****



**本 科 毕 业 论 文**

院 系 地球科学与工程学院

专 业 基地班

题 目 MPI/Hadoop在K-means算法上的性能比较分析

年 级 2011级 学 号 111150034

学生姓名 蒋鑫

指导老师 周会群 职 称 教授

论文提交日期

**南京大学本科生毕业论文（设计、作品）中文摘要**

题目： MPI/Hadoop在K-means算法上的性能比较分析

院系 地球科学与工程学院 专业 资源勘探与信息技术系

2011 级本科生姓名： 蒋鑫

指导教师（姓名、职称）： 周会群 教授

摘要：

过去几十年，由于计算机软硬件的进步，已有相当数量的科研领域开始利用计算机进行辅助处理，计算和模拟。然而随着现代科学研究的不断深入，处理数据海量化、计算过程复杂化、结果要求精细化，导致了很多情形下单机计算模型已经不能满足需要，并行计算成为解决这一问题的关键。

基于消息传递的MPI编程模型和基于HDFS的Hadoop编程模型是当前比较热门的两种并行编程框架。本文首先介绍了并行计算领域的一些基本概念，讨论了常见并行计算机结构模型和并行编程模型；然后分章节概要的介绍了MPI和Hadoop两种并行模型的特点和基本使用方式；接着对地学研究中比较常用的K-means聚类算法进行的简述；最后通过设计实验，将MPI和Hadoop同时作用于K-means算法，得到一系列不同维度的实验结果数据，对不同数据量，不同计算精度，不同集群规模所得到的两者性能数据进行了较详细的分析讨论，并指出了性能差异的可能原因，进而对两者的最适应用场景进行了描述，希望可以对今后的科研计算具有建议性作用。

关键词：并行计算， MPI， Hadoop， Mapreduce， K-means**南京大学本科生毕业论文（设计、作品）英文摘要**

THESIS：Comparative Analysis of MPI / Hadoop performance on the K-means algorithm

DEPARTMENT：Earth Science and Engineering

SPECIALIZATION: Exploration and Information Technology

UNDERGRADUATE: Jiangxin, 2011

MENTOR: Zhouhuiqun, Professior

ABSTRACT：

Over the past decades, due to the great progress in hardware and software, there has been a considerable amount of research fields began to use computer-assisted processing, calculation and simulation. However, with the deepening of modern scientific research, data processed gets bigger, calculation process gets more complicated, and the results requires more accurate. In many cases the stand-alone compute model cannot meet the requirement. So parallel computing has become the key to solving this problem.

Nowadays, MPI based on message passing and Hadoop based on HDFS are two popular parallel programming frameworks. This paper introduces the basic concepts of parallel computing, discusses the common model of parallel computer architecture and parallel programming model; in the next chapters I describe the characteristics and basic usage of the two MPI and Hadoop parallel model; then this thesis gives an introduction of K-means clustering algorithm, a commonly used algorithm in geoscience and other fields; and finally we designs experiments to make the MPI and Hadoop process K-means algorithm. A series of experimental results obtained in different amount data, different calculation accuracy, different cluster size. I give a detailed analysis and discussion on these experiment results and point out the possible reasons for differences in performance. I hope this thesis can give some scientific computing research an effective guide.

KEY WORDS: Parallel Computing, MPI, Hadoop, Mapreduce, K-means

[目录](#_Toc420777967)

[1. 绪论 6](#_Toc420777968)

[1.1 研究背景 6](#_Toc420777969)

[1.2 国内外关于该课题的研究现状 7](#_Toc420777970)

[1.3 本文的结构 9](#_Toc420777971)

[2. 并行计算基础理论 9](#_Toc420777972)

[2.1 并行计算的相关概念 9](#_Toc420777975)

[2.2 并行计算机结构模型 10](#_Toc420777976)

[2.2.1 并行向量处理机（PVP） 11](#_Toc420777981)

[2.2.2 对称多处理机（SMP） 11](#_Toc420777982)

[2.2.3 大规模并行处理机（MPP） 11](#_Toc420777983)

[2.2.4 分布共享存储多处理机(DSM) 12](#_Toc420777984)

[2.2.5 工作站机群（COW） 12](#_Toc420777985)

[2.3 并行编程模型 12](#_Toc420777986)

[2.3.1 数据并行编程模型 13](#_Toc420777992)

[2.3.2 消息传递并行编程模型 13](#_Toc420777993)

[2.3.3 消息传递与数据并行的对比 13](#_Toc420777994)

[3. MPI与Hadoop简介 14](#_Toc420777995)

[3.1 MPI简介 14](#_Toc420777999)

[3.1.1 MPI的通讯模式 15](#_Toc420778007)

[3.2 Hadoop简介 15](#_Toc420778008)

[4. K-means算法及实验设计 16](#_Toc420778009)

[4.1 K-means聚类算法概述 16](#_Toc420778014)

[4.2 K-means 算法基本步骤 17](#_Toc420778015)

[4.3 K-means算法复杂度分析 17](#_Toc420778016)

[4.4 基于Hadoop的并行K-means算法设计 17](#_Toc420778017)

[4.4.1 Map函数实现 18](#_Toc420778026)

[4.4.2 Reduce函数的实现 18](#_Toc420778027)

[4.5 基于MPI的并行K-means算法实现 19](#_Toc420778028)

[4.5.1 主进程的主要任务 19](#_Toc420778038)

[4.5.2 从进程的主要任务 19](#_Toc420778039)

[4.6 实验环境说明 19](#_Toc420778040)

[5. 实验结果 20](#_Toc420778041)

[5.1 使用Hadoop运行K-means算法性能分析 20](#_Toc420778047)

[5.2 使用MPI运行K-means算法性能分析 22](#_Toc420778048)

[5.3 Hadoop与MPI在K-means算法中的性能比较 28](#_Toc420778049)

[6. 结论 31](#_Toc420778050)

[参考文献： 31](#_Toc420778051)

[致谢 32](#_Toc420778052)

# 绪论

## 研究背景

随着科学技术的迅猛发展，越来越多的科研领域采用计算机作为辅助存储和运算工具，然而随着各项研究的深入，复杂化、精细化、智能化已经成为当今科学计算的主流。传统的单机存储、计算模型已经难以满足科研人员处理大数据量、大计算量的工作需要，特别是在天体物理学，分子生物学，地震学，材料化学，地理学等领域，该问题尤其突出。

以地质学中的矿产预测为例，由于矿产资源预测理论的持续进步，相关模型的不断完善，以及与其它地学信息的不断融合，以矿产资源预测理论为核心，以空间数据库为根基的矿产资源预测系统不断涌现[1]。由于该系统涉及到地学大数据融合分析、跨学科数据集成、矿产资源定位定量预测、地质空间建模等一系列复杂过程，所以涉及到的计算量十分庞大，传统的单机计算系统已经难以维持，必须使用现代的高性能计算或者分布式云计算系统进行支撑，以实现对地学大数据的快速处理，准确分析。因此基于现代云计算模型的矿产资源预测系统已经成为前沿研究方向。另外据曲寿利[2]等人有关地震资料勘探的研究中，也表达了相似的观点，由于勘探程度的持续提高，勘探对象也日趋复杂，矿产勘探的当前形势可以用“深，隐，低，稠，小”五个字进行概括，也就是说勘探对象埋深不断加大，勘探目标隐蔽性逐渐增加，低渗透性、特低渗透性油气藏所占的比例上升，稠油、超稠油油藏的比例持续增大，勘探对象的单元储量规模却越来越小。所以为了适应目前这种趋势，地震资料勘探的精度要求大幅提高，而精度的提高必然带来计算量的陡然上升，这之中高性能计算，分布式云计算在其中起到了关键性的作用。

传统高性能计算采用基于并行处理器为基础的超级计算机对复杂的现实世界进行计算模拟。随着相关科技的发展，现代高性能计算已经有了新的定义，开始融入集群计算、分布式计算、网格计算、普适计算等一系列新的技术。正是因为有以这种高性能计算为基础的地学计算，地理地质环境中以时空演变为特征的全球性或者大区域性现象（如地理信息系统、全球气候变化、数值天气预报、图像解译和遥感定量解译等）的模拟才能真正实现；原有的一些受单机条件限制而难以进行的模型才能得以完善、改进和运行。[3]

目前高性能计算模型层出不穷，各种新技术令人眼花缭乱，但其中最著名的两个流派就是MPI并行计算模型和以Hadoop分布式计算模型。

MPI(Message Passing Interface)是一种使用非常广泛的基于消息传递的并行编程模型。除了广泛应用于高校科研场所之外，MPI还用于许多商用计算机系统之中。互联网的存在意味着每两个计算机处理器之间都直接或间接的相连，也就是说每个处理器都可以和其它任何一个处理器进行通讯。而MPI并行编程模式就是依赖这种特性，它可以使集群中的任何两台计算节点在计算过程中进行消息传递，最大限度的利用每一台机器的处理器计算能力。

Hadoop是一种新兴的分布式基础架构。用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群的威力进行高速运算和存储。Hadoop主要由两大核心组成，HDFS和Mapreduce。HDFS处于底层，Mapreduce则利用HDFS高容错，高可靠，易扩展等特点，在其上构造了一个分布式并行计算引擎。MapReduce将分布式并行编程抽象为两个简单的基本操作（map 操作和reduce 操作），开发人员仅需要实现对应的接口即可，很好的屏蔽了其底层的容错、数据流、程序的并行执行等细节。这种设计无疑大大降低了开发分布式并行程序的难度。

## 国内外关于该课题的研究现状

目前MPI和Hadoop这两种并行计算模型都发展的比较成熟，其中MPI已经在地质学相关的高性能计算中得到了广泛的应用。中国地质大学的学者利用MPI开展了大地电磁三维正反演并行算法的研究[4][5]，上海超算中心的部分学者则利用MPI进行了黄河下游二维水沙数学模型的构建[6]，并且已经取得了初步的研究成果。由于Hadoop计算模型出现较晚，当前在地学领域的应用还不常见，但已有学者开始尝试，比如在Chaowei Yang等人的文章中提出希望利用Hadoop的Mapreduce计算框架进一步探讨数字地球可行性[7]。

虽然MPI和Mapreduce并行编程模型都属于高性能计算的范畴，但是由于两种模型差异较大，各有优缺点。到底哪种并行编程模型更加适用于地学的普遍应用场景，目前还没有一个定论。

MPI是一种基于消息传递机制的并行编程规范，它在程序设计上非常简单而且符合普通科研人员的使用习惯。但是MPI有一个比较大的缺点，即底层缺乏一个优秀的分布式的文件系统对其进行很好的容错支撑。数据处理节点和数据存储节点往往是相异的，这种数据的存储和处理分离的情况要求在每次计算开始时从集群的数据存储节点读取数据分配给各个计算节点，然后再进行接下来的数据处理。因此对于计算密集型的应用MPI能表现出相当良好的性能，但对于处理数据密集型应用，由于大量的数据需要在节点之间进行传递交换，网络IO时间将成为影响系统性能的瓶颈，用“计算换通信”也是MPI并行程序设计中的基本原则。当然这种情况正在改观，已经出现一些分布式文件系统开始在底层支持MPI模型，但是其中的大部分与HDFS相比在容错性、可扩展性等方面差距明显[8]。

在Hadoop中利用HDFS可以实现分布式存储，这就意味着可以使各个节点直接读取存储在本地的数据进行处理计算，避免了在网络IO上损失过多的性能。实现了“计算向存储迁移”。在数据密集型的应用中会有比较好的表现，已有研究表明Hadoop在PB级数据存储和处理中，仍然表现优良[9]。同时Hadoop可以很好的应对节点失效或者网络的通信终端情况。在某些情况下如果出现了异常情况，HDFS的备份和容错机制可以保证应用的持续稳定进行，而不用重新开始所有的计算处理，因而可靠性得到了保证。但是由于Hadoop对于并行模型的过于封装，导致人们很难实现应用特定的适配性调整。特别是在部分科学计算领域，很难深入到底层进行灵活控制。另外Hadoop的核心使用Java语言编写，其在运算性能上要低于MPI的C和Fortran绑定。而且使用Hadoop与MPI相比，在网络带宽提高的情况下，性能增幅较低。

为了比较两者哪个更适用于地学研究中的高计算量应用，本文通过实验比较了两者在地学研究中常见的K-means算法中的性能。并对实验结果进行了较为详细的说明和解释。

K-means算法是聚类分析中非常重要的一种算法，而聚类分析又在地学研究中普遍使用。例如王龙、王晓青等人使用K-means进行了水库地震烈度衰减的研究[10]。牛瑞卿等人“利用数据挖掘的滑坡监测数据处理流程”[11]中也多次使用了K-means算法。北京邮电大学的段炬霞提出可以使用K-means算法对地质勘探地震数据进行聚类，并尝试改进该算法以提高其有效性和适用性[12]。由此可见K-means算法在地学研究中有着广泛的应用场景，所以我在本文中通过对该算法在MPI和Hadoop两种并行编程模型中的实际性能进行比较分析，以希冀能为未来的地学高计算量应用提供参考性建议。

## 本文的结构

本文同过对MPI并行计算模型和Hadoop分布式计算模型在K-means聚类算法方面的性能进行比较分析，探讨两者的性能差异，为今后的科学研究特别是地学研究中的高计算量应用提供一定的参考。

本文各章节的安排如下：

第1章（绪论）概括性的描述了本选题的研究背景和国内外关于该课题的研究现状。给出了本文的结构和安排。

第2章（并行计算基础理论）主要介绍了并行计算的一些基础概念，常见的并行计算机结构模型和并行编程模型。

第3章（MPI和Hadoop简介）分别描述了MPI和Hadoop两种并行编程框架的基本内容和主要特点。

第4章比较详细的介绍了K-means算法的内容，基本算法步骤，复杂度分析，然后分别介绍了给予Hadoop和MPI的K-means算法设计，最后给出实验环境说明。

第5章分析了MPI和Hadoop两种并行编程模型在k-means算法方面的性能差异，并做了比较详细的解释。

第6章为结论部分，对之前的实验结果和性能分析进行概括性总结。

# 并行计算基础理论



## 并行计算的概念

并行计算是指同时使用多种计算资源对多任务、多指令或者多数据项进行处理，解决计算问题的过程[13]，并行计算是提高计算机处理能力的一种有效手段。其核心思想就是要把一个任务分解成多个子任务，利用多处理器的实时协作，共同求解该问题，从而达到快速求解大规模应用问题的目的。目前并行计算及其思想被认为是科研工作者和工程师用来解决各种复杂领域的问题的标准方法，如地质学中的古气候还原，天文学中银河系的演变过程，化学中分子动力模拟等一系列需要海量计算的问题[14]。

## 并行计算机结构模型

一般来说大型并行机系统结构一般可分为6类：

* 单指令流多数据流计算机（SIMD）
* 并行向量处理机（PVP, Parallel Vector Processor）
* 对称多处理机（SMP, Symmetric Multiprocessor）
* 大规模并行处理机 (MPP, Massively Parallel Processor)
* 工作站机群 (COW, Cluster of Workstation)
* 分布式共享存储 (DSM,Distributed Shared Memory)多处理机。

SIMD计算机多为专用，其余五中均属于多指令流多数据流计算机（MIMD）。其结构模型如图2-1所示。（此图重绘）



图2-1 5种并行机结构模型

* B(Bridge)：存储总线和I/O总线间的接口
* DIR(Cache Directory)：高速缓存目录
* IOB(I/O Bus)：I/O总线
* LD(Local Disk)：本地磁盘
* MB(Memory Bus)：存储器总线
* NIC(Network Interface Circuitry)：网络接口电路
* P/C(Microprocessor and Cache)：微处理器和高速缓存
* SM(Shared Memory)：共享存储器



### 并行向量处理机（PVP）

PVP的结构示如图2-1(a)所示。这样的系统中包含了若干个专门定制的高性能向量处理器（Vector Processor），一般来说每个VP都具有1GFLOPS以上的高速处理能力。在该系统中使具有高带宽、低迟延的交叉开关网络将各个向量处理器连向共享存储模块，共享存储器以每秒兆字节的速度向处理器提供所需处理的数据。一般来说并行向量处理机不使用高速缓存，而是使用数量较多的的向量寄存器和指令缓存器。

### 对称多处理机（SMP）

SMP的结构如图2-1(b)所示。对称多处理机系统中微处理器经由高速总线或者类似PVP中的交叉开关与共享存储器相连，给系统与PVP相比，最大的不同是该系统中的各个处理器是对称的，可以等价的方位共享处理器或者I/O设备。由于这种对称性，才赋予了SMP比PVP更高的并行度，但是由于使用了与PVP类似的共享存储，导致系统所负载的处理器一般低于64，于此同时总线和交互开关也是影响其并行度的关键因素。目前SMP重要用于数据仓库（data warehouse），在线事务处理（OLTP）等应用中。

### 大规模并行处理机（MPP）

MPP的结构如图2-1(c)所示。大规模并行处理机一般具有一下特性：

* 处理节点采用类似SMP的微型处理器；
* 系统中的存储器是分离的；
* 采用专门设计的高通信带宽；
* 能扩充至最多几千个并行处理器；
* 它是一种异步的MIMD机器，程序系由多个进程组成，每个都有其私有地址空间，进程间采用传递消息相互作用。

大规模并行处理机目前的主要应用领域包括高性能科学计算，复杂物理信号的处理以及高仿真的工程模拟。

### 分布共享存储多处理机(DSM)

DSM的结构如图2-1(d)。拥有高速缓存目录是DSM与MPP最大的不同，由于DIR的存在，DSM可以支持各个分布高速缓存的一致性。相对于最终的用户来说，SDM提供了一个单地址的编程空间，也就是说用户无需指定到底使用哪个存储器里的内容，简化了编程过程。

### 工作站机群（COW）

COW结构示于图1.15(e)。工作站机群一般都是成本较低并行处理机，COW的每个节点通过普通的低成本网络，例如以太网，ATM开关等进行互联，每个节点都是一个相对独立的工作节点，可以使一台PC也可以是SMP。与MPP有所不同，COW的每个节点都没有本地磁盘。现今，MPP和COW之间的界线越来越模糊。例如，IBM SP2 它虽视为MPP，但它却有一个机群结构。机群相对于MPP有性能/价格比高的优势，所以在发展可扩放并行计算机方面呼声很高。

## 并行编程模型

并行编程模型是指并行编程的一种形式，目前

并行编程模式，通俗的说就是指并行编程的一种形式，一种方式，就像串行编程时，你是采用过程式还是结构化一般。并行编程模式只要指并行编程时，程序员将程序各模块并行执行时，模块间的通信方式，目前并行编程模式主要包括以下三种：

1 共享内存模式，以openMP为代表；

2 消息传递模式，以MPI,PVM为代表；

3 数据并行模式，以Fortran为代表；

数据并行编程模型和消息传递编程模型是目前最重要的两种并行编程模型，也是本论文重点讨论的两种模型。数据并行编程模型（比如Hadoop）通过高度抽象，编程过程十分简单，有人把它比喻成常规编程中的Java语言。消息传递编程模型（比如MPI）抽象程度较低，但是可以操作的东西也更多，因此拥有更高的应用领域，有人把它比喻成常规编程中的C语言。当然还有一些其它并行编程模型，比如隐式模型和共享变量，隐式并行通过编译器和运行时环境的支持来挖掘程序的潜在并行性。共享变量一般适合在SMP系统和多核系统中，代码分别驻留在若干个并行处理器中，但是可以通过读写共享存储器中的变量进行通信[15]。



### 数据并行编程模型

　　数据并行编程模型利用将要处理的数据集合的无关性，将相同的操作并行作用于不同的数据，该模型为使用者提供了一个全局的地址空间，一般这种形式的语言本身就提供并行执行的语义，因此对于使用者来说，只需要简单地指明将要执行怎样的并行操作以及并行操作的作用对象，就可以实现数据并行编程。因此数据并行的表达是非常简单和简洁的，它不需要编程者关心并行机是如何对该操作进行并行执行的。

### 消息传递并行编程模型

　　消息传递即各个并行执行的部分之间通过传递消息来交换信息、协调步伐、控制执行。消息传递一般是面向分布式内存的，但是它也可适用于共享内存的并行机。

　　消息传递为编程者提供了更灵活的控制手段和表达并行的方法，一些用数据并行方法很难表达的并行算法，都可以用消息传递模型来实现，灵活性和控制手段的多样化，是消息传递并行程序能提供高的执行效率的重要原因。

　　消息传递模型一方面为编程者提供了灵活性，另一方面，它也将各个并行执行部分之间复杂的信息交换和协调、控制的任务交给了编程者，这在一定程度上增加了编程者的负担，这也是消息传递编程模型编程级别低的主要原因。虽然如此，消息传递的基本通信模式是简单和清楚的，学习和掌握这些部分并不困难，因此目前大量的并行程序设计仍然是消息传递并行编程模式。

### 消息传递与数据并行的对比

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数据并行与消息传递并行编程模型   |  |  |  | | --- | --- | --- | | 对比内容 | 数据并行 | 消息传递 | | 编程级别 | 高 | 低 | | 适用的并行机类型 | SIMD/SPMD | SIMD/MIMD/SPMD/MPMD | | 执行效率 | 依赖于编译器 | 高 | | 地址空间 | 单一 | 多个 | | 存储类型 | 共享内存 | 分布式或共享内存 | | 通信的实现 | 编译器负责 | 程序员负责 | | 问题类 | 数据并行类问题 | 数据并行、任务并行 | |

不同的并行编程模型也就是不同的通过并行程序解决问题的模式，数据并行的主要特征是以数据为中心，通过对数据的划分和并行处理来解决问题，消息传递当然也可以实现上述功能，但是消息传递在问题的表述上更具体，更低级，可以解决的问题相对于数据并行模型来说也更广泛。在一定程度上，可以把数据并行看作是消息传递的特殊形式。

# MPI与Hadoop简介



## MPI简介

MPI(Message Passing Interface)是由科研机构和工程部门共同设计的一套基于消息传递并行编程模型的标准规范，使用标准的MPI规范编写的程序，可以不加修改的运行在异构的并行计算机上。目前比较知名的MPI标准实现是MPICH以及OPENMPI，它们都已经支持最新的MPI 3标准，并都经过了严格的功能和性能测试。一般来说MPI的实现都会提供C，C++和Fortran的语言绑定，因为这三种语言编写出来的程序运行速度较快，而且对于科研人员也比较熟悉。但是现在已有组织和个人开始尝试MPI的GoLang，RUST等新兴并发语言的绑定。

MPI的并行编程主要依靠一组消息传递库函数进行。这组函数可以在不同的运算机器上进行数据交换，提供了并行任务间的发送，接收和同步接口。MPI有两种基本的并行程序设计模式：总从模式和对等模式。这两种模式可对应于Web开发中的C/S模式和C/C模式。

主从模式：该种模式中一个主进程（master）对整个并行程序进行总体监控和协调，各个从进程（slave）在主进程的控制下对数据进行并行的处理和运算。这是当前的一些程序使用比较多的一种并行程序设计模式。也是本文对应实验所采用的方式。

对等模式：并行程序中的各个进程的地位完全等价，不同的只是它们处理的数据对象有所不同。



### MPI的通讯模式

MPI共有四种不同的点对点通信模式：标准通信模式（standard）、缓存通信模式（buffer）、同步通信模式（synchronization）、就绪通信模式（ready）

标准通信模式：是否缓存数据由MPI决定。如果缓存数据，发送操作的进行与接收操作是否执行无关。数据缓存完毕时发送操作就会正确返回。如果不缓存数据，只有当相应的接收调用执行，并且发送数据完全到达接收缓冲区后，发送操作才算完成。这时，发送缓冲区可以被覆盖。对于非阻塞通信，发送操作虽然没有完成，但发送调用已经正确返回。

缓存通信模式：程序员直接对通信缓冲区进行控制。用户必须保证发送前必须有缓冲区可用，否则失败返回。阻塞发送后，缓冲区可覆盖，但非阻塞发送正确返回后，缓冲区不能立刻被覆盖。这种通信模式下发送调用是否正确返回只和有无缓冲区有关。另外，请注意区别MPI的发送/接收缓冲区与系统缓冲区。

同步通信模式：这种通信的开始不依赖于相应的接收操作是否起动，但是同步发送必须等相应的接收进程开始后才可正确返回。这个返回意味着发送缓冲区中的数据被系统缓冲区缓存并且开始发送，就是说，返回时发送缓冲区可以被覆盖了。

就绪通信模式：只有当接收操作起动时，发送操作才能开始，否则发送操作会出错。对于非阻塞发送操作的正确返回，不等于发送完成，但对于阻塞发送的正确返回，发送缓冲区可被覆盖。这是一个特别的通信模式，所以，一般在使用时，会先起动接收操作，然后再向发送进程发消息，发送进程收到这个消息后，才会开始进行发送操作。

## Hadoop简介

Hadoop是一个由Apache软件基金会开发的一个开源分布式系统基础架构。Hadoop允许用户在不了解分布式底层细节情况下，开发分布式应用程序。充分利用分布式集群进行高速运算和存储。Hadoop主要由两部分组成，底层是一个分布式文件系统HDFS（Hadoop Distributed File System），上层是Mapreduce并行编程框架。HDFS有高容错性的特点，并且设计用来部署在低廉的硬件上；而且它提供高吞吐量（high throughput）来访问应用程序的数据，适合那些有着超大数据集（large data set）的应用程序。HDFS放宽了POSIX的要求，可以以流的形式访问文件系统中的数据。Hadoop是一个能够让用户轻松架构和使用的分布式计算平台。用户可以轻松地在Hadoop上开发和运行处理海量数据的应用程序。它主要有以下几个优点：

高可靠性。Hadoop按位存储和处理数据的能力值得人们信赖。

高扩展性。Hadoop是在可用的计算机集簇间分配数据并完成计算任务的，这些集簇可以方便地扩展到数以千计的节点中。

高效性。Hadoop能够在节点之间动态地移动数据，并保证各个节点的动态平衡，因此处理速度非常快。

高容错性。Hadoop能够自动保存数据的多个副本，并且能够自动将失败的任务重新分配。

低成本。与一体机、商用数据仓库以及QlikView、Yonghong Z-Suite等数据集市相比，hadoop是开源的，项目的软件成本因此会大大降低。

Hadoop带有用Java语言编写的框架，因此运行在 Linux 生产平台上是非常理想的。Hadoop 上的应用程序也可以使用其他语言编写，比如 C++。

# K-means算法及实验设计



## K-means聚类算法概述

K-means算法是数据挖掘过程中的一种重要聚类算法。K-means聚类要求将n个数据对象分配到k个分区中，并且要求分区满足以下条件：在同一个分区中的数据对象相似度要达到最大，不同分区中的数据对象的相似度要达到最小。该算法的形式化描述如下：

对于给定的一组数据对象集合，每个数据对象都是一个d维的实数向量，K-means聚类算法就是要把这n个数据对象分配到k个分区之中，使得下式的值最小：

其中是的分区中心，与数据对象相同，也是一个d维的实数向量。

## K-means 算法基本步骤

K-means算法的具体实现步骤如下：

1. 从 n个数据对象集合中随机选择 k 个数据对象作为分区中心；
2. 依次对集合中的每个数据对象进行分区操作，即分别求该对象与k个分区中心的欧式距离（或者其它需要的度量），得到一个最小距离，然后将数据对象划分到该分区中。
3. 按照每个分区所包含的数据对象，重新计算该分区的分区中心；
4. 计算标准测度函数，当满足一定条件，如函数收敛时或者已经达到所需精度，则算法终止；如果条件不满足则回到步骤b)。

## K-means算法复杂度分析

由于我们并不能保证K-means算法一定会收敛于全局的最优解，通常该算法会终止于某个局部最优解，该最优解取决于初始分区中心的选择。在实际使用过程中，通常用不同的分区中心进行多次K-means运算，以得到比较优良的结果。

K-means聚类算法的复杂度是O（nkt），其中n是数据对象的数目。k是分区数量或者说簇数，t是迭代的次数。一般情况下，k和t是远小于n的。这也就给我们在实际应用中很大的伸缩性。

## 基于Hadoop的并行K-means算法设计

使用Hadoop进行并行计算，用户一般只需要进行Map和Reduce函数的设计。通常包括该函数的输入输出键值类型以及两者的具体逻辑实现。之前已经提过，K-means算法的主要工作是根据不同数据对象到分区中心的距离大小，把不同的数据对象聚类到各个分区之中。这些操作是彼此相互独立的，可以实现并行计算，在每次迭代过程中，使用相同的Map和Reduce操作即可完成。在设计这两个函数之前，我们首先需要在n个数据对象中随机选择k个数据对象作为初始的分区中心。然后将n个数据对象和k个分区中心所在的文件保存到Hadoop的HDFS之上（为了方便叙述，文件分别命名为cluster.txt和centers.txt）



### Map函数实现

首先选取需要的输入键值对，我们按照惯例做法，用cluster.txt文件中的每条数据相对于cluster.txt文件的起始位置的偏移量作为输入Key，用表示当前数据对象的d维实数矢量作为Value值。由于它们在输入文件中是以文本形式描述的，所以我们需要将String格式的Value进行初步解析，得到一个d维的实际坐标数据。然后同样解析centers.txt文件中数据，得到k个d维的分类中心坐标数据。接着依次算出每个数据对象与各个分区中心的欧式距离，找出其中与该数据对象距离最短的那个分区中心，最终的输出键值对表示为（Key’, Value’），其中Key’代表与该观察点最近的分区中心，Value’代表该数据对象的坐标。Mapreduce将这些（Key’, Value’）发送给之后的Reduce函数进行处理。

### Reduce函数的实现

Reduce函数接收到Map函数发送过来的（Key’, Value’），对所有Key’相同的Value’进行求均值操作，也就是将所有Key’值相同的Value’进行聚合，然后利用聚合后的数据求出一个新的分区中心，然后用这个分区中心更新该聚类初始的中心。反映在实际操作中就是更新centers.txt文件。如果已经达到了所要求的精度或者最终的结果已经收敛则停止运算，否则进行下一轮迭代。

## 基于MPI的并行K-means算法实现

在基于MPI的并行K-means算法中，我们采用主从模式进行设计，在我们的程序之中有若干个进程，他们可以通过进程号进行识别，我们假设0号进程是我们的主进程，其他进程为从进程。与基于Hadoop的并行K-means算法实现类似，我们同样拥有两个数据文件，cluster.txt文件中放置的是所有n个数据对象的坐标数据，centers.txt文件中放置的是随机选取的k个初始分区中心的坐标数据。



### 主进程的主要任务

主进程主要负责初始化MPI进程环境，检查文件的合法性，检测集群的状态。然后读取cluster.txt文件，获取一些初步的信息，比如n的值，各个观察点的维度信息等。然后将数据对象进行划分，尽量保证每个从进程可以分得近似的数据量和计算量。然后将将划分结果发送给各个从节点，让每个从进程进行计算，当计算完成，主进程收集各个从进程的运算结果得到新的聚类中心。

### 从进程的主要任务

从进程得到主进程发送过来的数据划分结果，然后读取文件进行计算。首先检查每个属于自己的数据对象，通过计算每个数据对象到分区中心的欧式距离，找到距离他们最近的分区中心。然后进行一次Reduce操作，将与某个分区中心最近的所有观察点进行聚合，算出这些这些数据对象的中心点，如果该中心点与出原始分区中心的欧式距离在要求的精度之内则停止运算，否则进行下一次迭代。

## 实验环境说明

本次实验的实验环境所用集群由5台普通商用服务器组建，5台机器的主机的IP地址以及主机名称分别为：

* 10.1.100.106 cluster-ib.ipoib
* 10.1.100.254 compute-ib-0-0.ipoib
* 10.1.100.253 compute-ib-0-1.ipoib
* 10.1.100.252 compute-ib-0-2.ipoib
* 10.1.100.251 compute-ib-0-3.ipoib

其中cluster-ib.ipoib的参数为：

* CPU: 4×32 Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.60GHz
* RAM: 128G
* System: Linux 2.6.32-431.11.2.el6.x86\_64
* OS: CentOS release 6.5 (Final)
* network: infinite band

另外四台服务的参数为：

* CPU: 2×8 Intel(R) Xeon(R) CPU E5-2650 v2 @ 2.13GHz
* RAM: 8G
* System: Linux 2.6.32-431.11.2.el6.x86\_64
* OS: CentOS release 6.5 (Final)
* network: infinite band

在Hadoop集群实验中，共设有1个NameNode，1个SecondNamenode，4个DataNode，1个ResourceManager（JobTracker），4个NodeManager（TaskTracker），其中cluster-ib.ipoib作为NameNode，SecondNameNode，ResouceManager载体，其余4台机器作为DataNode核NodeManager载体。实验时，Map和Reduce数量均根据集群配置和输入数据规模进行自动选择。集群之间通过SSH进行关联。

在MPI集群实验中，五个节点均作为等价计算节点，主节点和从节点在程序运行时自主选择。底层集群之间通过SSH进行关联。

所有数据对象均使用随机算法产生，并不代表实际物理意义。

# 实验结果



## 使用Hadoop运行K-means算法性能分析

在此次试验中，我们考虑不同运算精度、不同数据量对于运算时间的影响。运算精度是指经过某次迭代后，新旧聚类中心的欧式距离的偏差。例如：精度为1则代表，新旧聚类中心的欧式距离偏差如果小于1，即停止迭代，程序终止，否则继续进行下一次迭代。之所以没有直接采用迭代次数作为程序终止条件，是因为Hadoop和MPI两种编程模型中，迭代相同次数，并不能保证最终输出结果精度完全相同，而在实际科研领域，我们一般只考虑结果，而忽略具体的计算过程。

在Hadoop试验中，我们设置精度分别为1/0.1/0.01。输入数据规模分别为10万/20万/40万/80万/160万，最终在满足精度要求的情况下，将输入数据聚集到5个集合中。得到的计算时间与精度要求以及输入数据规模的关系可见下表：

表 5‑1 Hadoop计算时间与计算精度（数据规模）关系表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据量精度 | 100000 | 200000 | 400000 | 800000 | 1600000 |
| 1 | 1398.4 | 1888.9 | 1909.3 | 2377.2 | 3351.9 |
| 0.1 | 8117.4 | 10951.9 | 15527.8 | 28090.3 | 49565.6 |
| 0.01 | 20495.6 | 25281.0 | 42860.0 | 74050.4 | 141924.8 |

为了更清楚的分析使用Hadoop分布式计算模型运行K-means算法的性能，我们从不同角度进行绘图，首先考虑Hadoop计算时间与计算精度的关系。图5-1展示了Hadoop实际计算时间与计算精度的关系，可以清楚的看到随着计算精度的提高，也就是说随着运算迭代次数多增多，运算时间持续增加。这种关系对于不同的数据规模具有普适性。在图中我们还能看出，这种增长的趋势是逐渐加剧的。也就说精度每提高10倍，时间的增加量会越来越大，体现在K-means算法中，就是随着精度要求的提高，继续提高相同的精度，需要迭代的次数会越来越多。

图 5‑1 Hadoop计算时间与计算精度关系图

接下来，我们考虑Hadoop计算时间与数据规模的关系，如图5-2所示，随着数据规模的增加，K-means算法在Hadoop集群中的运算时间持续增加，结合图5-3的对数图观察，很容易发现这种增长趋势趋于线性。也就是说Hadoop并行计算模型对于输入数据具有良好的可扩展性。

图 5‑2 Hadoop计算时间与数据规模关系图

图 5‑3 Hadoop计算时间的对数与数据规模关系图

## 使用MPI运行K-means算法性能分析

为了更好地分析MPI计算模型的性能特性，我们分别统计了在K-means算法中的I/O时间，计算时间，总时间及其与集群规模，数据规模，计算精度的关系。

其中集群规模通过进程数来体现，通过在运行程序时制定host文件，设置进程数目与集群关系的规模。为了叙述方便，下文将不再区分进程数和集群规模这两个术语。对于数据规模，同Hadoop中一样，我们设置了10万/20万/40万/80万/160万5组。对于计算精度我们设置了1/0.1/0.01/0.001/0.0001共5组。

我们首先考虑MPI程序中的I/O时间与集群规模，计算精度的关系。其中具体数据可以参考下表（表5-2）。

表 5‑2 MPI I/O时间与集群规模（计算精度）关系表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 精度  进程数 | 1 | 0.1 | 0.01 | 0.001 | 0.0001 | 0.00001 |
|
| 1 | 516.0 | 513.3 | 556.6 | 522.5 | 520.7 | 521.6 |
| 2 | 521.6 | 519.6 | 508.0 | 540.0 | 522.9 | 511.0 |
| 4 | 520.4 | 522.3 | 539.3 | 523.0 | 531.9 | 530.1 |
| 8 | 517.8 | 508.4 | 522.6 | 529.3 | 507.0 | 520.8 |
| 16 | 514.1 | 517.5 | 518.1 | 546.2 | 527.2 | 523.6 |

我们将该表进行绘图，在图5-4中我们可以发现MPI中I/O时间与集群规模和阈值精度之间并没有必然关系，当然这与具体算法有关，体现在在本实验中，虽然进程数增加，但是每个进程所分到的数据量会减少，导致总的I/O时间并没有太大变化。另一方便，阈值精度提高，也就是说需要迭代的次数增多，但是仍没有导致I/O时间的上升，主要是因为I/O时间主要是数据文件初始化的文件I/O，在计算过程中的传递的信息主要是一些观察点的序号，以及数量很少的中心点坐标，这些导致的I/O与开始文件读取的时间相比，所占比例很小。所以即使迭代次数增加，网络传递的次数增加，但是由于基数很小，仍然不会对整体I/O造成很大影响。

图 5‑4 MPI I/O时间与集群规模关系图

接下来，我们考虑真正的计算时间与集群规模（计算精度）的关系。具体数据可以参考表5-3：

表 5‑3 MPI 计算时间与集群规模（计算精度）关系表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 精度  进程数 | 1 | 0.1 | 0.01 | 0.001 | 0.0001 | 0.00001 |
|
| 1 | 9.7 | 40.4 | 249.9 | 474.6 | 724.0 | 1132.4 |
| 2 | 4.8 | 21.7 | 125.2 | 237.0 | 305.1 | 648.2 |
| 4 | 3.0 | 10.4 | 62.8 | 119.0 | 181.9 | 329.7 |
| 8 | 1.3 | 5.3 | 32.1 | 60.1 | 93.9 | 157.1 |
| 16 | 0.7 | 2.8 | 16.2 | 30.6 | 46.6 | 72.0 |

针对该图，我们分别绘制了下面3幅图，图5-5展示了实际计算时间与计算精度的关系。图5-6展示了实际计算时间与集群规模的关系。图5-7是对应的将实际计算时间转换成对数形式。

从图5-5可以很明显的看出，MPI计算时间与计算精度呈现正相关，且变化趋势与Hadoop类似。具体原因前面已经叙述。

同图5-6和图5-7可以得到，MPI计算时间与集群规模呈现比较严格的线性负相关。说明MPI有着比Hadoop更加优良的可扩展性，可以通过扩展集群规模出来更大规模的计算量。

图 5‑5 MPI 计算时间与计算精度关系图

图 5‑6 MPI 计算时间与集群规模关系图

图 5‑7 MPI 计算时间的对数与集群规模关系图

接着我们考虑总时间（I/O时间+计算时间）与计算精度（集群规模的关系），具体实验数据见表5-4：

表 5‑4 MPI 总时间与集群规模关系图

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| MPI 总时间与集群规模关系图 | | | | | | |
| 精度  进程数 | 1 | 0.1 | 0.01 | 0.001 | 0.0001 | 0.00001 |
|
| 1 | 525.7 | 553.7 | 806.5 | 997.1 | 1244.7 | 1654.0 |
| 2 | 526.4 | 541.3 | 633.2 | 777.0 | 828.0 | 1159.2 |
| 4 | 523.4 | 532.7 | 602.1 | 642.0 | 713.8 | 859.8 |
| 8 | 519.1 | 513.7 | 554.7 | 589.4 | 600.9 | 677.9 |
| 16 | 514.8 | 520.3 | 534.3 | 576.8 | 573.8 | 595.6 |

我们用总时间的对数作为纵轴进行作图，得到图5-8，我们看到随着集群规模的增加，总时间逐渐减少，但不是呈现线性下降，但是当精度越高，也就是计算时间占总时间的比重越大，线性就越明显。从这里我们可以看出，MPI对于高计算量，高迭代运算可以简单的通过扩充集群规模进行性能提升。

图 5‑8 MPI 总时间的对数与集群规模关系图

在分别考率了与计算精度，集群规模之后，我们统计MPI程序的I/O时间，计算时间与输入数据规模的关系，具体数据见表5-5：

表 5‑5 MPI运行时间与数据量的关系

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 数据量  进程数 | 100000 | | 200000 | | 400000 | | 800000 | | 1600000 | |
| I/O | 计算 | I/O | 计算 | I/O | 计算 | I/O | 计算 | I/O | 计算 |
| 1 | 128.8 | 10.3 | 256.8 | 20.3 | 513.1 | 40.7 | 1483.2 | 80.4 | 2051.3 | 161.8 |
| 2 | 128.9 | 5.2 | 257.4 | 10.2 | 548.3 | 20.3 | 1082.3 | 40.4 | 2214 | 81.3 |
| 4 | 127.7 | 2.6 | 264.7 | 5.2 | 508.8 | 10.3 | 1008.4 | 20.4 | 2066.5 | 41 |
| 8 | 128.5 | 1.4 | 257.1 | 2.7 | 519.8 | 5.3 | 1016.5 | 10.4 | 2091.2 | 20.8 |
| 16 | 129.2 | 0.8 | 255.2 | 1.5 | 745.7 | 2.8 | 1090.9 | 5.4 | 2837 | 10.6 |

我们分别用I/O时间与计算时间的对数与输入数据规模进行作图（图5-9， 图5-10），可以发现无论是I/O时间韩式计算时间都与数据规模呈现正相关，这就导致了总时间与I/O时间是线性正相关的，而且这种线性要比Hadoop更加优良。

图 5‑9 MPI I/O时间的对数与数据规模关系图

图 5‑10 MPI I/O时间的对数与数据规模关系图

## Hadoop与MPI在K-means算法中的性能比较

下面我们对Hadoop与MPI进行对比，从5-11的柱状图可以发现对于目前的集群规模，输入数据量，以及相同的计算精度下，MPI所使用的时间要比Hadoop少的多，两者相差几十倍。这在一定情况下说明对于K-means这类需要较多迭代次数的算法，MPI更加占优势，这是因为Hadoop是一种近似离线计算，在每次运算之后的结果都需要存储在HDFS里面，下次迭代再使用的话，还需要读出来进行一次计算，磁盘IO开销比较大。而MPI使用消息传递机制，类似于少量的内存中间量需要经过高速网络直接传递，无需重复读写磁盘。

我们可以看到如果我们以数据规模作为横坐标，两者的比率作为纵坐标进行组图，结果呈现负相关。但是如果我们以计算精度作为横坐标，两者的比率作为纵坐标进行组图，结果呈现正相关。就是说明Hadoop对于数据增长比较友好，相反MPI对于计算量，迭代量增长更加友好。

表 5‑6 Hadoop与MPI计算时间与数据规模关系表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数据规模 | 100000 | 200000 | 400000 | 800000 | 1600000 |
| Hadoop | 8117.4 | 10951.9 | 15527.8 | 28090.3 | 49565.6 |
| MPI | 130.0 | 256.7 | 728.5 | 1096.3 | 2847.6 |
| Hadoop/MPI | 62.4 | 42.7 | 21.3 | 25.6 | 17.4 |
| 注：相同集群规模，阈值精度0.01 | | | | | |

图 5‑11 Hadoop与MPI计算时间与数据规模关系图

图 5‑12 Hadoop与MPI计算时间比率图

表 5‑7 Hadoop与MPI计算时间与计算精度关系表规模

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 计算精度 | 1 | 0.1 | 0.01 | 0.001 |
| Hadoop | 1909.3 | 15527.8 | 42860.0 | 115722.0 |
| MPI | 514.8 | 520.3 | 534.3 | 576.8 |
| Hadoop/MPI | 3.708819 | 29.843936 | 80.217106 | 200.627601 |
| 注：相同规模集群，400000规模数据量 | | | | |

图 5‑13 Hadoop与MPI计算时间比率图

# 结论

我们通过一系列的实验数据比较了Hadoop和MPI并行程序在K-means算法中的性能特点。我们可以得到下面的结论：

从整体来看，对于类似K-means这类需要数次迭代的算法，MPI相对于Hadoop有着较大优势，两者有着数十倍的性能差异。

对于精度提高，MPI和Hadoop所需时间的增加都近似指数增加，但是相对来说MPI的表现更加优异。也就说随着计算精度的提高，MPI相对Hadoop的优势会越来越大。

对于数据规模的提高，MPI和Hadoop所需时间的增加都近似线性增加，但是Hadoop的表现更加优良，也就说随着数据规模的提高，MPI相对Hadoop的优势会越来越小，甚至被超越。

MPI更加适合搞计算量，低IO的应用，而且在此类应用中，MPI拥有较好的可扩展性，可以通过增加集群规模，线性降低运行时间。

# 参考文献：

[1]张嘉桐. 基于云计算的地学 G~ 4I 系统结构设计[D]. 吉林大学, 2013.

[2] 曲寿利. 地震勘探技术的发展促进油气勘探新发现[J]. 石油地球物理勘探, 2005, 40(3): 366-370.

[3] 薛勇, 万伟, 艾建文. 高性能地学计算进展[J]. 世界科技研究与发展, 2008, 30(3): 314-319.

[4] 李焱. 基于 MPI 的大地电磁三维正反演并行算法研究[D]. 中国地质大学, 2011.

[5] 张治宏. 基于 MPI 的并行计算研究 [D][D]. 中国地质大学 (北京), 2006.

[6] 余欣, 杨明, 王敏, 等. 基于 MPI 的黄河下游二维水沙数学模型并行计算研究[J]. 人民黄河, 2005, 27(3): 49-50.

[7] Yang C, Xu Y, Nebert D. Redefining the possibility of digital Earth and geosciences with spatial cloud computing[J]. International Journal of Digital Earth, 2013, 6(4): 297-312.

[8] <https://issues.apache.org/jira/browse/MAPREDUCE-2911>

[9] 余飞. 电信运营商大数据应用典型案例分析[J]. 信息通信技术, 2014, 6: 014.

[10] 王龙, 王晓青, 郑友华, 等. 水库地震烈度衰减的研究[J]. 地震地质, 2009, 31(4): 758-767.

[11] 牛瑞卿, 韩舸. 利用数据挖掘的滑坡监测数据处理流程[J]. 武汉大学学报: 信息科学版, 2012, 37(7): 869-872.

[12] 段炬霞. K-means 算法的改进及其在地质勘探地震数据分析中的应用[D]. 北京邮电大学, 2012.

[13] Valiant L G. A bridging model for parallel computation[J]. Communications of the ACM, 1990, 33(8): 103-111.

[14] 伍少坤 基于影元胞技术的城市元胞自动机——以珠江三角洲与东莞市为例 2007

[15] 张武生，薛巍，李建江，郑伟民 MPI并行程序设计实例教程 清华大学出版社 2009.2

# 致谢

伴随着本科毕业论文的完成，我的大学生活也将告一段落了。在这里我真心想给与我帮助和指导的老师，同学和我的父母表示衷心的感谢。

首先真的很感谢我指导老师周会群周老师。在真个论文撰写过程中，周老师都给予了我很大的帮助。特别是在实验开始阶段，由于在集群配置过程中出现了这样或那样的问题，周老师都在百忙之中给与了我指导。并对我给与了很大的信任，也教会了我很多科研态度和精神。在这个过程中我学习到了很多额外的知识。

另外由衷的感谢南京大学地球科学与工程学院的老师们。在大学四年之中，地科院老师们的严谨治学的态度，钻研求实的科研精神都深深地影响了我。相信在以后的工作生活中都将受益匪浅。另外在生活中，老师特别是辅导员也无时无刻不在替我们着想。真的很感谢你们。

另外我的同窗好友让我们的课余大学生活变得丰富多彩。感谢2011级的全体同学，当然还有那些曾经帮助我们的学长学姐们。有了你们的陪伴，大学四年的生活必将是我今后人生中最美好的记忆。

感谢这段时间，我敬爱的父母在生活中的关怀。是你们一直支持着我前进。

最后感谢答辩组的各位老师在百忙之中的指点。