武漢理工大學

（申请工学硕士学位论文）

Hadoop 作业调度算法分析与优化

培养单位：计算机科学与技术学院
学科专业：计算机科学与技术
研 究 生：李词超
指导教师：钟 珞 教授

2015 年 4 月

分类号_____ 密　　级_____

UDC _____ 学校代码_____10497_____

武漢理工大學

学　　位　　论　　文

题　目_____Hadoop 作业调度算法分析与优化_____

英 文　Analysis and Improvement of Job Scheduling Algorithms

题 目_____in Hadoop_____

研究生姓名_____李词超_____

指导教师　姓名___钟珞___职称___教授___学位___博士___

单位名称 ___计算机科学与技术学院_____邮编___430070___

副指导教师　姓名_____职称_____学位_____

单位名称_____邮编_____

申请学位级别_____硕士_____学科专业名称___计算机科学与技术___

论文提交日期___2015 年 4 月___论文答辩日期___2015 年 5 月___

学位授予单位___武汉理工大学___学位授予日期_____

答辩委员会主席_____李琳　副教授_____评阅人___刘洪星 教授___

_____石兵　副教授___

2015 年 4 月

独 创 性 声 明

本人声明，所呈交的论文是本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，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得武汉理工大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

签 名：_____日 期：_____

学位论文使用授权书

本人完全了解武汉理工大学有关保留、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。本人承诺所提交的学位论文（含电子学位论文）为答辩后经修改的最终定稿学位论文，并授权武汉理工大学可以将本学位论文的全部内容编入有关数据库进行检索，可以采用影印、缩印或其他复制手段保存或汇编本学位论文。同时授权经武汉理工大学认可的国家有关机构或论文数据库使用或收录本学位论文，并向社会公众提供信息服务。

（保密的论文在解密后应遵守此规定）

研究生（签名）：_____ 导师（签名）：_____ 日期: _____

摘　要

随着技术的快速发展，互联网的使用人群急剧增长，数字化信息呈爆炸性增长，大数据的分析处理成为研究的热点。仿照 Google 公司 MapReduce 计算模型和 GFS 文件系统设计思想实现的 Hadoop 开源软件迅速发展并成为大数据处理的首选工具。Hadoop 平台为用户提供了方便的编程接口并合理安排用户作业的调度执行，用户只需专注于 map 和 reduce 处理过程。作业调度器合理安排作业任务的执行来合理利用集群计算资源，是 Hadoop 平台的核心模块之一，目前有 FIFO 调度算法、计算能力调度算法和公平份额调度算法三种实现供选择使用。FIFO 调度算法思想简单，易于实现，但不支持多用户多作业共享集群资源，计算能力调度算法和公平份额调度算法实现了多用户多作业对集群资源的共享，提高系统吞吐率，降低作业响应时间，但需要系统管理员对集群资源状况和用户和作业类型有充分了解从而合理进行各项配置。

本文在介绍国内外 Hadoop 作业调度研究现状的基础上，分析 Hadoop 现有各调度算法设计思想与调度策略并针对公平调度算法中 slot 资源分配方法提出改进，然后分析了各调度算法的优缺点并指出现有调度算法难以正确配置的不足，提出了基于贝叶斯分类的作业调度算法，该算法通过贝叶斯学习与分类让作业在计算节点上的运行尽可能使计算节点不过载，在此基础上，对作业进行资源需求分类预处理，使 CPU 密集型作业和 I/O 密集型作业得到合理调度，从而更高效利用计算资源。主要研究内容如下：

（1）分析 Hadoop 中 FIFO 调度算法、计算能力调度算法和公平调度算法的算法核心思想、使用配置并给出了伪码和流程图形式的算法描述、复杂度分析和算法特点及优缺点分析。同时，对公平调度算法的 slot 资源分配方法进行了改进，在原有 slot 资源分配方法基础上增加对剩余 slot 资源的尽可能平均分配，使得各资源池获得更为公平的份额。

（2）为克服使用现有调度算法时难以正确配置的不足，提出基于贝叶斯分类的作业调度算法，该算法通过分析作业任务执行历史信息将作业对资源使用的特征和计算节点的资源状态特征构造贝叶斯分类器，在学习作业任务执行时资源使用历史信息将作业在某计算节点分为可调度和不可调度两类，使得作业任务在计算节点上的执行尽可能不使计算节点过载，从而提高调度准确率并使得计算资源得到合理使用。

（3）为进一步提高计算节点的资源使用率，将作业按照资源需求类型情况分为 CPU 密集型作业和 I/O 密集型作业的预处理，进行分类调度使得计算资源得

到更充分的利用。

（4）结合实验对该算法的作业任务调度准确率、作业响应时间、集群资源利用率等性能指标做出评估并与现有调度算法进行比较分析。

关键词：Hadoop，MapReduce，作业调度，贝叶斯分类

Abstract

With rapid development of the Internet, the number of people using the Internet grows rapidly and digital information has an explosive growth and it becomes a hot spot for big data analyzing and processing. After Google introduces its big data computation framework MapReduce and distributed file system GFS, open-source software Hadoop has developed rapidly and becomes the most popular platform for big data processing, which was designed on their ideas. Hadoop provides an easy interface for developers who can only focus on the map and reduce functions and reasonably arranges the execution of jobs and tasks through job scheduling without user intervention. Job scheduler is one of core modules in Hadoop, and its goal is maximizing the use of the cluster resources through reasonable order of execution of many jobs and reasonable selection of tasks. Hadoop currently offers three job scheduling algorithms, which are FIFO scheduler, Capacity scheduler and Fair scheduler. FIFO scheduler is simple and easy to implement, but it does not support sharing resources for multi-users and multi-jobs. Capacity and Fair scheduler support sharing cluster's resources, increase throughput, decrease response time, but they need complicated configuration and administrator's fully understanding of the cluster's resources and types of users and jobs.

Based on domestic and foreign research on Hadoop, The paper analyzes the core idea and scheduling policy of existing scheduling algorithms and improves the slot allocation algorithm in Fair Scheduler. Then it analyzes the advantages and disadvantages and put up an scheduling algorithm based on Bayesian classification to overcome complicated configurations of the existing scheduling algorithms. The algorithm ensures jobs' running on nodes without overloading based through Bayesian learning and classifying. The paper then pre-processes jobs to CPU intensive and I/O intensive according to requirements of jobs to use computing resources more effectively. The paper's contents are as follows.

Firstly, deeply analyze and compare the FIFO Scheduler, Capacity Scheduler and Fair Scheduler in Hadoop, including their core idea, configuration, displaying of the pseudo-code, flowchart form with complexity description, features, advantages and disadvantages. Then it improves the slot allocation algorithm in Fair Scheduler to

allocate remaining slots as fair as possible.

Secondly, the paper puts up an scheduling algorithm to decrease and overcome the complicated configurations in existing scheduling algorithms. The algorithm classifies jobs for schedulable and not schedulable using Bayesian classifier which uses the job scheduling and executing history for learning according to features of jobs and nodes. Thus it schedules jobs to execute without nodes overloading as far as possible to improve the scheduling accuracy and resource usage of nodes.

Thirdly, the paper puts up an pre-processing step to classify jobs for CPU intensive and I/O intensive and schedules them separately to improve resource usage.

Fourthly, the paper chooses different types of typical jobs for experiment, and give assessment methods for the algorithm them. Then give results for scheduling accuracy, response time and cluster's resources usage ratio and analyzes the results comparing with the existing scheduling algorithms.

Key Words: Hadoop, MapReduce, Job Scheduling, Bayesian Classification

目 录

第一章 绪论

1.1 研究背景

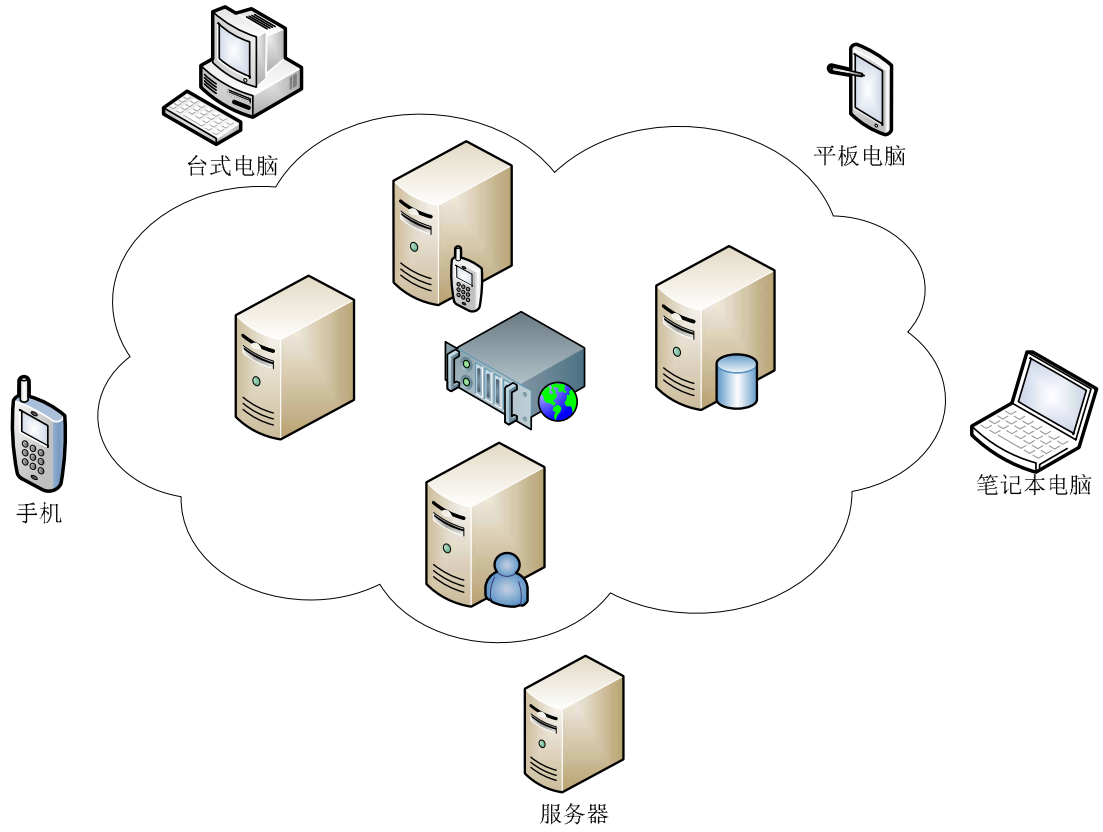
随着互联网、物联网技术的快速发展，互联网的使用人群急剧增长[1]，数字化信息呈爆炸性增长，海量的信息内容不断产生，让我们生活在数据时代。大量数据的产生给信息系统的开发以巨大的压力。尽管半导体技术的进步使得芯片处理速度呈现摩尔定律，但是仍然无法满足人们对海量数据的处理需求。为了处理日益增长的数据，信息系统的工作方式也由最初的单机系统逐渐演变成多机分布式系统。对信息的处理方式也在悄然发生变化，由最初的单机线性方式发展成高性能计算和多机并行计算，多机并行计算的发展促使了网格计算、分布式计算的产生与发展，进而发展到今天的云计算[2]。

云计算带来了一种全新的商业模式[2]，IT 服务商提供云端强大的计算、存储等服务，通过网络为企业或个人提供强大的计算或存储能力来获得商业报酬，而用户也减少或根本不需要筹建自己的的服务器，从而大大降低了成本。因此，这种模式迅速发展，具有巨大的市场前景和发展空间。

分布式技术可以说是云计算的核心基础[3]。由于业务功能的复杂和单台计算机计算和存储能力有限，系统各个模块不得不拆分，分布式技术为此提供了强大的支持。

现实的社会需求促使着云计算的快速发展，大数据时代信息系统面临的存储、分析、计算方式等问题成了云计算在技术方面的核心问题[4-6]。为此，学术界和工业界在这方面做了不断的努力和探索，使得云计算从概念走向现实。2006 年，Google 公司在世界搜索引擎大会上首次提出了云计算，随即得到迅速响应，各大科研单位、政府部门、企业在全世界掀起了云计算浪潮。国内外各大科技公司纷纷加入云计算产业中。Google、Yahoo!、Amazon、IBM、Oracle、Microsoft、EMC 等计算机行业巨头纷纷提出自己的云计算概念并推出了相应的产品，如 Google 搜索、Amazon EC2 与 S3、IBM 蓝云计划、Microsoft Azure 等；国内云计算发展也甚为迅速，阿里巴巴为此成立了阿里云子公司，IBM 在北京和无锡成立的云计算中心，网易、百度、中国电信、华为等也都推出了自己的云计算产品，试图一展身手，夺取先机，抢占其蕴含的巨大商业价值、应用前景和市场。

云计算的核心思想是集中大量计算、存储等共享资源，通过网络为用户提供本地机器难以做到的计算或存储工作，以最大化地有效利用这些共享资源。同时，云对用户是透明的，用户无需关注云端的实现细节，如图 1-1 所示。



1-1 云计算视图

如图 1-1 所示，手机、台式计算机、平板电脑、服务器等都作为云终端使用云提供的服务来完成各自的工作，而云中所有的计算、存储、通讯等细节对用户都是透明的，用户只需根据相应的规约使用即可。

根据用户实际不同的需求，云计算体系主要分为三个层次：基础设施即服务、平台即服务、软件即服务。基础设施即服务[2]（IaaS, Infrastructure as a Service）是指将云中服务器组作为一个整体对外提供硬件级别的计算服务，并同意管理 CPU、内存、I/O、网络、磁盘等计算资源。平台即服务[2]（PaaS, Platform as a Service）是指云中心为用户提供同意的应用软件运行平台，用户只需按照平台的编程接口完成应用自身的功能即可，而无需关心应用程序运行的平台特性。软件即服务[2]（Software as a Service）是指云中心提供各种完成某种功能的软件服务，用户只需获得相关许可即可使用该软件服务而无需自行开发。

1.2 研究现状与意义

为了实现云计算的目标，各大组织和单位纷纷开发出了自己的大数据处理平台，其中，Apache 开源基金会组织开发的 Hadoop[7]迅速成为主流产品。Hadoop

是 Apache Lucene 项目创始人 Doug Cutting 创建的，作为该网络搜索引擎的重要模块。

作为云计算的核心大数据处理平台，Hadoop 自提出以来经过快速迭代开发得到迅速发展，比较重要的版本有 0.20.X 系列、0.21.0/0.22.X 系列、0.23.X 系列，并且之间并没有取代和被取代，而是各自独立发展，具有不同的特点。其中，0.21.0/0.22.X 系列中将 Hadoop 正式划分为 Common、HDFS[10-12]和 MapReduce[13]三个模块，0.23.X 系列为了克服 Hadoop 在可扩展性和通用性方面的不足正式将 Hadoop 划分为 Common、HDFS、MapReduce 三个模块，分述如下：

（1）Hadoop Common 是 Hadoop 项目中的核心基础组件，为 Hadoop 其他模块提供一些常用工具，主要包括系统配置工具 Configuration、序列化机制、远程过程调用 RPC 等。他们都是一个分布式系统中的核心基础服务，例如远程过程调用 RPC 和序列化机制为分布式系统中各个节点提供基础通信服务，抽象文件系统为分布式系统提供一个统一的文件系统。

（2）Hadoop 分布式文件系统（HDFS，Hadoop Distributed File System）是为 Hadoop 平台提供高吞吐率数据访问的分布式文件系统，支持超大文件、流式数据访问、硬件配置低等。

（3）MapReduce 计算框架，是基于 YARN 框架的一个并行大数据离线计算框架。Hadoop YARN 框架和 MapReduce 框架是 Hadoop 最新版本中的两个重要模块。目前 YARN 框架已经实现的作业调度方法包括先来先服务调度算法（FIFO Scheduler，First In First Out Scheduler）、计算能力调度算法[14]（Capacity Scheduler）和公平调度算法[15]（Fair Scheduler）。

FIFO 调度算法沿袭了操作系统作业调度中的 FIFO 算法，核心思想是先进入系统的作业应当优先得到调度执行。这种方法比较简单易懂，实现也非常容易，但是缺点也是非常明显的，根本没有考虑作业的其他特性，如作业运行时长差异、资源需求差异、作业性质差异等，一个作业的运行会独占真个 Hadoop 系统资源。随着多用户提交作业，Hadoop 系统在多用户多作业场景下使用 FIFO 调度算法时系统性能比较低下，甚至有些大作业的运行会严重影响交互型短作业的正常运行，使得短作业的响应时间不可接受。

计算能力调度算法允许多作业并行运行，它将每个进入 Hadoop 系统的作业分配到某一个队列中去，每个队列中的所有作业共享队列拥有的计算资源。这样，只要作业所属的队列拥有计算资源，那队列中的作业就可以运行。计算能力调度算法的设计思想是将资源按比例分配给各队列并防止个别队列独占资源，同时当其他队列有空闲资源时可以共享使用。计算能力调度算法为每个队列分配一定的计算资源保障每个队列中作业的运行，特别是那些交互型短作业能够得到及时的

调度运行，克服了 FIFO 调度算法在处理交互型短作业时集群资源利用率低的缺点。但是使用计算能力调度算法需要手动进行队列各参数的配置，难以做到正确配置[18]。

公平调度算法同样也允许多作业同时运行，共享 Hadoop 系统计算资源并尽可能地保证系统中的所有作业都能够获得等量的资源份额。公平调度算法通过最大最小公平方法来进行资源分配。最小共享量保障了资源池在有作业运行时拥有一定的资源，公平共享量是资源池在获得最小共享额度的基础上其他资源池还有资源空闲时可以获得的资源量。资源池中的作业共享资源池的计算资源，当有新的作业提交至资源池运行时，资源池将分配出相应的资源给该作业。这样，提交至 Hadoop 系统中运行的小作业能够在合理的时间内运行完成，同时又保障大作业有资源运行而不处于长期饥饿状态。和计算能力调度算法一样，公平调度算法的使用也需要手动进行资源池各参数的配置，难以做到正确配置。

1.3 研究内容

本文在分析国内外 Hadoop 作业调度研究现状的基础上，确定研究目标是为 Hadoop 作业调度模块设计一个作业自动分类，使 Hadoop 作业调度智能化。在此作业调度智能化上，无需或尽可能少地要求用户对作业调度进行配置。

Hadoop 框架控制流程错综复杂，设计优秀的智能作业调度算法需要在深度分析现有作业调度算法及其应用场景的基础上进行。本文首先深入分析 Hadoop 框架的体系结构、各模块功能的基础上进行了如下工作：

（1）分析 Hadoop 中 FIFO 调度算法、计算能力调度算法和公平调度算法的算法核心思想、使用配置并给出了伪码和流程图形式的算法描述、复杂度分析和算法特点及优缺点分析。同时，对公平调度算法的 slot 资源分配方法进行了改进，在原有 slot 资源分配方法基础上增加对剩余 slot 资源的尽可能平均分配，使得各资源池获得更为公平的份额。

（2）为克服使用现有调度算法时难以正确配置的不足，提出基于贝叶斯分类的作业调度算法，该算法通过分析作业任务执行历史信息将作业对资源使用的特征和计算节点的资源状态特征构造贝叶斯分类器，在学习作业任务执行时资源使用历史信息将作业在某计算节点分为可调度和不可调度两类，使得作业任务在计算节点上的执行尽可能不使计算节点过载，从而提高调度准确率并使得计算资源得到合理使用。

（3）为进一步提高计算节点的资源使用率，将作业按照资源需求类型情况分为 CPU 密集型作业和 I/O 密集型作业的预处理，进行分类调度使得计算资源得到更充分的利用。

（4）结合实验对该算法的作业任务调度准确率、作业响应时间、集群资源利用率等性能指标做出评估并与现有调度算法进行比较分析。

本文的核心工作是将贝叶斯分类学习方法用于 Hadoop 作业调度中，即通过多次的作业调度学习后将进入系统的作业进行分类，然后将作业任务分配到合适的 TaskTracker 上执行。例如，可将作业分类为可调度和不可调度两类，可调度作业可在后续的作业调度中被调度执行。

1.4 论文组织结构

本文的组织一共分为六章节，每一章节内容安排如下：

第一章介绍了论文的研究背景、现状和意义、研究内容并给出了整个文章的组织结构。

第二章介绍了 Hadoop 平台中的 HDFS 和 MapReduce 计算框架，并分析了 Hadoop 中的分布式文件系统 HDFS、Hadoop MapReduce 计算框架、MapReduce 应用程序运行时环境，分析了 Hadoop 平台现有作业调度算法、做出对比并给出了优缺点分析。同时还对公平调度算法中的 slot 资源分配方法进行了改进，使得资源池间的缺额补充更为公平合理。

第三章详细介绍设计的基于贝叶斯分类的 Hadoop 作业调度算法，并给出了算法核心思想和详细推导过程与算法流程。该算法通过分析作业任务执行历史信息将作业进行分类调度并不断进行这一学习过程，从而将作业任务分配到合适的 TaskTracker 上执行，让其通过不断学习最终做出正确分配决策，提高 Hadoop 系统性能。为进一步提高计算节点的资源使用率，将作业按照资源需求类型情况分为 CPU 密集型作业和 I/O 密集型作业的预处理，进行分类调度使得计算资源得到更充分的利用。

第四章在第三章的基础上给出实验进行验证，同时给出了实验评估方法并对实验结果进行了详细分析，最后给出实验结论。

第五章对全文进行总结和展望，提出了一些需要改进的方面。

第二章 Hadoop 作业调度算法分析

本章首先介绍了 Hadoop 平台 Hadoop 分布式文件系统 HDFS 和 MapReduce 计算框架两个核心组件，分析 MapReduce 模型和作业执行流程。然后介绍并分析 Hadoop 中 FIFO、计算能力、公平份额三种现有调度算法，并对公平份额调度算法中的 slot 分配方法进行了改进，使得每个资源池的缺额资源补充更为公平，最后对个调度算法进行对比分析并总结优缺点。

2.1 Hadoop 平台架构

Hadoop 数据处理平台主要由 HDFS 和 MapReduce 构成，HDFS 负责可靠的数据文件操作，MapReduce 负责作业的调度与执行，他们相互配合共同完成大数据的处理任务。

2.1.1Hadoop 分布式文件系统 HDFS

HDFS 是一个分布式文件系统[11]，它处理的数据集（如 Google 搜索引擎在搜索时产生的网页数据量）远远超过单台物理计算机的存储承受范围，必须将数据进行拆分成多块存储在计算机集群中并进行有效管理。Hadoop 中的 HDFS[9]实现和 Google 的 GFS[10]十分相似，其为应用程序提供了非常重要的文件存储管理功能，具有如下特性：

（1）支持超大文件的存储和管理。超大文件是相对文件系统中存储的文件大小而言的，通常是以 MB 或 GB 甚至是 TB 为单位的文件。一般来说，Hadoop 分布式文件系统中存储和管理的文件总量在 TB（1TB=1024GB）或 PB（1PB=1024TB）级别。

（2）高效容错能力。Hadoop 分布式文件系统一般是部署由上百台甚至上千台普通计算机组成的集群上，单台计算机出现故障是正常现象，因此系统必须具有很好的故障检测和自动修复功能，这也是 HDFS 的一大特色。

（3）流式数据访问。Hadoop 应用程序处理的数据集一般比较大，每次都需要访问大量的数据，因此流式数据访问方式具有高吞吐率。

（4）简化的一致性模型。由于 HDFS 操作文件方式一般是一次写入、多次读取，因此在 HDFS 中一个文件一经创建并写入数据后，一般就不需要修改了，这样的一致性模型，提高了系统吞吐率。

HDFS 系统是一个典型的主从结构[11]，如图 2-1 所示。在 HDFS 集群中，只有

一个 Namenode 和若干个 Datanode。Namenode 是整个 HDFS 系统的控制中心，维护着整个 HDFS 系统的控制中心，维护整个文件系统的文件目录树、文件/目录的元数据信息，同时管理和操纵各数据节点。Datanode 是 HDFS 系统中文件存储和管理的实际执行者，一般一个 HDFS 集群中有着大量的 Datanode，它们的工作是将 HDFS 文件分块后的 Block 数据块写到 Linux 本地文件系统的中并进行管理。

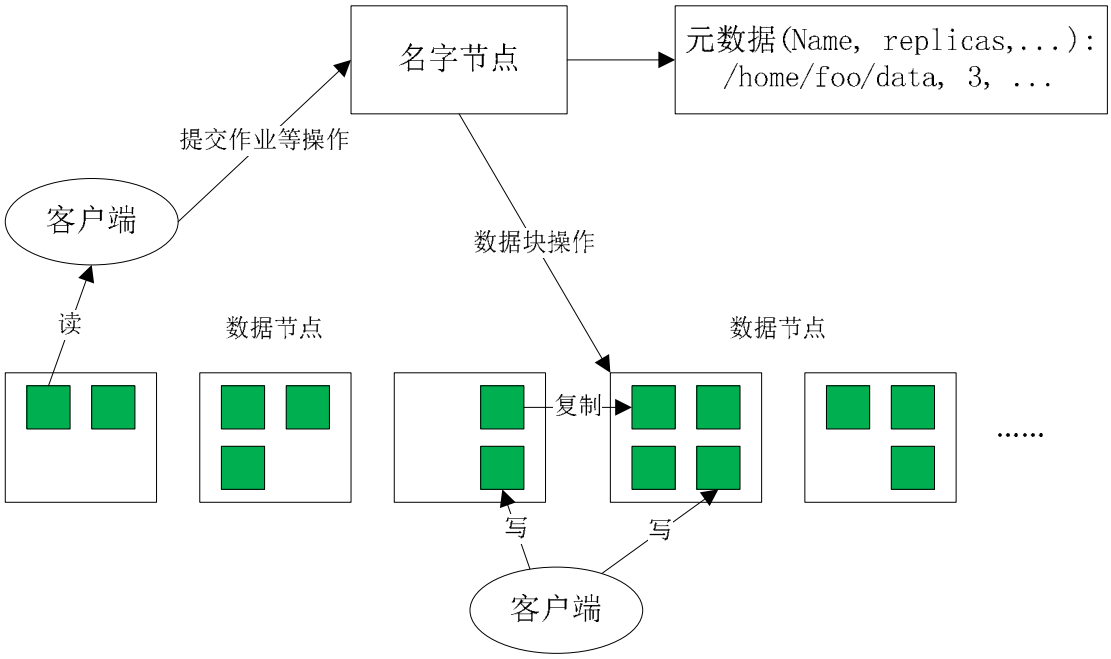


图 2-1 HDFS 体系结构

文件在 HDFS 中是分块存储的，为便于操作和管理大文件，HDFS 在存储大文件时将其切分为多个数据块（Block），和普通文件系统一样，数据块是 HDFS 中各种操作的基本单位。HDFS 是针对大文件而设计的分布式系统，数据块的设计有如下有点：（1）HDFS 可以存储超过单一节点存储能力的大文件。采用将大文件划分为快，分布式存储的设计，即使是超过单一计算机存储能力的大文件，HDFS 也能足够容纳。（2）简化存储子系统设计。由于处理文件时并不需要全部文件数据而仅仅是其中某一部分数据，因此将管理文件和数据块的功能分开，简化了存储管理，简化了分布式管理文件元数据的复杂性。（3）方便容错和数据复制。在 HDFS 中，为了应对损坏的数据块以及磁盘或机器故障，数据块会在不同的机器上进行备份（Replication 默认值为 3，即一数据块会存储在 3 个 Datanode 上）存储管理。如果某一数据块副本丢失或损坏，系统可以在其他的 Datanode 上读取。

HDFS 文件和目录命名空间是树状结构，并由 Namenode 维护。在 Namenode 上，文件和目录是以 inode 的概念呈现的，它记录了文件的诸如权限、修改与访问时间、命名空间等信息，这些被称为文件/目录元信息，这样 Namenode 可以为客户端提供文件操作。同时，Namenode 维护着文件与数据块的映射信息以及

数据块与 Datanode 的映射信息, 因此 Namenode 可以管理与文件对应的数据块。Datanode 实际管理文件的各个数据块，并通过“ 心跳 ”协议（hartbeat protocol）与 Namenode 通信。存储在 Datanode 上的每个数据块实际上包含两个文件，一个是数据块本身，一个是数据块元信息。在 Datanode 启动时，它与 Namenode 通过握手协议加入到 Datanode 集群中。启动成功后，Datanode 通知 Namenode 其上的所有数据块副本均可被访问。“ 心跳 ”中包含的信息还包括 Datanode 自身的总存储空间、已使用存储空间等信息，用于 Namenode 在进行文件存储时的数据块空间分配和负载均衡。

HDFS 客户端是 HDFS 提供的一个接口，用户应用程序通过该接口访问 HDFS 文件系统。HDFS 客户端封装了管理 HDFS 文件系统的接口，包括文件/目录的创建、读写文件、删除文件等操作。用户应用程序直接通过这些接口访问 HDFS, 并不需要了解 HDFS 中文件数据块副本的具体存储等细节。HDFS 中涉及到的相关实体如表 2-1 所示：

表 2-1 HDFS 中的实体

HDFS 中的对象	中文含义	解释
Namenode	名字节点	整个 HDFS 系统的控制中心, 维护整个文件系统的文件目录树、文件/目录的元数据信息，同时管理和操纵各数据节点。
SecondaryNamenode	备份名字节点	备用的 Namenode，每隔一段时间获取 Namenod 的状态, 当 Namenode 出现故障时作为 Namenode 提供服务。
Datanode	数据节点	HDFS 文件系统中保存具体文件的服务器节点，Datanode 保存的文件是以 Block 块为单位的。
HDFS Client	HDFS 客户端	客户端提供了用户应用程序与 HDFS 系统交互的接口，接口屏蔽了 HDFS 的实现细节
Block	数据块	存储在 Datanode 上的数据单位，为了解决大文件的存储问题, HDFS 将大文件切分成块进行分别存储和管理。
Packet	数据包	客户端每次写入文件系统时

		的数据量单位，客户端向 HDFS 中写入文件时并不是一个字节一个字节的进行，而是当写入的字节数积累到一定量时进行的
Chunk	传输块	HDFS 系统中各节点或与客户端之间进行传输时的数据量单位，每个传输块配有一个奇偶校验码。

当客户端通过 FileSystem.open()接口打开某个特定文件时，某个 HDFS 具体文件系统，如 DistributedFileSystem 会创建一个输入流 FSDataInputStream（一般是其具体类 DFSInputStream）返回给客户端，客户端使用这个输入流进行数据读取。DFSInputStream 会在其构造函数中通过 ClientProtocol 协议与 Datanode 通信，获取文件数据块的分布情况，对于文件中的每个数据块，Namenode 返回存储该数据块副本的数据节点地址。除此之外，HDFS 还支持对文件的多种操作，如创建文件、打开文件、读写文件、修改文件属性、删除文件等。

2.1.2MapReduce 计算框架

MapReduce 是一个分布式大数据计算框架，是 Hadoop 平台的核心组件，其实现和 Google 的 MapReduce 十分相似[17]。顾名思义，MapReduce 的核心思想是 map 和 reduce 操作，每个 map 操作处理一部分数据，并形成中间结果，然后由 reduce 操作将多个 map 操作的结果进行处理形成最终结果。本小节将从如下几个方面介绍 Hadoop MapReduce 框架：体系结构、编程模型、数据处理引擎和运行时环境。编程模型为用户提供统一的编程接口，编写应用程序时只需根据接口编写处理过程的 map 和 reduce 两个函数即可而无需关心其他繁琐细节。数据处理引擎为 MapReduce 作业的运行提供完整的运行流程。运行时环境为 MapReduce 作业的正常运行提供保障，作业本身无需关心作业如何被切分成任务，如何被调度执行，这些都由运行时环境处理。

MapReduce 则构建在 HDFS 之上，对存储在分布式文件系统中的数据进行计算。和 HDFS 一样，MapReduce 框架采用了 Master/Slave 架构[17]（主从结构），即一个充当 Master 角色的 JobTracker 和众多充当 Slave 角色的 TaskTracker，具体结构如图 2-2 所示。

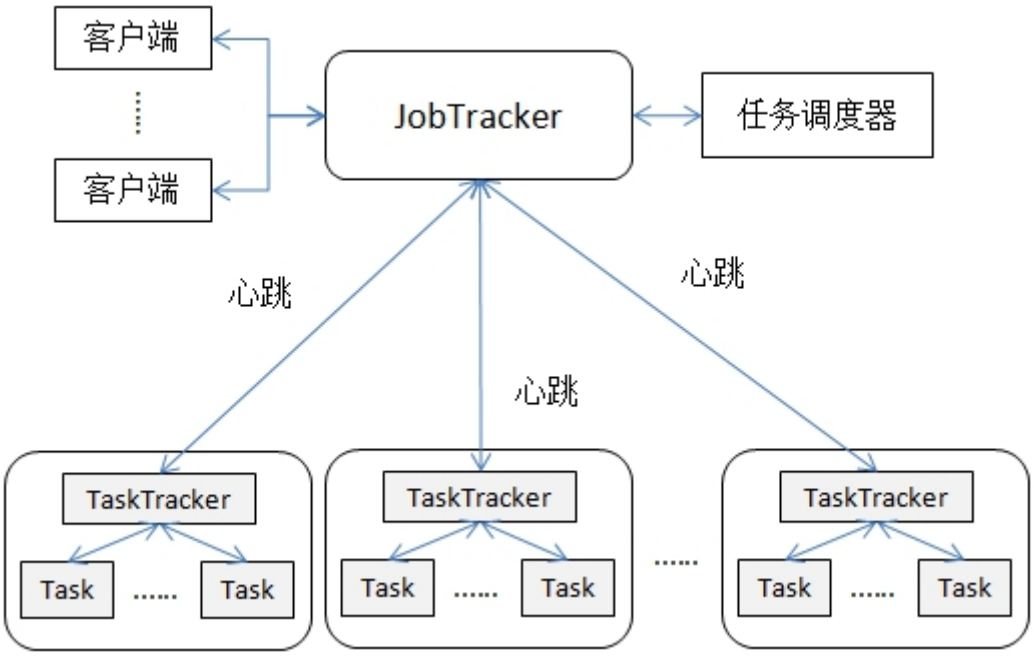


图 2-2 MapReduce 框架体系结构

从图 2-2 中可看出，MapReduce 架构由 Client、JobTracker 和 TaskTracker 三类实体组成，其作用和充当的角色分述如下：

（1）客户端。用户编写的 MapReduce 作业通过客户端提交至系统的控制中心 JobTracker，用户可以通过客户端实时查询已提交的作业的运行状态。

（2）JobTracker。JobTracker 被称为 MapReduce 计算框架的控制中心，负责维护作业信息和众多 TaskTracker 信息、监控系统资源和作业调度。JobTracker 会维护所有作业的信息，包括作业任务划分情况、作业处理进度等。同时，JobTracker 会监控集群所有资源并根据这些资源信息进行正确的作业调度。此外，JobTracker 还会对所有作业任务进行调度以将合适的任务分配到合适的 TaskTracker 上运行。

（3）TaskTracker。TaskTracker 是 MapReduce 框架中的计算执行实体。TaskTracker 通过“心跳”与 JobTracker 通信，报告其资源使用情况和任务执行情况，并接受相关命令操作，如结束某个任务、报告任务执行进度等。

此外，TaskScheduler 是 JobTracker 中的一个可插拔模块，专门负责进行作业调度，在搭建一个 Hadoop 集群时，系统管理员可根据要处理的作业特性选用不同的作业任务调度器。

2.2 MapReduce 作业处理流程

2.2.1MapReduce 运行时环境

MapReduce 框架为运行一个作业程序做了大量的准备工作，这些就构成了 MapReduce 运行时环境或工作机制，包括 MapReduce 作业的提交、初始化、任务分配和执行等过程。

MapReduce 应用程序中的 JobClient.runJob(conf)语句用于提交一个作业，这个简短的代码隐藏了 MapReduce 运行时环境中进行的大量处理细节，本小节将结合图 2-3 介绍 MapReduce 作业运行机制。MapReduce 作业运行环境由客户端、JobTracker、TaskTracker 和 HDFS 组成，他们相互配合完成作业的提交、初始化、调度和执行等过程。

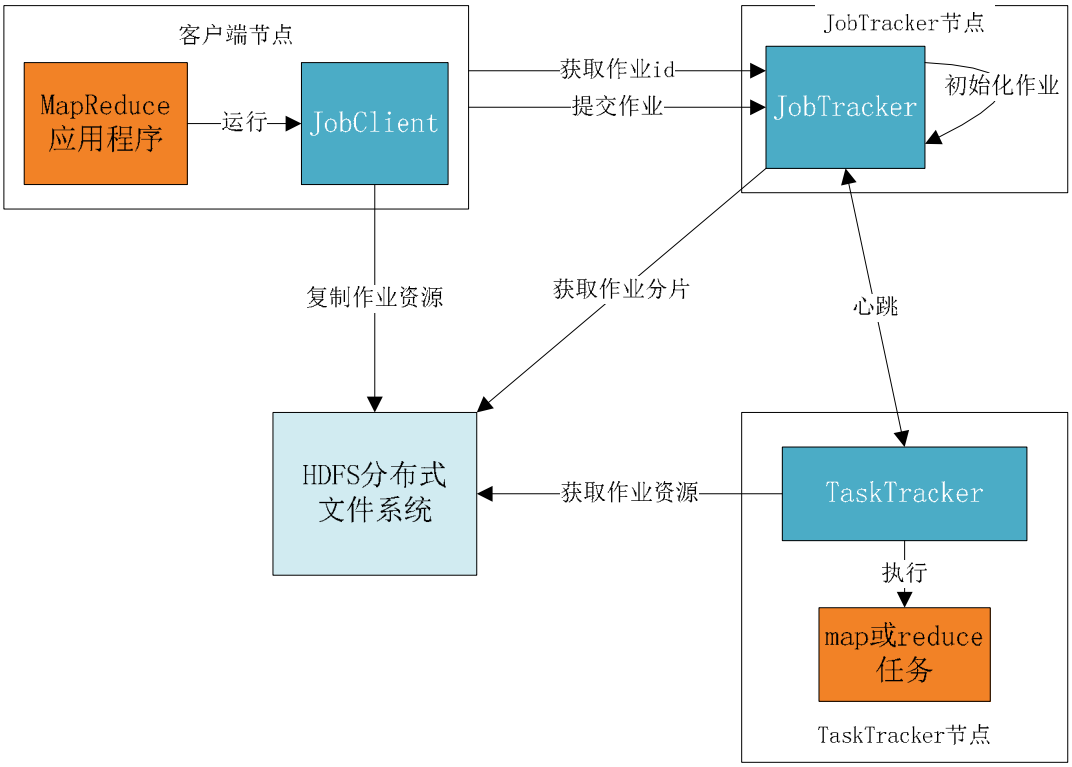


图 2-3 Hadoop 运行 MapReduce 作业的工作原理

（1）提交作业

客户端通过 runJob()方法来创建一个 JobClient 对象，并通过 JobClient 对象的 submitJob()方法向 JobTracker 提交一个作业。提交时每个作业拥有唯一的作业 id 并将相应的程序、数据、配置文件等提交至 HDFS 中存储。

（2）初始化作业

JobTracker 接收 JobClient 提交作业的请求后会将此调用请求放入一个内部队

列中，交由作业调度器进行调度并初始化。作业初始化时将创建一个JobInProgress 实例对象来封装任务和记录信息以便跟踪任务的状态。同时，调度器会获取 JobClient 提交的输入分片信息并为每个分片创建一个 map 任务，从而形成了作业的任务列表，最后交由调度器调度执行。

（3）调度作业

TaskTracker 通过“心跳”同 JobTracker 进行通信[14]，当 TaskTracker 准备执行一个新任务时通过心跳信息告知 JobTracker, JobTracker 的作业调度器模块会根据作业调度算法选择一个作业并从中选择一个任务交由 TaskTracker 执行。在当前的 Hadoop 版本中，任务分配是以 slot 为单位进行的，一个任务可能需要一个或多个 slot 的资源运行。map 任务的分配比较简单，reduce 任务的选择则需考虑数据本地性，理想情况是任务是数据本地化的（node locality），即任务和输入分片数据在同一节点上。

（4）执行作业

TaskTracker 被分配一个任务后就可以运行该任务了，有如下几个步骤：①通过 HDFS 把作业的 JAR 文件和所需配置、数据文件复制到本地文件系统；②为任务创建一个本地工作目录并将 JAR 文件内容解压到该文件目录下；③新建一个 TaskRunner 实例运行该任务。TaskRunner 启动一个新的 JVM 来运行每个任务，这样各个任务的运行相互之间没有任务影响。

2.2.2MapReduce 数据处理引擎

MapReduce 作业提交至 JobTracker 后经过调度到某个 TaskTracker 上执行，执行的过程由数据处理引擎驱动, 主要分为 map 和 reduce 两部分, 如图 2-4 所示。

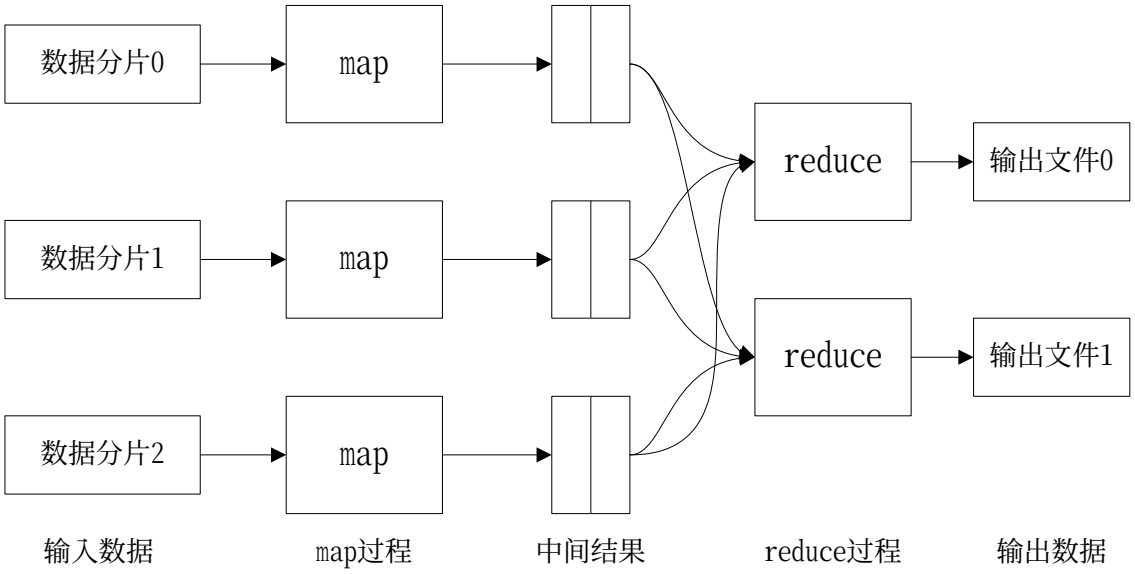


图 2-4 多个 reduce 任务时的 MapReduce 数据处理流程

如图 2-4 所示，我们可以将 MapReduce 过程分解为 5 个步骤：（1）map 任务输入数据的初始化；（2）map 计算处理；（3）将 map 任务处理结果存储到本地磁盘；（4）将 map 任务处理结果交给 reduce 任务进行 reduce 操作；（5）将 reduce 任务处理结果输出至 HDFS 中。

map 操作开始产生输出时，并不是简单地将结果写到磁盘，而是利用缓冲的方式先写到内存，并处于效率的考虑进行预排序。每个 map 任务都有一个环形内存缓冲区用于存储任务的输出，一旦缓冲区内容达到阈值，一个后台线程便开始将内容写到磁盘中。在写入磁盘过程中，map 输出将继续写到缓冲区，但是如果缓冲区已满，map 操作会阻塞直至缓冲区有空闲空间。

在写入磁盘前，处理线程会首先根据数据最终要传送到的 reducer 把数据划分成相应的分区，在分区中也会对数据进行内排序。如果设置了 combiner，并且溢出写次数至少为 3 次时，combiner 就会在输出文件写到磁盘之前运行，combiner 的作用是使 map 输出更加紧凑，数据更有效，但并不会影响到 map/reduce 的最终结果。

在 reduce 操作时，如果需要集群上若干个 map 任务的输出作为其特殊的分区文件，而每个 map 任务的完成时间可能不同，因此只要有一个 map 任务完成 reduce 任务就会开始复制 map 任务的输出结果。如果 map 任务的输出相当小，则会被复制到 reduce 任务的 TaskTracker 的内存，否则复制到磁盘。复制完所有 map 任务的输出后，reduce 任务开始排序并对排好序的最终文件进行 reduce 操作。

在 map 任务过程中，map 操作和 shuffle 过程十分重要，一个 MapReduce 作业的众多 map 任务具有相同的操作过程和数据格式，因此，可以通过这一过程来观察其对计算节点资源利用率的使用情况，如 CPU 密集型作业和 I/O 密集型作业在 map 阶段由不同的资源使用，由此可用于改进作业调度算法，将作业按资源需求分类并调度执行，具体见 3.3 节。

2.3 Hadoop 作业调度算法分析与改进

Hadoop 平台可供选择使用的传统调度算法包括 FIFO 调度算法、计算能力调度算法和公平调度算法，其中 FIFO 算法是系统默认的调度算法。Hadoop 以插件的形式调用作业调度器，允许用户设计实现满足特定需求的调度算法并应用到 Hadoop 系统中。

2.3.1 FIFO 调度算法分析

先进先出（FIFO，First In First Out）调度算法是 Hadoop 系统中最初设计采用也是最简单的作业调度算法，是系统默认的调度方法。先进先出调度算法与操作系统发展早期作业调度采用的单道批处理方法非常相似，进入系统中的所有作业按照时间和优先级排序并依次被调度执行。在执行过程中，作业占用系统全部计算资源直到运行完成。

在具体实现中，先进先出调度算法采用单一的全局队列来存放所有作业并将其按照作业优先级和进入系统的时间排序，作业的调度执行也采用先进先出的方式并且不支持作业抢占，前一个作业运行完成后一个作业才能使用开始执行。当 TaskTracker 有空闲的 slot 且请求 map 任务时，FIFO 调度器查看队列首部的作业是否用未执行的 map 任务，有则分配给该 TaskTracker，否则才会去检查队列首部作业后面的作业的 map 任务。如果 TaskTracker 请求 reduce 任务，调度器同样会先检查队列首部的作业，仅当有空闲 reduce slot 时才会考虑队列后面作业的 reduce 任务。

FIFO 调度算法思想与设计都比较简单，也易于实现，并且对于整个 Hadoop 系统来说，FIFO 调度算法的运行开销非常小，对系统性能的影响微乎其微。因此，在最初处理单用单一类型的大作业时，FIFO 调度算法十分合适。

但是，随着用户对处理的作业的类型多样化，以及多用户提交作业，多种多样的作业性质迥异、资源需求不同，运行时长差异大等，使得整个 Hadoop 系统在多用户多作业场景下使用 FIFO 调度算法时运行效率比较低下，甚至有时会严重影响其他作业的运行[15]。

2.3.2 公平份额调度算法分析与改进

公平调度算法的核心思想是尽可能地保证系统中的所有作业都能够获得等量的资源份额[15]。它设置多个资源池（和计算能力调度算法中的作业队列类似），每个资源池具有一个理论上应该获得的资源量（即公平共享量）和实际上资源池中有作业运行时获得最小资源量（最小共享量）。公平调度算法通过最大最小公平方法[15]来进行资源分配。最小共享量保障了资源池在有作业运行时拥有一定的资源，公平共享量是资源池在获得最小共享额度的基础上其他资源池还有资源空闲时可以获得的资源量（最小共享量小于等于公平共享量）。这样，各个资源池随着时间的推移将得到越来越公平的资源。

公平调度算法也采用三级调度策略：①选择资源池，当存在资源使用量小于

最小资源量的资源池时，优先选择资源使用率最低的资源池，否则选择任务权重比最小的资源池，由计算公式 3-1 确定；②选择作业和任务，当确定作业资源池后，调度器在该资源池中选择一个作业和任务，算法描述如图 2-5 所示。

算法 2 公平调度算法
输入：计算节点状态信息 输出：要调度执行的作业任务

for 资源池集合中的每个资源池
计算当前资源池中的可运行和正在运行的 Map 任务和 Reduce 任务
计算集群中总的 map 和 Reduce 的 slot 数量
将各个作业池按照公平算法进行排序
end for
for 可调度资源池集合
对当前资源池中的作业按照 FIFO 或公平份额排序
for 当前资源池
if 需要调度 Map 任务
根据数据本地性和延迟调度机制从作业 Map 任务集合中
选择一个任务
将该任务所属作业添加到已启动作业列表
更新作业中已调度 Map 任务、正在运行 Map 任务数
将选择的任务放入任务执行集合中
else if 需要调度 Reduce 任务
从作业 Reduce 任务集合中选择一个任务
将该任务所属作业添加到已启动作业列表
更新作业中已调度 Reduce 任务、正在运行 Reduce 任务数
将选择的任务放入任务执行集合中
end if
end for
end for
返回要运行的任务

图 2-5 公平调度算法

从算法描述和流程图可以看出，复杂度和计算能力算法一样为 $O(n+m+t)$，其中n表示资源池的数量，m表示系统中所有作业的数量，t表示所有任务的数量。在选择可调度资源池时有些资源池达不到要求而被放弃，所以实际复杂度要小于理论值。由于在作业任务选择的方法有区别，因此具体的n、m、t值不同，一般均会小于理论值。

在公平份额调度算法中，各资源池的 slot 分配十分重要，它决定着各资源池拥有的计算资源量。现有的公平份额调度算法中的 slot 分配方案采用了最大最小

公平算法[19]。每个资源池均有 m_i 和 d_i 两个参数，m_i 表示资源池的最小份额，d_i 两表示资源池的资源需求量，则资源池 slot 分配过程如下：

（1）如果 m_i 大于 di，则分配 di 的 slot 给该资源池

（2）将剩余的 slot 资源分配给各个资源池，满足各资源池的最小份额 mi

（3）将所有 mi 小于 di 的资源池按照缺额量从大到小排序，并将仍有剩余的 slot 资源依次分配给各缺额资源池。

分配示例如图 2-6 所示。

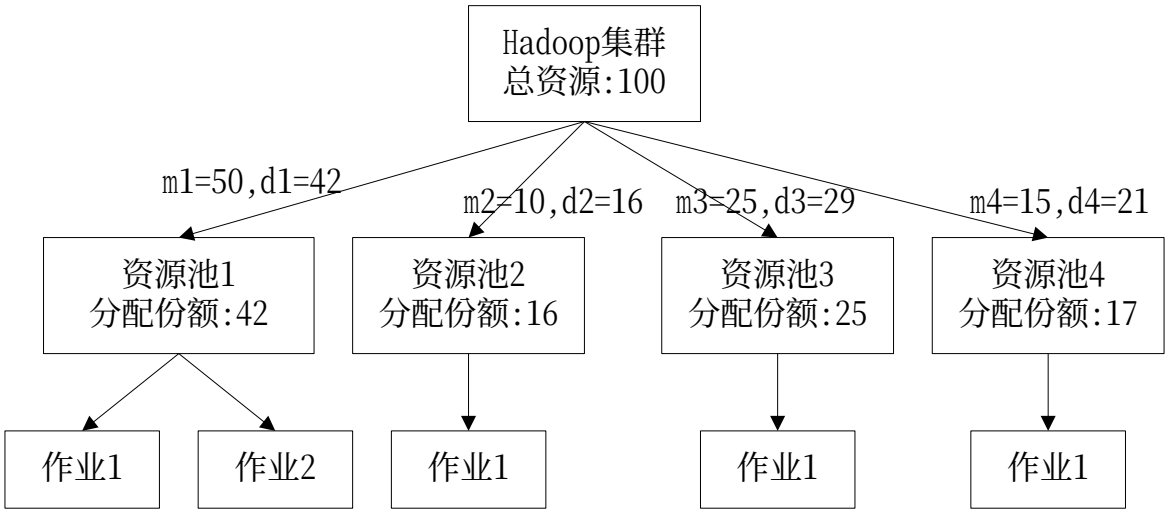


图 2-6 公平份额调度算法中的 slot 资源分配示例

此处，slot资源分配方法仍有不足之处，例如当资源缺额差别不大时，该方法将资源优先满足了缺额最大的几个资源池，而缺额相差不多的资源池却没有得到一定的资源，由此考虑更为公平的方法，将剩余资源尽量均分给各资源池，改进后的slot资源分配算法如图 2-7 所示。

公平调度算法中 slot 分配方案改进

输入：未分配 slot 资源的资源池集合，未分配的资源量

输出：分配 slot 资源后资源池集合

```
SA = S   //位被分配的资源池集合
SB = Φ //已被分配资源的资源池集合
M = F   //未被分配的资源
for SA 中的每个资源池 i   //如果需求量小于最小份额，则满足需求量
    if di < mi
        fi = di; M -= di;
        从 SA 中去除资源池 i 并将其加入到 SB 集合中
    end if
end for
for SA 集合中的每个资源池 i //对剩下的需求量大于最小份额的，满足最小份额
    fi = mi; M -= mi;
end for
while((SA≠Φ) and (M>0)    //新增资源池缺额公平补充
    Δs = M 除以 SA 中所有资源池的缺额并取整
    for SA 集合中的每个资源池 i
        资源池 i 增加获得的资源Δs * 缺额取整
        M -= Δs * 缺额取整
    end for
end while
while((SA 中仍有资源池有缺额) and (M>0)) //将取整后剩余的再补充给缺
                                          额多的资源池
    SA 中资源池按缺额大小排序并按缺额从大到小满足
end while
```

图 2-7 改进后的公平调度算法中的slot资源分配算法

按照图 2-20 所示改进后的 slot 资源分配方法对图 2-19 所示的示例重新实验，会发现差额差不多的各资源池获得了相对更为公平的 slot 资源量，如图 2-21 所示。原来的 8 个 slot 资源首先满足了 2 号资源池的 6 个缺额，然后满足了 4 号资源池的 6 个缺额中的 2 个，而 3 号资源池的缺额没有任何补充。改进后的方法满足了 2 号资源池 6 个缺额中的 3 个，3 号资源池 4 个缺额中的 2 个和 4 号资源池 6 个缺额中的 3 个，这样，使得缺额补充更为公平了。

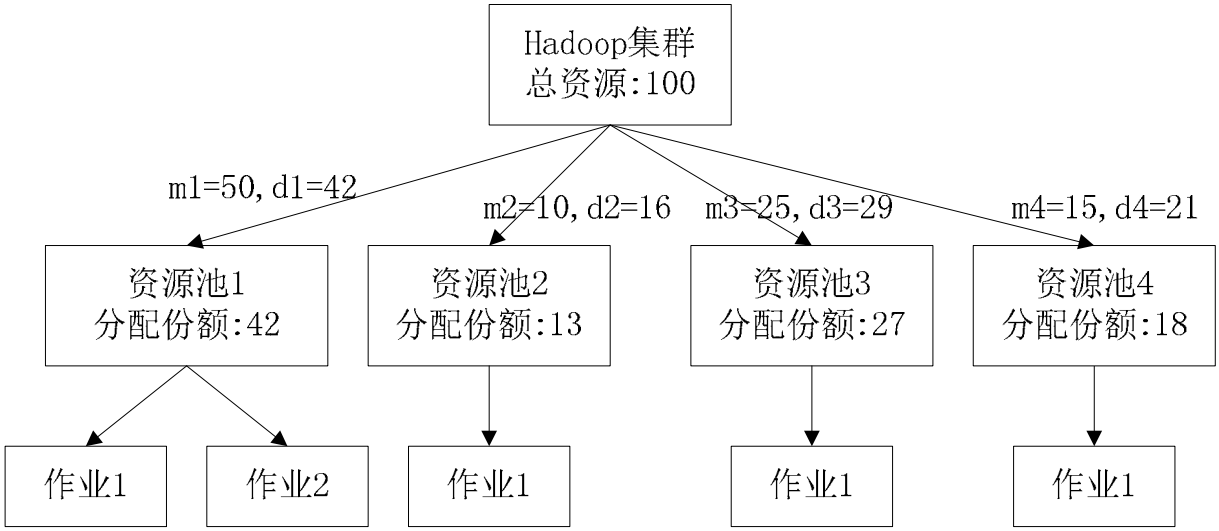


图 2-21 改进后的公平份额调度算法中的slot资源分配示例

公平算法调度器启动时首先加载 fair-scheduler.xml 配置文件进行初始化，然后等待作业提交并定时获取更新资源池和作业信息。使用公平调度算法可以通过配置文件中的配置项改变调度策略，其中重要参数及含义说明如表 2-5 所示。

表 2-5 公平调度算法重要配置参数及含义说明

*.fairscheduler.preemption	Fair Scheduler 是否支持资源抢占，默认为 false，表示不支持
*.fairscheduler.assignmultiple	是否在 TaskTracker 一次心跳中同时分配 Map 任务和 Reduce 任务
*.fairscheduler.assignmultiple.maps	TaskTracker 一次心跳中最多分配的 Map 任务数, -1 表示不限制
*.fairscheduler.assignmultiple.reduces	TaskTracker 一次心跳中最多分配的 Reduce 任务数，-1 表示不限制
*.fairscheduler.locality.delay.rack	当等待一个满足 rack 本地资源作业最长等待时间
*.fairscheduler.weightadjuster	用户可配置的作业分配资源权重调整器
minMaps	资源池中 map 资源的最小量
maxMaps	资源池中 map 资源的最大量
minReduces	资源池中 reduce 资源的最小量
maxReduces	资源池中 reduce 资源的最大量
maxRunningJobs	资源池中同时运行的作业最大

	数
schedulingMode	资源池中作业调度模式，可以选择 FIFO 或者 Fair 策略

公平调度算法采用作业池作为集群资源主要分配单位，保障了每个用户作业的并发执行。公平份额调度算法支持作业任务在未获得较好数据本地性情况下进行延迟调度，使作业获得较好数据本地性，但一定程度上牺牲了作业响应时间性能[21-22]。公平份额调度算法在使用时需要手动配置资源池参数（如表 2-6 所示），这在大集群环境下难以做到正确配置。

2.3.3 计算能力调度算法分析

计算能力调度算法是雅虎公司贡献的，使得多用户的作业可以共享 Hadoop 集群计算资源。计算能力调度算法的核心思想是预先设置多个具有一定计算资源的队列（一般是一个用户或用户组一个队列），将每个进入 Hadoop 系统的作业分配到相应的队列中去。当某个队列中有剩余的计算资源时可以被其他队列暂时共享，当原队列重新需要一些计算资源时，曾暂时被共享的计算资源将在完成其当前任务后重新回到原所属队列。这样，每个队列都能尽可能地提供资源运行其中的作业，从而整个集群能尽可能地完成各种各样的作业运行，从而实现运行多用户多作业的目标。

在计算能力调度算法中，每个队列预先就设置有一定的计算资源，这些计算资源的量是根据 Hadoop 集群运行多种作业的配置计算出来的。例如，计算密集型作业队列所拥有的 CPU 资源相对较多，非计算密集型作业队列所拥有的 CPU 资源相对较少；按照设定的优先级高的作业队列拥有的计算资源相对较多，低优先级的作业队列拥有的计算资源相对较少等。

计算能力调度算法支持各作业队列间多余计算资源的共享[14]。当某一队列中的作业运行时，如果其计算资源足够并且有余量，那么这些余量的计算资源会被所有其他队列共享使用。已经被共享使用的计算资源所属的队列中增加新的作业而需要增加计算资源时，这些被共享的资源会在其完成当前任务后，重新回到所属的队列以供新的作业运行使用，队列间资源共享与回归机制如图 2-7 所示。

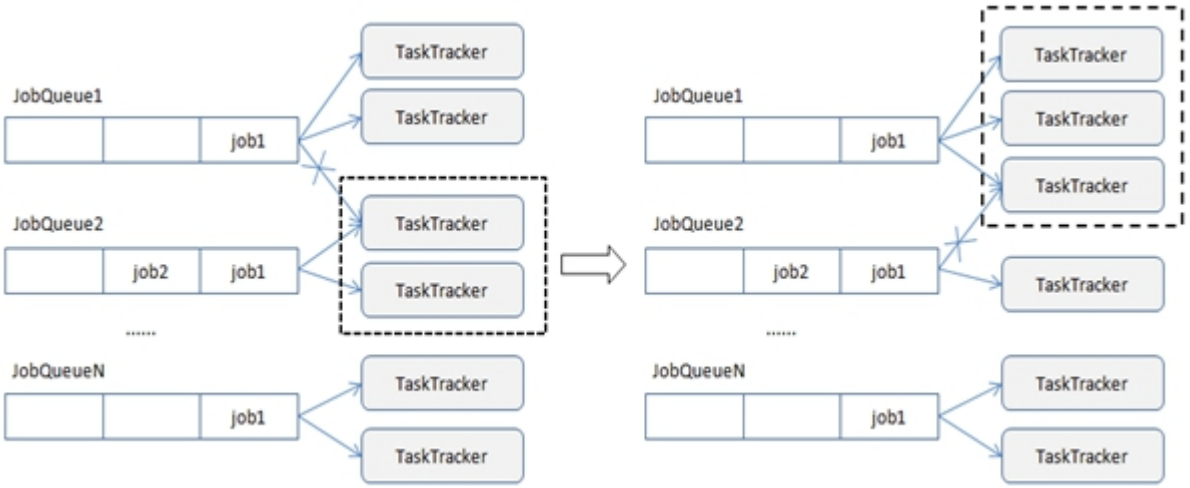


图 2-7 队列间计算资源共享与回归

当有 TaskTracker 请求任务时，计算能力调度算法从所有作业队列中选出一个作业的任务去请求的 TaskTracker 上运行。首先，Capacity Scheduler 更新各个队列信息，包括队列资源容量、资源使用上限、正在运行的任务和已经被使用的资源等；然后，根据队列的这些信息选择队列、作业和任务。(1) 选择队列，计算能力调度器总是优先选择资源使用率最低的队列，这样队列中的作业有充足的资源去执行任务。(2) 选择作业和任务。和初始化作业排序策略一样，如果支持优先级则按照优先级排序来选择作业，否则按照 FIFO 顺序进行。当选择任务时，调度器会依次遍历排好序的作业，并检查当前 TaskTracker 剩余资源是否足以运行当前任务，如果满足则从该作业中选择一个任务添加到已分配任务列表中，具体流程如图 2-8 所示。

计算能力调度算法
输入：计算节点状态信息 输出：要调度执行的作业任务

按照每个队列的 capacity 情况并从高到低将各个队列进行排序
for 队列集合中的每个队列&该队列未达到资源使用上限{
按照时间和优先级为队列中的所有作业进行排序
for 队列中的每个作业{
if taskTracker 能满足当前作业中一个任务的运行
if taskTracker 需要 Map 任务且当前作业有符合的 Map 任务
按照数据本地性原则选择一个 Map 任务
更新作业中已调度 Map 任务、正在运行 Map 任务数
将该任务放入任务执行集合中
else if 调度 Reduce 任务且当前作业有符合的 Reduce 任务
按照遍历的先后顺序选择一个 Reduce 任务
更新作业中已调度 Map 任务、正在运行 Map 任务数
将该任务放入任务执行集合中
end if
end if
end for
end for
返回要运行的任务

图 2-8 计算能力调度算法

由算法描述可知，计算能力调度算法的复杂度为$O(n+m+t)$，其中 n 表示队列个数，m 表示系统中所有作业的数量，m 表示所有任务的数量。由于有些队列已经超量使用资源，在选择队列时直接放弃，因此实际复杂度要小于理论值。

计算能力调度器实现了计算能力调度算法，它作为独立的模块进行开发并作为可插播的模块给 JobTracker 使用。为在 Hadoop 系统中使用 Capacity Scheduler，需要进行相应的配置，将 mapred.xml 文件中的参数 taskScheduler 设置为 org.apache.hadoop.mapred.CapacityTaskScheduler 即可，这样 JobTracker 启动时会加载 capacity scheduler 模块。计算能力调度器运行后加载 capacity-scheduler.xml 配置文件进行初始化并开始提供调度服务。使用计算能力调度算法可以通过配置文件中的配置项改变调度策略，其中重要参数及含义说明如表 2-4 所示。

表 2-4 计算能力调度算法重要配置参数及含义说明

配置参数名	说明
mapred.capacity-scheduler.maximum-system-jobs	Capacity Scheduler 当前允许的最大能被初始化的作业数目
mapred.capacity-scheduler.queue.X.capacity	队列 X 所拥有的资源量
mapred.capacity-scheduler.queue.X. maximum-capacity	队列 X 能够拥有的最大资源量
mapred.capacity-scheduler.queue.X. supports-priority	队列 X 是否支持作业优先级，如果是，则队列中作业按优先级排序，否则按提交顺序排序
mapred.capacity-scheduler.queue.X. minimum-user-limit-percent	队列 X 中某用户可使用资源的最小限制值
mapred.capacity-scheduler.queue.X. user-limit-factor	队列 X 中资源量的倍数，可通过设置多倍的资源供用户使用
mapred.capacity-scheduler.queue.X. maximum-initialized-active-tasks	队列 X 中所有作业的任务数总和限制
mapred.capacity-scheduler.queue.X. maximum-initialized-active-tasks-per-user	队列 X 中某用户作业任务的总和限制

计算能力调度算法为每个队列分配一定的计算资源保障每个队列中作业的运行，特别是那些交互型短作业能够得到及时的调度运行。克服了 FIFO 调度算法在处理交互型短作业时集群资源利用率低的缺点。计算能力调度算法支持多用户多类型作业的并发执行，既为大作业提供了资源运行又有效防止大作业独占集群资源，同时为小作业的运行提供计算资源，从而在各类型作业综合执行时提高了系统吞吐率。但是计算能力调度算法在使用时需要手动配置作业队列参数信息（如表 2-5 所示），Chauhan[18]等人对计算能力调度算法重要配置参数进行了研究，认为这些参数配置信息设置对 MapReduce 作业运行性能有较大的影响，因此做到正确配置并不容易，在大规模 Hadoop 集群中使用时需要管理员丰富的经验并熟悉应用场景。

2.3.4Hadoop 现有作业调度算法对比分析

由于 FIFO 调度算法的特点，其非常适合调度大作业，大作业拥有众多 map 和 reduce 任务，作业运行时可充分利用 Hadoop 集群资源。但是 FIFO 调度算法适用于大作业的运行，使得交互型短作业在提交后将长时间得不到调度执行，严重影响交互体验，这是其重要缺点。但对于计算能力调度算法和公平份额调度算法则不会出现这个问题，因此，按照是否支持多用户多类型作业运行来划分，可以将 FIFO 调度算法和计算能力和公平份额调度算法区分开。

计算能力调度算法和公平份额调度算法均克服了 FIFO 调度算法这一缺点，但他们之间仍有不同特点，例如计算能力调度算法的设计思想是将资源按比例分配给各队列并防止个别队列独占资源，而公平份额调度算法是基于最大最小公平算法将集群资源分配给各个资源池或用户，他们之间的特点对比如表 2-6 所示。

表 2-6 计算能力和公平份额调度算法特点对比

	多用户共享	最小资源保障	作业内存约束	空闲资源共享
计算能力调度算法	支持	不支持	无	支持
公平份额调度算法	支持	支持	有	支持

通过前面对 FIFO 调度算法、计算能力调度算法和公平份额调度算法的深入分析，本文总结他们的优缺点如表 2-7 所示。

表 2-7 FIFO-计算能力-公平调度算法优缺点

	优点	缺点
FIFO 调度算法	算法简单、易于实现，适合大量大作业的应用场景使用	对多用户多类型作业的调度缺乏考虑，尤其影响交互型短作业的运行
计算能力调度算法	考虑多用户多类型作业的资源需求且作业响应时间相对更少，适合各类型作业并发执行，共享集群计算资源	需手动配置作业队列参数，难以做到正确配置

公平调度算法	充分考虑多用户多类型作业的资源需求，在集群资源利用率和用户隔离方面获得较好平衡	作业响应时间一般，并且需手动配置作业资源池参数，难以做到正确配置

2.4 本章小结

本章介绍并分析了 Hadoop 平台主要相关技术，包括 Hadoop 分布式文件系统、MapReduce 计算框架，并详细分析了 MapReduce 作业执行流程与 map 和 reduce 操作处理过程，并在此基础上分析了 Hadoop 现有调度算法并改进了相关算法，主要内容如下：

（1）分析 MapReduce 作业中的 map 和 reduce 操作过程，并分析操作过程中各步骤间的关系；

（2）分析先进先出、公平份额、计算能力调度算法，并给出了各算法对比分析与优缺点，并分析公平份额调度算法中的 slot 资源分配方法与不足，对其进行了改进，使得有缺额的资源池间的 slot 资源补充更为公平合理；

第三章 基于分类的 Hadoop 作业调度算法

本章首先分析现有 Hadoop 作业调度算法需要预先配置计算节点参数以及资源利用率不高的不足，深入分析预先配置的作用并为尽量减少预先配置方面做出改进，在此基础上提出了基于贝叶斯分类的作业调度算法，并依次给出了算法思想、理论推导、算法流程等。然后提出基于作业资源需求类型的分类预处理方法，其将作业分类为 CPU 密集型和 I/O 密集型并分类进行调度，提高计算节点中各类资源的利用率。3.1 节分析了 Hadoop 现有作业调度算法的不足；3.2 节介绍了基于分类的 Hadoop 作业调度算法的产生背景、算法思想、理论推导过程，调度算法的描述等；3.3 节提出基于作业资源需求类型的分类预处理方法；3.4 节对本章进行小结。

3.1 Hadoop 传统作业调度算法的不足

Hadoop 的资源管理模块包括两部分内容，一个是整个 Hadoop 集群资源的表示，一个是 Hadoop 集群资源的分配，这两部分是紧密耦合的，即 Hadoop 系统的资源分配方式依赖于其资源的表示方式。Hadoop 系统中采用的各种作业调度算法就是其资源分配方式。在 Hadoop 0.20.2 版本中，资源的表示采用的是基于 slot 的，现有的 FIFO 调度算法、计算能力调度算法、公平调度算法都是基于 slot 表示模型的[22]。

对于一个处理多用户多类型作业的 Hadoop 系统的作业调度算法，有两点非常重要的目标：①在众多作业中选择一个合适的作业并将其任务分配到合适的 TaskTracker 上执行；②合理调配计算资源合理分配任务以达到集群资源的最大利用率。H.Herodotou 等人[23]和 G.Wang 等人[24]经过研究认为 MapReduce 作业运行性能和集群的配置、作业配置和输入数据紧密相关。但在现有的 Hadoop 调度算法中，正确合理的配置并不容易做到。

在现有 Hadoop 调度算法中，要做到调度一个合适的作业任务到合适的 TaskTracker 执行，需要进行一些预先配置，例如配置 mapred.xml 和特定调度算法的配置文件。首先，为使用现有调度算法，Hadoop 系统管理员在启动前需要对所有 TaskTracker 上可以运行的最大任务数目进行正确配置，如果最大任务数目配置过大，那么运行过程中通过调度，TaskTracker 上运行的任务数就增多，从而每个任务所获得计算资源变小，容易导致任务运行时过载而严重影响作业的完成；如果最大任务数目配置过小，那么相反会导致每个任务获得计算资源过量，从而浪费系统资源。但是正确配置 TaskTracker 上的最大任务数量需要系统管理

员对作业的运行和资源使用非常地情况明白是比较困难的, 尤其是在大量多类型作业的情况下更是如此。

同时，现有的 Hadoop 作业调度算法，如 FIFO、计算能力、公平份额调度算法通过不同集群资源分配方式保证作业拥有一定的计算资源执行。但是，他们都没有充分考虑不同类型作业在集群中运行时对计算资源的需求和对集群资源利用率的影响，因为像 CPU 密集型和 I/O 密集型这样不同类型的作业具有不同的资源负载。不考虑这些不同类型作业的工作负载情况可能会对系统的资源利用率产生较大的影响，而 CPU 密集型作业和 I/O 密集型作业的运行需要计算节点上的不同资源，他们的并行运行可以更为充分的利用集群计算资源。

3.2 基于分类的 Hadoop 作业调度算法

为了克服现有调度算法正确配置不容易做到这一不足, 可以尽量减少或消除预先配置对 MapReduce 作业调度执行时的影响。首先深入分析预先配置的使用场景和目的, 发现预先配置能够是每个作业任务得到足够合理的计算资源去运行, 同时也尽可能地是 Hadoop 集群资源得到较合理的使用（资源利用率最大化）。在理想情况下, 合理的预先配置会得到理想的结果, 但是在实际情况中难以做到, 特别是多种类型作业共享集群计算资源时。

基于 Hadoop 集群中 slot 资源表示，本文提出一种基于贝叶斯分类的作业调度算法，它以作业任务实际资源需求（CPU、内存、磁盘容量等）多维度资源表示, 同时不需要在计算节点上预先配置, 而是通过贝叶斯分类方法不断学习训练, 最后将作业任务分配到最合适其运行的 TaskTracker 上执行的算法。该算法消除了原先为作业调度而在 TaskTracker 进行的各种配置，而是通过一个基于贝叶斯分类的学阶段来替代预先配置，这样不仅免去了大量额外的工作，而且还使得各个作业任务获得了更为合适的资源去执行。经过一定学习，作业调度器可以使用该分类结果来预测作业任务类型以便更好地调度。

3.2.1 算法思想

Tian 等人[18]提出了作业任务的特征信息可以从其历史信息中获取得到并假设 MapReduce 作业中的任务特征近似, 特别是 Map 任务, 并且得到实验的验证。由于同一作业中的各个任务近似, 作业中先运行的任务的特征就可以作为未运行的任务的特征，先运行的任务的执行情况也可以为未执行的任务提供参考，如哪些条件下任务可以成功执行，哪些条件下任务任务执行失败的可能性大。

基于分类的作业调度算法的核心思想就是基于这一假设, 通过分析作业任务

的执行历史信息，将同一作业的后续其他任务进行分类，为其调度到合适的TaskTracker 上执行。

分类的类别可以依据不同的标准确定，依据作业能否在某个 TaskTracker 上被调度执行可以将作业分类为可调度和不可调度两类；依据作业运行时需要的资源特性可以将作业分类为 CPU 密集型和 I/O 密集型两类；依据作业运行时长可以将作业分类为短作业、一般长度作业和长作业三类。不同的分类方法产生不同的分类结果，可以根据这些分类结果设计作业调度算法。本文采用是否可被调度执行分类以及 CPU 密集型和 I/O 密集型这两种分类，它们均可使用基于概率计算的贝叶斯分类方法。本小节的讨论采用贝叶斯分类方法和是否可被调度执行的分类类型，CPU 密集型和 I/O 密集型的分类将在后面介绍。

当 TaskTracker 向 JobTracker 发送心跳请求为其分配一个作业运行时，JobTracker 通过该调度算法遍历作业队列，标识出每个可调度和不可调度作业，然后再从所有可调度作业中选择一个作业并从其中选择一个任务分配给TaskTracker 去执行。当作业任务在 TaskTracker 上运行完成后，通过心跳信息报告该次任务分配运行的结果来影响后面的作业调度，这是一个不断学习的过程：分类->调度->反馈结果->分类->调度->反馈...。基于分类的调度算法通过不断学习每次任务分配决策对 TaskTracker 资源造成的结果反馈，来不断地影响及改善以后的作业分类，然后通过作业调度达到一种良好的调度执行，即每次都能为请求作业任务的 TaskTracker 选择最合适在其上运行的作业任务，这样就提高了整个Hadoop 集群的资源利用率。

TaskTracker 能够为作业任务提供必要的计算资源是作业任务能够正确运行完成的重要因素。为了能够选择一个最为合适某个 TaskTracker 运行的作业任务，需要将作业任务运行需要的计算资源量和 TaskTracker 上具备的计算资源量进行对比判断，这些量就是前面所述的 CPU、磁盘 I/O、内存容量、磁盘容量、网络 I/O 等，本文将这些变量称为节点特征变量，作业任务的资源需求称为作业特征变量，例如作业任务的平均 CPU 使用率和最大内存需求量，TaskTracker 拥有的资源情况称为节点特征，例如 TaskTracker 上 CPU 的使用率、物理内存大小、物理内存剩余量、磁盘空间大小、磁盘空间剩余量等，这些量统称为特征变量。

作业特征描述了作业任务的运行对资源的使用需求量。这一类型的变量的值可以通过统计作业执行的历史信息得到或者由用户评估并在提交作业时给出。变量值以 1 为下限，表示对该项资源的最小使用量，以 100 为上限，表示第该项资源的最大使用量。分析 MapReduce 作业的特征，我们选择作业的如下几项资源作为作业特征变量：平均 CPU 使用率、最大内存使用量、最大磁盘使用量。它们分别描述了作业对 CPU、内存、磁盘等资源的使用需求，如果特征值越高，表

明作业任务对资源的需求量越大，特征值越低，表明其对资源的需求量越低。

节点特征描述了 Hadoop 集群中某个 TaskTracker 上的资源状况，分为静态变量和动态变量两类，如处理器个数、处理速度、磁盘空间大小、内存大小等属于静态变量，CPU 使用率、空闲内存容量、空闲虚存容量、空闲磁盘容量等属于动态变量。为方便与作业特征进行比较，我们选择的节点特征变量如下：空闲 CPU 使用率、空闲内存容量、空闲虚存容量、空闲磁盘容量。它们描述了整个 TaskTracker 拥有的计算资源，如果节点特征值越高，表明该 TaskTracker 节点拥有的计算资源越多，相反，节点特征值越低，表明其拥有的资源量越少。

不同的作业特征和节点特征组合会导致作业需求的不同满足结果，有些情况下作业需求得到满足并且不会导致 TaskTracker 节点过载（过载是指 TaskTracker 节点不能满足作业任务的资源需求），而另一些情况下作业的需求得到满足或者会导致 TaskTracker 节点过载。这样，所有这些过载临界值的特征变量的集合组成了一个超平面，可以使用一个线性分类器以该超平面作为决策面划分作业。我们用 C1、C2……Cn 表示作业特征和节点特征的集合，原则上他们任意配合均可，满足独立假设条件，符合贝叶斯对条件变量独立假设的要求，同时，通过分析同一作业中任务执行历史信息可以得到特征变量的在任务执行成功失败时的先验概率。因此，本文选择贝叶斯分类方法作为具体的分类方法来构造作业分类器和任务调度器。

3.2.2 基于贝叶斯分类的调度算法理论推导

本小节的讨论基础是将所有作业划分为可调度和不可调度两类，并且以作业任务资源需求条件下 TaskTracker 是否会过载为衡量标准。特征变量取值使得 TaskTracker 过载的作业称之为可调度的作业，否则称之为不可调度作业。这样可调度作业和不可调度作业的判断就可以使用贝叶斯公式[22]描述如下：

$$P(\tau_i = \text{可调度} \mid C_1, C_2, \ldots\ldots C_n) \tag{3.1}$$

和

$$P(\tau_i = \text{不可调度} \mid C_1, C_2, \ldots\ldots C_n) \tag{3.2}$$

公式 3.1 表示作业 i 在特征变量 C_1、C_2……C_n 在某取值情况下为可调度作业的概率，而公式 3.2 则表示其为不可调度作业的概率。一个作业是否为可调度作业取决于公式 3.1 和 3.2 的结果大小，即若某作业在特征变量某取值条件下为可调度作业，那么说明公式 3.1 计算结果大于公式 3.2 计算结果。

这样我们就将作业调度问题转化为公式 3.1 和 3.2 的计算问题，应用贝叶斯

理论，公式 3.1 推导如下所示：

$$P(\tau_i = 可调度 \mid C_1, C_2, \ldots\ldots C_n) = \frac{P(C_1, C_2, \ldots\ldots C_n \mid \tau_i = 可调度)P(\tau_i = 可调度)}{P(C_1, C_2, \ldots\ldots C_n)} \tag{3.3}$$

由于$C_1, C_2, \ldots\ldots C_n$满足条件独立假设，故而公式 3.4 成立：

$$P(C_1, C_2, \ldots\ldots C_n \mid \tau_i = 可调度) = \prod_{j=1}^{n} P(C_1 \mid \tau_i = 可调度) \tag{3.4}$$

将公式 3.4 代入公式 3.3 中，得公式 3.5、3.6 如下：

$$P(\tau_i = 可调度 \mid C_1, C_2, \ldots\ldots C_n) = \frac{P(\tau_i = 可调度)\prod_{j=1}^{n} P(C_1 \mid \tau_i = 可调度)}{P(C_1, C_2, \ldots\ldots C_n)} \tag{3.5}$$

$$P(\tau_i = 不可调度 \mid C_1, C_2, \ldots\ldots C_n) = \frac{P(\tau_i = 不可调度)\prod_{j=1}^{n} P(C_1 \mid \tau_i = 不可调度)}{P(C_1, C_2, \ldots\ldots C_n)} \tag{3.6}$$

公式 3.5 为作业 i 在特征条件$C_1, C_2, \ldots\ldots C_n$时为可调度作业的概率计算公式，公式 3.6 为作业 i 在特征条件$C_1, C_2, \ldots\ldots C_n$时为不可调度作业的概率计算公式。由于要比较公式 3.5 和 3.6 在同一条件$C_1, C_2, \ldots\ldots C_n$下的大小，因此在计算时$P(C_1, C_2, \ldots\ldots C_n)$可约去，则公式 3.5 和 3.6 分别转化为公式 3.7 和 3.8：

$$P(\tau_i = 可调度 \mid C_1, C_2, \ldots\ldots C_n) = P(\tau_i = 可调度)\prod\prod_{j=1}^{n} P(C_j \mid \tau_i = 可调度) \tag{3.7}$$

$$P(\tau_i = 不可调度 \mid C_1, C_2, \ldots\ldots C_n) = P(\tau_i = 不可调度)\prod\prod_{j=1}^{n} P(C_j \mid \tau_i = 不可调度) \tag{3.8}$$

其中，$P(\tau_i = 可调度)$表示作业 i 为可调度作业的概率，$P(\tau_i = 不可调度)$表示作业 i 为不可调度作业的概率，$P(C_j \mid \tau_i = 可调度)$表示作业 i 为可调度作业时条件 C_j 取某值时的概率，$P(C_j \mid \tau_i = 不可调度)$表示作业 i 为不可调度作业时条件 C_j 取某值时的概率，这些概率都是先验概率，它们的值随着作业任务分配执行结果而变化，即作业任务被分配到某 TaskTracker 上运行后如果信息表明过载了则表示改作业任务不可调度作业，否则被认为是可调度作业，在此基础上通过统计信息计算这些先验概率。本文采用直方图来统计各特征变量的先验概率，当所有这些先验概率都计算出后可以将其代入公式 3.7 和 3.8，然后比较他们的大小，从而判断判断作业是否为可调度作业。

在计算$P(\tau_i = 可调度)$等这些先验概率时，TaskTracker 是否过载至关重要，过载则表示作业任务分配运行不合理，不应该调度此次作业任务，从而该作业应该

被认为是不可调度的作业。任务分配运行时，判断其是否导致 TaskTracker 过载的判断规则称为过载规则。要判断是否过载，必须让 TaskTracker 收集该作业任务运行时的相关信息并将这些信息放在心跳中传送至 JobTracker 进行判断，如 JobTracker 通过 TaskTracker 传送来的 TaskTracker CPU 使用率、内存使用量、磁盘空间使用量等来与作业需求量进行对比判断。过载规则对于基于贝叶斯分类的调度算法十分重要，其直接决定了调度算法的分类结果与作业任务调度，本文采用贝叶斯分类中的多项条件的综合作为过载规则。

在贝叶斯分类决策中，过载规则的判断结果需要反馈给分类器，更新各种变量后重新计算$P(\tau_i = 可调度)$、$P(\tau_i = 不可调度)$、$P(\tau_i = 可调度 \mid C_1, C_2, \ldots\ldots C_n)$和$P(\tau_i = 不可调度 \mid C_1, C_2, \ldots\ldots C_n)$等先验概率值，为下次进行贝叶斯分类做准备。作业分类完后，形成可调度作业集合与不可调度作业集合。调度器将在可调度作业集合中选择作业任务进行分配执行，而不可调度做业集合将不再考虑范围内，直接进入下次分类器的输入队列。如果所有作业经过分类器分类后都是可调度的，说明 Hadoop 集群满足所有作业的资源需求。如果所有作业经过分类器后都是不可调度的，说明 Hadoop 集群不能满足作业的资源需求，这时可考虑适当增加系统资源或者调整过载规则。

在可调度作业集合中选择作业任务进行调度时有多种方法可供选择，最简单的就是 FIFO（First In First Out，先进先出）方法，在此不做叙述。本文采用一种基于资源需求量与可用资源量的方法，即选择所需资源量与当前可用资源量比（称为资源匹配比，如公式 3.9 所示）最高的作业任务进行调度，这样使得集群的资源利用率最大化。

$$P_i = \frac{R_i}{S} \qquad (0 < P_i < 1) \tag{3.9}$$

其中，R_i 表示第 i 个作业的资源需求量，S 表示当前可用资源量，P_i 表示作业 i 的需求量与当前可用资源量的比。P_i 越大，表示作业 i 的资源需求量与可用资源量越契合，调度作业 i 执行后可用资源量越小，当前浪费资源越少；P_i 越小，调度作业 i 执行后可用资源量越大，当前浪费资源越多。当遍历可调度作业集合并计算所有作业的资源匹配比后，调度器选择比值最高的作业，从中选择一个任务到某个 TaskTracker 上执行。

在选择可调度作业时，尽量考虑 Hadoop 集群分布式文件系统中数据本地特性，即首先选择有数据在本地的作业任务。如果不存在这样的作业，再选择数据不在本地的作业任务进行调度。当作业任务在 TaskTracker 上执行完成后，TaskTracker 都将 CUP、内存、磁盘等特征变量的使用情况通过心跳上报给 JobTracker 中的调度器模块。调度器通过这些信息判断本次执行的作业任务是否过载，从而得知此次作业任务的调度是否合理，计算各种先验概率值，由此来训

练分类器。如果此次作业任务调度不合理，表明作业任务分配失败，调度器下次将避免同样的分配方案。

在可调度作业集合中选择作业进行调度时，除了上述基于作业资源需求量与可用资源百分比的方法外，还可以采用其他方法选择一个可调度作业。这些方法是在$P(\tau_i = 可调度 \mid C_1, C_2, \ldots\ldots C_n)$的基础上添加一个效用函数 W(I)来实现类似优先级的调度方法，通用模式如公式 3.10 所示。

$$R.W.(I) = W(I)P(\tau_i = 可调度 \mid C_1, C_2, \ldots\ldots C_n) \quad (3.10)$$

其中，R.W(I)称为预期效用，W(I)称为和作业 i 相关的效用函数，$P(\tau_i = 可调度 \mid C_1, C_2, \ldots\ldots C_n)$表示作业为可调度作业的概率。

在添加效用函数后，通过提供多种效用函数来实现选择可调度作业的多样化。如果不使用效用函数，调度器会选择使得$P(\tau_i = 可调度 \mid C_1, C_2, \ldots\ldots C_n)$值最大的作业任务执行，这样会导致调度器永远会选择耗用资源最少、最不会导致 TaskTracker 过载的作业而不会给其他可调度作业执行的机会。不同的效用函数侧重点不同，因此不同效用函数的构造方法将形成不同的调度策略，在设计时应多方位、多角度考虑。

上述推导过程同样适用于使用其他的作业类别，只是会有计算每个作业类别概率的公式个数不同而已。

3.2.3 算法描述

从前面的描述可以看出，整个基于贝叶斯分类的作业调度器包含两部分：贝叶斯分类器和作业任务分配器。贝叶斯分类器负责将当前所有作业分为可调度作业和不可调度作业两类，作业任务分配器负责在可调度作业集合中选择一个作业任务分配到 TaskTracker 上运行，算法描述如图 3-1 所示。

算法 3 基于贝叶斯分类的调度算法
输入：计算节点状态 输出：可调度执行的作业任务

获取作业特征变量值、节点特征变量值
for 作业集合 //选择可调度作业进入可调度集合
计算前一次任务调度过载结果
更新各先验概率
根据公式 3.7 和 3.8 计算作业可调度和不可调度概率
if 可调度概率 > 不可调度概率
将作业放入可调度作业集合
else
将作业放入不可调度集合
end if
end for
for 可调度作业集合中每个作业 //在可调度作业集合中选择一个作业和任务
根据效用函数计算当前可调度作业的效用值
end for
选择效用值最大的可调度作业
根据数据本地性和延迟调度机制选择作业中的一个任务
更新系统状态
更新作业信息
返回要运行的任务

图 3-1 基于贝叶斯分类的作业调度算法

基于贝叶斯分类的调度算法时间复杂度为$O(n+m)$，其中 n 表示作业数量，m 表示所有作业中任务数量的最大值。与计算能力调度和公平调度算法相比，少了选择队列或资源池的工作量。算法流程图如图 3-2 所示。

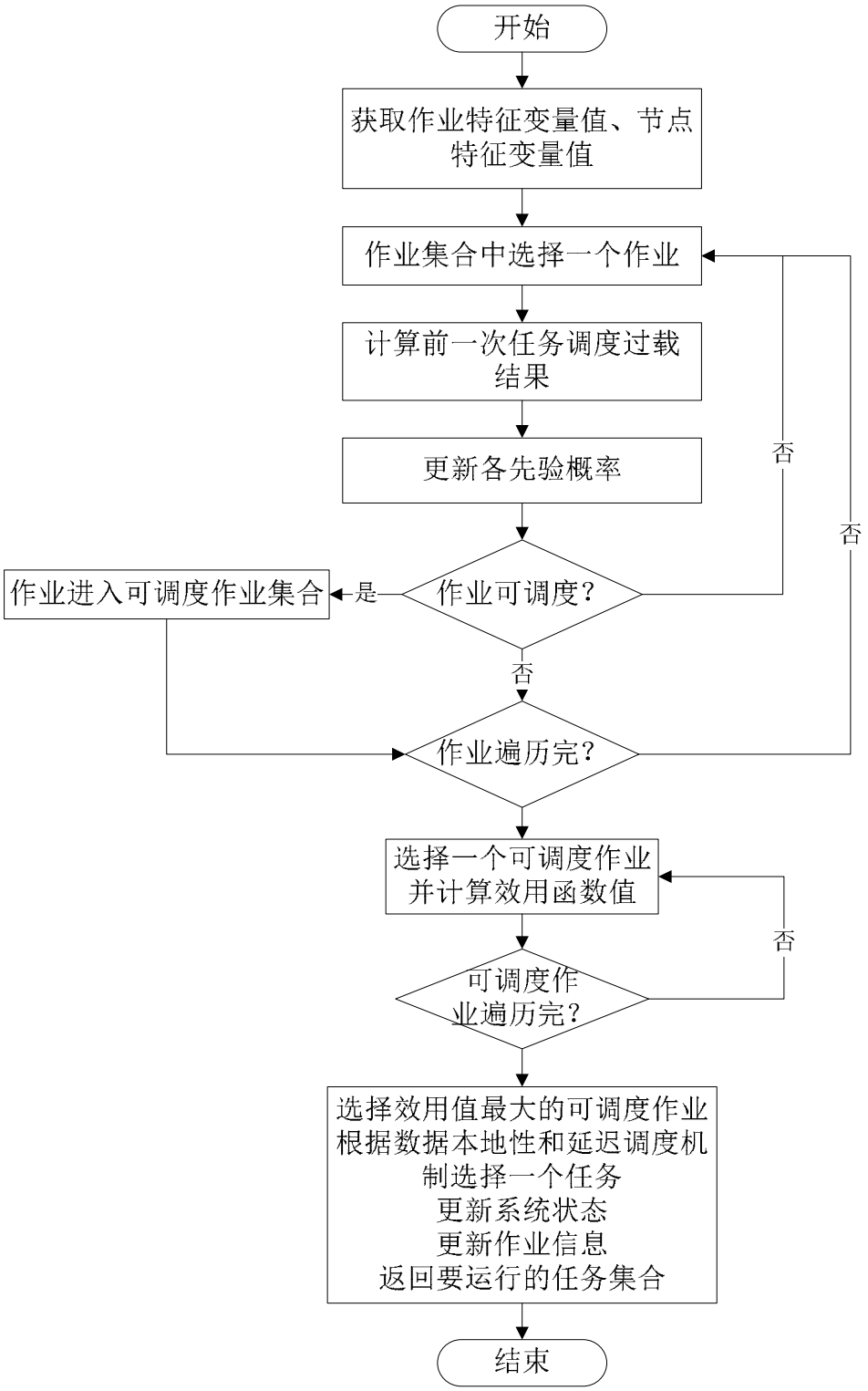


图 3-2 基于贝叶斯分类的作业调度算法流程

（1）获取特征变量值。当 Hadoop 集群中有 TaskTracker 具有空闲的资源时，它会通过心跳向 JobTracker 请求作业任务。JobTracker 会将为 TaskTracker 分配作业任务的工作交给作业调度器。作业调度器首先从心跳信息中提取所需的作业特征和节点特征两类特征变量值，包括 CPU、内存、磁盘、I/O 等资源需求、空闲和使用情况，作为贝叶斯分类器的输入。其中节点特征变量值有两个用途，一个是

用于当前贝叶斯计算器的输入，一个作为过载决策器的输入用于判断上次作业任务执行情况是否过载．过载决策器的结果直接影响$P(\tau_i = 可调度)$、$P(\tau_i = 不可调度)$、$P(C_j|\tau_i = 可调度)$、$P(C_j|\tau_i = 不可调度)$等先验概率值。

（2）贝叶斯分类器将所有作业分为可调度和不可调度作业两类。贝叶斯计算器遍历所有作业，根据公式 3.7 和 3.8，计算所有作业为可调度作业和不可调度走也的概率，并根据这两个概率大小将作业分类为可调度作业和不可调度作业两个集合。其中，如果作业为可调度作业则将其放入可调度作业队列集合中作为作业选择模块的输入；如果作业为不可调度作业，则将其放入不可调度作业队列，并可直接忽略。

（3）作业调度器遍历所有可调度作业集合，使用效用函数计算每个可调度作业的预期效用值。不同的效用函数侧重点不同，计算出的预期效用值也不同，导致最终选择分配给 TaskTracker 上运行的的作业任务不同。

（4）在计算出所有可调度作业的最终预期效用值后，dispatcher 选择预期预期效用值最大的可调度作业，从中选择一个任务分配给 TaskTracker 运行。

（5）当有新的 TaskTracker 通过心跳请求分配任务执行时，转到步骤 1。

整个作业调度器的结构和工作流程如图 3-3 所示。

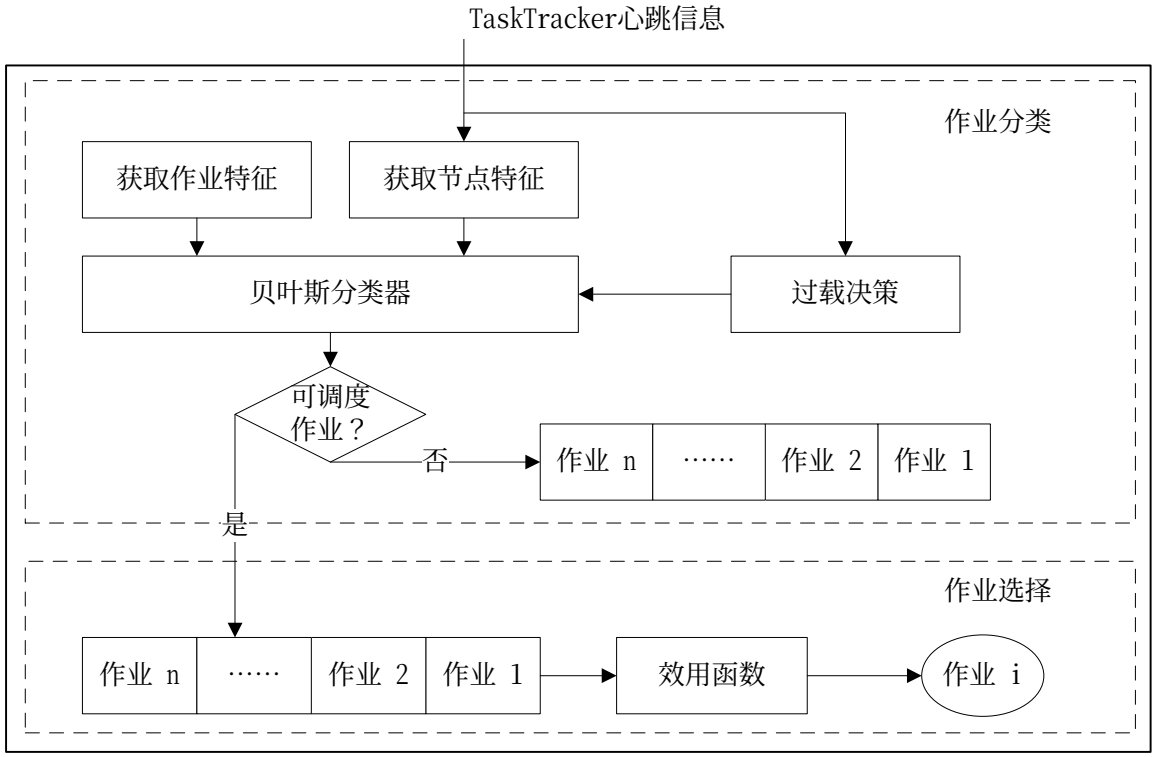


图 3-3 基于贝叶斯分类的作业调度器结构与工作流程

3.3 基于资源需求类型的分类预处理

3.3.1 算法思想

由于同一个作业的多个 map 任务通常具有相同的资源需求特性，而现有调度算法在调度作业任务执行时并未考虑这点，例如多个同类型的 map 任务在计算节点上执行时会产生对相同类型资源的激烈竞争而却浪费了其他类型资源。当资源需求类型不同的作业在计算节点上执行时可以弥补这一不足，因为他们使用的资源类型不同，对同一资源的相互竞争会少很多，能相对互不影响地使用计算节点上的资源。

本文在考虑不同作业不同资源需求类型的基础上提出了基于资源需求类型的作业预处理方法，该方法利用作业负载预测机制来将作业分类为 CPU 密集型、I/O 密集型和暂不确定三类并对不同类型作业进行分类调度，然后对这些不同资源需求类型的作业分类调度，以最大化利用计算节点资源。

3.3.2 基于资源需求类型的分类预处理

如 2.2.2 小节描述，Map 任务的 Shuffle 过程对作业任务运行的重要影响，可以在此阶段对作业的工作负载进行分类预处理。在 Map-Shuffle 阶段，map 任务的输入数据量（MID）和输出数据量（MOD）大小的比例依赖于作业任务运行时的资源负载情况，CPU 计算量大的 map 任务对输入数据处理后产生的数据量往往较小，而 I/O 工作量大的 map 任务往往产生的数据量较大，将这个比例定义为 β，则 map 过程的输入输出数据量的关系可描述如下：

$$MOD = \beta * MID \tag{3.11}$$

Shuffle 步骤的输出数据（SOD）和 MOD 数据相同，因为 MOD 数据时 Shuffle 步骤的输入数据。与此不同的是 Shuffle 步骤的输入数据 SID 并不由 map 任务的数量决定而是由作业的本地 reduce 任务数和作业总 reduce 任务数量的比值决定。依据 Map-Shuffle 阶段的特性，我们将作业任务分为 CPU 密集型、I/O 密集型和咱不确定三种类型。

第一种作业类型为 CPU 密集型作业，这类作业通过任务并行运行可使 CPU 利用率最高达到百分之百。本文假设同一个作业的 map 任务均具有相同的这一比值，并且用 n 表示一个计算节点上同时并行运行的任务数目，这样，我们设置 MTCT 表示 map 任务完成时间，定义 DIOR 为磁盘 I/O 速率。这样，可得到 CPU 密集型作业分类的定义，如公式 3.12 所示：

$$\frac{\mathrm{n(MID+MOD+SOD+SID)}}{\mathrm{MTCT}}=\frac{\mathrm{n((1+2\beta)MID+SID)}}{\mathrm{MTCT}}<DIOR \tag{3.12}$$

对于map任务而言，I/O操作包括输入、输出、shuffle输出和shuffle输入，在作业运行过程中，每个计算节点都有n个map任务运行并且这些map任务共享磁盘I/O。因此，如果n个map任务的（MID+MOD+SOD+SID）之和除以MTCT的值（即公式 4.2 左边的计算值）小于磁盘I/O速率，则可认为这类作业任务为CPU密集型的，我们以此来预测作业的分类。

第二中作业类型不确定类型作业，它与 CPU 密集型作业不同，其 map 任务中少了 shuffle 步骤，但是仍然有 CPU 密集型作业特点。这类作业任务一旦 shuffle 操作时，任务将 I/O 阻塞，因为其 map 任务的 I/O 比例通常小于 DIOR，而当有 reduce 操作时则会有大量 I/O 操作。这类作业的 CPU 利用率不可能会达到 100%，其定义如公式 3.3 所示：

$$\frac{\mathrm{n(MID+MOD+SOD+SID)}}{\mathrm{MTCT}}=\frac{\mathrm{n((1+2\beta)MID+SID)}}{\mathrm{MTCT}}\geq \mathrm{DIOR} \tag{3.13}$$

另一类作业为 I/O 密集型作业，这类作业的 map 任务都会有大量 I/O 操作，同一个计算节点上的大量 map 任务的执行会导致对 I/O 操作的竞争现象。这类作业尽管其 reduce 任务还没开始，其 map 任务仍然会竞争 I/O 资源，该类作业定义如公式 3.14 所示：

$$\frac{\mathrm{n*(MID+MOD)}}{\mathrm{MTCT}}=\frac{\mathrm{n*(1+\beta)MID}}{\mathrm{MTCT}}\geq \mathrm{DIOR} \tag{3.14}$$

由于每个 map 任务具有差不多的输入输出数据量，因此可以假设 reduce 任务也具有差不多的输入数据量。这样，作业的 SID 取决于集群中 reduce 任务的分布情况，每个计算节点上的 shuffle 输入数据取决于正在运行的 reduce 任务和所有 reduce 任务的比值，如公式 3.15 所示

$$\mathrm{SID}=\frac{\mathrm{RRN}}{\mathrm{WRN}}*\mathrm{SOR}*\mathrm{nodesNum} \tag{3.15}$$

在上述公式 3.11-3.15 中，MTCT受计算节点运行时状态的影响，由于作业负载情况，计算节点上的MTCT值可能大于理论值。这表明，如公式 3.12 所示，如果一个作业被认为是I/O类型作业，那么其一定是I/O密集型作业，但是如果一个作业被认为是CPU密集型则不一定确定是CPU密集型作业。因此，I/O密集型作业分类方法是一个保守型方法，而CPU密集型作业分类方法则并不如此。

CTCT、MID、MOD，β 等变量的值在集群计算节点中作业运行过程中计算获得，MTCT 表示 map 任务执行时间、MID 为 map 任务的输入数据量大小，MOD 表示

map 任务的输出数据量大小，β为 MID 与 MOD 的比值，这些均在作业任务的执行后反馈给给调度器用于后续作业任务的调度执行。

基于资源需求类型的分类预处理算法
输入：未被调度作业 输出：按资源类型需求分类后的作业

if 公式 3.12 计算值为真//如果是 CPU 密集型则放入 I/O 密集型作业队列中
将该作业放入 I/O 密集型作业队列中
end if
if 公式 3.13 为真或者公式 3.14 为真 //如果是非 CPU 密集型则放入 CPU 密集型作业队列中
将该作业放入 CPU 密集型作业队列中
标记该作业为不确定的 CPU 密集型作业
end if
if 要运行的作业为 I/O 密集型作业
在拥有 I/O 密集型 slot 的计算节点上运行该作业
监控该作业运行
end if
if 要运行的作业为不确定的 CPU 密集型作业
在拥有 CPU 密集型 slot 的计算节点上运行改作业
监控并计算正在运行的 I/O 密集型作业的最新 MTCT 值
while 最新的 MTCT 值 < Threshold * arg MTCT
将该作业移入 I/O 密集型作业队列中
end while
end if

图 3-4 基于资源需求类型的分类预处理算法

基于资源类型需求的作业调度算法需要三个队列来分别存放三种类型的作业，为此我们设置三个作业队列来存放不同类型的作业，分别为 CPU 密集型作业队列、I/O 密集型作业队列和暂不确定的等待队列。分类算法动态如图 3-5 和图 3-6 所示。

CPU密集型作业队列中存放的均为正在执行的CPU密集型作业和暂不确定的CPU密集型作业，I/O密集型作业队列中存放的均为I/O密集型作业，等待队列中存放的是提交至系统还未确定类型的作业。当有作业提交至系统时，作业首先被放入等待队列，然后调度器将改作业放入CPU密集型队列中并试图调度该作业的

一个map任务到空闲的CPU密集型slot上运行（如图 3-5 所示），并统计计算I/O密集型作业的最近时间的平均MTCT值，如果刚运行的作业任务导致MTCT值增长较快（有个阈值限定），则认为该作业为I/O密集型而不是CPU密集型，因此需要将其视为I/O密集型作业而转到I/O密集型作业队列中，如图 3-6 所示。

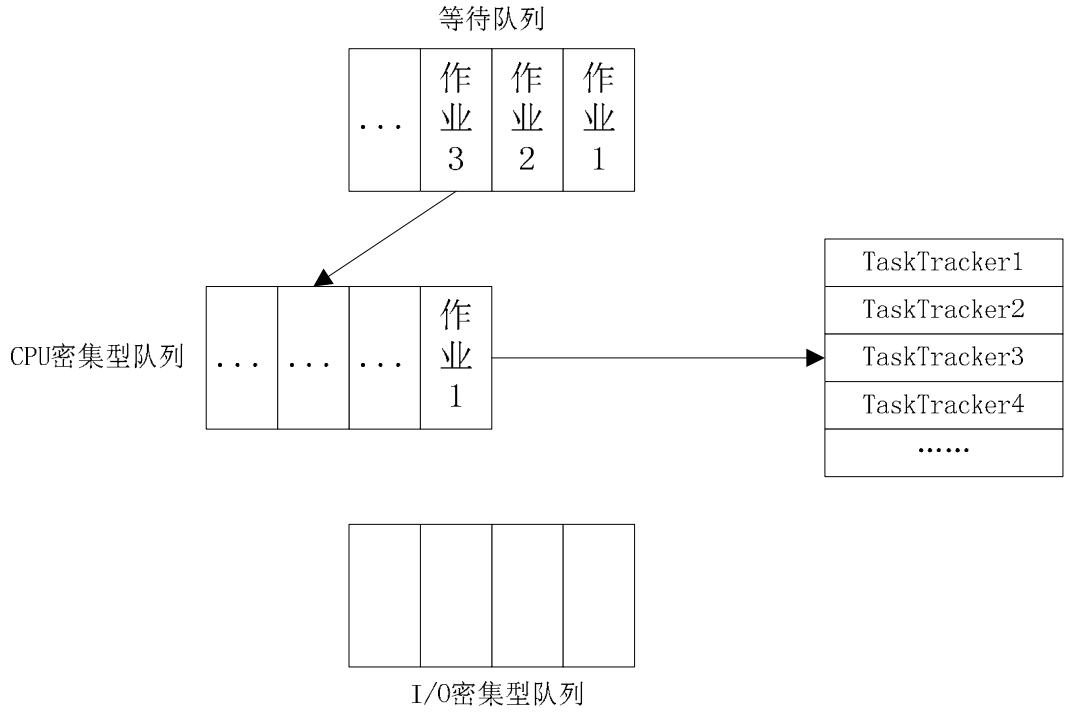


图 3-5 CPU 密集型作业队列和 I/O 密集型作业队列为空时的状态

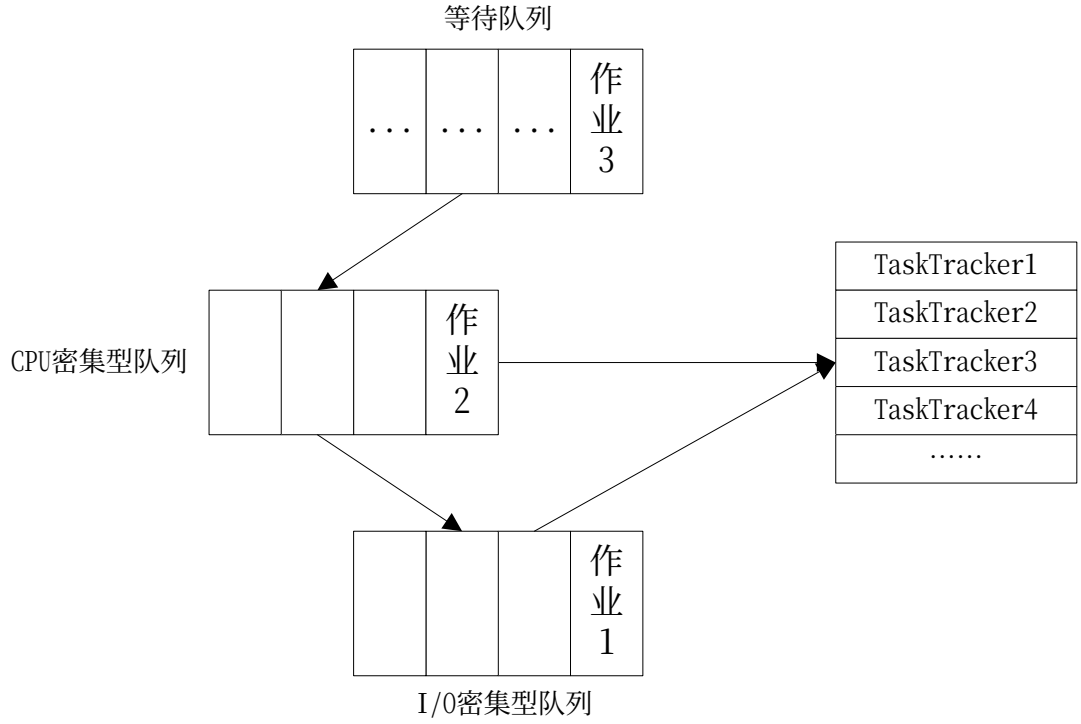


图 3-6 CPU 当检测作业分类错误时更正作业所属队列

在如图 3-4 所示的基于资源需求类型的作业分类算法中，公式 3-12、3-13、3-14 的计算时算法的关键，其中，β 可由map任务的输入输出数据量确定，MTCT表示计算节点上map任务完成时间，可通过监控map任务的运行获得，DIOR表示磁盘I/O速率，可由计算节点配置或者通过运行I/O密集型作业任务测试得到。在检测作业分类正确与否时需要将当前作业放入CPU密集型slot中运行并同时监测I/O密集型作业任务的MTCT值，如果当前放入CPU密集型slot中的作业的运行导致I/O密集型作业map任务完成时间增长较多，则说明该任务的运行也占用了较多的I/O资源导致原有I/O密集型作业任务运行减慢，完成时间增长，从而说明该作业不应该被定义为CPU密集型作业而应该是I/O密集型作业。增长幅度的阈值可通过多次实验观察获取，一般定义为 40%-50%。

可以将基于作业资源需求类型分类方法作为基于贝叶斯分类调度算法的预处理过程，这样可以更好地提高资源利用率。

3.4 本章小结

本章在前面章节的基础上分析了 Hadoop 平台中可供选择使用的几种作业调度算法并指出不足之处，在此基础上提出一种基于贝叶斯分类的作业调度算法和基于资源需求类型的分类预处理。

（1）分析现有调度算法需要大量配置的不足，提出了应用贝叶斯分类的作业调度算法，并描述了核心思想、算法原理、流程和应用该算法的调度器的结构及工作流程。

（2）为进一步提高计算节点的资源使用率，将作业按照资源需求类型情况分为 CPU 密集型作业和 I/O 密集型作业的预处理，并描述了预处理算法的核心思想、算法原理和处理流程。

第四章 实验结果与分析

在前面的章节中，我们针对 Hadoop 系统中现有作业调度算法存在的不足，提出了基于贝叶斯分类的作业调度算法并给出了算法的理论基础、系统建模和算法工作流程。本章将搭建实验环境对基于贝叶斯分类的作业调度算法从调度准确率、作业响应时间、集群资源利用率三个方面进行验证，并将其与其他现有作业调度算法进行比较，对所有实验结果进行分析。

4.1 实验环境搭建与配置

（1）网络拓扑结构

为了更好地模拟真实资源表示方式，我们使用不同的计算机配置来构成一个 Hadoop 集群，集群网络拓扑结构如图 4-1 所示。集群中 node0 用作 NameNode 和 JobTracker，其他 5 个节点（node1-node5）用作 DataNode 和 TaskTracker，各个节点之间用交换机相连。5 台服务器机器的资源配置相同，均配有 Intel core i3 处理器、4G 内存和 320G 磁盘，在此基础上，集群环境配置如表 4-1 所示。

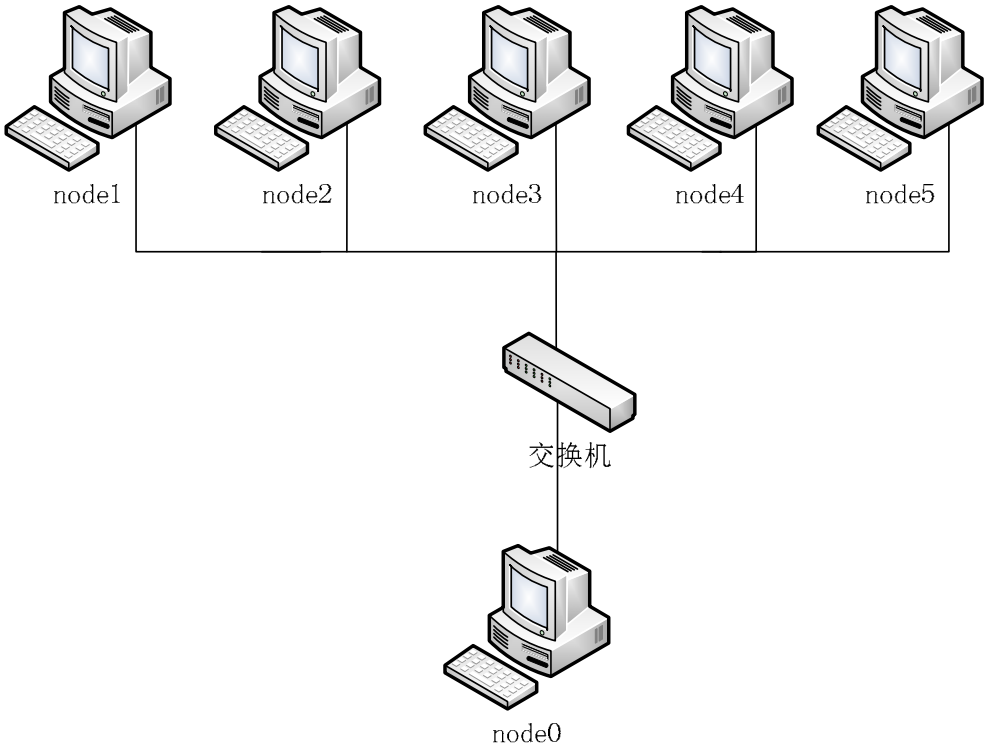


图 4-1 实验环境网络拓扑结构

此 Hadoop 集群一共有 6 个节点，其中一个节点作为 HDFS 的 Namenode 和 MapReduce 的 JobTracker 节点，其余 5 个节点均作为 HDFS 的 Datanode 和

MapReduce 的 TaskTracker 节点。

（2）安装配置 Java 开发环境

所有 NameNode、JobTracker、DataNode 和 TaskTracker 都运行在 Ubuntu Linux 10.04 上，本文实验中 Java 开发环境为 Eclipse 3.5 和 JDK-1.6.0_45。

（3）安装并配置 Hadoop 环境

在安装好 Linux 系统、SSH 后，创建创建一个用户 hadooptest123。为保证集群各个节点间正常通信，每个节点需要配备一个 IP 地址和主机名称，并修改/etc/hosts 文件和/etc/hostname 文件。结合集群的网络拓扑结构，为每个节点分配的 IP 地址如表 5-2 所示。

表 4-2 Hadoop 集群各节点配置

节点 IP	节点名称	用途
192.168.2.2	node0	NameNode、JobTracker
192.168.2.3	node1	DataNode、TaskTracker
192.168.2.4	node2	DataNode、TaskTracker
192.168.2.5	node3	DataNode、TaskTracker
192.168.2.6	node4	DataNode、TaskTracker
192.168.2.6	node4	DataNode、TaskTracker

在完成上述工作后，就可以安装 Hadoop 并进行相关配置，Hadoop 系统的版本为 0.20.203，具体步骤如下：

①安装 Hadoop。将 Hadoop 包解压缩到/home/hadooptest/hadoop-0.20.203，并在/etc/profile 文件中添加 HADOOP_HOME 和 PATH 的配置。

②修改 hadoop-env.sh 文件，添加 JAVA_HOME 路径配置。

③修改 masters 和 slaves 文件，分别添加各自节点主机名称。

④分别配置 core-site.xml、hdfs-site.xml 和 mapred-site.xml 文件，相关重要的参数设置如表 4-3 所示。

表 4-3 Hadoop 相关重要参数设置

参数名	设置值
dfs.replication	3
dfs.block.size	64M
mapred.map.tasks	2
mapred.reduce.tasks	1
mapred.tasktracker.map.tasks.maximun	2
mapred.tasktracker.reduce.tasks.maximun	2
heartbeat interval	5s

4.2 实验评估方法

为验证基于贝叶斯分类的作业调度算法的有效性，我们选择不同类型的作业多次运行并使用基于贝叶斯分类的作业调度算法进行调度。Hadoop wiki 和 Hadoop 自带的基准测试中提供了大量的各种使用 Hadoop 系统进行研究的不同类型的作业，我们从中选择了两类作业：WordCount 作业（计算文本词频）、CPUActivity（对 key-value 进行数值计算）和 Sort（对大文件数据进行排序），他们对 CPU、内存、I/O 等计算资源的需求量不同。因此使用他们来做测试用例具有很强的代表性。

（1）WordCount 是一个统计输入文本中英文单词出现次数的作业。其处理过程是先由 map 任务将输入分片数据中的单词切分并将其出现次数设置为 1，生成<word, 1>形式的中间结果。然后由 reduce 任务统计所有相同单词出现的次数并最终生成<word, count>形式的输出结果。WordCount 作业对内存的需求量大。

（2）Sort 是一个对大数据集进行排序的作业。其处理过程先由 map 任务对各个分片数据进行局部排序并生成中间结果，然后由 reduce 任务对所有数据进行全排序。Sort 是 CPU 密集型作业，对 CPU 资源的需求量大。

（3）CPUActivity 是一个对 key-value 做数值计算的作业。其处理过程主要由 map 任务进行，将<key, value>形式的输入进行复杂的计算得到结果 result 并以<key, value, result>形式进行输出。CPUActivity 作业对 CPU 资源的需求量大。

在 Hadoop 系统上运行各个作业时，我们侧重于观察各个作业经过分类器后的分类结果以及当作业作为可调度作业时，经过调度器被调度执行后其执行结果来观察算法运行情况。具体而言，就是当某个作业在经过贝叶斯分类器被分类为可调度作业并被调度到某个合适的 TaskTracker 上执行后，我们分析执行后 TaskTracker 通过心跳上报给 JobTracker 的是否过载的结果信息。判定一个作业正确调度的条件是改作业经过调度后正确完成工作任务并没有使 TaskTracker 过载。

通过心跳中这些信息来判定该调度算法的调度准确率、作业运行时间和集群资源利用率，同时本文也将该算法与现有作业调度算法的比较。

1、作业任务调度准确率。作业任务调度准确率准确率描述该算法在一段时间内调度某类作业时能够做出正确调度的概率越来也大。为了验证基于贝叶斯分类的作业调度算法的学习能力，我们运行 WordCount 作业多次，统计其各次任务分配结果，通过计算$P(\tau_i = \text{可调度} \mid C_1, C_2, \ldots\ldots C_n)$概率来验证，如果该概率逐次变大，则说明算法的学习能力较好，能够经过一段时间的学习进行较好地任务调度。

2、作业运行时间。作业运行时间是衡量 Hadoop 系统的重要标准之一，如果作业在越短时间内完成说明 Hadoop 系统性能越好。作业运行时间取决于作业任务本身的运行计算时间和数据到达 TaskTracker 的时间。如果调度器能将作业任务调度到离输入数据近的 TaskTracker 上运行，浪费在拷贝数据上的时间越少，作业运行总的时间也越少。

3、集群资源利用率。集群资源利用率是衡量集群资源使用情况的重要标准之一，使用计算机或计算机集群的重要目标之一就是尽可能地发挥计算机的各部件功能。我们分别适用 FIFO 调度算法、计算能力调度算法、公平调度算法和基于贝叶斯分类的调度算法来多次运行作业，考虑集群资源利用率。

4.3 实验结果与分析

本文在讨论作业分类对提高作业调度成功比例的基础上，提出了基于分类的 Hadoop 作业调度算法，本小节将从作业调度准确率、作业响应时间和集群资源利用率三个方面进行实验与分析。

4.3.1 作业分类调度对作业调度准确率的影响

在本实验中依次使用 FIFO 调度算法、计算能力调度算法和公平调度算法和贝叶斯调度算法调度 4 个 WordCount 作业（命名为 job1、job2、job3 和 job4）执行，每隔一定时间统计任务调度的准确率，任务调度准确与否通过作业任务调度是否导致 TaskTracker 过载与否来判定，任务调度导致 TaskTracker 过载的判定为任务调度失败。四个作业配置相同，作业数据大小均为 4GB，优先级均为 NORMAL，map 任务输入分片大小为 64MB。实验中每隔 1 分时间依次将 job1、job2、job3、job4 提交至系统执行，计算调度准确率时以每 50 次任务调度为一小组统计任务调度时 TaskTracker 未过载次数。共进行 10 次实验，计算结果取平均值测试结果分别如表 4-4、4-5、4-6、4-7 所示。

表 4-4 采用 FIFO 调度算法时作业调度准确率

调度次数(次)	50	100	150	200	250
准确次数(次)	48.5	48.2	48.5	48.0	48.1
准确率(%)	97.00%	96.40%	97.00%	96.00%	96.20%

表 4-5 采用计算能力调度算法时作业调度准确率

调度次数(次)	50	100	150	200	250
准确次数(次)	46.6	46.5	46.3	46.5	46.1
准确率(%)	93.20%	93.00%	92.60%	93.00%	92.20%

表 4-6 采用计算能力调度算法时作业调度准确率

调度次数(次)	50	100	150	200	250
准确次数(次)	46.4	45.8	45.1	45.7	45.1
准确率(%)	92.80%	91.60%	91.20%	91.40%	91.20%

表 4-7 采用基于贝叶斯分类调度算法时作业调度准确率

调度次数(次)	50	100	150	200	250
准确次数(次)	46.4	46.8	47.7	48.4	48.4
准确率(%)	92.80%	93.6%	95.4%	96.8%	96.8%

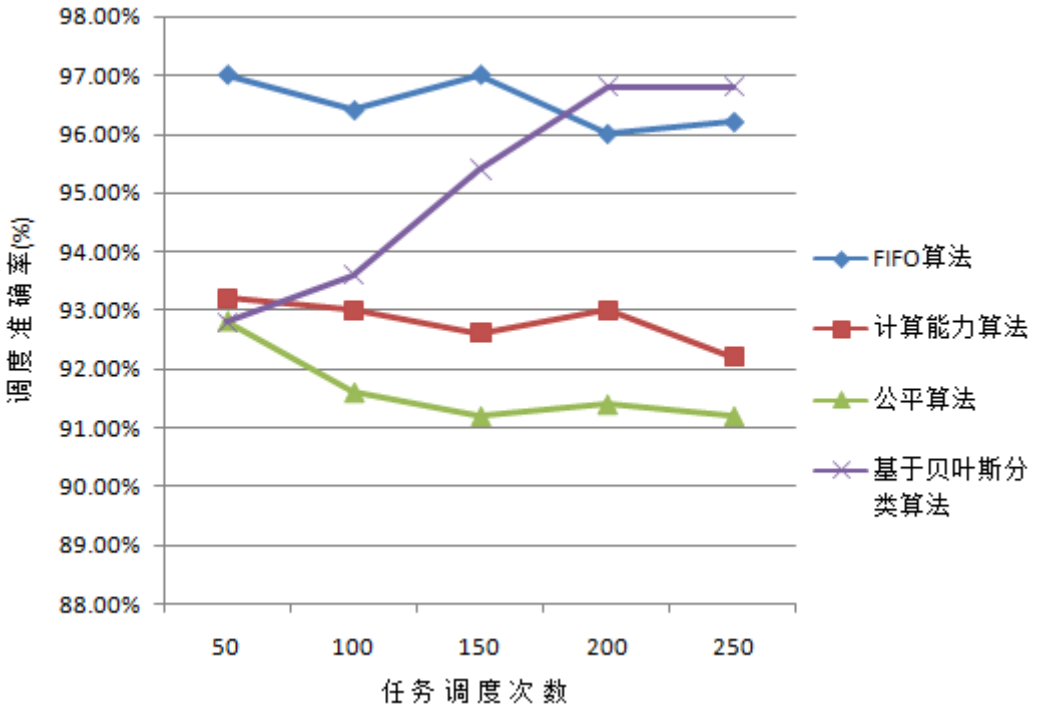


图 4-2 各调度算法调度准确率对比

从图中可知，各作业调度算法均具有较高的调度准确率，这是由于 Hadoop 平台提供了巨大的计算资源量，保障了作业任的运行所需，但各个调度算法也会

有些差异。在传统调度算法中，FIFO 调度算法的特性使得它具有相对更高的准确率，因为 FIFO 算法按照作业在队列中的先后顺序进行调度，后一个作业的任务不会被调度执行直到前一个作业的所有任务都开始调度运行，所以正在运行的作业占用集群所有计算资源，因此任务运行时过载的可能性比较小，而计算能力调度算法和公平调度算法都是多用户作业共享集群资源，作业运行时使用的资源受限，作业任务运行相对容易导致 TaskTracker 过载。同时，计算能力算法在调度作业任务时会对内存资源进行匹配，必要时会分配更多的 slot 的内存资源给作业，内存资源对作业任务运行的限制较小，而公平算法则没有这一功能，因此计算能力算法相对公平算法有更高的调度准确率。

基于贝叶斯分类的作业调度算法在多作业运行初始阶段没有作业任务运行历史信息参考，因此初始阶段调度准确率和计算能力算法和公平算法类似，但是随着任务调度次数的增加，基于贝叶斯分类调度算法通过不断学习资源分配作业任务的匹配情况后能将任务分配到更合适的 TaskTracker 上运行，因此过载情况会越来越少，任务调度准确率迅速提升，甚至超过了使用 FIFO 算法时的准确率。

从上述实验可知，在进行作业任务调度时通过分析作业任务执行历史信息可以将作业任务分类并调度到合适的节点上执行可获得更高的调度准确率。

4.3.2 作业分类调度对作业运行运行时间的影响

作业运行响应时间是评判作业调度算法的一项重要标准，本实验将依次使用 FIFO 算法、计算能力算法、公平算法和基于贝叶斯分类算法多次运行 WordCount 作业，分别统计每个作业运行时间，最后取三个作业运行时间平均值。三个作业均为 WordCount 作业，只是处理的数据量不同，如表 4-8 所示。

表 4-8 响应时间实验作业用例

作业名称	数据量	map 任务数	优先级
WordCount1	256MB	4	NORMAL
WordCount2	512MB	8	NORMAL
WordCount3	1GB	16	NORMAL

实验中未设置的参数取系统默认值，实验按 WordCount1、WordCount2 和 WordCount3 顺序依次提交三个作业，各种算法下各执行 10 次，每次记录 WordCount3 作业的运行时间，结果取平均值，测试结果如表 4-9 所示。

表 4-9 使用各算法调度时的作业运行时间

	WordCount1	WordCount2	WordCount3
FIFO 算法	110s	257s	692s
计算能力算法	106s	249s	588s
公平算法	178s	316s	596s
基于贝叶斯分类算法	133s	241s	498s

表 4-9 中的数据表明基于贝叶斯分类算法相比 FIFO 算法从作业 WordCount1 时响应时间高出 20.91%，到 WordCount2 时反而少了 6.23%，再到 WordCount3 时少了 28.03%，相比计算能力算法也有相似的结果，分别为 25.47%、3.21%和 19.81%；相比公平算法一直都有优势，提升比例分别为 25.28%, 23.73%和 15.31%。将表 4-9 数据整理成柱形图表示，如图 4-3 所示。

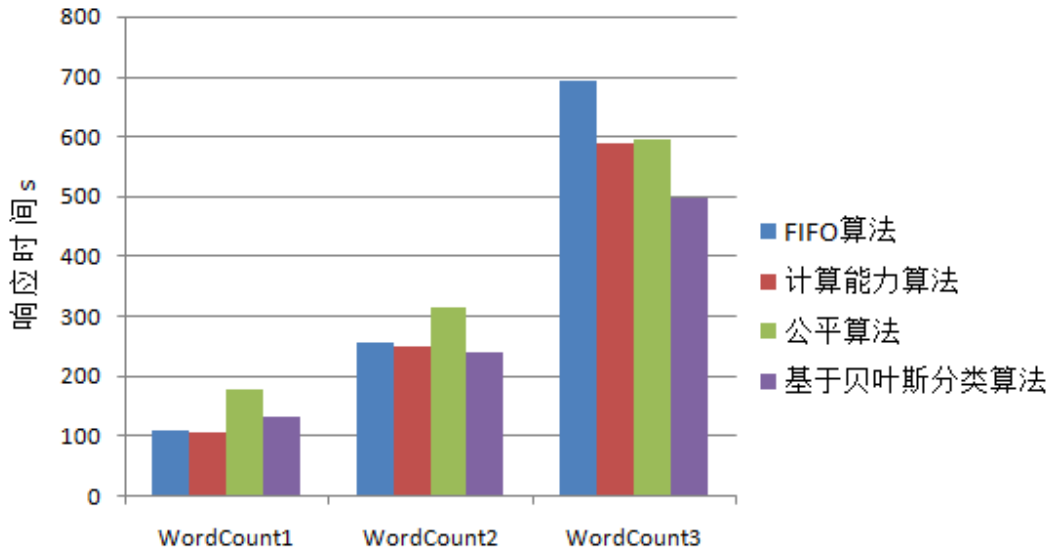


图 4-3 使用各调度算法时作业运行时间对比

从图 4-3 可知，FIFO 算法、计算能力算法和公平算法在作业处理的数据量增加时作业运行时间显著增加，而使用基于贝叶斯分类算法时响应时间相比其他三种算法增长越来越缓慢，这表明基于贝叶斯分类的作业调度算法经过分析作业任务历史信息为后面的任务调度提供更多的 TaskTracker 特征信息，便于作业任务调度将合适的作业任务分配好资源匹配的 TaskTracker 上运行，从而降低了作业的运行时间。

从上述实验可知，在进行作业任务调度时通过分析作业任务执行历史信息可以将作业任务分类并分配到合适的节点上执行可获得更短的作业响应时间。

4.3.3 作业分类调度对集群资源使用率的影响

集群资源利用率是衡量 Hadoop 作业调度算法的另一项重要标准，本实验将测试使用各调度算法调度作业任务时 Hadoop 集群的资源利用情况。为了最大化利用 CPU、内存和 I/O 这些计算资源，我们使用各调度算法调度 CPU 密集型、I/O 密集型和普通类型的作业，以此查看作业分类对集群资源利用的影响情况。

在实验中，我们使用如表 4-10 所示的三种类型作业，CPUActivity 为 CPU 密集型作业，Sort 为 I/O 密集型作业，WordCount 为一般类型作业，通过 FIFO 算法、计算能力算法、公平算法和基于贝叶斯分类算法调度这三个作业。

表 4-10 资源利用率实验作业用例

作业名称	数据量	map 任务数	优先级	作业类型
Sort	256MB	4	NORMAL	I/O 密集型
CPUActivity	256MB	4	NORMAL	CPU 密集型
WordCount	256MB	4	NORMAL	一般

实验按照 Sort、CPUActivity、WordCount 顺序提交三个作业，并每隔 100 秒时间记录作业运行过程中对 CPU 资源的使用率。共进行 10 次实验，结果取 CPU 利用率平均值，测试结果如表 4-11 所示。

4-11 使用各调度算法时集群平均 CPU 利用率

	100	200	300	400	500	600
FIFO 算法	27%	31%	84%	81%	45%	19%
计算能力算法	53%	66%	59%	62%	57%	33%
公平算法	61%	62%	54%	68%	57%	39%
基于贝叶斯分类算法	62%	70%	73%	75%	63%	16%
基于作业资源需求分类预处理和贝叶斯分类算法	73%	90%	91%	84%	62%	16%

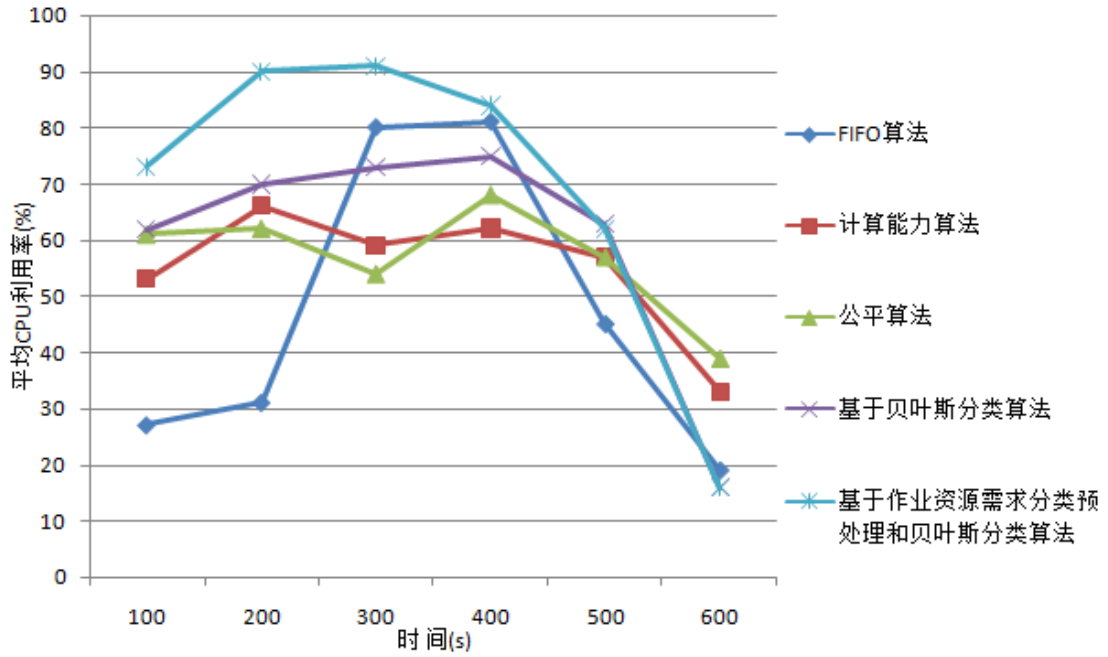


图 4-4 使用各调度算法时集群平均 CPU 利用率

从图 4-4 可知, FIFO 算法由于调度特性在调度 CPUActivity 作业执行时 CPU 利用率较高，而在 Sort 作业和 WordCount 作业执行时 CPU 利用率较低。计算能力算法和公平算法都对各作业综合调度，没有明显的作业分类，CPU 利用率比较高并稳定直到作业运行结束。基于贝叶斯分类的调度算法在多作业运行期间对集群 CPU 资源利用率较高，在作业运行尾声时 CPU 资源有空闲。

由于基于贝叶斯分类分类的调度算法将作业依照特征进行分类，使得作业任务的运行尽量不使计算节点过载，这样计算节点既不过载又能尽可能利用资源，从而获得较好的资源利用率。

在添加基于作业资源需求类型分类预处理后，预处理过程能够识别出系统中运行作业的类型（CPU 密集型或 I/O 密集型作业），并进行分类调度，使得 CPU 密集型作业能够以更大概率获得和 I/O 密集型作业在同一个节点上执行，这样 CPU 密集型作业任务能够充分利用 CPU 资源，I/O 密集型作业任务能够更充分利用 I/O 资源，从而使得计算节点资源利用率更高。

从上述实验可知，在进行作业任务调度时通过分析作业任务执行历史信息可以将作业任务分类并分配到合适的节点上执行可获得较高的集群 CPU 资源利用率。同时，I/O 资源的使用与 CPU 资源有互补特点，通过合理安排 CPU 密集型作业和 I/O 密集型作业的调度运行，可以获得较高的集群 I/O 资源利用率。

4.4 实验结论

在使用 Hadoop 传统作业调度算法（FIFO 算法、计算能力算法和公平算法）

时，正确的配置显得十分重要，因此系统管理员需要十分熟悉 Hadoop 集群的资源特性和用户提交运行的各类作业的特性，而正确做到这些并不容易。本文在分析 MapReduce 作业运行机制和传统作业调度算法的基础上，提出基于分类的作业调度算法。利用贝叶斯分类方法分析作业任务运行历史信息，获取作业和节点资源特征，试图对 MapReduce 作业进行分类，将作业任务分配到拥有合适计算资源的节点上运行，并在作业任务运行完后反馈任务运行情况来调整后续任务分类与分配执行。通过这一不断学习的过程，最后获得较高的作业任务调度准确率、较短的作业运行时间和较高的集群资源利用率。

本章实验使用 FIFO 算法、计算能力算法、公平算法和基于贝叶斯分类的算法调度 WordCount、CPUActivity 和 Sort 三个作业的执行，测试了作业任务调度准确率、作业响应时间和集群 CPU 资源利用率，得到如下结论：

（1）基于分类的作业调度算法比传统调度算法有了更高的任务调度准确率，且随时间准确率上升后逐渐稳定。

（2）基于分类的作业调度算法比传统调度算法有更少的作业执行时间。

（3）基于分类的作业调度算法比传统调度算法有更高的集群资源利用率，特别是在多 CPU 密集型、I/O 密集型等作业情况下工作得更好。

综上所述，在 Hadoop 作业调度过程中通过分析作业任务执行信息，依据这些信息将作业任务分类调度能够获得较好的性能。因此，在 Hadoop 作业调度算法中加入作业任务分类调度方法具有一定的优越性。

4.5 本章小结

本章对第三章提出的基于贝叶斯分类的作业调度算法进行了实验分析，给出了实验环境搭建与配置步骤、方法，描述了进行实验的各个作业并给出了实验评估方法、测试指标，针对各个测试指标给出了相应的实验内容、实验结果与分析。实验分析结论为：

（1）基于分类的作业调度算法比传统调度算法有了更高的任务调度准确率，且随时间准确率上升后逐渐稳定；

（2）基于分类的作业调度算法相比传统调度算法有更少的作业运行时间；

（3）基于分类的作业调度算法比传统调度算法有更高的集群资源利用率，特别是在多 CPU 密集型、I/O 密集型等作业情况下工作得更好。

第五章 全文总结与展望

5.1 总结

本文介绍并详细分析了 Hadoop 平台的分布式文件系统 HDFS、MapReduce 计算框架、MapReduce 应用程序运行时环境。然后详细介绍分析了 Hadoop 现有 FIFO 调度算法、计算能力调度算法和公平份额调度算法的核心思想、实用配置参数、算法流程，对其做出对比分析并阐述总结了各调度算法的优缺点。现有的计算能力算法和公平份额算法这两种多用户多类型作业调度算法都需要系统管理员 Hadoop 集群各节点资源情况和运行在 Hadoop 集群上 MapReduce 作业的资源使用特性有充分的了解，同时还需要和集群使用用户确定作业的资源需求。这些最终都是由系统管理员通过正确的配置完成的，但正确的配置往往难以做到。

本文在深入分析预先配置的作用并为尽量减少预先配置方面做出改进，在此基础上提出了基于分类的作业调度算法，并详细给出了算法思想、理论推导和算法流程描述等。算法通过分析作业任务执行历史信息，不断学习作业任务被分配执行对节点计算资源的影响来调整后面作业任务的调度决策。这种作业调度方式可以减轻 Hadoop 系统管理员的工作量，同时提高了集群性能。

本文选择不同类型的作业来进行实验，给出了评估基于贝叶斯分类的调度算法的评估方法，并给出了各个实验结果与分析。最后，结合实验与分析给出了基于贝叶斯分类分类的调度算法在调度正确率、作业响应时间和集群资源利用率方面具有一定优越性的实验结论。

本文主要工作总结如下：

（1）分析 MapReduce 计算框架及作业执行控制流程等 Hadoop 相关技术。

（2）分析比较 Hadoop 中 FIFO 调度算法、计算能力调度算法和公平调度算法的核心思想、使用配置并给出了伪码和流程图形式的算法描述、复杂度分析和算法优缺点分析，并对公平份额调度算法中的资源池 slot 分配方法进行了改进，改进后对资源池差额的补充更为公平合理。

（3）为减少使用现有 Hadoop 作业调度算法时的预先配置作出改进，提出基于贝叶斯分类的学习算法，并给出了算法核心思想、理论推导和算法流程描述等。为进一步提高计算节点的资源使用率，将作业按照资源需求类型情况分为 CPU 密集型作业和 I/O 密集型作业的预处理，进行分类调度使得计算资源得到更充分的利用。

（4）结合实验对该算法的作业任务调度准确率、作业响应时间、集群资源利用

率等性能指标做出分析并给出实验结论。

5.2 展望

现实的社会需求促使着云计算的快速发展，云计算产业的发展离不开大数据处理平台的支持，因此 Hadoop 在今后相当长一段时间内仍是大数据处理的重要平台。在 Hadoop 中，如何进行更好地作业调度进一步提高 Hadoop 系统的资源利用率仍然十分重要。本文提出基于贝叶斯分类的作业调度算法并给出了算法模型与实现，具有一定优越性，但是该算法模型仍有许多可以进一步深入研究、改进和扩展的地方：

（1）作业特征的来源和节点特征的表示。作业特征和节点特征的选择直接影响到系统计算资源的表示方式，因此可以通过多次试验合理选取作业特征和节点特征，例如节点特征可以考虑系统的网络拓扑结构、集群中新老机器设备计算能力等因素。

（2）效用函数。不同的效用函数通过公式 3-7 计算结果十分影响选择正确的作业进行任务调度，因此设计更加合理的效用函数仍是未来需要努力的方向。

（3）不同的作业和节点资源分类方法会影响作业最终的被调度执行，因此尝试不同的分类方法具有一定的意义。

同时，整个 Hadoop 平台的各个模块都有一定的相关性，各模块的高效运行都能很好地提高整个系统的效率，因此未来也需要进一步更深入地研究。

致　谢

时光荏苒，光阴似箭，三年的研究生生涯转眼即将结束。在本科学习的基础上，研究生生活、学习和工作中让我更加善于自我学习、自我思考，学会如何与人沟通，学会如何正确地平衡生活与学习间的关系，学会了如何认识自己、认识世界。在研究生生活和学习中，我的性格变得更加坚强，综合素质不断地提高。我清楚地认识到研究生生涯将会是我整个人生中不可或缺的一部分，也是我日后最值得回味的一段光辉岁月。

首先，特别感谢我的导师钟珞教授。钟老师丰富的学识、严谨的治学态度、精益求精的工作作风、祥和的待人态度使我受益匪浅。他对学生学术上的认真指导、学习及生活上的关心使我们感动，在此深表敬意和谢意。

同时，感谢团队的夏红霞、袁景凌、李琳、向广利、邹承明、宋华珠等老师，他们渊博的知识，严谨的治学作风，宽以待人的态度以及在学习和生活上的指导和帮助令我感动，在此深表谢意。

另外，还要感谢武汉理工大学计算机科学与技术学院，是学院提供的良好学习氛围和生活环境，在此表示感谢。

感谢所有关心和支持我的同学和朋友们。感谢室友胡雪龙、段崇聪、丁师邦和万轶同学，我们一起渡过美好的时光；感谢 lab6 实验室的同学们，谢谢你们在学习生活中对我的帮助。

感谢父母的教诲与培养！

参考文献

[1] CNNIC. 第 33 次互联网络发展状况统计报告[EB/OL]. http://www.cnnic.net.cn/hlwfzyj/hlwxzbg/hlwtjbg/201403/t20140305_46240.htm, 2014-03-05

[2] Wikepedia. Cloud Computing[EB/OL]. http://en.wikipedia.org/wiki/Cloud_computing, 2014-03-13

[3] G.Goulouris, J.Dollimore, T.Kindberg, G Blair 著,金蓓弘,马应龙译.分布式系统[M].北京:机械工业出版社,2012.

[4] 罗军舟,金嘉晖,宋爱波等.云计算:体系架构与关键技术[J]. 通信学报,2011,32(7): 3-21.

[5] 陈全,邓倩妮. 云计算及其关键技术[J]. 计算机应用, 2009, 29(9):2562-2567.

[6] 王意洁,孙伟东,周松等. 云计算环境下的分布式存储关键技术[J]. 软件学报,2012,23(4): 962-986.

[7]刘正伟,文中领,张海涛. 云计算和云数据管理技术[J]. 计算机研究与发展,2012,49: 26-31.

[8] Apache. Hadoop[EB/OL]. http://hadoop.apache.org/, 2014-05-16

[9] 蔡斌,陈湘萍. Hadoop 技术内幕:深入解析 Hadoop Common 和 HDFS 架构设计与实现原理[M].北京:机械工业出版社,2013.

[10] Apache. HDFS Architecture. http://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html, 2014-05-17

[11] K.Shvachko, H.Kuang, S.Radia, et al. The Hadoop Distributed File System[C]. 2010 Symposium on Massive Storage Systems and Technologies, Incline Village, NV, 2010:1-10.

[12]Ghemawat S, Gobioff H, Leung S. The Google File System[C] //Proceedings of the 19th Symp on Operating Systems Principles(SOSP'03). New York: ACM, 2003:29-43.

[13] Apache. MapReduce Tutorial. http://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html, 2014-07-06

[14] Capacity Scheduler[EB/OL]. http://Hadoop.apache.org/common/docs/ r0.20.2/Capacity_Scheduler.html, 2014-07-06

[15] M.Zaharia, D.Borthakur, J.S.Sarma, et al. Job Scheduling for Multi-User MapReduce Clusters[C]. In Technical Report UCB/EECS, 2009,55.

[16] Fair Scheduler[EB/OL]. http://Hadoop.apache.org/common/docs/r0.20.2/Fair_Scheduler.html, 2014-07-10

[17] J.Dean, S.Ghemawat. MapReduce:Simplified Data Processing on Large Clusters[J].

Communication of ACM, 2008, 51(1):107-113.

[18] J.Chauhan, D. Makaroff, W.Grassmann. The Impact of Capacity Scheduler Configuration Settings on MapReduce jobs[C].// 2012 Second International Conference on Cloud and Green Computing(CGC), 2012, 96:667-676.

[19] Max-Min Fairness [EB/OL]. http://en.wikipedia.org/wiki/Max-min_fairness, 2014-07-10

[20] A.Demers,S.Keshav,S.Shenker. Analysis and simulation of a fair queueing algorithm[J]. In Journal of Internetworking Research and Experience, 1990:3-26.

[21] Tom White. Hadoop 权威指南(中文版)[M]. 北京：清华大学出版社,2010.

[22] 董西成. Hadoop 技术内幕：深入解析 MapReduce 架构设计与实现原理[M]. 北京：机械工业出版社, 2013.

[23] H.Herodotou,S.Babu, Profiling, what-if analysis,and cost-based optimization of MapReduce Programs[C].// VLDB Endowment(PVLDB), 2011, (4): 1111-1122.

[24] G.Wang,A.Butt.,P.Pandey, and K.Gupta, A simulation approach to evaluating design decisions in MapReduce setups, MASCOTS ’09, 2009: 1-11.

[25] Tian C, Zhou H, He Y,et al. A dynamic MapReduce Scheduler for Heterogeneous Workloads[C] //Proceedings of the 8th International Conference on Grid and Cooperative Computing(GCC’09). Piscataway, NJ:IEEE, 2009: 218-224.

[26] YANG Y. An evaluation of statistical approaches totext categorization[J]. Journal of Information Retrieval, 1999, 1(1):67-88.

[27] 冀俊忠,刘椿年,沙志强. 贝叶斯网络模型的学习、推理和应用[J].计算机工程与应用,2003(5): 24-27.

[28] 张璠. 多种策略改进朴素贝叶斯分类器[J]. 微机发展. 2005(4):35-36.

[29] Quinlan J. C4.5:Programs for Machine Learning[M]. San Matteo, CA:Morgan Kaufmam Publishers,1993.

[30] MapReduce-4305[EB/OL]. https://issues.apache.org/jira/browse/MAPREDUCE-4305 2014-12-13

[31] 夏祎.Hadoop 平台下的作业调度算法研究与改进[D],广州:华南理工大学,2010.

[32] 陈艳金. MapReduce 模型在 Hadoop 平台下实现作业调度算法的研究与改进[D]. 广州: 华南理工大学, 2012

[33] 张密密.MapReduce 模型在 Hadoop 实现中的性能分析及改进优化[D].成都：电子科技大学，2010.

[34] 王凯. MapReduce 集群多用户作业调度方法的研究与实现[D]. 长沙: 国防科技大学, 2010

[35] 马冯.数据密集型计算环境下贝叶斯网络的学习、推理及应用[D].昆明：云南大学，2013.

[36] Kc K, Anyanwu K. Scheduling Hadoop Jobs to Meet Deadlines[C].//Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science(CloudCom'10). Piscataway,NJ:IEEE,2010:388-392.

[37] 余正祥. 基于学习方式对 Hadoop 作业调度的改进研究[J]. 计算机科学, 2012,39(6A):220-222

[38] 顾宇,周良,丁秋林. 基于优先级 Three-Queue 调度算法研究[J].计算机科学,2011,38(B10):253-256.

[39] Sandholm T, Lai K. Dynamic Proportional Share Scheduling in Hadoop[C].//Job Scheduling Straitegies for Parallel Peocessing(JSSPP'10). Berlin:Springer, 2010:110-131.

[40] Ghodsi A, Zaharia M, Hindman B, et al. Dominat Resource Fairness:Fair Allocation of Multiple Resource Types[C].//Proc of the 8th USENIX Symp on NetWorked Systems Design and Implementation(NSDI'11). Berkeley, CA:Usenix Association, 2011.

[41] 许丞,刘洪,谭良. Hadoop 云平台的一种新的任务调度和监控机制[J].计算机科学,2013,40(1):112-117.

[42] 李丽英,唐卓,李仁发.基于 LATE 的 Hadoop 数据局部性改进策略[J].计算机科学,2013,30(3):38-40.