华中科技大学

硕士学位论文

基于MapReduce模型的分布式计算平台的原理与设计

姓名：张文峰

申请学位级别：硕士

专业：软件工程

指导教师：刘小峰

2010-01-16

摘 要

互联网应用的高速发展与应用为企业带来了非常巨大的发展机遇，各种个性化的应用与服务散发着无限魅力。然而随之而来的是海量数据。如何正确而高效的从海量数量中分析有用信息并做出决策是目前所有新型互联网企业必须面对的一件极具挑战性的工作。

传统上，人们往往选择使用分布式计算系统来处理这种复杂而庞大的任务。传统的分布式计算平台往往依赖高端大型服务器，并且需要专业分布式与并行计算的程序员进行长期设计与维护。这往往使得新型互联网企业面临巨大的经济压力。因此，设计一个采用大量廉价机器组成的可扩展的分布式计算平台变得尤其重要。

MapReduce 是一种并行编程模型，它用于处理大型的数据集的程序设计中。基于这种功能的程序能够在大规模的廉价机器上并发地执行任务。基于 MapReduce 编程模型的分布式计算系统解决以下细节：分割输入数据，在集群上的调度，机器的错误处理，管理机器之间必要的通信。这样就可以让没有并行编程经验的程序员利用大量分布式系统的资源了。

基于 MapReduce 编程的优点，本文在分析当前各种分布式计算系统的基础上，设计了一个运行于普通廉价机器上的可扩展的分布式计算平台。首先我们对比当前几种流行的分布式计算技术，总结各自的优缺点，提出了更适合于分析海量数据的分布式计算平台框架。然后从平台总体架构上进行设计，合理地设计了各个功能子模块。我们花了大量笔墨于系统 I/O 模块和 MapReduce 模块，因为系统 I/O 的好坏将直接影响到系统的整体性能。而 MapReduce 模块是整个系统的核心，精心设计的 MapReduce 子模块是对系统良好运行的保证。最后，我们对影响到系统性能的关键策略进行探讨，包括作业和任务调度，容错机制等。

关键词：数据处理　　映射规约　　分布式计算　　任务调度　　容错机制

Abstract

The rapid development of Internet applications brings a plenty of opportunities for enterprise development, and a variety of personalized applications and services were distributed with infinite charm. However, it results in a mass of data. Therefore, it is a challenging job for all new Internet companies to properly and efficiently make decisions from the analysis of massive amount of useful information.

Traditionally, people often choose to use the distributed computing system to deal with such complex and huge task. The traditional distributed computing platform is often dependent on high-end large-scale servers, and needs professional programmers of distributed and parallel computing for long-term design and maintenance. This often makes the new Internet companies feel tremendous economic pressures. Therefore, designing a scalable distributed computing platform composed of a large number of low-cost machines has become particularly important.

MapReduce is a parallel programming model, which can be used to handle large data sets in the process of program design. Programs that are based on this function can be complicated by large-scale low-cost machines to perform tasks. Distributed computing systems based on mapReduce model can be used to solve the problems such as partitioning the input data, scheduling in the cluster, handling machine error, controlling necessary communication among machines. This allows programmers without a parallel programming experience to make use of a large number of distributed system resources.

On the basis of the merits of mapReduce programming, after an analysis of various exsiting distributed computing system, this paper gives an idea to design a common scalable distributed computing platform that runs on low-cost machines. First, we proposed a more suitable framework of distributed computing platform for analyzing mass, after comparing several current popular distributed computing technology and summing up the advantages and disadvantages of each. Then multi- functional sub-modules were properly designed from the whole structure. We wrote a lot in the terms of system I / O modules and MapReduce modules, because the function of system I / O will directly affect the system's overall performance, While the MapReduce module is

the core of the whole system. Well-designed MapReduce sub-module is a guarantee of good functioning of the system. Finally, this paper discussed the key strategies that may affect the system performance, including task scheduling, fault-tolerant mechanisms and so on.

Key words：Data processing MapReduce Distributed computing Scheduling Fault tolerance

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。

本论文属于　保密□，　在_____年解密后适用本授权书。
　　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

1 绪论

1.1 研究背景

在 21 世纪初的高度信息化的社会里，各种各样的信息数据呈指数式增长。特别是随着以电子政务和电子商务为代表的互联网新型应用的不断发展和日益普及，互联网上的电子信息量在爆炸性增长。据统计，纽约证券交易所每天产生 1terabyte（TB）的数据；Internet 主干网存储了大约 2petabytes 数据，并且以每月 20terabytes 的速度增长；阿里巴巴数据仓库[5]数据总量超过了 100TB。因此，对于政府统计部门和大型互联网企业来说，如何对海量的数据进行存储和分析从而提炼出有用信息是一件极具挑战性的工作。

近年来磁盘的存储容量的发展速度大大地快于磁盘的吞吐能力。上个世纪九十年代磁盘的容量为数 GB，而访问速度可达到 4.4MB/s，因此读完全盘数据需要不到 5 分钟的时间。20 年后的今天 TB 级的商业硬盘的访问速度只有 100MB/s，要读完整个磁盘竟然要二个半小时。一种显而易见的方法是同时从多个硬盘中并行地读取数据来提高数据吞吐量。

然而这种多磁盘访问方式却带来两种问题：一是硬件错误。使用多磁盘使得其中一个出错的概论大大增加。防止出错的磁盘上数据丢失的一般方法是使用备份。本文实现的分布式系统所采用的文件系统与此类似。文件系统不是本文件探讨的对象，有兴趣的可以看看 GFS 论文。二是如何对并行访问数据后的得出的结果进行整合。目前虽然有多种分布式系统可以将多源数据进行整合，但是进行正确的整合是极具复杂和挑战的任务。本文件针对分布式计算系统的复杂性，采用 Map Reduce 编程模型[1, 3-4]对多磁盘读写问题进行抽象化，并以此为模型进行设计和实现一个通用的分布式计算平台。

1.2 国内外研究状况

分布式计算[10-12]是一门计算机科学，它研究如何把一个需要非常巨大的计算能

力才能解决的问题分成许多小的部分，然后把这些部分分配给许多计算机进行处理，最后把这些计算结果综合起来得到最终的结果。很多计算任务在概念上很容易理解，然而由于输入的数据量很大，以至于只有计算被分布在成百上千的机器上才能在可以接受的时间内完成。怎样并行计算[15]、分发数据、处理错误，所有这些问题综合在一起，使得原本很简单的计算，需要大量的复杂代码来处理这些问题。

1.2.1 分布式计算系统

分布式计算系统一般要满足以下基本条件：

（1）各计算节点之间能够互相通信；

（2）需要有具体的任务调度策略和容错机制；

（3）各计算节点之间应该相互独立。

分布式计算由于具体的数据存储、分发数据、结果整合等的实现方法不同，而形成相去甚远的系统。下面简单的介绍几种形式的分布式计算系统。

1.2.2 中间件技术

中间件[20-21]是基础软件，是处于操作系统和应用程序之间的软件，也有人认为它应该属于操作系统中的一部分。人们在使用中间件时，往往是一组中间件集成在一起，构成一个平台(包括开发平台和运行平台)，但在这组中间件中必需要有一个通信中间件，即中间件=平台＋通信，这个定义也限定了只有用于分布式系统中才能称为中间件，同时还可以把它与支撑软件和实用软件区分开来。

基于中间件的分布式计算技术以中间件为桥梁，通过对分布式应用的开发者隐藏底层信息，屏蔽网络和分布式应用的复杂性，并为网络和分布式应用提供相应的服务，使得开发者可以集中致力于应用逻辑。中间件通过把数据转移到各个计算节点之处，使得一个网络系统内的所有组件成为一个统一可操作的异构计算系统。

中间件技术出现时仅具有消息通讯和事务管理等简单功能。然而随着应用需求的多样化和广泛性，目前出现了各种各样的中间件系统。如：消息中间件(MOM ：Message-Oriented Middleware)、数据库中间件(Database Middleware)、远程过程调用中间件（RPC：Remote Process Call）、对象请求代理中间件(ORB：Object Request

Broker)和事务处理中间件(TP Monitor：Transaction Process Monitor)等。目前中间件技术已经形成一个完整的技术结构，并正在向上（应用框架和基础服务）和向下（融合操作系统、数据库管理系统的功能）两个方向不断发展，并在应用领域不断地进行拓展新业务。当前，世界各大主流软件提供商都在积极开展自己的中间件服务。

1.2.3 网格计算

网格计算（Grid Computing）[22-23]，简单来说是一种造价低廉而数据处理能力超强的计算模式。网格计算是随着 Internet 的广泛应用而迅速发展起来的，它专门针对复杂科学计算的新型计算模式。

通过 Internet 把分散在不同地理位置的计算机组织为一个巨大的虚拟超级计算机，其中每一台参与计算的计算机就是一个节点，而整个计算系统是由分散在各处的众多节点组成的一张网格，即网格计算。网格计算使得人们能够利用分散各地的资源，完成各多种大规模的、复杂的数据处理任务；同时网格计算能够为充分利用互联网上闲置的处理能力。

显然，网格计算是分布式计算的一种，它发展非常迅速，不仅受到时需要大型科学计算的国家级部门，如航天、气象部门的关注，目前很多大公司也开始追捧这种计算模式。网格计算无疑是分布式计算技术通向计算时代的一个非常重要的里程碑。

网格计算的目的是，通过任何一台计算机都可以提供无限的计算能力，可以接入浩如烟海的信息。这种环境将能够使各企业解决以前难以处理的问题，最有效地使用他们的系统，满足客户要求并降低他们计算机资源的拥有和管理总成本。网格计算的主要目的是设计一种能够提供以下功能的系统：

提高或拓展型企业内所有计算资源的效率和利用率，满足最终用户的需求，同时能够解决以前由于计算、数据或存储资源的短缺而无法解决的问题。

建立虚拟组织，通过让他们共享应用和数据来对公共问题进行合作。

整合计算能力、存储和其他资源，能使得需要大量计算资源的巨大问题求解成为可能。

通过对这些资源进行共享、有效优化和整体管理，能够降低计算的总成本。

1.2.4 P2P 技术

P2P[24-25]即 Peer to Peer，称为对等连接或对等网络。P2P 起源于最初的联网通信方式，如在建筑物内 PC 通过局域网互联，不同建筑物间通过 Modem 远程拨号互联。其中建立在 TCP/IP 协议之上的通信模式构成了今日互联网的基础，所以从基础技术角度看，P2P 不是新技术，而是新的应用技术模式。

简单地说，P2P 就是一种用于不同 PC 用户之间，不经过中继设备直接交换数据或服务的技术，它允许 Internet 用户直接使用对方的文件。每个人可以直接连接到其他用户的计算机，并进行文件的交换，而不需要连接到服务器上再进行浏览与下载。目前 Internet 的存储模式是“内容位于中心”，而 P2P 技术的运用将使 Internet 上的内容向边缘移动。这将带来以下改变:首先，客户不再需要将文件上传到服务器，而只需要使用 P2P 与其他计算机进行共享；其次，使用 P2P 技术的计算机不需要固定的 IP 地址和永久的 Internet 连接，这使得占有极大比例的拨号上网用户也可以享受 P2P 带来的变革。

P2P 技术可以让用户可以直接连接到其他用户的计算机，进行文件共享与交换，同时 P2P 在深度搜索、分布计算、协同工作等方面也大有用途。图 1.1 是目前常见的 P2P 部署结构。

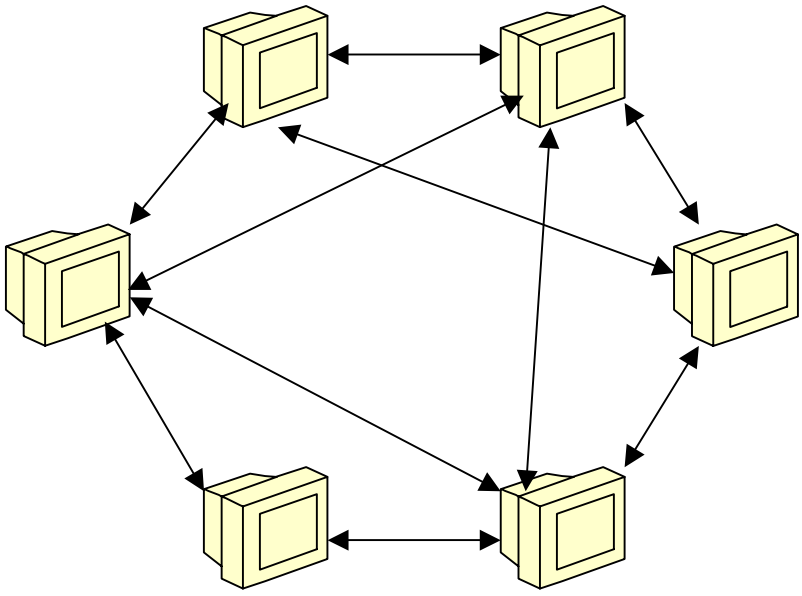

图 1.1 P2P 分布图

1.2.5 Volunteer 计算

目前国际上流行的 Volunteer 计算主要有 SETI@home、Folding@home 等项目。SETI（the Search for Extra-Terrestrial Intelligence）是目前最出名的 Volunteer 计算项目。该项目志愿者捐出自己计算机 CPU 空闲时间来计算和分析外太空射电信息来寻找外星生命迹象。其他项目包括解决较为复杂的数学问题，例如寻找最大的梅森素数 GIMPS；研究寻找最为安全的密码系统，例如密码破解项目 RC-72；生物病理研究，例如研究蛋白质折叠，误解，聚合及由此引起的相关疾病的 Folding@home；各种各样疾病的药物研究，例如寻找对抗癌症的有效药物 United Devices 等。

Volunteer 计算把一个巨大的计算任务分成很多任务小块，称为工作单元，然后把各个工作单元分发到全球各个志愿者计算机上进行计算与分析。例如，SETI@home 的工作单元为约 0.35MB 的外射电信息数据，在普通家用计算机上要运行数小时。当志愿者完成任务时计算机把结果传回给服务器。为了进行容错处理和预防欺骗，每个工作单元分发到三个不同的志愿者，只有当两个以上的计算结果一致时才会被服务器接受。

Volunteer 计算与 MapReduce 模型有很大的相同之处。本文将在下一章讲述 MapReduce 编程模型的原理，并分析与上面各种现有流行的分布式计算系统间的优缺点。

1.3 本文的主要内容和工作

本文针对目前各种流行的分布式计算平台各自的优缺点，在借鉴传统上的 MapReduce 编程模型的基础上对分布式计算进行深入研究，设计并实验性实现了一个基于 MapReduce 模型的分布式计算平台。对分布式系统上的调度及其容错机制进行细致的研究并提出改进方案。最后通过实际应用对该分布式计算平台性能进行分析。

1.4 论文组织

本文分为 6 章：

第一章是绪论，简要介绍了数字时代数据的海量的特点。并由此引出了分布式计算系统的相关概念。介绍并分析了当前流行的几种分布式计算平台。介绍了本文的主要内容和工作。

第二章介绍了 MapReduce 相关原理。详细说明了 MapReduce 的背景、编程模式、实现框架、用途和影响。

第三章对分布式系统从全局的高度进行设计。详细描述了各个系统模块和详细。包括数据存储、分发数据、结果整合等。

第四章针对影响系统性能瓶颈的关键策略进行详细而全面的论述，包括分布式计算系统的作业和任务的调度策略以及容错机制等。

第五章通过现有的实验环境条件，设计了一个简单的应用程序来对分布式计算平台进行性能分析。

第六章对研究进行总结和展望。总结了本文的工作和创新点，指出了需要进一步深入的工作。

2 MapReduce 模型概述

2.1 编程模型

MapReduce 是一个用于数据处理的编程模型[1]，它简化了复杂的数据处理计算过程：它将数据处理过程分为两个阶段，即 map[1]阶段和 reduce[1]阶段。每个阶段都将一系列 key/value 对作为输入和输出，其中的键和值的类型为 MapReduce 用户指定。用户同时指定两个函数：map 函数和 reduce 函数。

用户自定义的 map 函数，接受一个输入 key/value 对，然后产生一系列临时中间 key/value 对。我们把所有具有相同中间 key 的临时 key/value 对聚合在一起,然后把它们传递给 reduce 函数。

用户自定义的 reduce 函数，接受一个中间 key 和相关的一个 value 集。它合并这些 value，形成一个比较小的 value 集。通常，每次 reduce 调用只产生 1 个输出 value。

以这种函数式编写的程序能自动的在大规模的普通机器上并行的执行。我们设计基于 MapReduce 模型的分布式系统时要特别关注以下细节：分割输入数据；在机群上的执行调度；机器的错误处理；管理机群内机器之间必要的通信。这样就允许系统用户在没有任何并行或分布式系统经验的情况下容易地利用大量分布式系统的资源。下面将列举描述这个模型实际如何工作。

2.2 气象应用

在实现生活中，气象数据的处理是在非常复杂的。全球各地气象站收集每小时的数据总量是惊人的。如何对这些海量数据进行分析从而对气象进行预测足以对巨型计算机都形成挑战。气象数据格式基于行存储的，一行代表一条气象数据。为了简单起见，这里仅仅用 MapReduce 分析某年份的最高气温。

2.2.1 用 MapReduce 进行分析

Map 阶段的输入为一个或一组气象原始数据。由于气象数据格式基于行，因此，我们选择以而行的内容作为 value 的值，行的偏移量作为 key 的值。因为这里 key

的值没有意义，因此我们忽略其值。Map 输出以年份和气温作为 key 和 value 的类型。在这里，Map 阶段只是作为数据准备阶段。而 Reduce 阶段对 map 阶段的输出数据进行处理，找出每个年份的最高气温。Reduce 输入类型和 map 阶段输入类型一样，而输出类型分别为：年份和气温。

假设以下为原始数据：

0067011990999991950051507004...9999999N9+00001+99999999999...

0043011990999991950051512004...9999999N9+00221+99999999999...

0043011990999991950051518004...9999999N9-00111+99999999999...

0043012650999991949032412004...0500001N9+01111+99999999999...

0043012650999991949032418004...0500001N9+00781+99999999999...

Map 阶段的输入为：

(0, 0067011990999991950051507004...9999999N9+00001+99999999999...)

(106, 0043011990999991950051512004...9999999N9+00221+99999999999...)

(212, 0043011990999991950051518004...9999999N9-00111+99999999999...)

(318, 0043012650999991949032412004...0500001N9+01111+99999999999...)

(424, 0043012650999991949032418004...0500001N9+00781+99999999999...)

Map 函数只是简单的提取年份和气温数据，输出为：

(1950, 0)

(1950, 32)

(1950, −11)

(1949, 36)

(1949, 27)

Map阶段的输出在被送到Reduce阶段时一般要进行排序与分组，reduce阶段的输入为：

(1949, [36, 27])

(1950, [0, 32, −11])

Reduce 函数遍历各 key 的列表，计算出最大的气温：

(1949, 36)

(1950, 32)

2.2.2 伪码实现

以上简单地描述了整个 MapReduce 工作流程。图 2.1 给出数据流程图。

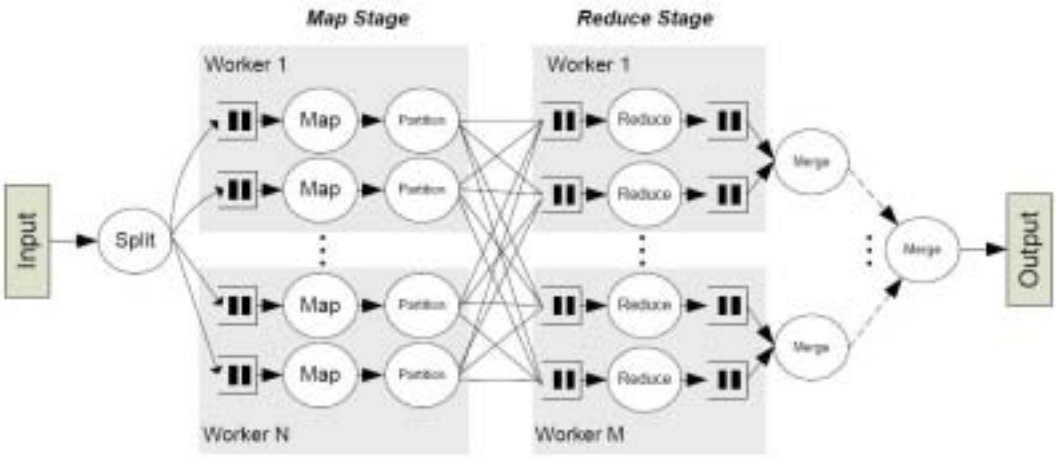

图 2.1 MapReduce 数据流程图

以上描述可以用下面的伪码表示：

//key：文件偏移量

//value：一条气象记录，也就是一行数据

Map (int key, String value):

ClawYear (value,&year,pos);

ClawTemp (value,&temp,pos);

Collect (year,temp);

//key：年份

//value：气温列表

Reduce(int key,Iterator value):

Traverse each temperature in value:

max(temperature,…);

需要说明的是：map 和 reduce 间的输入的键值和输出的键值可以属于不同的域的。而中间 key, value 和输出 key，values 的域相同。具体的有着如下的关系：

map (k1,v1) ----> list(k2,v2)

reduce (k2,list(v2)) ---->list(v2)

2.3 分布性和可靠性

在 Map 阶段，通过把输入数据进行分割，这样就可以把 map 操作分布到多台机器上进行运行。数据输入块能够在不同的机器上并行处理。Reduce 操作是通过对中间产生的 key 的分布来进行分布的，中间产生的 key 可以根据某种分区函数进行分布(比如 hash(key) mod R)，分布成为 R 块。分割数量(R)和分割函数由用户来指定。

图 2.2 是我们实现的 MapReduce 操作的整体数据流.当用户程序调用 MapReduce 函数，就会引起如下的操作。

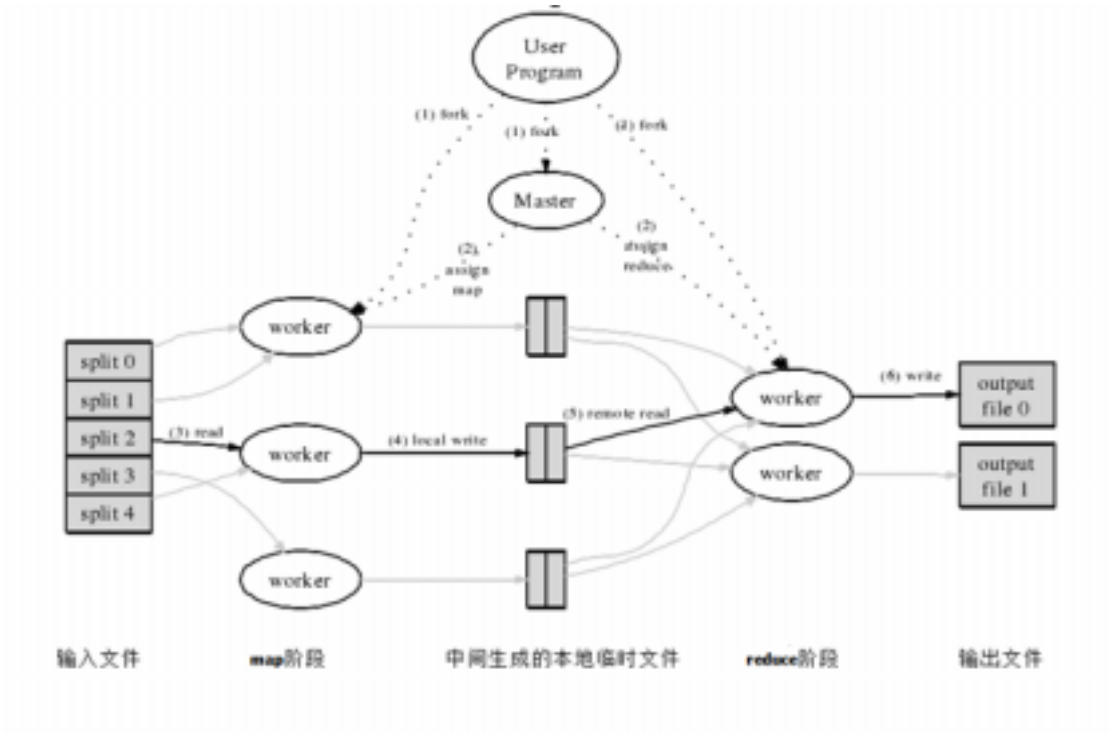

图 2.2 执行流程图

MapReduce 通过把对输入数据集的大规模操作分发给网络上的每个计算节点来实现可靠性；每个节点会周期性的把完成的工作和状态的更新报告回来。如果一个节点保持沉默超过一个预设的时间间隔，主节点记录下这个节点状态为死亡，并把分配给这个节点的数据发到别的节点上进行重新计算。

化简操作工作方式很类似，但是由于化简操作在并行能力较差，主节点会尽量把化简操作调度在一个节点上，或者离需要操作的数据尽可能近的节点上了；另一

方面，由于化简操作的输入数据一般比 Map 操作的少得多，因此化简操作的节点往往比 Map 操作所需的节点要少。如果要将最终结果进行汇总，我们一般只取 reduce 节点数为 1。

2.4 与其他分布式系统的比较

表面上看，基于 MapReduce 模型的分布式系统似乎和第一章简述的几种其他形式的分布式系统极为相似。然而细微的设计和目标差别往往使得各种分布式系统有着巨大的不同。下面我们就把基于 MapReduce 模型的分布式系统与其它几种流行的分布式系统进行比较。

2.4.1 与中间件技术比较

为解决分布异构问题，人们提出了中间件的概念。中间件是位于平台(硬件和操作系统)和应用之间的通用服务。这些服务具有标准的程序接口和协议。针对不同的操作系统和硬件平台，它们可以有符合接口和协议规范的多种实现。中间件是构件化软件的一种表现形式。中间件抽象了典型的应用模式，应用软件制造者可以基于标准的中间件进行再开发，这种操作方式其实就是软件构件化的具体实现。中间件产品对各种硬件平台、操作系统、网络数据库产品以及 Client 端实现了兼容和开放。图 2.3 为中间件的模型图。

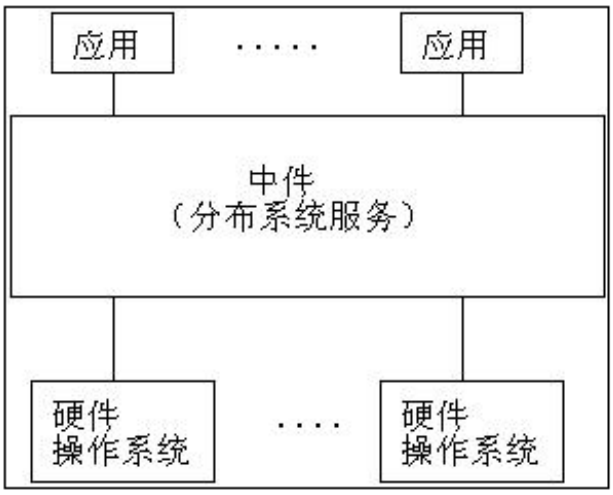

图 2.3 中间件模型图

与中间件关注异构的软硬件不同，MapReduce 模型更关注计算本身的分割和整合。用于布署 MapReduce 的各节点往往需要相同的软硬件环境。这是不同是由于不同的应用需求决定的。中间件广泛用于 INTERNET 上，因此必须对各种硬件平台、操作系统、网络数据库产品以及 Client 端实现兼容和开放。其次，应用于 INTERNET 上的中间件往往由于带宽的原因不能满足那些对运行时间有着严格限制的任务。而 MapReduce 往往布署在企业的内部网络中，因此能够满足很多实时性的要求。

2.4.2 与网格计算比较

高性能计算（HPC）和网格计算用于大规模数据处理已经有几年了。通常，高性能网格计算平台通过存储域网络（Storage Area Network，SAN）提供的共享的文件系统来把计算任务分布到集群中各个节点上。因此这种计算模式特别适合那些计算紧凑型的任务，而不适合那些各节点间有大量数据迁移的计算。因为网络带宽在这种情况下往往成为瓶颈，以至于很多计算节点处于空闲状态。

而 MapReduce 系统因为往往是在内部网络构建的，因此其数据访问速度是非常快的。并且系统会采用一种数据本地化的策略是计算节点和存储节点尽量在同一个节点上。简单的说，MapReduce 更倾向于计算的移动而不是数据的移动。同时，MapReduce 系统的用户会发现自己完全不需要了解任何分布或并行编程的知识，而网格计算往往需要借助 MPI（Message Passing Interface）编程。因此，MapReduce 系统比网格计算易用性更好，但也失去了一定的通用性。

2.4.3 与 P2P 比较

目前 P2P 技术已经延伸到几乎所有的网络应用领域，如分布式科学计算、文件共享与下载、流媒体直播与点播、语音通信及在线游戏支撑平台等方面。然而在 P2P 技术的发展道路上，有许多尚待解决的问题。版权问题一直是 P2P 发展的一个不确定因素，如何在技术层面支持合法文件的分发是需要解决的重要问题。安全问题也是 P2P 领域的重要研究课题，如何在 P2P 网络中实现数据存取安全、路由安全、用户身份认证和身份管理都需要进一步研究。此外，如果能够实现 P2P 应用之间的统一资源定位，统一路由，使得 P2P 技术有一个统一开发标准，那么就能够融合 P2P 技术，提升 P2P 应用的整体性能。

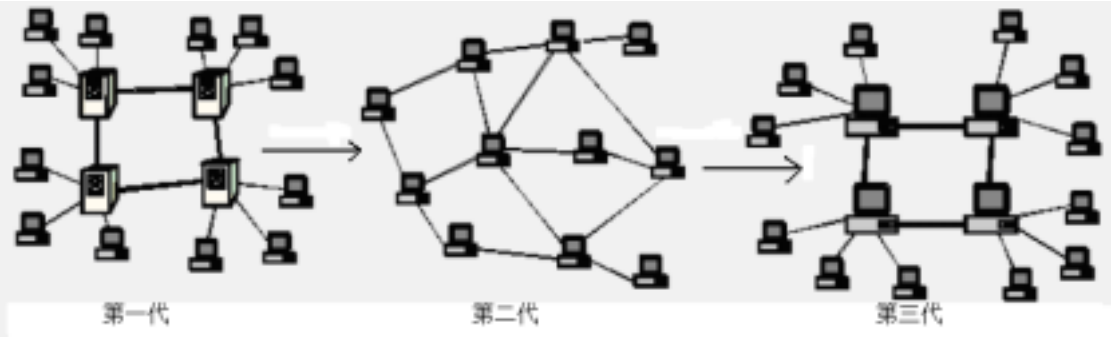

图 2.4　P2P 体系的发展

P2P 技术与 MapReduce 模型的差别就相当大了。MapReduce 系统中节点分为任务分配节点 Master 和任务计算节点 Slave。系统通常只有一个 Master， Master 主要负责数据和计算程序的分发、调度及容错处理等。而 Slave 节点只与 Master 进行通信，而不能感知其他 Slave 的存在的。这点和 P2P 技术有着本质的区别。显然易见，P2P 技术在数据分发方面有着极高的性能，因此目前 P2P 技术大行其道。国内软件诸如迅雷下载工具，**PPLive** 流媒体直播与点播，Skype 语音传输等都可见到其踪影。P2P 这种充分利用各节点间的通信的优点，在遇到各节点间任务相互独立时却显得力不从心了。这也是目前 P2P 技术局限于数据分发方面应用的原因。

2.4.4　与 Volunteer 计算比较

直观来看，MapReduce 模型和 Volunteer 计算表现形式极为相似：两者都对数据进行分割，并对计算程序和分割数据分发到各节点进行并行计算，然后传回计算结果。然而由于两者在实际应用目标上还是有着一些重大的差别。例如 SETI@home 项目是一个 CPU 紧凑型的任务。分析外太空射电信息需要大量的计算工作量，因此特别适合将其运行在全世界数以亿计的 CPU 上。相比于计算任务耗费的时间，分割数据的传输时间就显得不那么重要了。因此，志愿者捐献的是 CPU 周期，而不是网络带宽。

MapReduce 系统分割数据的一个重要因素就是运行时限性。MapReduce 用户往往要求分割的任务能够在可信赖的时间内完成，因此节点间的传输带宽就显得特别重要了。相比之下，运行在 Internet 上各节点的 Volunteer 项目往往不可能对运行时间进行严格控制和依赖了。

总而言之，各个分布式系统之间都有着一定的相似之处。然而由于设计和应用目标的不同，各系统实现间的表现和运行差别就非常巨大了。

2.5 实际应用

MapReduce 编程模型最先由 Google 公司提出和实现的。如今 MapReduce 在在 Google 内部很多应用领域都有广泛的应用。包括：

（1）大规模的机器学习问题。

（2）Google News 和 Froogle 产品的集群问题。

（3）从公众查询产品(比如 Google 的 Zeitgeist)的报告中抽取数据。

（4）从 web 网页作新试验和抽取新的产品(例如,从大量的 webpage 中的本地查找抽取物理位置信息)。

（5）大规模的图型计算。

紧随 Google 之后的是著名的开源项目 Hadoop。搜索技术界和开源社区中大名鼎鼎的 Doug Cutting 一手倡导和组织了 Hadoop 社区。2006 年 Hadoop 从 Nutch 中分离出来，正式成为 Apache 组织中一个专注于 DFS 和 MapReduce 的开源项目。Hadoop 如火如荼地发展到现在，已经成为一个可以更容易开发和并行处理大规模数据分布式平台，未来几年内它可能具有与 Google 系统架构技术相同的竞争力。

Hadoop 如今已经相当成熟，众多国际公司更是直接参与或应用 Hadoop。2008 年雅虎宣布搭建出一个世界上最大的基于 Hadoop 的生产集群系统 Yahoo! Search Webmap，即雅虎网页搜索抓取的所有站点和网页其及关系的数据库。在雅虎，Hadoop 目前除了被用于网页搜索中的 Webmap 中，还广泛地被用到雅虎的日志分析、广告计算、科研实验中。

Amazon 的搜索门户 A9.com 中的商品搜索的索引生成就是基于 Hadooo 完成的。另外，Amazon 发布的 GrepTheWeb Web Service 内部使用了基于 EC2（Elastic Compute Cloud）的 Hadoop 集群，承担其中的并行计算工作。

著名 SNS 网站 Facebook 用 Hadoop 构建了整个网站的数据仓库，它目前有 320 多台机器进行网站的日志分析和数据挖掘。此外，在 IBM2007 年底的蓝云计

算集君中也采用了 Hadoop 进行并行计算。国内企业如阿里巴巴也使用 Hadoop 分析其电子商务的日志。本文设计的分布式计算平台也在很多方面借鉴 Hadoop 的成功经验。

3 分布式计算系统分析与设计

上一章全面论述了 MapReduce 模型的原理，并以气象应用方面的简单示例描述了 MapReduce 运行时简单的工作流程。本章在此基础上对分布式系统进行论述。首先，我们会介绍此系统的整体架构和计算流程，然后详细介绍系统各子模块的结构。并在下章对分布式系统的一些影响系统性能瓶颈的关键策略进行详细而全面的论述。

3.1 总体结构

我们设计的分布式计算系统采用 master/slave 架构，即由一个中心服务器 JobTracker 和一定数目的任务服务器 TaskTracker 组成。JobTracker 负责管理运行在此框架下所有作业的，也是调度各个作业分配任务的核心。为了简化 master 间高度任务的同步复杂性，一个分布式系统的 JobTracker 是作为单节点存在的。TaskTracker 具体负责执行用户定义的操作，每一个作业被拆分成很多的任务，包括 Map 任务和 Reduce 任务等。任务是具体执行的基本单元，它们都需要分配到合适任务服务器上去执行，任务服务器一边执行一边以发送心跳消息的方式向作业服务器汇报各个任务的状态，以此来帮助中心服务器了解作业执行的整体情况，分配新的任务等等。每个 TaskTracker 只直接和 JobTracker 通信，它们互相之间是不可能感应对方的。这样也是为了保证各任务间的独立性，便于后面的容错处理。

我们的分布式系统采用了协议接口来进行 master 和 slave 的交流。实现者作为 RPC 服务器，调用者经由 RPC 的代理进行调用，如此完成大部分的通信，具体服务器的架构和其中运行的各个协议状况，参见图 3.1。由于这当中涉及过多的网络知识，本文不再就此进行赘述了。

除了以上讨论的 master 和 slaves，我们的分布式系统同时提供了一个客户端接口，用于平台客户向我们的分布式系统提交作业。客户提交作业后，只需要监视和等待作业的完成即可，其余工作全部由分布式系统后台执行。

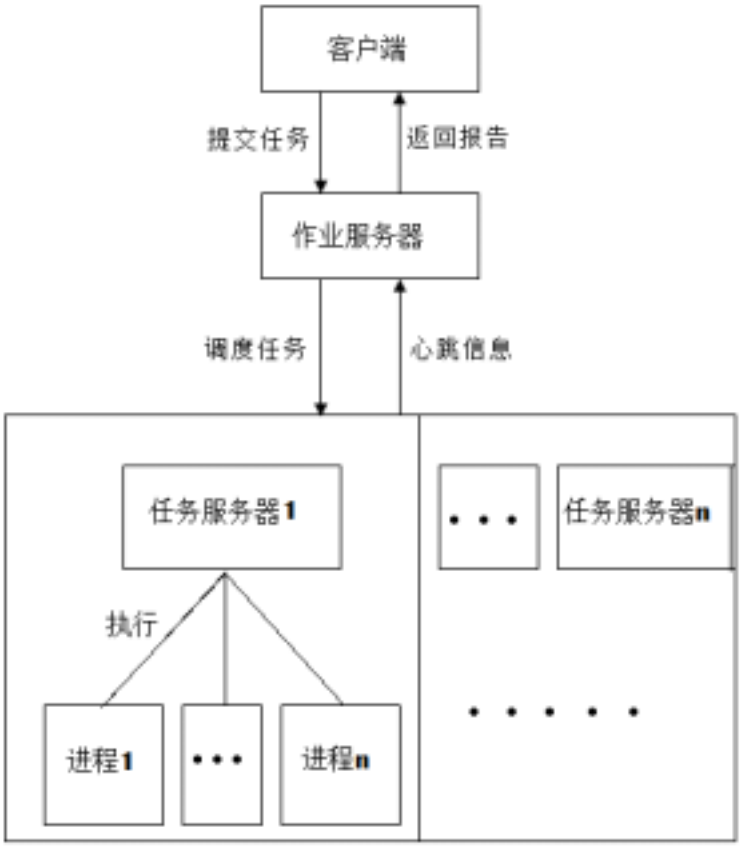

图 3.1　系统架构图

因为中心服务器要为每个 map 和 reduce 任务存储它们的完成状态（idle，in-progress 或者 completed），并且要能识别出不同的节点，所以中心服务器需要保存一定的数据结构。中心服务器就像一个管道，通过它，中间文件区域的位置从 map 任务传递到 reduce 任务。因此，对于每个完成的 map 任务，中心服务器存储由 map 任务产生的 R 个中间文件区域的大小和位置。对于这个位置和大小信息是当接收到 map 任务被动局面得时候做的。这些信息是增量推送到处于 in-progress 状态的 reduce 任务的节点上的。

3.2　计算流程

整个作业的计算流程从客户端开始，客户通过系统提供的接口进行作业提交，整个流程如下：作业的提交 -> Map 任务的分配和执行 -> Reduce 任务的分配和执

行 -> 作业的完成。而在每个任务的执行中，又包含输入的准备 -> 算法的执行 -> 输出的生成，三个子步骤。下面将详细地介绍各个子阶段的具体流程及相关设计理由。具体过程如图 3.2 所示。

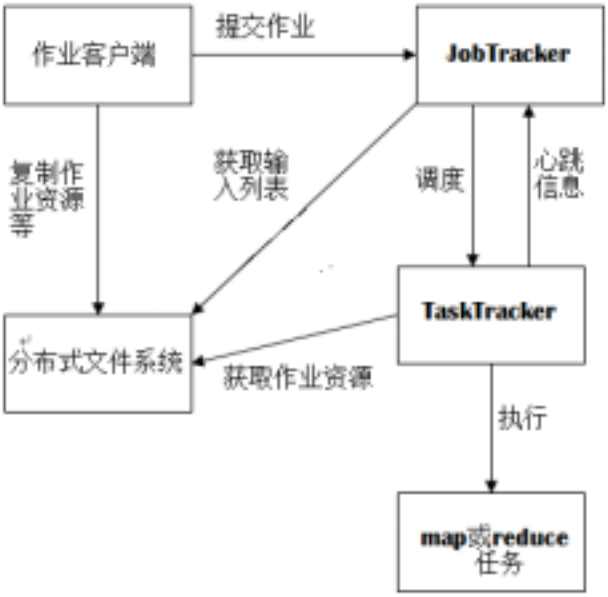

图 3.2　MapReduce 计算流程图

3.2.1　作业的提交与初始化

客户通过系统提供的接口进行作业提交时，需要现实两个接口：Mapper 和 Reducer。这两个接口分布需要用户实现两个函数：也即是我们前面谈到的 map 函数和 reduce 函数。作为通用的计算框架，需要面对的各种不同的问题、不同的输入、不同的需求，因此我们系统内置了多种数据类型供用户使用。用户在定义 map 函数和 reduce 函数时可以自由地选择内置中的类型来作为键和值的类型。用户书写好所有自定的代码后，只要进行简单的一些配置即可提交作业了，我们要求用户采用 XML 文档来提交详细配置项。这些配置包括 master 和 slave 地址，输入输出的文件路径，Map 和 Reduce 的并行数目，甚至包括读取写入文件所需的格式支持类，等等。大部分配置都有默认值，因此用户只需简单配置即可。

最后，用户通过调用 JobClient 的 runJob 方法，提交此作业。至此，所有的工作都将转移到我们的分布式系统中。用户只需监督作业的执行，等待作业完成即可。

这里需要说明的是：当用户提交后作业后，分布式系统会分发作业定义的代码到各个 TaskTracker 中。JobTracker 根据用户的配置会将作业的输入进行分割，然后对 map 任务进行调度。具体来说 runJob 方法需要做如下的事情：

（1）向 JobTracker 申请一个新的作业编号 jobID。

（2）检查作业配置的输出目录。如果用户没有指定输出目录或指定的目录已存在（避免覆盖用户已存在目录中的数据），作业不会被提交，相反，系统将会抛出异常指出输出目录错误。

（3）根据设定的 map 任务数量将输入数据进行分割。如果数据无法分割，例如没有指定输入目录，作业不会被提交，相反，系统将会抛出异常指出输出目录错误。

（4）将作业所需要的资源复制到分布式文件系统中，资源包括用户配置文件，用户实现 map 和 reduce 的源文件，分割的数据块等。

（5）通知 JobTracker 作业已准备就绪，等待执行。

当 JobTracker 收到客户端提交作业后，JobTracker 将该作业放入一个内部作业队列中。作业调度器将调度将作业并完成初始化。初始化时 JobTracker 会创建一个作业对象来表示正在运行的作业，该对象封装了该作业的任务数目类型，个数以及一些方法来实时获取任务的状态和进度。

为了建立作业的一系列子任务（map 任务或 reduce 任务），作业高度器首先要遍历先前计算好的数据分割块。一个数据分割块对应一个 map 任务。而 reduce 任务由用户配置文件中指定，我们系统默认 reduce 数目为 1。调度器据此建立一系列 reduce 任务。

3.2.2 Map 任务的分配与执行

当一个作业提交到了作业服务器上，作业服务器会生成若干个 Map 任务，每一个 Map 任务，负责将一部分的输入转换成格式与最终格式相同的中间文件。通常一个作业的输入都是基于分布式文件系统的文件，因为它可以很天然的和分布式的计算产生联系。本系统采用本实验室自行开发实现的分布式文件系统 KiddenFS。KFS 的设计目标之一便是基于大文件存储与分析，因此能很好地满足本系统对文件系统

的需要。

对于一个 Map 任务而言，它的输入往往是输入文件的一个数据块，或者是数据块的一部分，但通常不跨数据块。因为一旦跨了数据块，就可能涉及到多个服务器，带来了不必要的复杂。当一个作业从客户端提交到了作业服务器上，作业服务器会生成一个 JobInProgress 对象，作为与之对应的标识，用于管理。作业被拆分成若干个 Map 任务后，会预先挂在作业服务器上的任务服务器拓扑树。这是依照分布式文件数据块的位置来划分的，比如一个 Map 任务需要用某个数据块，这个数据块有三份备份，那么，在这三台服务器上都会挂上此任务，可以视为是一个预分配，这是为容错处理准备的。关于任务管理和分配的大部分的真实功能和逻辑的实现，则依 TaskScheduler 子类。TaskScheduler，顾名思义是用于任务分配的策略类。任务分配是一个重要的环节，所谓任务分配，就是将合适作业的合适任务分配到合适的服务器上。这里面蕴含了两个步骤，先是选择作业，然后是在此作业中选择任务。和所有分配工作一样，任务分配也是一个复杂的活。不良好的任务分配，可能会导致网络流量增加、某些任务服务器负载过重效率下降，等等。不仅如此，任务分配还是一个无一致模式的问题，不同的业务背景，可能需要不同的算法才能满足需求。关任务调度将在下一章进行详细地论述，在此就不赘述了。

对于作业服务器来说，把一个任务分配出去了，并不意味着它的工作已经完成了。因为任务可以在任务服务器上执行失败，可能执行缓慢，这都需要作业服务器帮助它们重新执行。因此在 Task 中，记录有一个 TaskAttemptID，对于任务服务器而言，它们每次运行其实都只是一个 Attempt 而已，Reduce 任务只需要采信其中一个的输出。

任务服务器是通过心跳消息，向作业服务器汇报此时此刻其上各个任务执行的状况，并向作业服务器申请新的任务的。

3.2.3 Reduce 任务的分配与执行

比之 Map 任务，Reduce 的分配极其简单，基本上是所有 Map 任务完成了，有空闲的任务服务器，来了就给分配一个 Reduce 任务。因为 Map 任务的结果星罗棋布，且变化多端，真要搞一个全局优化的算法，绝对是得不偿失。而 Reduce 任务的

执行进程的构造和分配流程，与 Map 基本完全的一致。但其实，Reduce 任务与 Map 任务的最大不同，是 Map 任务的文件都在本地存储，而 Reduce 任务需要到处采集。这个流程是作业服务器经由此 Reduce 任务所处的任务服务器，告诉 Reduce 任务正在执行的进程，它需要的 Map 任务执行过的服务器地址，此 Reduce 任务服务器会于原 Map 任务服务器联系来获取数据。这个隐含的直接数据联系，就是执行 Reduce 任务与执行 Map 任务最大的不同了。

3.2.4 作业的完成

当 jobtracker 收到一个通知表明最后一个任务完成时，所需数据都写到了分布式文件系统上，整个作业才正式完成了，此时 jobtracker 将 job 的状态设为 successful。此时，jobClient 查询作业状态时，即可知作业已成功完成了，此时将打印一条成功信息给客户端。之后，jobtracker 将清除整个作业内包括 tasktracker 中内的所需要的中间状态和信息。

3.3 系统 IO 模块

分布式计算系统处理的是大规模海量的原始数据，并且各系统节点间数据传输量非常巨大，因此系统 IO 模块的设计将直接影响到系统的最终性能。本系统的 IO 模块主要包括基本输入输出流、序列化接口、数据压缩以及一些内置工具等组成，如图 3.3 所示。本节就系统的数据整合、数据压缩、数据的序列化和基本输入输出等进行详细说明。

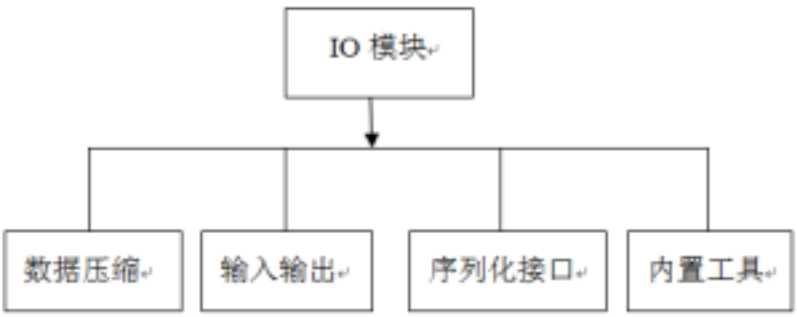

图 3.3 I/O 模块结构图

3.3.1 基本输入输出流

MapReduce 模型的数据处理过程都依赖于 map 任务和 reduce 任务的输入和输出。输入和输出的类型即组成一个健值对。在第二章我们简单地以下列形式表示输入和输出。

map (k1,v1) ----> list(k2,v2)

reduce (k2,list(v2)) ---->list(v2)

我们的分布式系统的输入输出继承图如图 3.4、图 3.5 所示。

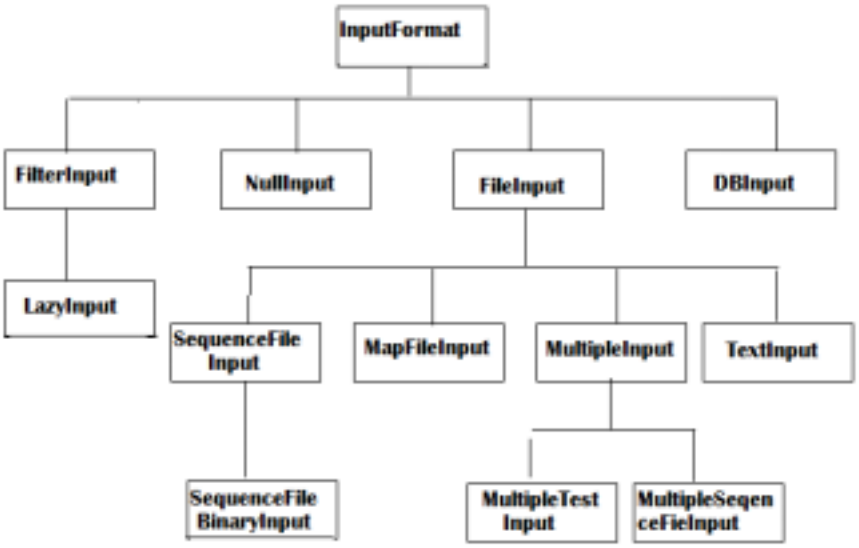

图 3.4 输入继承图

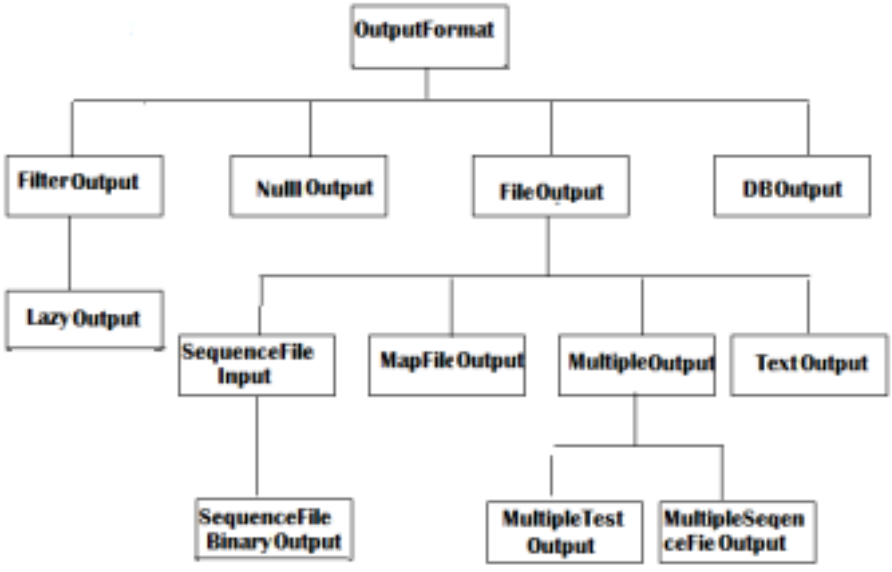

图 3.5 输出继承图

3.3.2 数据完整性

系统用户绝不希望看到系统在存储或处理他们的数据时发生数据丢失或损坏。然后一般的基于磁盘或是网络的 IO 操作在数据读写的时候却存在较低的概率丢失或破坏数据。当分布式系统采用这样的 IO 操作时，由于系统各节点间有大量的数据转移和读写，发生错误的概率将大大地增加。因此，我们的分布式计算平台有必要进行 IO 控制，确保用户的数据的无错误运行。

目前通常用于远距离通信中保证数据的完整性和准确性的是计算检验和（checksum）。当数据首次提交给系统时，系统计录下数据的检验和。当数据要系统内部进行分发和转移时，再次记录检验和。当两次记录的检验和不能精确的

匹配时，我们即认为数据被破坏了。这项技术不提供任何方式来修复数据而只是进行错误检测。常用的检验和是循环冗余码 CRC-32。CRC-32 码对原始数据进行某些运算得到一个 32 位的整形数字，大都被采用在一种称为 Point-to-Point 的同步传输中。

MD5 的全称是 Message-digest Algorithm 5（信息-摘要算法），用于确保信息传输完整一致。MD5 算法的描述和 c 语言源代码在 Internet RFC 1321 中有详细的描述。当用户输入原始数据时，系统检测是否附带有 MD5 验证码。如果有 MD5 码，系统会使用 IO 内置的 MD5 算法计算出接收的数据的 MD5 并进行匹配，匹配相同系统则认为原始数据提交成功。如果没有附带 MD5 码，系统则默认接收的数据准确无误。因此我们强烈建议用户提交数据时附带 MD5 码。确认用户原始数据提交后，我们的分布式计算平台直接将原始数据计算的校验和透明地写进输入文件。我们系统默认每隔 512 个字节计算一个校验和，这个值可以通过配置 XML 文件进行设置。因为一个校验和占 4 个字节，因此整个系统的数据冗余不超过 1%。

任务服务器 TaskTracker 负责验证它从客户端或其他节点分发的数据。确认无误后才进行数据的储存和计算，否则直接抛弃并将结果反馈给作业服务器。

3.3.3 数据压缩

文件压缩至少有两个好处：减少所需要的存储空间以及减轻网络文件传输负荷。当处理大规模数据时，效果尤其明显。所以分布式计算平台应当尽可能地考虑如何

在系统中使用数据压缩技术。

目前有很多不同的压缩格式，工具和算法，每种都有不同的特征。表 3.1 列出了目前常用的一些压缩格式。

表 3.1　常见压缩格式

压缩格式	压缩工具	压缩算法	文件后缀名	支持多文件	是否可分割
DEFLATE	N/A	DELATE	.delate	否	否
Gzip	Gzip	DELATE	.gz	否	否
Zip	Zip	DELATE	.zip	是	是
Bzip2	Bzip2	Bzip2	.bz2	否	是
Lzo	Lzop	LZO		否	否

我们目前采用的是 zlib 压缩函数库。zlib 是提供数据压缩的函数库，由 Jean-loup Gailly 与 Mark Adler 所开发，初版 0.9 版在 1995 年 5 月 1 日发表。zlib 使用 DEFLATE 算法，最初是为 libpng 函式库所写的，后来普遍为许多软件所使用。由于 zlib 压缩函数库为自由软件，使用 zlib 授权。我们不再对 zlib 进行更详细地介绍了。我们对 zlib 函数库进行简单面向对象地封闭，以便更方便地进行文件压缩。需要说明的是：所有的压缩算法都需要在时间和空间上权衡，更快地压缩和解压往往使得压缩率很低。我们默认地使用中等速度进行压缩，这样能在时间和空间上达到某种最优。用户可以在配置文件中进行设定，压缩级别分为九种：越高级别表示压缩率最高，但所需要的时间也是最长的，其中 0 级只是进行简单地数据格式包装，没有应用任何压缩算法，因此速度也是最快的。

当分布式系统检测出输入文件是压缩类型时，它会自动地进行解压缩。根据文件后缀名来决定使用哪种压缩编码。值得一提的是，即使用户提交给分布式系统的是未压缩的文件，分布式系统将 map 阶段的输出文件进行压缩依然能够提高性能。因为 map 任务的输出被写出磁盘并通过网络传输给 reducer 任务节点，通过压缩可以极大地减少网络传输所需要的时间，由于网络传输往往是系统运行性能的瓶颈之一，因此压缩能在很大程度上提高系统性能。

3.3.4 对象序列化

序列化是将结构化的对象状态转换为可持久性或传输的字节流格式的过程。与序列化相对的是反序列化，它将字节流转换为一系列结构化的对象。这两个过程结合起来，可以轻松地存储和传输数据。序列化的目的有两个：一是以某种存储形式使自定义对象持久化；二是将对象从一个地方传递到另一个地方。

通常有两种序列化技术：二进制序列化和 XML 序列化。二进制序列化保持类型保真度，这对于在应用程序的不同调用之间保留对象的状态很有用。例如，通过将对象序列化到剪贴板，可在不同的应用程序之间共享对象。您可以将对象序列化到流、磁盘、内存和网络等等。远程处理使用序列化“通过值”在计算机或应用程序域之间传递对象。XML 序列化仅序列化公共属性和字段，且不保持类型保真度。当您要提供或使用数据而不限制使用该数据的应用程序时，这一点是很有用的。由于 XML 是一个开放式标准，因此，对于通过 Web 共享数据而言，这是一个很好的选择。SOAP 同样是一个开放式标准，这使它也成为一个颇具吸引力的选择。显然我们的系统要使用二进制进行序列化，我们要保持类型高保真度地转移。一般而言，序列化要尽可能满足以下设计原则：

（1）紧凑：紧凑型的格式能最大限度地使用网络带宽，而网络带宽往往是数据中心最重要最稀缺的资源，这也是我们选用二进制序列化的最大原因。

（2）快速：进程间通信构成了分布式系统的骨干，因此它是必须有尽可能少的执行序列化开销和反序列化的过程。

（3）可扩展：系统应该随着时间的推移不断满足新的需求，因此应该有一种可控的接口来管理客户端和服务端对未来数据类型序列化的要求。

依照以上的设计原则，我们的分布式计算平台针对目前各种主流语言所支持的数据类型进行序列化。这些序列化都必须实现一个公共的接口，即 Writable 接口。这个接口有两个方法需要用户实现，即 write 和 readFields 方法。分别对应数据类型的序列化和反序列化。用户只要继承并实现这两个方法，即可轻松地按自己的意愿对任何数据类型进行序列化了。这也是分布式系统强大的扩展性的表现之一。

当我们想用来储存 key 时，我们应该继承 WritableComparable 接口，因为分布

式系统在 MapReduce 阶段会按 key 值进行排序。因此 WritableComparable 接口比 Writable 接口多了一个 comparable 接口，用于自定义“比较”语意。

目前我们支持大部分主流语言有的基本类型，具体如表 3.2 所示。

表 3.2　基本类型

基本类型	序列化类型	序列化大小（字节）
bool	BoolWritable	1
byte	ByteWritable	1
int	IntWritable	4
float	FloatWritable	4
long	LongWritable	8
double	DoubleWritable	8

序列化类型具体的继承关系如图 3.6 所示。

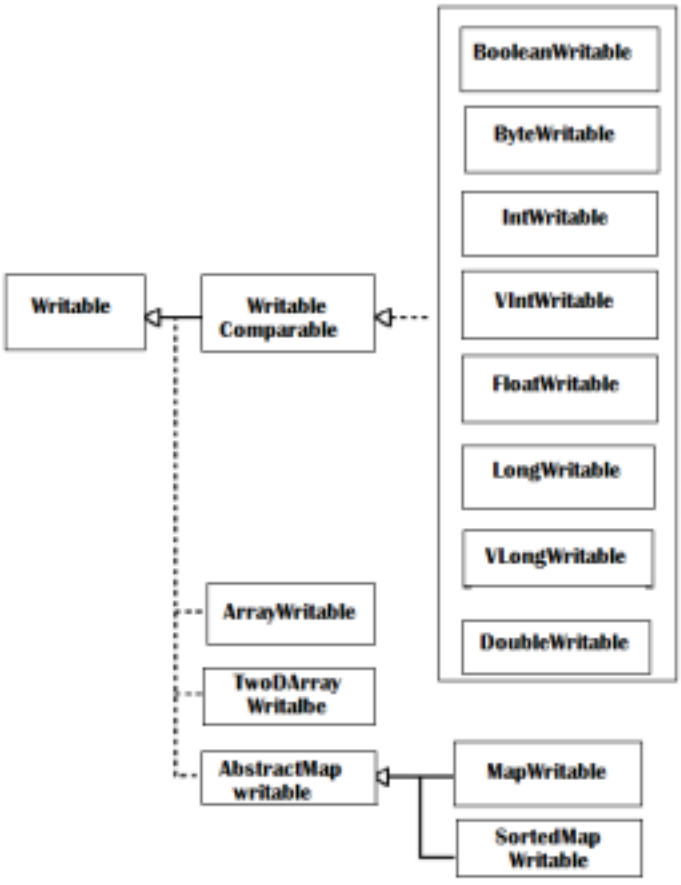

图 3.6　序列继承图

由上图可见，我们分布式系统还支持一些其它数据类型的序列化，包括字符串 Text、对象 Object 等等的序列化。Text 是针对 UTF-8 字符串的序列化封装。我们用一个 int 类型来存储字符串的字节数，因此 Text 类型最高可支持 2GB 的单一字符串。此外，我们之所以选择 UTF-8 编码格式，是因为目前 UTF-8 格式的普遍流行，这样会使得系统与现行的其它系统能够很好的兼容。在此就不再对其它内置数据类型序列化进行介绍了。

虽然我们尽量提供更多的数据类型的序列化，然而有时依然不能简单地满足用户要求。这样情况下，用户可以自定义数据类型的序列化。用户自定义序列化需要对二进制表示进行全面控制，同时也需要提交数据类型的 compare 方法。具体来说就是用继承前面描述的 WritableComparable 接口，实现其中的 write 和 readField 方法。我们建议用户尽量按照前面序列化的设计原则进行设计，以提高 MapReduce 性能。虽然内置的序列化设计良好，但有时用户精心自定义的序列化会更加符合用户的需求。

3.3.5 基于文件存储的数据结构

前面我们提到采用文件来存储 MapReduce 处理的数据。下面我们就文件的具体结构进行详细描述。

我们的系统支持两种类型的文件格式：序列文件和映射文件。序列文件是一个实现了紧凑存储键/值对的文件。支持压缩键/值对。同时增加了同步机制。映射文件以序列文件为基础。映射是一个目录。包含一个数据文件和一个索引文件。

序列化文件有着广泛的应用。考虑一个日志文件，其中每一行文本代表一条日志。如果你想记录其中二进制数据的类型，普通的文本文件是不合适的。而序列化文件能够很轻松地表示这种情况：指定一个键/值对，然后按照这种固定的数据类型存储。例如指定一个 LongWritable 数据类型来存储时间戳作为 key，而 value 则可以指定为 Writable 类型的记录字节数。

序列化文件的操作支持分别由 SequenceFileWriter 和 SequenceFileReader 类提供。分别实现对序列文件的写/读操作。

创建 sequenceFilewriter/sequenceFilereader 对象需要传入键/值对的类型。并且要指定是否有压缩记录值。对序列文件的读写大量使用了缓冲技术。读写操作需要遵

循序列文件格式。其他可选的参数包括是否压缩等等。系统会在创建时将一些元数据写进文件头部。创建完序列文件后，就可以用 append 方法写进键/值对了。写完后就像一般文件流一样调用 close 方法即可。

最有效地排序或合并序列文件就是使用 MapReduce 模型了。MapReduce 内在的并行性以及可以指定 reduce 数目，使得排序或合并序列文件特别容易。例如我们可以指定一个 reduce，并指定好 map 输入文件为要排序或合并的序列文件并指定键值类型，那么我们将得到一个有序输出序列文件。对排序的工作机制我们将在下一章进行详细描述。下面简单介绍序列文件的格式。

一个序列文件由一个文件头和一系列记录构成，如图 3.7 所示。

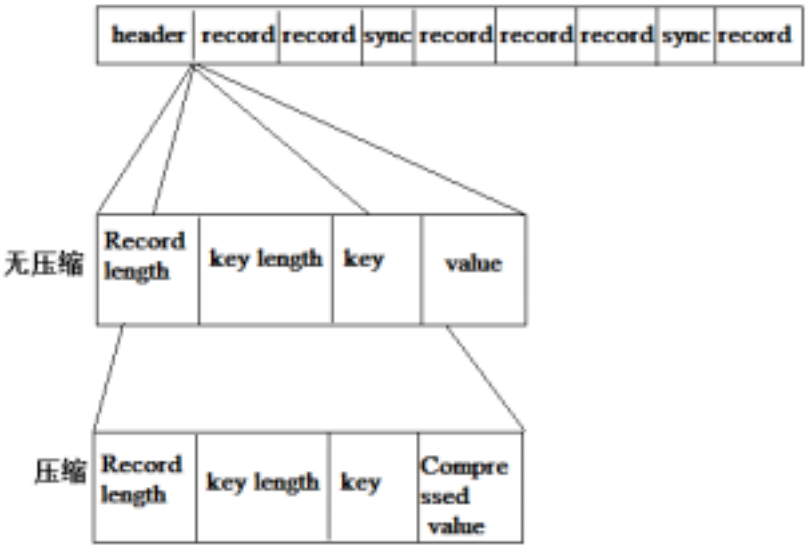

图 3.7　压缩格式图

其中文件头的具体构成如下图所示：SEQ1 表示序列文件版本号，为方便以后升级时进行版本检查。目前版本为 SEQ1，括号里的 4 表示占 4 个字节。接下来写入键和值类型的名字，然后选择是否采用压缩算法。最后是一次同步标记。

同步标记用于允许文件同步地读取文件的任意部分的记录。每个序列文件都在头文件写入一个随机产生的同步标志。如上图所示，同步标志随后出现在记录间。由于要控制数据冗余率，我们会每隔 2000 字节写一次同步标志。这样会保证数据冗余率不超过 1%。

图 3.8　文件头格式

在序列文件头后就可以追回键/值对记录了。序列文件中间的记录的格式取决于是否采用压缩算法以及采用的是记录压缩还是块压缩。目前我们的系统仅支持记录压缩，以后会完善块压缩。如果没有采用压缩算法，那一条记录由记录长度、key 长度、key 和 value 组成。其中记录长度的数据类型均为 4 字节的整型。记录压缩大部分和无压缩相同，只是对其中的 value 进行压缩，而并不对 key 进行压缩。由于压缩率并不高，除去压缩算法本身所需要的运行时间，记录压缩在性能上未能有大幅度地提高。因此我们急需要用块压缩。块压缩对多个记录进行一次压缩，同记录压缩相比不仅压缩率有大幅度地提高，压缩次数也大大地减少了。我们目前设计了如图 3.9 的块压缩结构，其实包括记录数、压缩后的 kay 长度、压缩的 key、压缩后的 value 长度、压缩的 value。

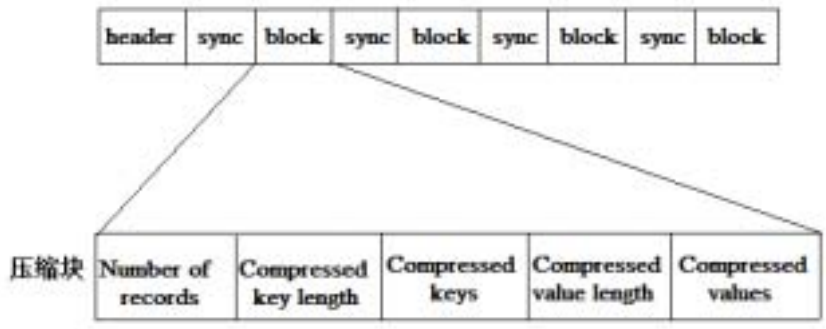

图 3.9　块压缩格式

我们系统支持的另一个文件类型是映射文件。映射文件以序列文件为基础，包含一个排序的数据文件和一个基本 key 值的索引文件。数据文件包含映像中的全部键值对。而索引文件相对而言要小得多，包含部分键和索引信息。目前索引文件只支持一级索引。索引文件包括数据文件的键的一部分，索引间隔可以由用户在配置

文件里进行设置，索引间隔默认为 128。由于索引文件要全部装入内存，因此 key 类型应尽量保持较小的范围。映象中的项必须按照非降序添加。要对映象文件更新，可以对编辑排序，然后顺序合并。

映射文件支持的操作和序列文件类似。MapFileWriter 和 MapFileReader 类实现对映射文件的写\读操作。MapFileWriter 对象包括两个序列文件的指针。分别代表数据文件和索引文件。对映射文件的写操作即通过序列文件的写操作完成。MapFileReader 对象包括一些索引项。用于对映象的数据文件键\值对进行定位。索引文件采用二分法进行查找。因此对映射文件的索引时间复杂度为 O（logN）。

下面具体了解映射文件的结构。系统默认将映射文件中的数据文件命名为 data，索引文件命名为 index。这两个文件都是序列文件。假设用户定义的 key 类型为 int，value 类型为 text。则数据文件一般如下所示：

1 One, two, buckle my shoe
2 Three, four, shut the door
3 Five, six, pick up sticks
4 Seven, eight, lay them straight
5 Nine, ten, a big fat hen
6 One, two, buckle my shoe
7 Three, four, shut the door
8 Five, six, pick up sticks
9 Seven, eight, lay them straight
10 Nine, ten, a big fat hen
……

假设索引间隔采用默认的 128，则索引文件的内容如下所示：其中的 key 类型与数据文件中相同，而 value 存储的是相应的 key 值在数据文件中的偏移位置。

1 128
129 6079
257 12054
385 18030
513 24002

641 29976

769 35947

897 41922

……

索引间隔的不同往往能影响系统的性能。一般来说，增大索引间隔会使得索引文件变小，从而占用更少的内存空间。相反，减少索引间隔会使得索引更加高效，这种高效的代价就是内存的增大。因此，用户在构建自己的应用时有必要进行权衡。

假设现在我们要对上例中的 key 值为 496 的键值对进行定位。通过二分法我们最后定位在索引文件中的 key 值为 385 的位置，得到其在数据文件中的偏移量为 18030。接着我们在数据文件中找到偏移量为 18030 的地方开始查找直到找到或大于（即不存在）我们的目标 key 值 496，从而获取其 value 值。显然通过这种索引的方法，我们随机访问键值对是非常高效的。

3.4 map/reduce 核心模块

3.4.1 mapReduce 模块概要

Map/reduce 模块是整个分布式计算系统的核心模块，它完整地实现了第二章的 map/reduce 模型原理，并在其基础上进行了很多必要的补充，以使的系统拥有更强的健壮性、扩展性和易用性。

具体来讲，Map/reduce 模块实现完整的作业建立，作业设置提交以及作业调度。其中作业调度又包括任务的调度与具体的执行。任务调度的具体细节又涉及到用户自定义的 Map 函数与 Reduce 函数。本模块由于涉及大量设计策略，因此我们将在下一章专门描述。

3.4.2 mapReduce 模块的一些特性

mapReduce 模块除了提供上面描述的基本框架之外，还提供了一些非常有用的高级特性，这些特性包括计数器、排序以及连接数据集等。这些高级特性极大地提高了系统的易用性和扩展性。

（1）计数器

当我们分析数据时往往希望获取关于数据的各种周边统计。比如，如果你正在统计文本中的非法记录，结果发现非法记录在文件中的比例非常的高。这时你不禁要去检查为什么有这么多的非法记录了：是统计程序错误还是原始数据本身质量问题？分布式计算用户往往需要先在小范围内做出正确性检查，然后才开始大规模地运行数据。

我们内置的计数器特别适合于这种用于采集统计信息。我们为每个作业内置了许多计数器。诸如输入文件的大小或是记录的条目等。

每一个 tast 维护各自的计数器由并周期性地发送给 tasktracker 和 jobtracker。这样 jobtracker 就能够在整个 job 内进行整合了。Task 每次发送所的有计数，而不是按增量发送。这主要是有两个原因：一是可以有效地防止数据丢失，二是计数可能会减少，如果之前那个 task 失败的话。计数值的确定只是在整个作业完成时才开始统计。

当然，内置的计数器往往也有局限性。这时用户可以自定义计数器。比如，用户可能想统计所有已经索引的文档数量或者已经处理了多少单词的数量，等等。为了使用这样的特性，用户代码创建一个叫做 counter 的对象，并且在 map 和 reduce 函数中在适当的时候增加 counter 的值。例如:

```
Counter * uppercase;
uppercase=GetCounter("uppercase");
map(String name,String contents):
    for each word w in contents:
        if(IsCapitalized(w)):
            uppercase->Increment();
        EmitIntermediate(w,"1");
```

这些 counter 的值,会定时从各个单独的 worker 机器上传递给 master(通过 ping 的应答包传递)。master 把执行成功的 map 或者 reduce 任务的 counter 值进行累计，并且当 MapReduce 操作完成之后返回给用户代码。当前 counter 值也会显示在 master 的状态页面，这样用户可以看到计算的进度。当累计 counter 的值的时候，master 会检查是否有对同一 map 或者 reduce 任务的相同累计，避免累计重复。(backup 任

务或者机器失效导致的重新执行 map 任务或者 reduce 任务或导致这个 counter 重复执行，所以需要检查，避免 master 进行重复统计)。

counter 特性对于 MapReduce 操作的完整性检查非常有用。比如，在某些 MapReduce 操作中。用户程序需要确保输出的键值对精确的等于处理的输入键值对，或者处理的文档数量是在处理的整个文档数量中属于合理范围内。

（2）排序

MapReduce 编程模型的性能核心之一就是数据排序。即便用户的应用程序中不需要排序，MapReduce 模块也会在其组织数据的过程中使用排序，特别是应用在 map 阶段的输出上。因此，如何选择高效的排序算法，将对整个分布式计算系统来说极为关键。

我们以第二章的气象数据中按气温排序进行举例。回顾一下第二章，原始气象数据存储于文本文件当中。由于无法简单地直接对文本文件进行排序，我们将气象数据存储于序列化文件中。其中 IntWritable 类型的 key 值代表气温，而 Test 类型的 value 值是该行的数据。

此时排序由于 map 任务输入一般为 64Mb 的数据，所以使用一般的内部排序足以应付了。我们的内部排序算法采用的是堆排序算法，时间复杂度为

O(NlogN)。

堆是一种数组，但是以树的结构形式来看待它，如下标 i 节点的求解 Parent 和 Children 节点如下：

PARENT(i) return [i/2]

LEFT(i) return 2i

RIGHT(i) return 2i + 1

堆分为 MAX 堆和 MIN 堆：

MAX 堆满足的条件为： $A[PARENT(i)] \geq A[i]$,

MIN 堆满足的条件为： $A[PARENT(i)] \leq A[i]$.

MAX 和 MIN 堆的维持：

这里只对 MAX 堆，MIN 对类似：数组 A 的 LEFT(i) 和 RIGHT(i) 都是 MAX 堆，但可能 A[i]可能小于它的 Children 节点，所以需要调整，调整伪代码如下：

```
MAX-HEAPIFY(A, i)
 1 l ← LEFT(i)
 2 r ← RIGHT(i)
 3 if l ≤ heap-size[A] and A[l] > A[i]
 4     then largest ← l
 5     else largest ← i
 6 if r ≤ heap-size[A] and A[r] > A[largest]
 7     then largest ← r
 8 if largest ≠ i
 9     then exchange A[i] ↔ A[largest]
10   MAX-HEAPIFY(A, largest)
```

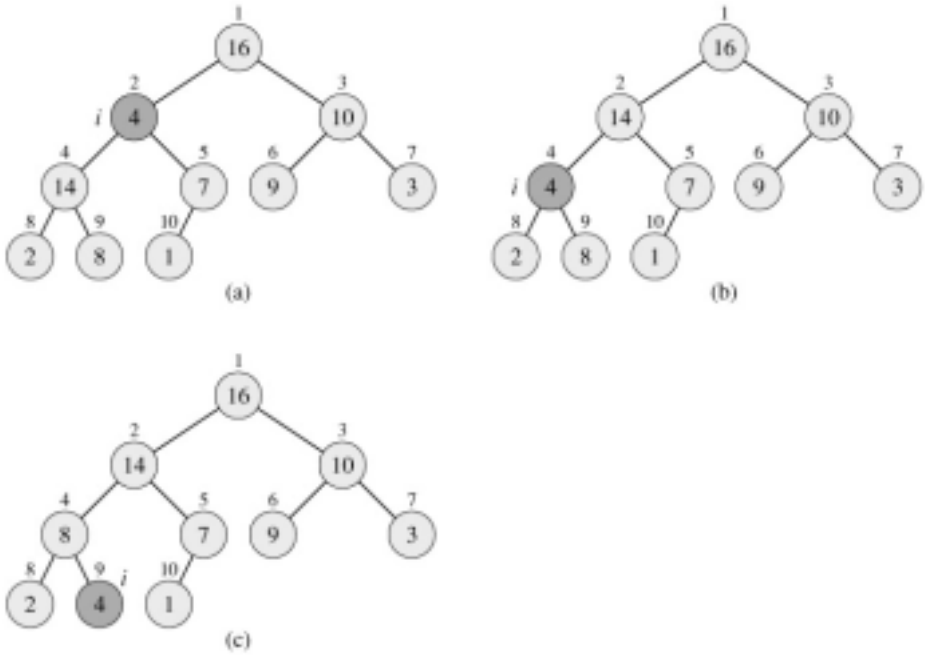

图 3.10　构建堆

MAX 堆建立：

```
BUILD-MAX-HEAP(A)
1   heap-size[A] ← length[A]
2   for i ← □length[A]/2□ downto 1
3       do MAX-HEAPIFY(A, i)
```

。如下面建立 MAX 过程：

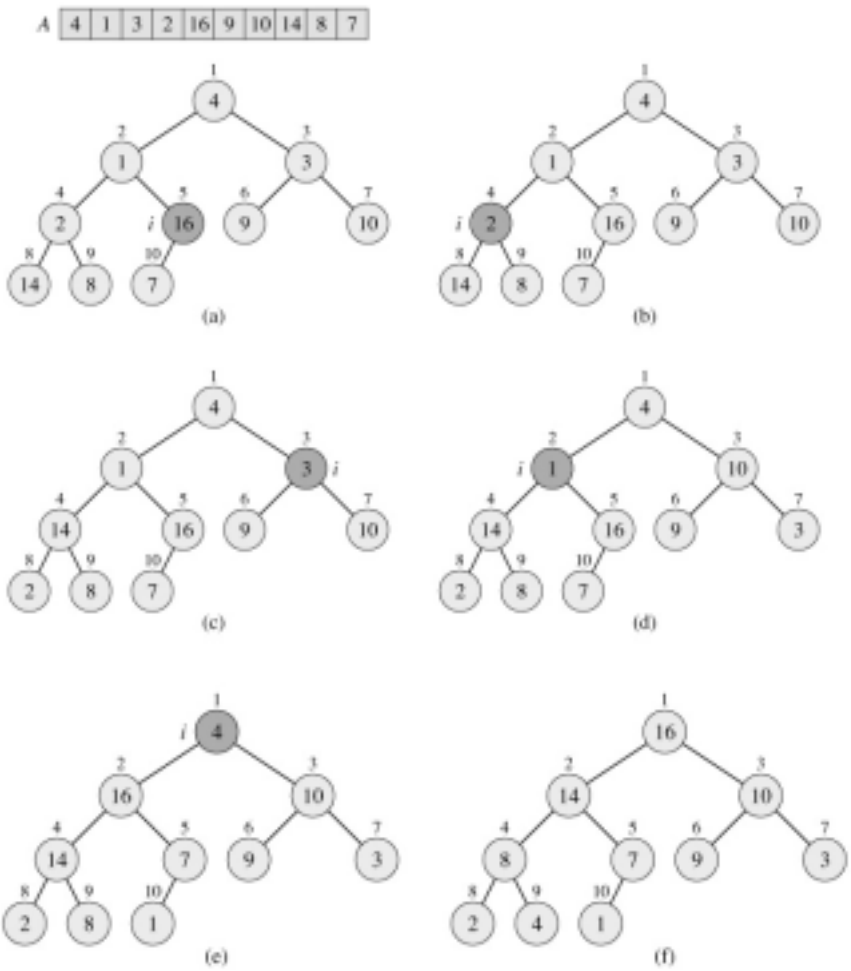

图 3.11　堆序过程

堆排序(HeapSort)

首先是把一个给定的数组变成 MAX 堆，然后把根节点和堆的最后一个交换堆的大小减 1，再调整堆，这样下去，直到堆的大小为 1。伪代码如下：

HEAPSORT(A)

```
1 BUILD-MAX-HEAP(A)
2 for i ← length[A] downto 2
3     do exchange A[1] ↔ A[i]
4       heap-size[A] ← heap-size[A] - 1
5       MAX-HEAPIFY(A, 1)
```

堆排序的图解过程如下：

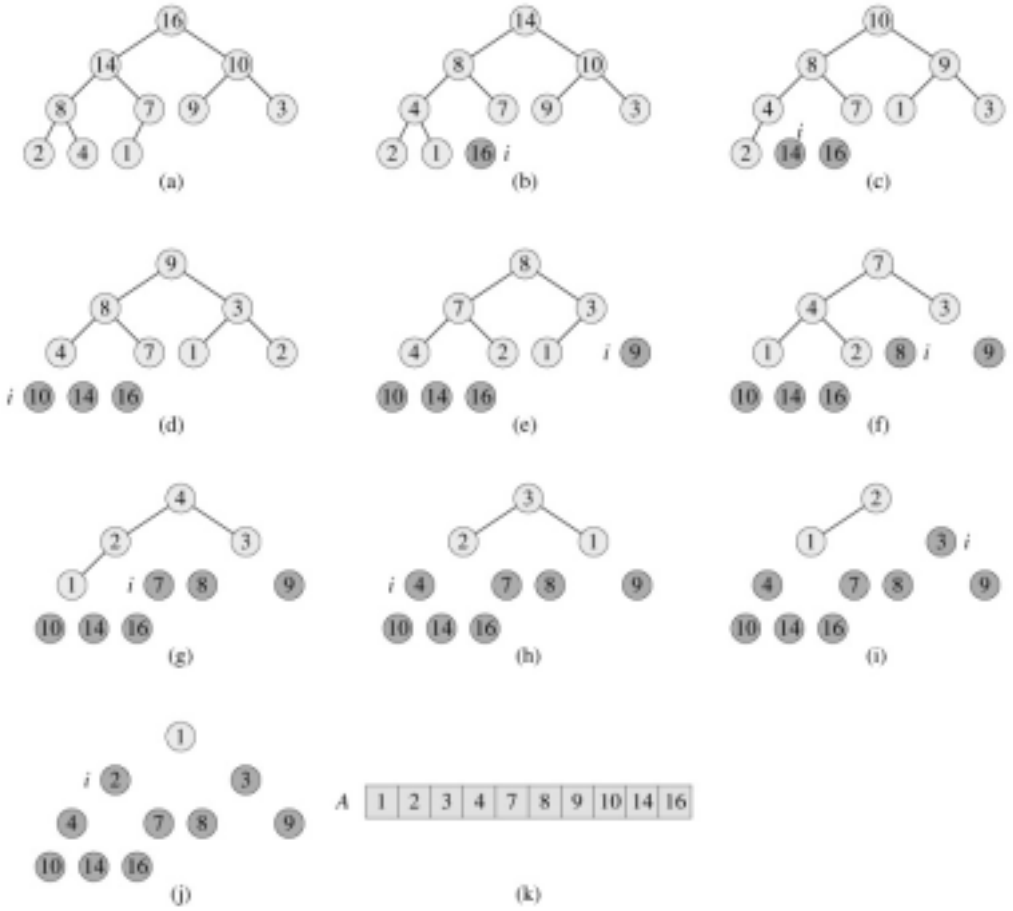

图 3.12　堆序结果

但这里在存在一个问题，假设用户设置了 30 个 reduce 任务，那么最后系统会产生 30 个已排序的输出文件。然后，如何对这 30 个文件进行合并形成一个单一的已排序文件呢？虽然对多数应用来说，没有必要进行合并了。

一种自然想到的解决方案便是只设置一个 reduce 任务，这也是系统默认的数目。然而对于一些非常大的 map 阶段输入文件，采用单一的机器来处理必定会使性能急剧下降。这样也就是将 MapReduce 模型的并行性抛到一边去了。

另一种方法便是采用如同数学分类的方法。如上面的气温例子，我们可以设置四种分类：

第一类：温度低于-10 度；

第二类：温度介于-10 到 0 度；

第三类：温度介于 0 到 10 度；

第四类：温度大于 10 的度。

这种方法具有可行性，但用户必须非常仔细地选择分类区间才能确保各个分区能够比较平均的获得数据。如同上面的分类区间，很可能得到如下的分类结果：

温度区间：< –10　[–10, 0)　[0, 10)　>= 10

记录比例：11%　13%　17%　59%

显然，上面的比例是非常不平均的。为了得到较为平均的比例，我们需要了解整个温度的分布状况。用 MapReduce 是非常容易计算出各温度点出现的次数，如图 3.13 所示。

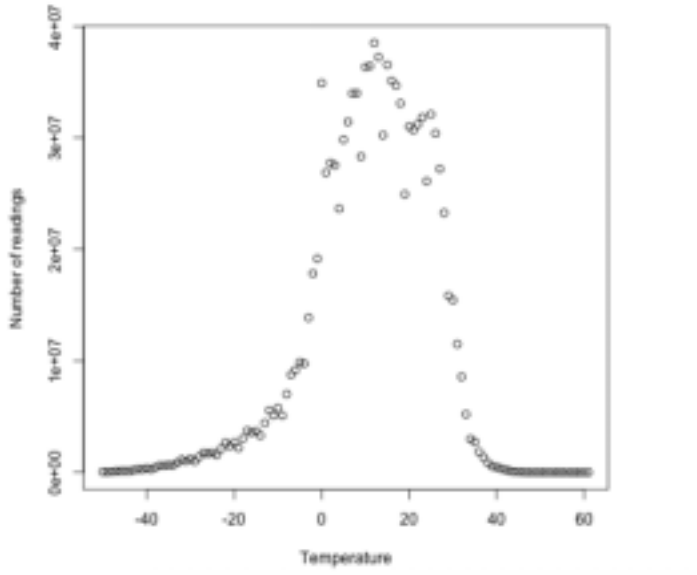

图 3.13　温度频度

我们可以据此来构造一个非常好的分类区间了。但使用全部的原始数据来构造分类区间是不实现的，这样会增加大量的额外开销。相反，我们采用统计学中的“采样”来模拟整个数据，即只需要少量数据来模拟温度分布。请注意，由于采样后数据量极其小，因此完全不需要分布式计算，只需要单机上运行即可。

通过一次取样模拟，我们得出如下四个区间：

<-5.6，(-5.6,13.9)，(13.9,22.0)，>22.0。从而得到如下的分类结果：

温度区间：　<-5.6　[-5.6,13.9)　[13.9,22.0)　>22.0

记录比例：　29%　24%　23%　24%

目前我们的分布式系统并不支持内置取样接口。用户需要在指定 reduce 数目（大于 1）时指定相同数目的分类区间和 key 类型。至此，整个合并排序即得到了解决。

然而，在分布式计算系统中常见的另一种排序是外部排序。外部排序指的是大文件的排序，即待排序的记录存储在外存储器上，待排序的文件无法一次装入内存，需要在内存和外部存储器之间进行多次数据交换，以达到排序整个文件的目的。外部排序最常用的算法是多路归并排序，即将原文件分解成多个能够一次性装人内存的部分，分别把每一部分调入内存完成排序。然后，对已经排序的子文件进行归并排序。

外部排序主要的时间开销用在信息的内、外存交换上，所以减少 I/O 时间成为要解决的主要问题。外部排序的基本过程由相对独立的两个步骤组成：

（1）按可用内存大小，利用内部排序方法，构造若干个记录的有序子序列外存，通常称这些记录的有序子序列为“归并段”；

（2）通过“归并”，逐步扩大有序子序列的长度，直至外在中整个记录序列按关键字有序为止。

本系统外部排序采用 4 路归并排序法。此方法非常常见，这里就不再详细描述了。

3.5 系统其他模块简述

除去上面介绍的两大模块，分布式系统还包括以下几个模块：

（1）fs 模块，定义抽象的文件系统 API。

（2）kiddenfs 模块，项目组实现的分布式文件系统的实现。

（3）tools 模块，定义了一些常用的工具，诸如一些类型转移等。

（4）util 模块，定义了一些公有的 API，其中包括一些外部排序算法等。

（5）net 模块，用于简单封装一些网络 API。

（6）ipc 模块，用于封装远程过程调用协议，提供一些实用的远程过程调用接口。

（7）zlib 模块，由前面介绍的，此模块由开源组织开发与管理，本系统在 IO

模块里对其接口进行了一些 OO 封装。

（8）log 模块，用于记录系统作业完成的日志。

由于本文关注重点在于 map/reduce 模型的原理及实现，并限于篇幅有限，以上各个模块就不详细论述了。值得一提的是，分布式计算系统并不局限于分布式文件系统之上，也不局限于本系统所用的 kiddenfs，但经过实践表明，采用 kiddenfs 能极大地提高系统性能，特别是那些局限于内部网络的分布式计算系统。

4 MapReduce 关键策略研究

上一章介绍系统的整体架构、计算流程以及系统各子模块的结构。分布式系统中作业调度以及容错机制[26-27]是模型的重要一部分，本章将对这些影响系统性能瓶颈的关键策略进行详细而全面的论述。

4.1 作业调度

分布式计算系统的应用十分广泛诸如生产性应用：数据加载、统计值计算、数据分析等；批处理作业：机器学习等；交互式作业：SQL 查询、样本采集等。我们经常面临要处理各种等待作业，因此如何对作业进行调度来保证公平性便成为了衡量一个分布式系统好坏的重要标准。参考了操作系统的任务调度策略，目前我们提供了如下几种调度方法。

4.1.1 单队列调度

单队列调度即先进先出调度策略，是最简单的调度方法。在这种策略下，master 总是把最先进入就绪队列的作业进行运行，而运行作业独享整个集群一直执行下去直至作业完成或者失败，因此其他的作业都必须等待该作业运行结束。

显而易见，这种调度策略实现极为简单，只需要维护一个队列即可。然而 FIFO 调度策略服务质量不佳，后续作业往往要等很长时间以至引起作业用户不满。

该策略的一个改进即引入优先级，即建立作业时加入一个优先级别，用优先队列维护作业的调度。由于实现简单且应用广泛，目前我们系统采用这种调度策略作为默认调度策略。客户端用户可以动态设置作业的优先级，当作业调度器选择下一个作业运行时，会选择优先级最高的那个作业来运行。目前该调度器并不支持抢占式调度，因此一个高优先级的作业也许要很长时间来等待一个正在运行的低优先级作业；同时该策略不支持并行执行。基于这些缺点，我们同时提供了以下两种调度策略供用户根据情况选择使用。

4.1.2 容量调度

容量调度策略考虑了集群执行作业的并行性，提供了一个并行执行作业的方法。该策略主要提供了如下特点：

（1）支持多队列，一个作业将被分配到其中的一个队列之中。

（2）系统为每个队列分配一定的系统集群容量(即 TaskTracker)。每个队列中的作业可以访问系统为队列分配的容量资源。

（3）空闲资源可以自由地分配给负载过重的队列。过剩的分配容量可以重新分配给其他队列。

（4）队列支持作业的优先级。

（5）在一个队列内，优先级高的作业比低优先级作业优先享用系统资源。然而，一旦作业正在运行，低优先级作业不会被高优先级作业抢占。

（6）为了防止资源垄断，即使某些队列没有竞争者的时候，系统仍然会强制分配给这些队列不高于设置的容量份额。

每当一个 TaskTracker 空闲时，容量调度器便会挑选一个最需要回收资源的队列。如果找不到这样的队列，便会分配给一个负载最重的队列。

容量调度策略有效地支持了作业间的并行执行，很大程度上提高了资源利用率。同时该策略动态地调整资源分配，使整个集群的利用率得到很大的提高，也提高了作业的执行效率。

容量调度策略缺点在于队列设置和队列选择无法自动进行，并且不同的设置对系统效率影响很大，因此系统用户需要了解大量系统。

4.1.3 公平调度

与上两节的调度策略相比，公平调度策略注重各个用户公平地使用集群时间，特别是改善小作业的响应时间，确保生产性作业的服务水平。当单一作业执行时，作业会使用整个集群资源。随着越来越多的作业被提交时，空闲的 TaskTrackers 会确保给予每个作业一个集群的公平份额。该策略主要有以下特点：

（1）将作业根据提供的用户进行分组，形成作业池。一个提交更多作业的用户不会获得更多的集群资源。

（2）给每个作业池分配最小共享资源。

（3）公平调度策略支持抢占，当一个作业池在一定时间内没有获得应用的公平份额时，调度器会强制停止那些超过份额执行作业内的任务以便给该作业池内作业执行。

（4）将多余的资源平均分配给每个作业。

在每个作业池内作业执行选择方面，调度器会优先选择调度资源小于共享资源的作业，在同等情况下，选择分配资源与所需资源差距最大的作业。

公平调度策略的优点在于支持作业分类调度，使不同类型的作业获得不同的资源分配，提高服务质量。同时支持抢占式调度，高度保证作业池内作业获得较公平的执行份额。由于公平调度策略过分强调作业执行的公平性，没有考虑节点的实际负载状态，因此有可能导致节点负载实现不均衡。

本节讨论了三种作业调度策略，并就三种策略的优缺点进行了简单的论述。用户在使用系统时应根据自己的应用场景，选择合适的调度策略进行作业调度。当然，我们系统非常支持并欢迎用户开发出适合自己应用的作业调度策略。

4.2 任务调度

我们设计分布式系统的目标之一是充分挖掘分布式计算的潜在能力，特别是处理多作业时充分利用各任务节点的 CPU 时间，尽可能减少部分节点的空闲时间，使整个系统的负载均衡达到理想的效果。上节我们通过某种策略，尽可能使各用户提交的作业能够得到公平地处理。本节我们关注如何高度计算节点来运行相应的任务，以此来提高系统的整体性能。

任务调度机制如下：

（1）所有的 worker 定期向 master 发送心跳包，心跳包中包括 worker 运行任务状态以及其他信息。

（2）根据 worker 发送的心跳包中的信息，master 按时间顺序选择空闲的 workers，然后分配给他们每个节点一个任务（map 任务或 reduce 任务皆可）。

（3）对于 map 阶段：map worker 从输入数据中通过用户提交的 map 函数产生

中间临时 key/value 对，再将中间 key/value 对写入本地磁盘并将其分发到由分割函数指定的 N 个区域中, 最后将这些缓存在本地磁盘上的具体位置传回给 master。

（4）对于 reduce 阶段：reduce worker 先从 master 获取中间 key/value 对在 map workers 本地磁盘上的具体位置，然后使用远程过程调用从这些 map workers 读取数据。读取所需要的数据后，reduce worker 根据 key 对这些数据进行排序, reduce worker 遍历排好序的中间数据，将具有相同 key 的 key/value 对传给用户定义的 reduce 函数, 此时即开始执行 reduce 函数。

这里，我们尽量让输入数据保存在构成集群机器的本地硬盘上的方式来减少网络带宽的开销，因为网络带宽在一般的计算机环境中是一个相当缺乏的资源。Master 有输入文件的位置信息，并且尝试分派 map 任务到存储了对应相关数据块的设备上执行。如果不能分配 map 任务到对应其输入数据的机器上执行，则尝试分配 map 任务到尽量靠近这个任务的输入数据的机器上执行，例如同一个交换机机内的 worker 节点上执行。由于这样大部分输入数据都是在本地机器上读取的，他们消耗比较少的网络带宽。

在 reduce 阶段，我们同样尽可能地将 worker 节点分配到已有相应输入数据的节点上。由于分布式系统中 reduce 任务的个数一般而言要远远低于 map 任务个数，因此我们通常的做法是尽可能将要执行 reduce 任务的节点分散到各个交换机中的一个节点上，然后将该交换机中的其他节点 map 输出数据传输到该 reduce 任务节点上，以此来提高系统数据复制的效率。

总之，分布式系统的任务调度策略由于各节点间的对等性，我们关注的是决定如何让最适当的节点来执行相应的任务，以此来减少系统中数据的传输量，从而提高系统整体性能。

4.3 容错处理

由于分布式计算系统被设计用于在成百上千台机器上处理海量数据，并且考虑到用户自定义的 map 函数或其他代码可能产生的错误，计算系统必须考虑到机器故障的容错处理。成熟的分布式计算系统应该考虑到各自错误发生的可能性，并保证

系统在这些错误发生的情况下依然能够有相应的操作使系统安全退出并记录错误到日志中。

4.3.1 任务错误处理

我们首先考虑任务失败的情况。最常见的导致任务执行失败是用户自定义的代码可能有 bug 并抛出运行时异常。发生这种情况时，由于不是系统本身的缺陷，错误本身无法修复。我们只能记录错误发生点到日志，并在退出之前将错误报告给 master。

4.3.2 worker 节点故障处理

Worker 节点故障是很容易获知的。因为 master 周期性的 ping 每个 worker 节点，一旦 master 在一个有效的时间段内没有收到 worker 传回的心跳包，那么它将把这个 worker 的状态标记为失效。一个 worker 节点失效后，所有原本分配给这个 worker 完成的 map 任务都会被重置为初始 idel 状态，这样就可以安排给其他的 workers 来完成。与此同时，所有在失败的 worker 上正在运行的 map 或 reduce 任务，也都会被重置为 idle 状态。

当一个 map 任务由于 worker 节点故障而重新分配给其他 worker A 节点后，所有执行 reduce 任务的 worker 都会被通知。所有还没有得及从原始失败节点上读取数据的 worker 都会从 A 上读取数据。

MapReduce 可以有效地支持到很大尺度的 worker 失效的情况。在一个 MapReduce 操作期间，如果正在运行的机群上进行由于某些原因导致一些机器故障，MapReduce master 只是简单的再次执行那些 workers 完成的工作，最终完成这个 MapReduce 操作。

4.3.3 master 故障处理

Master 故障是另一种节点故障，也是最为严重的一种故障。目前我们并没有有效的机制来应对这种错误。由于 master 并没有备用机器，因此一旦 master 发生故障，当前作业也就失败了。目前的处理是，一旦 master 失败，我们立即就中止 MapReduce 计算。系统会通知客户发生了这种故障，这样将控制权交给了客户来处理。

4.3.4 防止错误的其他策略

分布式文件系统 kiddenfs 被设计能够在一个大集群中跨机器可靠地存储超大文件。它将每个文件在逻辑上存储成一系列的数据块，除了最后一个，所有其他数据块都是同样大小的。为了容错，文件的所有数据都应该保留多个副本。每个文件的数据块大小和副本系数都是可配置的。

副本的存放是 kiddenfs 可靠性和性能的关键。我们将副本存放在不同的机架上。这样可以有效地防止当整个机架失效时数据的丢失，并且允许读数据的时候充分多个的带宽。这种策略设置可以将副本均匀分布在集群中，有利于当组件失效情况下的负载均衡。但是，因为这种策略的一个写操作需要传输数据块到多个机架，这增加了写的代价。相比优点而言，这点代价完全是值得的。

当系统任务运行需要访问这些数据时，为了降低整体的带宽消耗和读取延时，系统会尽量让读取程序读取离它最近的副本。如果在读取程序的同一个机架上有一个副本，那么就读取该副本。如果一个集群跨越多个数据中心，那么客户端也将首先读本地中心的副本。

5 MapReduce 系统应用

分布式计算平台是基础设施，我们应当如何在其上构建实际应用并检验其正确性与有效性呢？本章我们将利用实验室有限资源来构建一个小型计算平台，并在其上完成与第二章描述类似的气象统计任务。

5.1 部署环境

系统总共有 6 台机器。其中一台做为 master，另五台做计算节点，具体配置如表 5.1 所示。

表 5.1 硬件配置

cpu 类型	Intel
cpuMHZ	1.6GHz
内存大小	512M
cpu 缓存	2048KB
硬盘空间	至少保证 40G

6 台机器部署在一个交换机内，因此任意两点间的传输速率非常快，不会因为网络传输等因素给系统造成性能方面影响。

5.2 气温统计

在第二章第二节，我们讨论过气象原始数据的格式并分析了 MapReduce 处理数据的整个过程。现在我们将用设计的分布式计算平台处理一个更为复杂点的问题。

我们要找出最近十年武汉市一年当中月平均温度最高的一天，以此来分析季节的变化。这个问题较为复杂。如果采用单机程序分析，我们需要逐条访问气象数据，然后再统计最高值。采用 MapReduce 编程模型，我们可以用以下两个步骤完成：

（1）在 Map 阶段，编写函数解析气象原始数据，提取出（月份，气温）这样的键值对。

（2）在 Reduce 阶段，对所有相同的键（即月份）的气温相加，并设定 reduce 计算节点数目为 1。此时即可得到有序的（日期，温度）数据了。

气温最高所对应的月份即是我们想要的结果了。当然，如果有需要，可以以些输出结果作为 MapReduce 操作的输入，将所有温度除以 10（年数），得到一年当中每天气温的排行榜。

经过统计，我们发现武汉日平均最高温度大多发生在 7、8 月份。具体如图 5.1 所示。

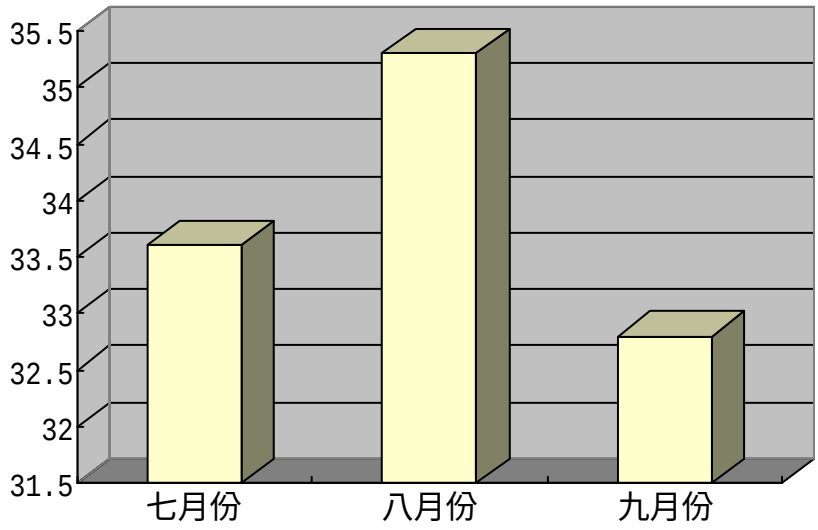

图 5.1　月均温度统计

通过与武汉权威气象公布的 50 年来的统计数据相比，我们得到的数据基本吻合，一些细微差别应该是由于我们统计的只是近十年来的数据。由于实验室环境所限，我们测试的数据量较小，运行时间大概只有单机的一半左右。

6 总结与展望

6.1 全文总结

分布式计算平台是目前互联网企业首选的用于存储和分析海量数据的系统。国外企业诸如 Google、Amazon、Facebook 等，国内企业诸如阿里巴巴、腾讯、网易、百度等也正积极探索分布式计算平台。本文基于目前流行的 MapReduce 编程模型，设计并实验性实现了通用可扩展性分布式计算平台。主要在以下几个方面做了一些工作：

（1）通过对当前几种主流分布式计算系统进行对比与分析，选择了适合处理海量数据的 MapReduce 编程模型，并在系统实现中对 MapReduce 模型中关键调度策略进行扩展。

（2）对整个分布式系统进行整体设计，并对功能模型进行了合适的划分。

（3）在开发的实验性产品上进行了简单应用，验证了本文所设计的分布式计算平台设计的合理与正确性。

在整个系统设计与实现阶段，我们也学习到了很多内容。首先，我们认识到网络带宽是系统的资源的瓶颈。我们系统的一系列优化都使因此针对减少网络传输量为目的的：本地优化使得我们读取数据时，是从本地磁盘读取的，并且写出单个中间数据文件到本地磁盘也节约了网络带宽。其次，我们发现很多实现问题都可以简单通过 MapReduce 来解决，而不仅仅用于离线的批处理。第三，意识异常处理对程序的重要性，这是以前小项目中没有过的经验。

6.2 展望

限于实验环境及研究生保贵的科研生涯，我们仅仅从主体上对所设计的分布式平台进行实现而忽略了一些重要细节。在现有的基础上，有以下几个方面等待改进或完善：

（1）改进、完善和更新影响系统性能的一些关键策略。特别应注重作业调度时

用户提交作业运行时间的公平性。在容错机制方面应该设计出一套能够处理 master 错误的合理解决方案。

（2）完善目前的实验性产品，增强其功能并增加用户使用接口。由于实验环境所限，目前并没有对系统进行稳定性等一系列严格测试。后续工作应该在这方面有所侧重。

（3）努力挖掘 MapReduce 编程模型应用范围，并在实际应用中适当改进 MapReduce 过程，扩大其通用性。

致 谢

研究生生涯即将画满句号了，我要对很多可爱可敬的人表示感谢。

首先要感谢我的指导教授刘小峰博士，分布式计算项目任务和学位论文都是在刘老师的悉心指导下完成的。从项目开始，刘老师便以严肃的科学态度要求我；在项目进行中，刘老师不辞辛苦地给予我细心的指导；在思想和生活上，刘老师给我以无微不致的关怀；在项目完成时，刘老师又给我指点缺陷与未来的方向。在此谨向刘老师致以诚挚的谢意和崇高的敬意。

项目与论文的完成亦离不开软件学院方少红老师大力支持。在进入项目时，方老师便给予我极大的帮助，让我很快适应新的环境。在项目开发时，方老师不厌其烦的指出我研究中的缺失，且总能在我迷惘时为我解惑，从各个方面给予体谅及帮忙，在此表示诚挚的谢意。

感谢项目成员程名同学，在项目中我们激烈的讨论让我受益良多。更难能可贵的是我们之间的友情。恭喜我们顺利进入同一公司，相信在未来我们有更多的合作。感谢我的好朋友陈雪姣，有她的帮忙，使得本论文更加完整与严谨。

在论文即将完成之际，我的心情无法平静，在研究生期间，有太多可敬的师长、同学、朋友给了我无数的帮助，在这里请再次接受我诚挚的谢意!最后我还要感谢培养我长大父母和姐姐，谢谢你们!

参考文献

[1] Jeffrey Dean, Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters, 2004

[2] Sanjay Ghemawat, Haward Gobioff, and Shun-Tak Leung. The Google File System. 19th ACM Symposium on Operation Systems Principles, Lake George, NY, 2003

[3] Jeff Dean Google, Inc. Experiences with MapReduce, an Abstraction for Large-Scale Computation. 15thm International Conference on Parallel Architectures and Compilation Techniques, 2006: 1-5

[4] Hadoop. Open source MapReduce implementation from Apache, 2008

[5] Preeti Priyadarshini Sharad Agarwal. Hadoop Recent Improvements and Roadmap. Hadoop china Salon, 2009

[6] Shouyan Wang, Scheduler used in Baidu, Computing Security and Data Security, Hadoop china Salon, 2009

[7] Zheng Shao. Hive-Progress and Future: Open-Source Solution for Data Warehousing. Hadoop china Salon, 2009

[8] Mr&Avro, Feature HighLights-Scheduler. Hadoop china Salon, 2009

[9] Huaming Liao/Yongqiang He. Experience of Studying Column Based Storage on Hadoop and Future Work. Hadoop china Salon, 2009

[10] 郭宏. 分布式对象技术新进展. 微电子世界, 2003, 22(1): 26-39

[11] 郭银章, 徐玉斌, 曾建潮. 分布式对象及主流技术比较研究. 太原重型机械学院报, 2004, 4(2): 14-18

[12] 熊江. 分布式计算模式的演变与发展. 重庆三峡学院学报, 2007, 3(4): 44-47

[13] 云计算理论及技术研究. 重庆交通大学学报, 2009(4): 11-16

[14] Hagit Attiya, Jennifer Welch. Distributed computing fundamentals, simulations, and advanced topics, 2008

[15] 迟学斌. 高性能并行计算: [硕士学位论文]. 武汉: 华中科技大学图书馆, 2007

[16] Tom white. Hadoop: The Definitive Guide, 2009, 7(4): 45-48

[17] Jason Venner. Pro Hadoop: Build scalable, distributed applications in the cloud, 2009: 177-185

[18] Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava. Data Preparation for Mining World Wide Web Browsing Patterns, 2008

[19] 赵波, 吉立新. 基于 UML 的电信网海量数据处理系统的分析与设计. 通信技术, 2007, 11(4): 238-240

[20] 张应刚. 分布计算中间件技术的探讨, 2009

[21] 陈俊, 华莹. 简析中间件. 科技信息(学术研究), 2007(5): 22-25

[22] 桂小林. 网格技术导论: [硕士学位论文]. 武汉: 武汉大学图书馆, 2005

[23] Maozhen li, Mark Baker. Grid core technologies, 2006: 254-262

[24] Fox G. Peer-to-Peer network. Computing in Science & Engineering, 2007

[25] 周文莉, 吴晓菲. P2P 技术综述. 计算机工程与设计, 2006, 27(1): 92-95

[26] 孙广中, 肖锋, 熊曦. MapReduce 模型的调度及容错机制研究. 微电子学与计算机, 2007, 9(24): 14-17

[27] Sun Guangzhong, Fan Bin, Chen Guoliang, et al. Study on scheduling strategy for global computing application. PDCAT, 2006: 368-372

[28] 许薇. 嵌入式数据库的海量存储技术研究. 微计算机信息, 2008, 7-2(7): 119-120

[29] 房友园, 杨树强, 贾焰等. 基于 CORBA 的海量数据加载并行任务调度技术研究与实现. 全国开放式分布与并行计算学术会议, 2005

[30] Glen Bruce, Rob Dempsey. Security in distributed computing, 2002

[31] Andrew S. Tanenbaum, Maarten Van Steen. Distributed systems: principles and paradigms, 2008

[32] George Coulouris, Jean Dollimore, Tim kindberg. Distributed system: concepts and design, 2008

[33] 张君. 分布式系统技术内幕. 北京: 机械出版社, 2006: 339-343

[34] Nicolai M. Josuttis. SOA in practice: the art of distributed system, 2008: 285-291

[35] 李西宁. 分布式系统. 北京: 机械出版社, 2006: 307-309

[36] John F. Buford, Heather Yu, Eng Keong Lua. P2P networking and applications, 2009

[37] 奚建清，游进国，汤德佑. 基于 MapReduce 的封闭立方体并行计算方法, 2009

[38] 郑启龙，房明，汪胜. 基于 MapReduce 模型的并行科学计算, 2009

[39] 刘封. 谈分布式计算模型 MapReduce 与搜索引擎系统, 2007

[40] 周锋，李旭伟. 一种改进的 MapReduce 并行编程模型. 科协论坛, 2009

[41] 吴宝贵，丁振国. 基于 Map/Reduce 的分布式搜索引擎研究. 现代图书情报技术, 2007

[42] 吴刚，王怀民，毛新军. 可成长的分布式系统. 计算机工程与科学, 2005