后的类 MyRegistrator。

Step3：设置 Spark 系统所占内存大小，这样愈发凸显 Spark 内存计算的优势。

Step4: 将 Spark 系统自带的的数据压缩功能启动，spark.rdd.compress 参数默认为 false。当磁盘 IO、GC 问题的确得不到比较好的解决办法时，这时启用 RDD 数据压缩功能是不错的选择。

Step5：优化 cache 大小，storage.memoryFraction 表示系统用来缓存 RDD 的大小所占 executor.memory 的比例，其缺省值为 0.6，剩下的 0.4 用于应用运行期间的对象创建。当 tasks 运行缓慢、内存空间不足、JVM 频繁执行内存回收操作，这时候就可以减小 cache 大小来进一步降低内存消耗。结合数据序列化，使用较小缓存可以处理大部分内存回收瓶颈。

Step6：设置 executor.memory 中 heap 比例，默认 0.2。

Step7-8：设置数据序列化方式并将类注册到 Kryo。

Step9：Hadoop HDFS 分布式文件系统的 Input 目录下读出数据对象集并生成初始 RDD data 以供后面计算所用。

Step10：将 RDD data 使用 map 操作并行向量化，将数据对象有文本形式转化为便于后面数据处理的向量形式。

Step11-12：使用 map 操作并行计算 data 中的数据对象和 Canopy 局部中心点的距离，当两者的距离大于阈值 T1，则该数据对象可以作为新 Canopy 中心点加入 Canopy 中心点列表，其中 Canopy_1 为 Scala 函数闭包。

Step13-14：合并各个节点计算生成的局部 Canopy 中心点列表，从而产生 Canopy

全局中心点列表。

Step15：对 RDD data 执行 map 操作，计算各数据对象和 Canopy 全局中心点的距离，距离小于 T2 则将其加入该 Canopy 子集，同时也要对经过标记后的数据对象执行 Cache 操作持久化在内存中，减少后面操作再从磁盘读出的消耗，其中 Canopy_2 为 Scala 函数闭包。

Step16：将 Canopy 中心点列表赋值为 Cluster 初始中心点列表。

Step18：对 RDD CanopyT2 执行 map 操作，计算各数据对象和属于同一 Canopy 子集中的 Cluster 中心点之间的距离，将其划分到距离最短的 Cluster 中心点所在的类簇中，其中 closestCenter 为 Scala 函数闭包。

Step19：对 RDD mapCanopyT2 执行 reduceByKey 操作，计算同一 Cluster 中数据对象的平均属性值，并将它作为新的 Cluster 类簇中心点。

Step21：对 newClusterCenterList 执行 Map 操作，计算各个新 Cluster 中心点所属的 Canopy 子集,计算新旧中心点的平方差，进而更新 Cluster 中心点。

CKM 算法基于 Spark 平台的并行化实现主要代码描述大致如下：

Algorithm 4-3: Code of Canopy_K-means algorithm general framework

Input：SparkMaster、DataInput、DataOutput、MaxIteration、ConvergeDist

OutPut：ClusterCenterList $L(L_1、L_2、\ldots、L_k)$

```
1:  class MyRegistrator extends KryoRegistrator {
          override def registerClasses(kryo: Kryo) {
            kryo.register(classOf[java.util.HashMap[_, _]])
          }  }//Kryo 数据序列化比默认的 Java 数据序列化性能通常能提高十倍；
2:  val sc = new SparkContext(SparkMaster).setAppName("Spark_Canopy_K-means")
3：          .set("spark.executor.memory", "2g")
4：          .set("spark.rdd.compress", "true")
5：          .set("spark.storage.memoryFraction", "0.4")
6：          . set("spark.shuffle.memoryFraction", "0.35")
7：          .set("spark.serializer", "org.apache.spark.serializer.KryoSerializer")
8：          . set("spark.kryo.registrator", classOf[MyRegistrator].getName)
9:   val lines = sc.textFile(DataInput)
10:  val data = lines.map(parseVector _).persist(StorageLevel.MEMORY_ONLY_SER)
11:  val CanopyMapCenterList = new HashSet()
```

续 **Algorithm** 4-**3**:Pseudo-code of Canopy_K-means algorithm general framework

```
12:  val crudeCenters = data.map{
         x=>(x._1,Canopy_1(x,CanopyMapCenterList,T1))}
          .filter(y=>y._2!=null).collect().toList
13:  val CanopyCenterList = new HashSet()
14:  for(i<- 0 util crudeCenters.size){
       Canopy_1(crudeCenters(i)._2 ,CanopyCenterList ,T1)}
15:  val CanopyT2 = data.map{
         x=>(x._1,Canopy_2(x,CanopyMapCenterList,T2))}.collect().toList
16:  val ClusterCenterList = new Hashset(CanopyCenterList)
17:  while(tempDist > ConvergeDist || iteration < MaxIteration){
18:       mapCanopyT2 = CanopyT2.map
          {do=>(closestCenter(do, ClusterCenterList),(do,1))}
19:  val newClusterCenterList = mapCanopyT2.reduceByKey
          {Case((s1,c1),(s2,c2))=>(s1+s2,c1+c2)}.map
          {Case(index,(s,c))=>(index,s/c)}.collect()
20:  val tempDist = 0.0
21:  for((index,value)<- newClusterCenterList){
         tempDist += ClusterCenterList.get(index).get.SquaredDist(value)
         ClusterCenterList(index)=Canopy_2(value,CanopyCenterList,T2)}
22:  }
```

4.5 本章小结

CKM 改进算法在 Spark 集群上的并行化实现主要从两个方面就行了改进和优化：一方面，研究利用 Canopy 粗糙聚类算法在及时简单处理大数据聚类的优势改进了 K-means 在初始 K 值和类簇中心点选取的算法自身缺陷，提高了聚类结果准确性的同时也加快了迭代计算的收敛速度，从而从算法自身提高了其整体执行效率；另一方面，由于 Spark 平台自身以内存计算为核心，为了充分利用 RDD 的特性优势，实验结合 Spark 平台自身的特性，从内存优化、数据压缩、数据序列化、运行内存占比、运行堆占比、缓存大小等系统配置方面进行 Spark 调优，从而利用 Spark 相比于 Hadoop 的平台优势进一步加快改进 CKM 算法的并行计算效率。

5 实验平台设计与结果分析

本章节使用 Yarn 资源管理器来搭建 Hadoop 和 Spark 两个分布式计算平台，实验通过对原有 K-means 算法和基于 Canopy 粗聚类算法的改进算法进行深入分析，进一步了解改进后的算法是否符合实验初始的期望，并且对两种计算平台进行比较，了解两个云计算平台在进行聚类和关联规则计算时的优缺点。

5.1 实验平台的搭建

5.1.1 硬件环境的配置

实验平台的集群系统在实验楼机房中搭建而成，集群由六台主机组成。表 6.1 描述了机器的具体配置。

本实验平台的分布式集群系统由 1 台主控节点（namenode）和 5 台计算节点（datanode）所组成，其中主控节点所在主机也作为计算节点使用。主控节点和计算节点之间以百兆以太网互联，硬件环境的具体配置如表 5-1 所示。

表 5-1 实验集群硬件环境配置详表

Name	Number	Description
Master	1	4GB RAM，2.1GHZ *core2 CPU，500GB Hard disk
Worker	5	3GB RAM，1.4GHZ *core4 CPU，981GB Hard disk
Ethernet	1	100Mb/s

5.1.2 软件环境的配置

作为搭建 Hadoop 和 Spark 分布式集群环境的必备条件，软件环境起着至关重要的作用，其中每台机器的软件环境具体配置如表 5-2 所示。

表 5-2 实验集群环境软件配置详表

Software	Configuration
OS	Ubuntu12.04
Java Environment	Jdk1.7.0_67
Hadoop(HDFS、Yarn)	Hadoop2.2.0
Scala Environment	Scala 2.10.4
Spark	Spark1.0.2

出于对实验室集群环境可扩展性考虑，实验对集群内的所有节点进行 IP 规划（修改系统中的“/etc/network/interfaces”文件实现手工配置静态 IP）和主机名的批次修改（通过修改“/etc/hosts”添加名字信息），同时对节点之间的 SSH 无密码连接进行了相关配置[42]。相关具体配置如表 5-3 所示。

表 5-3 实验集群环境节点 IP 配置描述

Hostname	IP Address
SparkMaster	211.69.197.84
SparkSlaver1	211.69.197.85
SparkSlaver2	211.69.197.86
SparkSlaver3	211.69.197.87
SparkSlaver4	211.69.197.88

5.1.3 集群管理平台 Yarn 的搭建

在将 Hadoop 部署在 Yarn 资源管理平台上前,首先需要将 Hadoop 分别安装到集群所有节点上,再各自通过修改配置文件将 Hadoop 的计算框架注册到 Yarn 集群。Yarn 作为最新一代资源管理平台，自 Hadoop2.0 版本之后都有内部 YARN 包集成，可采取比较便捷的方式来部署 Yarn 作为 Hadoop 和 Spark 的底层资源管理平台，只需要在下载 Hadoop2.2.0 版本后配置相应的环境变量和属性参数即可，具体参数配置如表 5-4 所示。

表 5-4 Yarn 资源管理器环境参数配置详表

FileName	Property	Value
yarn-env.sh	JAVA_HOME	/usr/lib/java/jdk1.7.0_67
mapred-env.sh	JAVA_HOME	/usr/lib/java/jdk1.7.0_67
core-site.xml	fs.defaultFS	hdfs://SparkMaster:9000
hdfs-site.xml	dfs.replication	5
mapred-site.xml	mapreduce.framework.name	yarn
yarn-site.xml	yarn.resourcemanager.hostname	SparkMaster
yarn-site.xml	yarn.nodemanager.aux-services	Mapreduce_shuffle

配置完成后，可以进入 SparkMaster 节点中 hadoop 安装目录并执行 bin/hadoop namenode -format 命令来格式化文件系统，然后执行命令 sbin/ start-dfs.sh 启动 HDFS 分布式文件系统，接着执行 sbin/start-yarn.sh 命令启动 Yarn 资源管理平台，或者执行

sbin/start-all.sh 命令替代前面两步来启动 Yarn 集群[43]。

5.1.4 在 Yarn 上搭建 Spark 开发环境

由于 Spark 集群是搭建于 Yarn 之上，所以只需将原有的 Hadoop 集群引入并修改部分文件即可完成 Spark 集群的搭建，主要需要修改的文件为 slaves 和 Spark-env.sh，其中 slaves 文件只需要添加集群中所有计算节点的主机名即可，而 Spark-env.sh 中具体参数配置如表 5-5 所示。

表 5-5 Spark 集群环境参数配置描述

Property	Value
JAVA_HOME	/usr/lib/java/1.7.0_67
SCALA_HOME	/usr/lib/scala/scala-2.10.4
HADOOP_HOME	/usr/local/hadoop/hadoop-2.2.0
HADOOP_CONF_DIR	/usr/local/hadoop/hadoop-2.2.0/etc/hadoop
SPARK_MASTER_IP	SparkMaster
SPARK_WORKER_MEMORY	2g

在对集群中的所有计算节点配置完毕上述文件后，首先利用 SSH 无密码登陆将所有计算节点 Datanade 在主控节点 Namenode 上注册，接着进入 Hadoop 安装目录执行命令/sbin/start-all.sh 启动 Hadoop 集群成功，再进入 Spark 安装目录执行命令/sbin/start-all.sh 启动 Spark 集群成功[44]。

5.2 实验数据准备

实验所准备的数据集都是经过数据预处理后的文本文件，为了能多方位地检测 K-means 算法及其改进算法在 Spark、Hadoop 平台上并行化的性能，本实验使用了不同大小的数据对象集来对比分析，数据对象集的规模及数据量如表 5-6 所示。

表 5-6 实验数据集详表

DataSet	FileSize	Instance
DATASET1	174.4 KB	14536
DATASET2	10.8 MB	901232
DATASET3	351.6 MB	27036960
DATASET4	1.72 GB	132262717
DATASET5	3.64 GB	279904820

5.3 实验过程及结果分析

实验主要从以下四个方面进行对比分析：

（1）分析 K-means 算法分别在单机、Hadoop 集群、Spark 集群下运行同一数据集时的性能对比；

（2）分析 K-means 算法、CKM 改进算法在 Spark 集群下运行不同数据集时的性能对比测试算法的准确性。

（3）分析 K-means 算法、CKM 改进算法在 Spark 集群下运行不同数据集时的性能对比测试算法的加速比；

（4）分析 K-means 算法、CKM 改进算法在 Spark 集群下运行不同数据集时的性能对比测试算法的可扩展性；

5.3.1 Hadoop、Spark 集群运行 K-means 算法的性能对比实验

首先为了验证 Spark 分布式集群在迭代计算上相对于 Hadoop 所具有的优势，本实验测试了 K-means 算法在 Hadoop、Spark 集群上的运行性能对比，以相同机器配置环境下相同计算节点数目时两者运行时间之比 $E1 = t_1 / t_2$ 作为对比条件,其中 t_1 表示 K-means 算法在 Spark 平台上所运行的时间，t_2 表示 K-means 算法在 Hadoop 平台上所运行的时间。实验使用使用数据集 DATASET1-DATASET5，运行结果如下表 5-7：

表 5-7 K-means 在 Spark、Hadoop 下运行性能对比表

DATA \ Worker	1	2	3	4	5
DATASET1	0.967	1.003	1.13	1.134	1.139
DATASET2	0.952	1.064	1.167	1.204	1.265
DATASET3	0.944	1.183	1.262	1.338	1.475
DATASET4	0.925	1.31	1.452	1.53	1.687
DATASET5	0.751	0.784	0.836	0.93	0.953

为了更直观地体现上述运行结果的对比情况，由上表所统计数据可以作出趋势曲线图如图 5-1 所示：

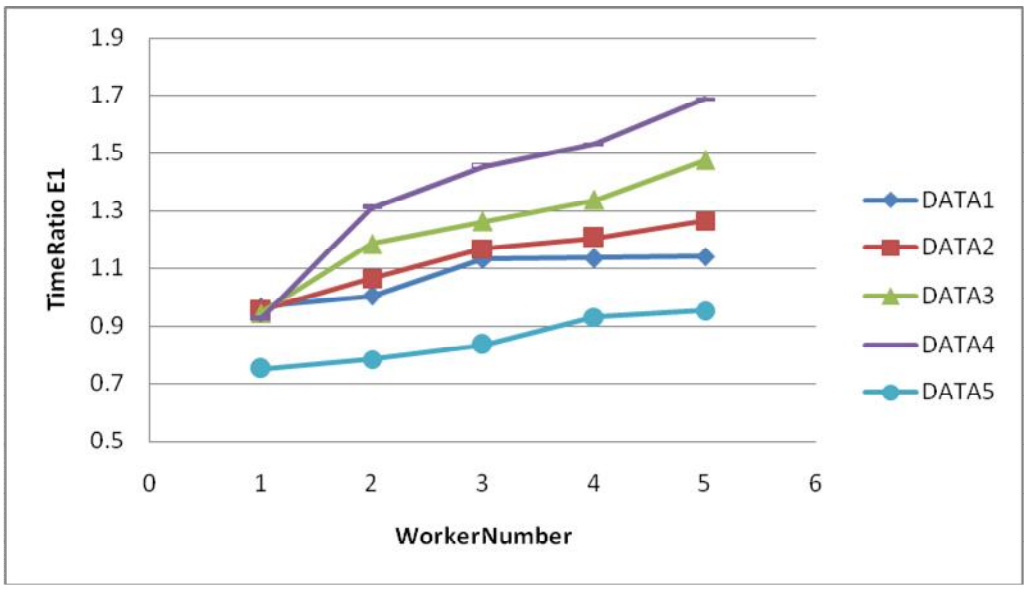


图 5-1 K-means 在 Spark、Hadoop 下运行性能对比图

由实验结果所得比率图 6-1 可以看出如下结论：

（1）当 WorkerNumber = 1 时，不论数据量 *DataSize* 是否大于计算节点的物理内存 *Memory*，$\sigma = 1/E1 = \frac{t_2}{t_1}$ 的比例值一直小于 1。由此可知：由于 Spark 集群是搭建于 Yarn 资源管理器（Hadoop2.x）之上，当其在单机上运行时发挥不了 Spark 内存计算的优势，反而因为其内部通信量占总运算时间比重比 Hadoop 要高而降低了数据处理效率[45]。

（2）当 workerNumber>1 且 *DataSize* < *Memory* 时，随着集群计算节点的增加，同一数据量在 Spark 集群下的运行时间都比 Hadoop 集群下运行时间要短，即 $\sigma = 1/E1 = \frac{t_2}{t_1}$ 的比例一直大于 1，而且大约呈线性上升趋势。由此可知：Hadoop 在每一次迭代计算过程中都重新访问 HDFS 读入数据对象集到本地内存，而 Spark 提供数据缓存功能可以大幅降低了迭代计算中数据 IO 的时间开销占总运行时间的比率，所以 Spark 集群环境下的运行时间现对于 Hadoop 就可以得到显著减少。而且由图 5-1 可知随着数据量的提高，Spark 相比于 Hadoop 在并行计算方面的优势愈发明显。

（3）当 *DataSize* > *Memory* 时，随着数据量的增加，不论是单机还是集群环境下，Spark 集群的运行时间都比 Hadoop 要长，即 $\sigma = 1/E1 = t_2/t_1$ 的比例一直小于 1；随着计算节点的增加，比例值会逐渐增加但任然不会大于 1。由此可知：当数据量大于计

算节点的可用 memory 时，Spark 集群不能一次性地将数据对象全部写入进内存中，从而造成部分加载而引起的磁盘和缓存间的频繁 IO，故由图 6-1 可知当内存足够大的情况下，Spark 集群的通信量比 Hadoop 小而且数据 IO 的消耗远比 Hadoop 要低；反之会因为缓存与磁盘间的频繁 IO 而损耗过多地降低计算效率，从而造成 Spark 的处理效率低于 Hadoop。

5.3.2 K-means 算法及其改进算法的准确性对比实验

上一小节实验验证了 Spark 集群相比于 Hadoop 集群在处理同一数据集时能更快地完成数据聚类任务，本小节通过 K-means 算法在 Spark 集群、Hadoop 集群环境下的聚类结果对比进一步验证 Spark 在聚类准确性能上是否同样也具有一定的优势。本实验使用 UCI Machine Learning Repository 中的四个数据集：Seeds、UKM（User Knowledge Modeling）、Perfume、TSU（Turkiye Students Evalution），其数据集特征如表 5-8 所示：

表 5-8 测试数据集特征表

DataSet	InstanceNumber	AttributeNumber	ClusterNumber
Seeds	210	7	3
UKM	403	5	4
Perfume	560	2	20
TSU	5820	33	5

Seeds 数据集由三种不同的小麦品种籽粒组成，分别是卡玛，罗萨和加拿大，每组 70 个元素；UKM 数据集用于对用户的知识分类建模，共 403 个样本对象，分为 very_low、low、middle、high 四个类别；Perfume 数据集统计了 20 种不同的香水品牌，属性比较单一只有两个；TSU 数据集用于统计土耳其学生评价等级，共 5820 条评价，评分有 1-5 五种等级。

在本实验中，K-means 算法的准确率是将算法的聚类结果与 UCI Machine Learning Repository 中的标准聚类相比较得出，准确率 $E2=\frac{\sum_{i=1}^{k} m_i}{n}$，其中 n 表示类簇的个数，m_i 表示最终结果中被正确划分到第 i 个类簇的数据个数。I 表示聚类结束时的迭代次数，实验结果如下表 5-9 所示。由表 5-9 可以清晰地看出 Spark 在聚类准确性能上相较于 Hadoop 同样具有一定的优势。

表 5-9 K-means 聚类算法在 Spark、Hadoop 上的准确率对比表

Worker / DATA		1		2		3		4		5	
		E2	I	E2	I	E2	I	E2	I	E2	I
Seeds	**Spark**	0.937	14	0.940	13	0.941	13	0.943	13	0.943	13
	Hadoop	0.912	16	0.929	14	0.931	13	0.934	13	0.934	13
UKM	**Spark**	0.862	17	0.875	16	0.879	14	0.883	14	0.884	14
	Hadoop	0.873	21	0.878	19	0.879	16	0.880	16	0.880	15
Perfume	**Spark**	0.730	37	0.737	34	0.739	33	0.742	32	0.746	30
	Hadoop	0.684	35	0.687	35	0.693	33	0.696	33	0.708	32
TSU	**Spark**	0.538	136	0.627	124	0.683	98	0.714	85	0.792	68
	Hadoop	0.452	183	0.487	166	0.517	139	0.573	117	0.672	96

接下来的实验通过传统 K-means 算法、CKM 算法在 Spark 集群环境下的聚类结果对比进一步验证 CKM 算法在聚类准确性上是否也有一定程度的提升。为了更直观地体现改进 CKM 在迭代准确率和迭代收敛率上的性能优势，本实验结果分析中以 $E3 = E2 * I$ 作为算法效率的综合度量标准，其中 $E2$ 表示聚类结果与标准聚类结果对比准确率，I 为迭代停止时迭代次数。

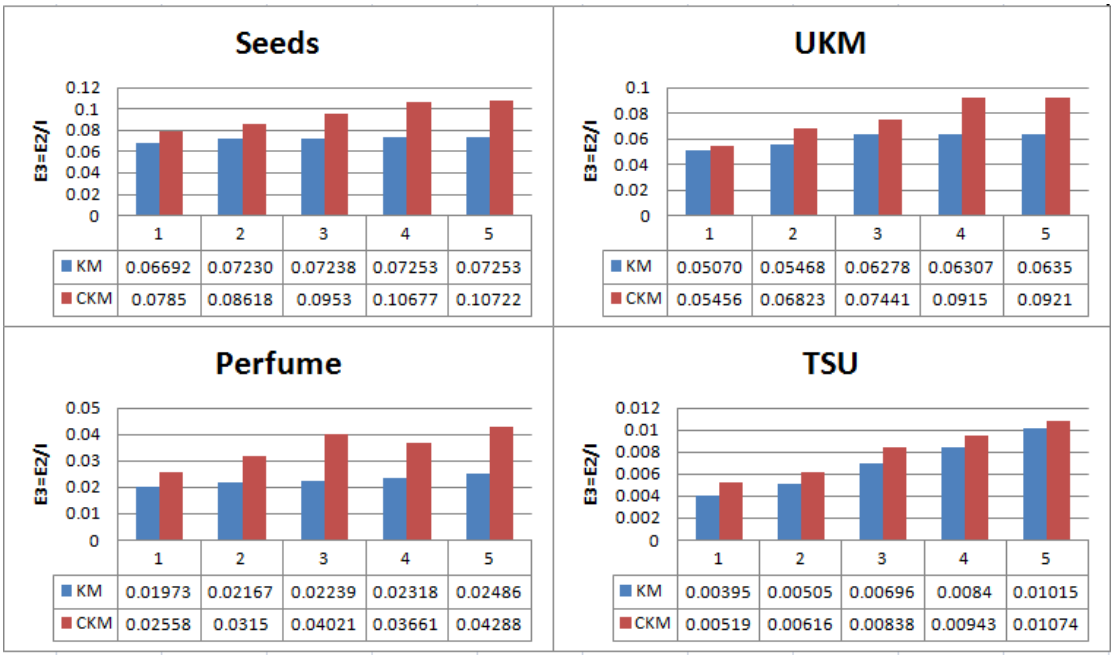


图 5-2 KM、CKM 算法在 Spark 集群上的准确率对比图

从图 5-2 可以很明显地看出改进的 CKM 并行算法的聚类结果比 K-means 并行算法更加准确可信，而且改进的 CKM 并行算法的收敛速度比 K-means 并行算法更快。综合而言，基于 Spark 集群实现的 CKM 并行聚类算法比传统 K-means 算法效率更高。

5.3.3 K-means、CKM 改进算法的加速比分析实验

前两小节首先分析了 Spark 相比 Hadoop 在并行计算效率（快速性和准确性）方面的优势，然后分析了基于 Spark 集群的 CKM 改进算法相比 K-means 算法在聚类准确性、收敛速率方面的优势。本小节在前两小节实验分析的基础上进一步分析传统 K-means、CKM 改进算法在加速比方面的性能。

加速比是测试并行算法性能的重要指标之一，它描述了通过并行化使算法运行时间缩短而获得的整体性能提升。加速比的计算公式为 $E_r = T_s / T_r$，其中 T_s 表示算法在单机环境下运行所需要的时间，T_r 表示算法在由 r 个计算节点所组成的集群环境下运行所需要的时间。加速比 E_r 愈大，则算法并行计算所需要的相对时间愈少，进一步说明并行化的效率愈高。本实验使用数据集 DATASET1-DATASET5，测试这些数据集在不同节点下传统 K-means 算法以及 CKM 改进算法在 Spark 集群环境下的并行聚类的加速比，由实验结果可得如下直观的曲线图 5-3、图 5-4。

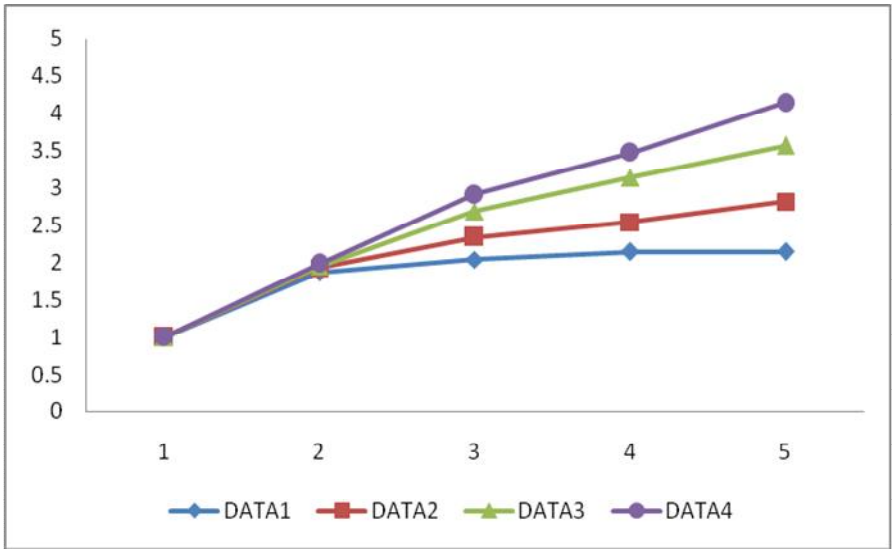


图 5-3：K-means 并行算法在 Spark 下的加速比曲线图

图 5-3 描述了 Spark 集群下传统 K-means 并行算法执行数据对象集 DATASET1-4 的加速比趋势图。当数据量比较小时，数据集在集群节点上可以一次性被读入内存形成 RDD 然后并行计算，随着计算节点的增加反而会在一定程度上拖慢整体计算性

能，因为节点增多后，集群中各节点的通信和配置初始化都会增加开销；当计算节点不变，在数据量很小时，算法的加速比并不明显。

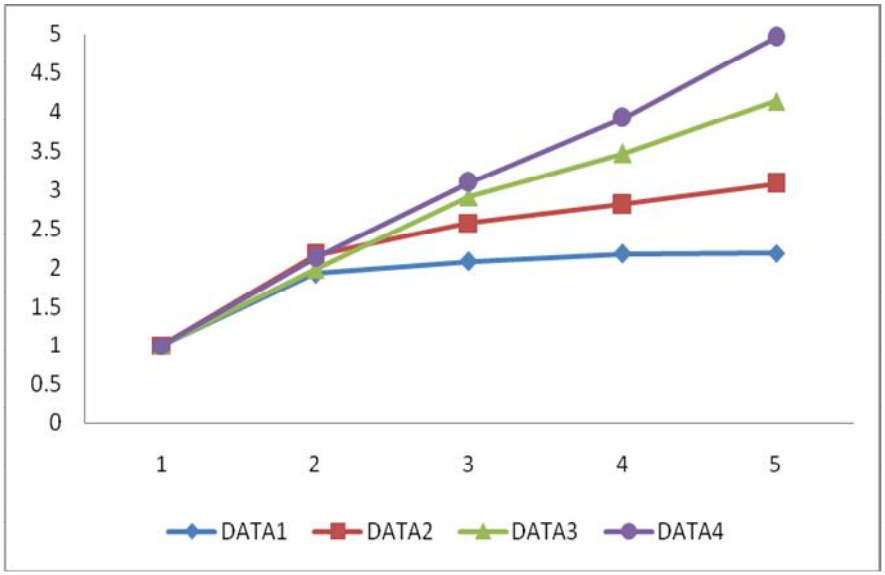
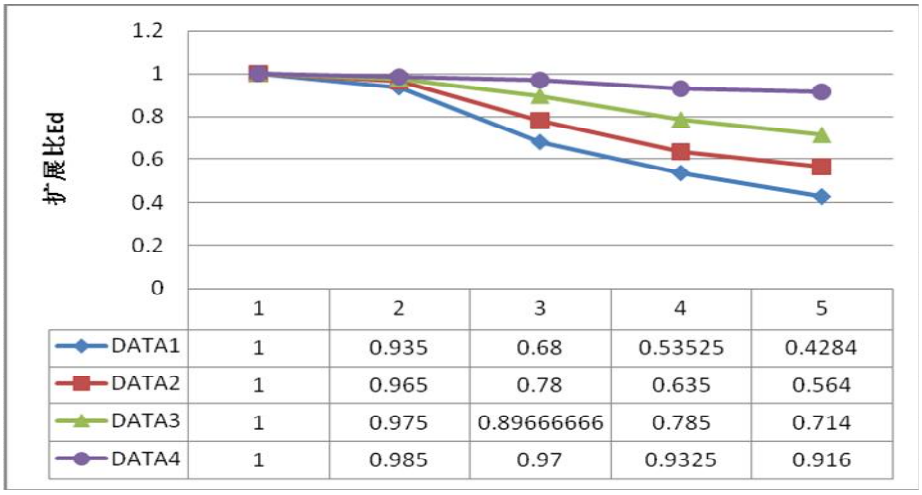


图 5-4：CKM 改进算法在 Spark 下的加速比结果

图 5-4 描述了 Spark 集群下 CKM 改进算法在执行数据集 DATASET1-4 的加速比趋势图。其加速比曲线趋势和 Spark 集群下 K-means 算法在执行数据集 DATASET1-4 的加速比趋势图大致相似，不同之处就是它比前者增幅要快，而且相同计算节点下处理相同数据集时的加速比普遍偏高，这也从另一方面说明了改进下 CKM 在算法性能上的确比传统 K-means 算法略高一筹。

5.3.4 K-means、Canopy_K-mean 改进算法的扩展比分析实验

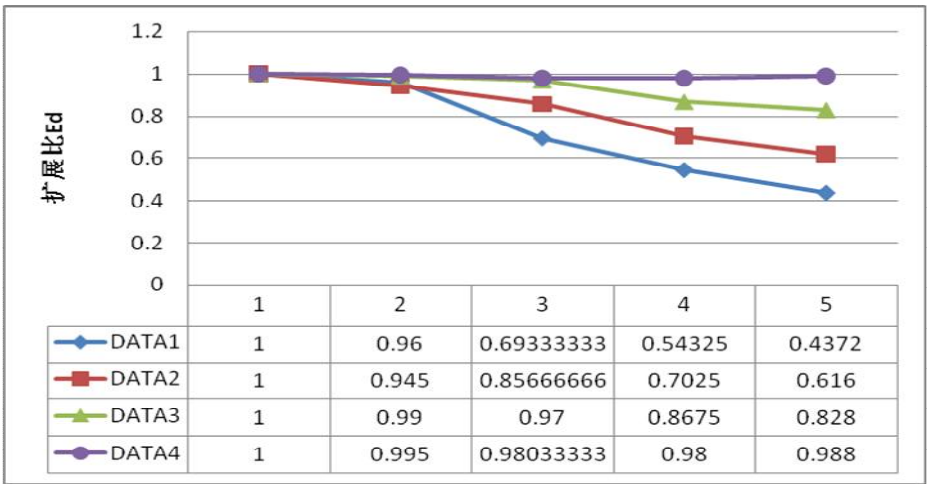
本小节在前三小节实验分析的基础上进一步分析传统 K-means、CKM 改进算法在扩展比方面的性能。扩展比也是检测并行算法性能的重要指标之一，它描述了通过不断提高集群中计算节点数目时集群的利用率情况，因为加速比不可能无限增大，此时就需要用扩展比来检验算法的并行效率。扩展比的计算公式为 $E_d = E_r / r$，其中 E_r 表示算法在由 r 个节点组成的集群环境下运行时的加速比。扩展比 E_d 愈大，说明该并行算法对集群的利用率愈高，进一步说明其并行化的效率愈高。本实验使用数据集 DATASET1-DATASET5，测试这些数据集在不同节点下传统 K-means 算法以及 CKM 改进算法在 Spark 集群环境下的并行聚类的扩展比。由实验数据可得扩展比数据统计如图 5-4、5-5 所示。



	1	2	3	4	5
DATA1	1	0.935	0.68	0.53525	0.4284
DATA2	1	0.965	0.78	0.635	0.564
DATA3	1	0.975	0.89666666	0.785	0.714
DATA4	1	0.985	0.97	0.9325	0.916

图 5-5：K-means 并行算法在 Spark 下的扩展比曲线图

由图 5-5 描述了可以直观地看出比较明显的规律：数据集为 DATASET1、DATASET2 时扩展比 E_d 下降比较快，当数据集为 DATASET3、DATASET4 时扩展比 E_d 下降速率比较平缓，而且随着计算节点数量的增加 E_d 逐渐收敛。以上现象表明 Spark 集群下的 K-means 并行算法虽然加速比 E_r 有显著上升，但是当数据集愈大、计算节点愈多，K-means 算法的扩展比逐渐收敛。



	1	2	3	4	5
DATA1	1	0.96	0.69333333	0.54325	0.4372
DATA2	1	0.945	0.85666666	0.7025	0.616
DATA3	1	0.99	0.97	0.8675	0.828
DATA4	1	0.995	0.98033333	0.98	0.988

图 5-6：CKM 并行算法在 Spark 下的扩展比曲线图

图 5-6 描述了 Spark 集群环境下 CKM 改进算法在执行数据集 DATASET1-4 的扩展比趋势图。其扩展比曲线趋势和 Spark 集群环境下 K-means 算法在执行数据集 DATASET1-4 的扩展比趋势图大致相似，不同之处就是它比前者要更快趋于收敛于某一稳定值，这也从另一方面说明了改进 CKM 算法比传统 K-means 算法基于 Spark 的并行化实现的鲁棒性更高，具有更好的可扩展性能。

5.4 实验总结与分析

本研究课题通过以上几组实验分别从不同侧面分析了传统 K-means 算法与改进 CKM 算法、Spark 分布式集群与 Hadoop 集群在性能上的对照实验。

实验一通过 K-means 算法在 Hadoop、Spark 集群上的运行性能对比，以相同机器配置环境下相同计算节点数目时两者运行时间之比 $E1 = t_1 / t_2$ 作为对比条件,验证了 Spark 分布式集群在迭代计算上相对于 Hadoop 所具有的速率优势。

实验二通过 K-means 算法在 Spark 集群、Hadoop 集群环境下的聚类结果对比进一步验证 Spark 在迭代计算上相对于 Hadoop 同样也具有准确性优势；接下来通过传统 K-means 算法、改进 CKM 在 Spark 集群环境下的聚类结果对比进一步验证了改进的 CKM 并行算法的聚类结果比传统 K-means 并行算法更加准确可信，而且改进的 CKM 并行算法的收敛速度比传统 K-means 并行算法更快。

实验三进一步分析传统 K-means、CKM 算法在加速比方面性能。实验结果表明后者比前者加速比增幅要快，而且相同计算节点下处理相同数据集时的加速比普遍偏高，这也从另一方面说明了改进下 CKM 在算法性能上的确比传统 K-means 算法略高一筹。

实验四在前三小节实验分析的基础上进一步分析传统 K-means、CKM 改进算法在扩展比方面的性能。实验结果表明后者比前者要更快趋于收敛于某一稳定值，这也从某一方面说明了改进 CKM 算法比传统 K-means 算法基于 Spark 的并行化实现的鲁棒性更高，具有更好的可扩展性能。

综合而言，基于 Spark 集群实现的 CKM 并行算法比传统 K-means 算法效率（准确性、迭代收敛速率、并行算法健硕性能）更高。

6 论文总结与前景展望

6.1 研究项目总结

本研究课题工作主要如下：

（1）首先概要地介绍了本课题的研究背景与目的以及聚类分析国内外研究现状，由传统 K-means 算法应用广泛的同时所遇到的计算瓶颈抛砖引玉，阐述本课题研究的可行性分析。

（2）接着详细介绍了海量数据处理中的数据挖掘技术的基础知识，结合海量数据挖掘的瓶颈提出了聚类算法的并行化的可行性分析。然后详述了并行计算及分布式集群环境下并行编程的思想，介绍了基于 MapReduce 编程模型的 Hadoop、基于 RDD 内存计算及 DAG（有向无环图）的 Spark，并对比了两者的性能优势。

（3）然后详述了传统 K-means 算法的流程、应用及其优缺点，并针对其在单机环境下串行计算效率低的瓶颈提出 K-means 算法在 Hadoop 和 Spark 分布式集群环境下的并行化实现。

（4）再传统 K-means 算法在 K 值选择的主观性和初始化中心聚类中心选取的随机性上的缺陷，提出了改进的 CKM 算法来更精确地选取 K 值和初始中心点，最后给出改进算法在 Hadoop 和 Spark 分布式集群环境下的并行化实现。

（5）最后详述了两个分布式计算平台 Hadoop、Spark 在 Yarn 下软硬件的环境配置，并实现了 K-means 算法和优化后的 CKM 算法在 Hadoop、Spark 平台上的并行化，通过实验结果进行聚类准确性、加速比、可扩展性、与其他平台比较以及资源调度稳定性的测试。

6.2 研究前景展望

本研究课题基于 Spark 分布式计算平台实现了传统 K-means 算法和改进 CKM 算法的并行化实现，通过大量对比测试进一步证明了 Spark 在分析需要频繁迭代计算的海量数据集时和 Hadoop 相比有较好的性能。但在研究过程中也发现了可以深入研究的地方。

（1）进一步扩大实验规模，通过设计更多不同情况下不同输入数据的对比实验得到更加立体有效的结论。

（2）进一步深入研究 Spark 内核代码，使算法和 Spark 基于 RDD 的内存计算特

性进一步的融合，从而发挥算法并行化更大的效率。

（3）进一步深入研究 Spark 调优方式（如数字序列化、文件压缩、堆占比、多级缓存、小分片、Action 顺序等方式），从而通过精通配置各项环境参数来加快集群运行的效率。

（4）通过进一步实现其他具有迭代计算特性的机器学习算法基于 Spark 的并行实现，来更加熟练的操作 Spark 平台上的大数据挖掘。

致 谢

两年研究生学涯即将结束之际，首先我要感谢导师韩建军老师，从研究生复试面试时的短促交流直接录取我，到这两年多期间韩老师悉心热情地学术培养和生活工作方面无微不至的关心，都让我感受到韩老师如父亲一般的温情和耐心。在学术科研方面，当我在研究方向选取屡屡碰壁之时，韩老师耐心开明地帮助我、指引我一步步慢慢摸索式地找准如今的研究方向并十分支持我的选择；当遇到技术难题时，韩老师总是耐心细致地给我讲透难题里面关键性的技术和基础原理，从未有过丝毫的不耐烦，有的只是满满的对学术认真细致地钻研态度和热情；当研究方向定下之后，考虑到研究方向对硬件性能的要求，韩老师没有犹豫地大批购买性能过硬的主机以供我快速地搭建起科研型的实验室集群环境，这为此论文中大量对比试验的反复运行提供了最基本的条件；在此论文从开始布局到整体构思、后期的具体实施，韩老师都给予我关键性的参考性建议，使我可以利用紧俏的在校时间去验证我的构想；在此篇论文初稿完成之际，韩老师秉承着认真负责的科研态度指导我反复地修稿，这才有了此篇论文呈现在大家面前。在生活方面，韩老师时刻关心我们的温饱，每月学校规定的科研补贴从不拖发一刻；实验室空调坏了以后韩老师立马联系维修人员修理直至完善才放心；天冷了也会像父亲一般提醒我们注意加减衣物以防感冒。感谢我的同门赵清伟同学，谢谢这两年你在学习和生活上对我无私的帮助，尤其是在找工作期间两个人一起互相鼓劲打气，也谢谢你在这两年里的各种谅解和体贴。感谢陶鑫师弟在这一年里给我带来的许多欢乐时光。感谢一群志同道合的同学陪我度过那么多科研之外的业余美好时光，谢谢室友付永刚同学如兄长般的提携，谢谢陈智、汪光练、赵庆杰、漆学志如兄弟般的关怀和帮助。感谢女友这段日子里给我那么一段美好的感情。感谢家人一如既往地支持和尊重我的选择。感谢华中科技大学提供了我继续深造的机会和良好的学习环境。感谢本论文中所引用的文献的作者与出版机构。在此也十分感谢从百忙之中抽空审阅本论文的老师们以及毕业答辩小组的老师和答辩秘书师兄！

参考文献

[1] Wu X, Kumar V, Quinlan J R, et al. Top 10 algorithms in data mining. Knowledge and Information Systems, 2008, 14(1): 1~37
[2] Mahdavi M, Fesanghary M, Damangir E. An Improved Harmony Search Algorithm for Solving Optimization Problems. Applied Mathematics and Computation, 2007, 188(2): 1567~1579
[3] Chieh-Yen Lin, Cheng-Hao Tsai, Ching-Pei Lee, et al. Large-scale Logistic Regression and Linear Support Vector Machines Using Spark. in: IEEE International Conference on Big Data, 2014. 519~528
[4] Agrawal R, Srikant R. Fast algorithms for mining association rules. in:Proceeding of the 20th VLDB conference. 2009. 487~499
[5] Hindman B, Konwinski A, Zaharia M, et al. Spark: cluster computing with working sets. in: Proceeding of the 2th usenix conference on Hot topics in cloud computing. 2010. 10~15
[6] 顾洪博. 基于 K-means 算法的 k 值优化的研究与应用. 海南大学学报, 2009, 16(4): 386-389
[7] 张逸清. 基于遗传算法的 K-Means 聚类改进研究: [硕士学位论文].重庆:重庆大学,2006.
[8] 程佳. 支持向量机与 K-均值聚类融合算法研究: [硕士学位论文]. 大连:辽宁师范大学, 2008.
[9] Eui-Hong Han, George Karypis, Vipin Kumar. Scalable Parallel Data Mining for Association Rules. In: IEEE Transactions on Knowledge and Data Engineering, 2000. 337~352
[10] Yiping Wu, Tiejian Li. Parallelization of a hydrological model using the message passing interface. Environmental Modelling & Software, 2013, 15(43): 124~132
[11] Sallis Jf, McKenzie Tl, Kolody B, et al. Effects of health-related physical education on academic achievement. Res Q Exerc Sport, 1999, 70(2): 127~134.
[12] 盛文峰. 面向数据挖掘的遗传算法的研究与应用: [硕士学位论文]. 上海: 上海交通大学, 2007.
[13] 彭崇. 聚类技术在车险业务分析中的应用研究: [硕士学位论文]. 成都: 电子科技大学, 2007.
[14] 胡俊. 集群环境下聚类算法的并行化研究与实现: [硕士学位论文]. 上海: 华东师范大学, 2012.
[15] 熊忠阳. 数据挖掘聚类算法的分析和应用研究: [硕士学位论文]. 重庆: 重庆大学, 2002.
[16] 刘超. 无预设类别数的大数据量聚类算法研究: [硕士学位论文]. 南京: 南京师范大学, 2012.
[17] 钱线, 黄萱菁, 吴立德. 初始化 K-means 的谱方法. 自动化学报, 2007, 33(4): 342~346
[18] 孙秀娟, 刘希玉. 基于初始中心优化的遗传 K-means 聚类新算法. 计算机工程与应用, 2008, 44(23): 167~168
[19] 汪中, 刘贵全, 陈恩红. 一种优化初始中心点的 K-means 算法. 模式识别与人工

智能, 2009, 22(2): 299~304
[20] 冯征. 一种基于粗糙集 K-means 聚类算法.计算机工程与应, 2006, 42(9): 141~142
[21] Zhu Jian, Wang Han-shi. An improved K-means clustering algorithm. in: The 2th IEEE International Conference on Information Management and Engineering, ICIME, 2010. 109~192
[22] 梁红. 基于 MPI 的并行小波聚类算法. 计算机科学, 2005, 32(7): 156~158
[23] Kantabutra S, Couch A L. Parallel K-means clustering algorithm on Nows. NECTEC Technical Journal, 1999, 1(1): 243~247
[24] 贾瑞玉, 管玉勇, 李亚龙. 基于 MapReduce 模型的并行遗传 K-means 聚类算法. 计算机工程与设计, 2014, 35(2): 657~660
[25] Zhao Weizhong, MA Huifang, FU Yanxiang, et al. Research on parallel k-means algorithm design based on hadoop platform. Computer Science, 2011, 38(10): 166~176
[26] Qing Liao, Fan Yang, Jingming Zhao. An improved parallel k-means clustering algorithm with mapreduce. in:Proceedings of ICCT, 2013. 764~768
[27] Son Lam Phung, Abdesselam Bouzerdoum, Douglas Chai. Skin segmentation using color pixel classification: analysis and comparison. in: IEEE Transactions on Pattern Analysis and Machine Intelligence, 2005, 27(1): 148~154
[28] 蔡伟杰, 张晓辉,朱建秋等. 关联规则挖掘综述. 计算机工程, 2001, 27(5): 31~33
[29] Halkidi M, Batistakis Y, Vazirgiannis M. Clustering algorithm and validity measures. in: The 13th International Conference on Scientific and Statistical Database Management. Fairfax:Virginia, 2001. 145~153
[30] Thilina Gunarathne, Tak-Lon Wu, Judy Qiu, et al. MapReduce in the clouds for Science. in: The 2nd IEEE International Conference on Cloud Computing Technology and Science, 2010. 565~572
[31] Holden Karau. Fast Data Processing with Spark. UK: Packet Publishing Ltd, 2013. 43~51
[32] Fahim A.M, Salem A.M, Torkey F.A. Unique distance measure approach for k-means clustering algorithm. in: IEEE Transactions on Pattern Analysis and Machine Intelligence, 2007. 1~4
[33] 李飞, 薛彬, 黄亚楼. 初始中心优化的 K-means 聚类算法. 计算机科学, 2002, 29(27): 94~96
[34] 周婷, 张君瑛, 罗成. 基于Hadoop的K-means聚类算法的实现. 计算机技术与发展, 2013, 16(7): 18~21
[35] Cay S. Horstmann. Scala for the Impatient. 高宇翔译. 北京:电子工业出版社, 2012. 32~37
[36] 贺瑶, 王文庆, 薛飞. 基于云计算的海量数据挖掘研究. 计算机技术与发展, 2013, 23(2): 69~72
[37] 于春深. 基于 MapReduce 并行聚类算法的研究: [硕士学位论文]. 西安: 西安电子科技大学, 2012
[38] 张雪凤, 张桂珍, 刘鹏. 基于聚类准则函数的改进 K-means 算法. 计算机工程与应用, 2011, 19(11): 123~127
[39] 冯波, 郝文宁, 陈刚, et al. K-means 算法初始聚类中心选择的优化. 计算机工程

与应用, 2013, 14: 182~185
[40] 毛嘉莉. 聚类 K-means 算法及并行化研究: [硕士学位论文]. 重庆: 重庆大学, 2003.
[41] 付腾达. 基于 GPU 并行聚类的加密分组密码算法的研究及实现: [硕士学位论文]. 南京: 南京理工大学, 2013.
[42] Jimmy Lin, Chris Dyer. Data-Intensive text processing with mapreduce. in: The 4th IEEE International Conference on Information Management and Engineering, ICIME, 2012. 54~62
[43] Dean J, Ghemawat S. MapReduce:simplified data processing on large clusters. in: IEEE Transactions on Pattern Analysis and Machine Intelligence, 2004: 137~150
[44] Xu X W, Jager J, Krigel H P. A fast parallel clustering algorithm for large spatial databases. Data Mining and Knowledge Discovery, 1999, 34(13): 263~290
[45] Wu X, Kumar V, Quinlan J R, et al. Top 10 algorithm in data mining. Knowledge and Information Systems, 2008, 14(1): 1~37