1814241

分类号 _____ 密级 _____

UDC[注1] _____

学 位 论 文

MapReduce 模型在 Hadoop 实现中的性能分析及改进优化

（题名和副题名）

张密密

（作者姓名）

指导教师姓名 刘 玓 副教授

电子科技大学 成 都

（职务、职称、学位、单位名称及地址）

申请专业学位级别 硕士 专业名称 计算机系统结构

论文提交日期 2010.03 论文答辩日期 2010.05

学位授予单位和日期 电子科技大学

答辩委员会主席 _____

评阅人 _____

2010 年 月 日

注 1：注明《国际十进分类法 UDC》的类号。

独创性声明

本人声明所呈交的学位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。据我所知，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，也不包含为获得电子科技大学或其它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谢意。

签名：张密密　日期：2010年 5月24日

论文使用授权

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将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、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

签名：张密密　导师签名：

日期：2010年 5月24日

摘 要

云计算的提出是对互联网的一个冲击，它实现了计算能力的商品化，其透明性和简单的编程模式为开发者带来了更便捷的服务开发和部署方式。2009 年被称为云计算元年，Amazon、Google、IBM 等诸多 IT 巨头都把目光聚焦在云计算，将其视为未来发展的主要战略方向。因此，对云计算进行研究即迎合了 IT 技术的发展趋势，又具有较强的实际意义和商用价值。

MapReduce 是一种简单的并行计算模型，它将简单的业务逻辑从复杂的实现细节中分离出来，提供了一系列简单强大的接口，通过这些接口可以实现大规模计算得自发的并发和分布执行。MapReduce 的这种特性使得它成为了云计算的首要选择。它不仅仅是编程模型，还是优秀的任务调度模型，其作业调度问题已成为业内最热烈的讨论话题之一，并成为云计算系统高效稳定运行的关键技术。

Hadoop 是对 Google 公司 MapReduce 模型的开源实现，它已成为当前应用最广泛的开源云计算平台，但 Hadoop 发展时间较短，仍有许多不足的地方需要改进。

本文对云计算的关键技术之一 MapReduce 编程模型做了深入的研究，并在 Hadoop 平台上对 MapReduce 的典型应用进行了关键性能指标的测试，科学的检测了 MapReduce 在公平性、可扩展性、加速比等关键指标上的性能。通过实验分析和对 Hadoop 调度算法的研究提出了一种创新的调度算法，文中命名为基于优先级加权的滑动窗口调度算法。它通过滑动窗口技术动态的监控系统中执行作业的数量，自适应的管理系统负载平衡，利用优先级来为不同类型的作业提供差别服务。并对 Hadoop 原始的推测执行算法进行了改进，新的算法采用更精确的方法来判断影响系统响应时间的掉队者任务，大大的提高了掉队者任务的命中率从而有效地提高了系统的响应能力。最后该算法考虑 Hadoop 计算平台中集群的异构性，根据每个计算节点的处理能力合理的分配任务。

本文在最后用实验比较了该算法和 Hadoop 的 FIFO 调度算法的性能差异。通过实验验证了新算法具有更好的响应时间和公平性，有着良好的负载均衡，并且大大提高了 Hadoop 系统在异构平台上的性能。

关键词： 云计算、MapReduce、Hadoop、调度算法

ABSTRACT

The presentation of Cloud Computing is a great impact to the Internet-mode Service. The Concept of Clout Computing means that we can treat computing as a commodity. It provides a simple and transparency programming model for internet users and developers to develop and archive I-mode Service. As 2009 becomes the first year of Cloud Computing in IT industry, business giant such as Amazon, IBM and Google has taken Cloud Computing the most important strategic direction in their near further.

As a new technology that runs on large clusters for dealing with massive data storage and computing, it is very important to find a way for organizing these large number of servers to guarantee high performance of Cloud Computing system. Meanwhile MapReduce shows its simply model for parallel data computing and task scheduling to be a suitable solution for these requirements. With MapReduce simple business logics are separated from the complex implementations. Yet how to schedule tasks efficiently and fairly in MapReduce model has become one of the most popular topic in MapReduce community.

Hadoop is a open source frame that implement Google's MapReduce model and is the most popular open source software for Cloud Computing. But it's still a young project, and there are a lot of points to be improved.

In this thesis we do a in-depth research on MapReduce which is the core technology of Cloud Computing. We test MapReduce's performance with its typical applications on fairness, scalability, speedup and response time. According to the result of our experiments we determine the inadequate of the scheduling in Hadoop platform and propose a new scheduling algorithms. With this new scheduling algorithms, Hadoop can have a better performance in a Heterogeneous Environments which Hadoop always runs on in practice. And finally we do some experiments and comparison to verify the advantage and usability our scheduling algorithms.

At last of this thesis we conclude our work and discuss the possible orientation of MapReduce model.

KeyWords: Cloud Computing, MapReduce, Hadoop, Scheduling algorithm

目录

第一章 引 言

1.1. 研究背景

互联网诞生至今保持着迅猛的发展速度，Web 2.0 的出现使互联网发生了革命性的变化，网络已不仅是单向的对静态信息的浏览，它为用户提供了联系更紧密内容更丰富的网页内容，已经成为了用户生活工作中的常用工具，互联网企业与用户之间的互动大大增加。

Web 2.0 最根本的特点是用户为本，互联网上的内容发布不再由权威的互联网企业所控制，网络上的绝大多数内容都是来自用户，我们称这些内容称为微内容（Micro Contents），例如喜爱的音乐播放列表、用户搜藏的图片文档书签。这种突出用户参与的特征必定会引发微内容在互联网上的数量剧增。与之前的网络内容相比，Web 2.0 具有开放性和可重用性等特征。我们可以利用 XML 等技术结构化管理这些微内容，并通过一些开放接口提供与应用无关的调用。这种变化使的互联网信息变成了一个完整的生态系统，而不是一个个相互隔离的信息孤岛。目前互联网的应用是以这些微内容为基础的，互联网企业为了生存和发展，势必会竭尽可能为网络用户提供免费的信息存放和处理的服务，大量的存放用户各方面的信息数据，并会为了这些数据展开竞争。例如 Google 公司的 Gmail 宣称永不删除用户邮件，并且每天都在扩展其 Gmail 的容量；再如网易、腾讯等公司都提供了免费网络硬盘的服务。Web 2.0 对互联网的模式带了了巨大的冲击，它不仅带来了丰富的应用和网络内容，同时也为互联网带来了海量数据传输、处理和存储的需求。

与此同时，中国的网络普及率正在逐年攀升，互联网用户的规模也迅速增加。如图 1-1 所示，据 CNNIC[1]调查报告显示，中国网民于 2009 年底达到了 3.84 亿，比 2008 年增长了 0.86 亿，增长率为 28.9%，网络普及率从 22.6%提升到了 28.9%，保持着稳定的增长势头。网络普及率虽然在中国稳步上升，但和世界发达国家比较而言，还有着非常大的差距，至 2009 年底网络普及率最高的国家是韩国，普及率为 77.3%。这说明中国的潜在互联网用户规模十分巨大，加之中国的家电下乡政策，也会带动中国农村互联网的发展，所以保守估计中国在 2010 年的互联网用户必定会超过 4 亿，并会在 2010 年上半年完整这个目标。

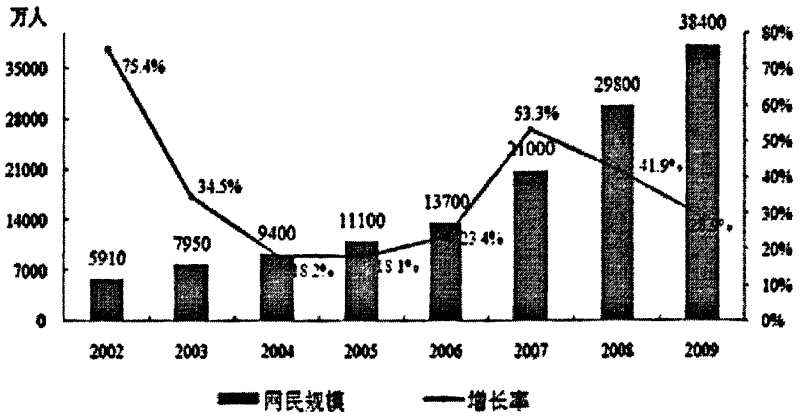


图 1-1 中国网民规模和增长率示意图

在用户数量和数据处理量同时增加的刺激下，互联网应用面临的问题随之而来。首先，海量的数据需要巨大规模的存储资源作为基础，其次网络应用对数据的依赖性增加，使得对海量数据进行计算和处理的能力的需求越来越强烈，维护这些应用程序的数据存储的成本和数据计算处理的成本越来越高。在这种对资源和计算能力的需求加剧的情况下，传统的 C/S 模型已经不能承担海量数据的存储和处理能力，于是点对点、海量文件系统、并行计算等技术相继出现。虽然它们在一定程度上缓解了用户需求和传统模式服务能力上的矛盾，但并没有在根本上解决问题，互联网的发展使得需要对互联网的模型在基础上进行转变，以适应信息爆炸给 IT 产业带来的巨大冲击。

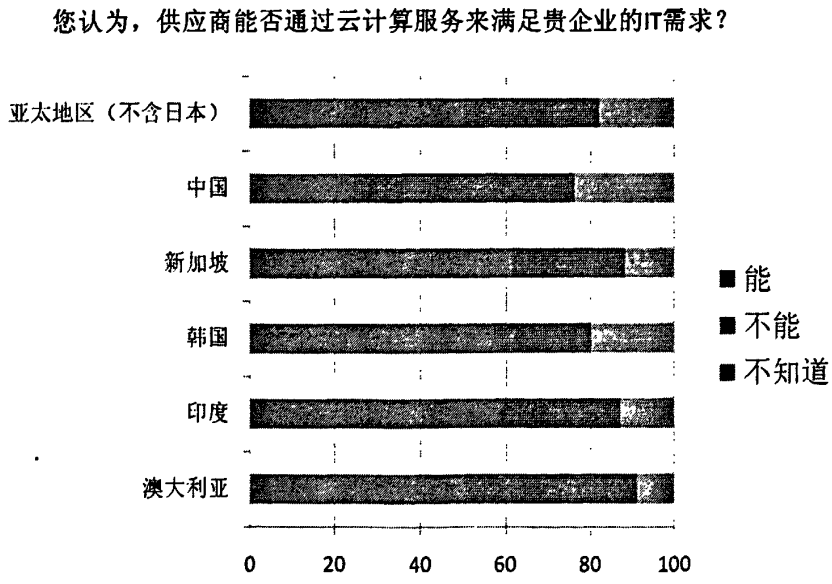
在应用需求和相关技术发展的推动下，云计算作为一种新的模型被提了出来。在 IT 业内，对云计算的解释不下于 20 种。其中维基百科对云计算的定义是："云计算(cloud computing)[2][3][21]，是这样一种计算方式，计算资源是动态易扩展而且虚拟化的，往往通过互联网提供。用户不需要了解'云'中基础设施的细节，不必具有相应的专业知识，也无需直接进行控制"。Accenture 咨询公司将云计算定义为："第三方提供通过网络动态提供及配置 IT 功能（硬件、软件或服务）"。

云计算的萌芽应该是从计算机的并行化开始，并行技术的出现使人们不满足于 CPU 摩尔定律的增长速度，希望把多台计算机并联起来以获得更快的计算速度和更强大的计算能力。这种看似朴素的实现高速计算的方法后来被证明是很成功的。此时的并行计算的主要应用是为了满足科学和技术领域的专业需要，现在并行技术已经成为各国的战略性技术，并行计算能力的强弱已经成为一个国家科技实力的衡量标准。在并行计算时代，人们追求的是高速的计算速度，采用昂贵的服务器，各国都在为超越别国的计算速度不懈努力，此时的计算能力和高性能机

群已经演变成了一种快速的消耗品。种种迹象表明，并行计算时代，计算能力和计算速度飞速发展，而应用却局限在了科学计算的专业领域，计算机技术的发展向普通用户的商业化色彩相对较弱。

云计算伴随着互联网的发展，使得普通用户不在关心数据存放在哪里，不必担心数据的安全性，不再关心应用程序的版本问题，不必为计算机病毒而烦恼，云计算通过将数据和应用隐藏在云后端，一切工作都交付给云计算中心负责解决，普通用户仅需要通过自己的喜好来决定购买云服务。对于普通用户，只需要任何形式的终端（可以使手机、PDA、个人电脑等），通过网络接入，就可以使用云后端的各中应用程序和网络信息。云计算的提出使得普通用户有了享受高性能计算的机会，因为云计算中心几乎可以提供无限制的计算能力，计算的弹性化和存储的弹性化也是云计算的一个重要特征。

自 2009 年以来，人们把 2009 年成为云计算元年，整个云计算的发展进入到一个计算机的“战国时代”，企业群雄纷起。Google 提出的云计算强调云的重要性，因为 Google 从来就没有端，并将 GFS 跑出作为诱饵欲占主动；Microsoft 依赖其市场占有率则强调“云”加上“端”才是云计算；VMware 则认为虚拟化是云计算的核心；Sun 公司则重提多年之前它们的老话“网络就是计算机”等等。



数据来源：IDC

（N=亚太区（不含日本）696为IT管理人员和首席信息官，2009年1月）

图 1-2 2009 年 1 月云计算企业满意度调查

图 1-2 说明大多数国家 IT 企业管理人员对云计算服务的满意度都是比较乐观的，其中新加坡、韩国和印度等地的应用满意度已超过 50%，可见云计算不仅成

为了当今 IT 行业领军人物的重要战略道路，而且在商业应用上已经有了非常好的成绩，对云计算的研究不仅有着深刻的科研意义，还有更广泛的实际需求和商业价值[4]。

因此对云计算在海量数据处理上面的研究和改进工作，无论是对企业还是对广大的互联网用户都有着非常重要的意义。

1.2. 本文的工作

本文详细研究了云计算的 MapReduce 并行计算模型的原理，发展状态和前景。并在此基础上，对 MapReduce 编程模式的性能进行全面深入的分析和研究。然后在 Hadoop 平台上提出了检验 MapReduce 性能的评估指标和方法，设计了基准测试程序集。通过对实验结果的分析以及现有 Hadoop 调度算法的研究提出一种全新的调度算法，叫做基于优先级加权的滑动窗口调度算法（PWSW），这个调度算法能更好的维护系统的负载平衡，并解决了当前 Hadoop 算法对于异构集群的不适应性，能够提高系统对任务的响应时间，其主要优势表现为以下几个方面：

1. 自适应的负载平衡，根据系统负载水平自适应的调整可调度作业滑动窗口的大小；
2. 更准确的推测执行。改进现有 Hadoop 中推测执行的策略，提出更准确的判断任务快慢的标准；
3. 更适合异构环境。考虑到 MapReduce 运行平台普遍的异构性，改进 Hadoop 调度策略中对异构性环境的考虑不足等问题；
4. 更快的响应时间。合理执行后备任务，提高系统响应时间。

最后本文通过实验测验证了 PWSW 对于 Hadoop 的 FIFO 调度算法的优越性。

1.3. 本文的结构

本文主要有五个章节。

第一章是引言，主要介绍海量数据处理需求的相关技术背景，发展现状和解决方案和未来的发展动向。总结了海量数据处理面临的几大问题。接下来介绍了本文的主要工作和内容组织。

第二章是相关技术和系统平台的研究。首先介绍了云计算的相关概念、设计模型、特点优势及其应用，其次详细介绍了 MapReduce 的概念模型、存储方式、

容错机制以及实现方式，然后介绍了 MapReduce 当前主流的调度算法。

第三章 MapReduce 在 Hadoop 实现上的系统性能评估及分析。基于开源的 Hadoop 平台对 MapReduce 设计模型进行了详细的性能评估，包括评估目标方法的确定以及制定出一系列的基准测试程序集。

第四章对 Hadoop 调度算法的改进优化方案。通过对 Hadoop 现有调度算法的分析和上一章的实验结果提出了对 Hadoop 调度算法的优化改进方案。并通过实验验证了优化方案的可行性。

第五章结论和展望。对本文的工作做了一个总结，并对今后 MapReduce 技术的发展方向做出了可能的展望。

1.4. 本章小结

本章首先介绍了互联网发现带来的海量数据处理上的需求，接着阐述了海量数据处理的技术背景、发展现状、解决方案和未来的发展动向，观察到云计算成为了目前海量数据处理应用最广泛的技术。最后阐述了本文的主要内容和组织结构。

第二章　相关技术和系统平台研究

2.1. 云计算概述

云计算简单来说是一种计算方式，它是由之前的多个计算机科学概念发展而来的，是分布式计算、并行处理和网格计算的结合，或者可以说云计算是这些科学概念的一种商业实现。通过将信息和计算分布到“云”后端的巨大的资源池上，为用户提供各种各样的应用和服务。

本文在引言中已经给出了维基百科和 Accenture 公司关于云计算的定义，但是云计算究竟是什么众说纷纭，我认为比较准确的涵盖出云计算含义的定义如下[2]：

云计算是一种资源交付和使用模式，指通过网络获得应用所需的资源（硬件、平台、软件）。提供资源的网络被称为“云”。“云”中的资源在使用者看来是可以无限扩展的，并且可以随时获取。这种特性可以形象的比喻为像水电一样使用硬件资源，按需购买和使用。

2.1.1. 云计算的基本概念

云计算的这种商业模式对当今 IT 业有着巨大的冲击，它标志着计算能力可以作为商品来流通，只不过这种商品流通的介质变成了互联网的传输功能。它的发展使得在未来，所有的用户无论在何时何地，只需要一个终端，如手机、笔记本等，就可以通过网络使用所需要的一切应用和信息。终端用户不用为在哪里管理云，以及如何存储云而担心，只要能够使用网络，就拥有了一个连接的移动设备，也就可以使用信息和服务了。云是一个基于网络的模型，有了它随时随地都可以调用云服务和云应用。

通过云计算可以使计算分布在大量的分布式计算机上，而不是把计算固定在本地计算机和远程服务器上，通过高速的网络连通，将计算过程从本地计算机或服务器转移到分布式的计算机集群中去。“云”即是一些自我监控的虚拟资源池，通常为一些大型服务器集群，包括服务器、存储服务器、宽带资源等等。云计算集中管理资源池中的全部资源，并通过软件实现自动化，自主的调节资源分配和负载均衡等工作，无需人为干预。这样减少了云计算应用提供者的日常工作，增大工作效率。

云计算的根本是关于建立数据中心的计算，它既包括了终端用户以服务方式获得的应用和信息，也包括了数据中心用来支持这些服务的硬件和软件资源，云计算的概念是从 Saas（Software as a service）演变过来的。我们将云计算看作 SaaS 平台上的一个应用，从这个角度看 SaaS 具有三个角色，SaaS 用户、SaaS 提供者/云用户、云提供者。云提供者即使服务运营商，他们掌握着 SaaS 的平台，即我们所说的云，应用开发商利用这个平台开发上层的服务业应用提供给 SaaS 用户使用，形成了一个完整的生态圈。这种模式已经应用于电信、政府等大企业，而且通过这种模式企业可以便捷的开发一些增值服务。如图 2-1 所示：

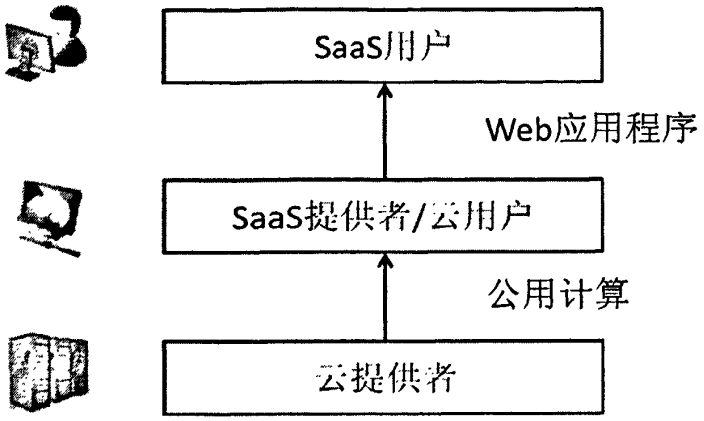


图 2-1 云计算的用户和提供商

SaaS 对终端用户和应用提供者都带来了不同的好处：服务提供者不再需要一次性购买昂贵的硬件和软件来应用，不用考虑资金不足和成本折旧问题，而是采用了按需付费的理念，这种方式非常经济，并能及时获得最新硬件平台和最佳解决方案，不再担心升级的问题，使用 SaaS 直接在网上升级，全部使用最新版本的软件；对终端用户而言，任何时间，任何地点都能用到云应用。SaaS 在加入云计算后并不改变其模式，更省去了服务提供者配置数据中心的麻烦，节省了工作量。云计算就像电力一样，每个企业都有自己的发电厂，自己配置发电量。当电剩余的时候就浪费了，电力不足时就影响了业务。云计算从业者也一样，现在每个企业都有自己的数据中心，数据中心的共性是非常受限制的，当业务增值时需要增加硬件，然而实际情况中很难精确地预算出明年的增长是怎么样，所以云计算可以灵活的调动资源，不管企业是在北京、上海、甚至国外。

从硬件方面讲，云计算在以下三方面有创新[2]：

1. 按需付费的、看似用之不尽的 IT 资源，这就避免了用户为了资源配置而提前做出规划；

2. 不需要一次性投资，使得公司可以在开始购买适量的资源，从小规模做起，随着业务的增加来再来增加资源；

3. 缩短了 IT 资源的使用和购买时间（比如按小时购买 CPU 使用和按天购买存储设备使用），再企业没有业务的时候，可以释放掉资源，从而节省企业开支，并提高了资源的整体利用率。

2.1.2. 云计算模型

云计算平台就是对一系列计算机资源的管理和配置，是这些资源可以根据用户的动态需求来实时分配。云计算的使用模式即服务化。所谓服务化就是服务消费者仅需要提供服务请求以及服务输入，而不需要了解服务怎样被调用、服务的局实现方法、服务流程等，直接获得服务结果。云服务消费者只需要通过该平台提供的简单接口就可以使用它所带来的丰富的服务。云计算平台通过把计算机资源隐藏起来，从而使用户可以更方便的使用服务。

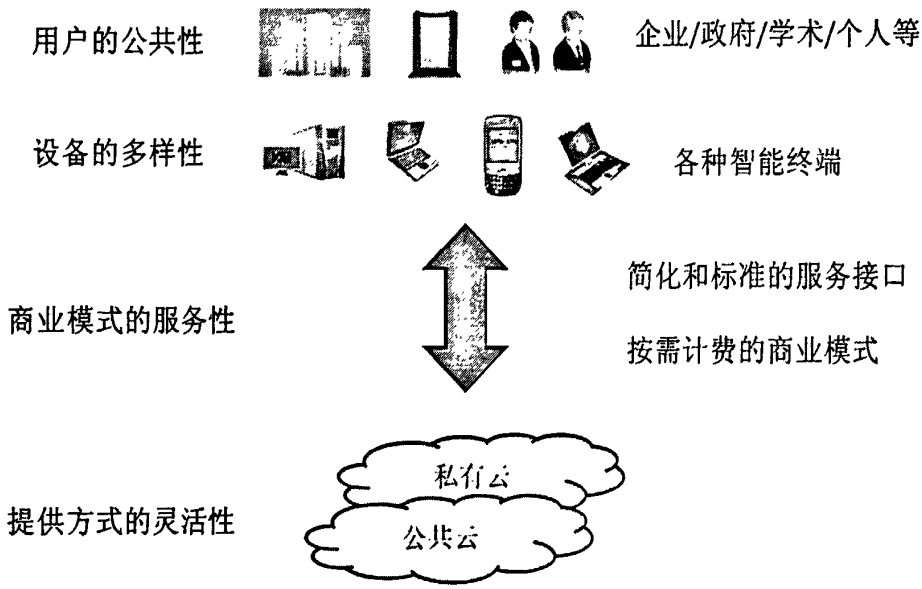


图 2-2 云计算概念模型

云计算平台可以既可以作为为公共用户提供服务的公共云，其维护和运行由第三方来进行，提供有偿服务给注册用户，如 Google、亚马逊等；也可以作为企业或者组织内部的私有云，私有云一般由企业自己搭建，为企业提供了更适合的云服务，但是规模一般比公共云要小。

2.1.3. 云计算的特性和应用

2.1.3.1. 云计算的特性

云计算使得普通用户可以享用高性能计算机，是由云计算的以下四个特性决定的：

1. 高可靠性和高安全性是云计算的一个显著特点。“云”通过数据多副本容错、不同用户的数据隔离及严格的安全级别划分，使得云计算可以为用户提供最可靠最安全的数据中心。因为在云的另一端，有最专业的管理团队，最先进的存储中心，使用户可以放心的存储并共享数据。
2. 云计算还为用户带来了更便利的操作和更低端的设备配置需求。只需要能够接入互联网的设备和浏览器，可以直接编辑存储文档，与朋友分享信息，在云后端有专业的计算机人员帮助用户完成硬件维护软件升级以及各种安全工作，完成之前用户需要在个人电脑上所做的一切。
3. 云计算可以实现跨设备的信息和应用共享。试想，当今我们经常需要异地办公（比方出差、在家办公），对于需要的数据往往只存在于固定的存储中，即便我们在每个计算机中都有数据的备份，而每次对数据修改后带来的数据同步也是非常头疼的。而云计算使这一切简单化，在它的网络应用模式中，数据有且只有一份，保存在“云”后端。只要能够连接互联网，就可以随时随地访问和编辑同一份数据。
4. 云计算为使用网络的云用户提供了近似无限的可能。他将最大化的提升个人计算机的存储能力和计算能力，以云自己的方式。终端用户的硬件配置和客户端应用是有限的，通过互联网，云计算赋予了用户无限的潜力，比如透过云用户可以使用数以千计的服务器集群所拥有的强大计算能力。当我们把核心数据和核心功能置入云上时，应用软件的概念，网络的概念都会发生翻天覆地的变化，也会因此改变云用户的生活。

2.1.3.2. 云计算的应用

IT 界未来的八大发展趋势中，云计算位居首位，足以说明云计算的代表性。Google、微软、Oracle、IBM、Sun、英特尔等 IT 巨头都面临着云计算的挑战，而谷歌在此领域一枝独秀，走在了各企业前面。云计算是由多个领域技术混合发展而来的，目前在商业上已经有了成熟的应用，特别是上面提到的几家公司，下面对各公司的代表产品作简要介绍：

1) Amazon 的 Web 服务[5]（Amazon Web Service）

亚马逊以经营网络书籍销售业务起身到后来成为美国最大的网络电子商务公

司，是最早开始经营网上电子商务的公司之一，也是最早进入云计算行业的企业之一，如今亚马逊已成为了世界最大的云服务提供商之一。

亚马逊的在线服务名为亚马逊网络服务，它有四个主要的服务：S3（Simple Storage Service）简单存储服务，它提供了无限制的存储空间供用户存放文档、视频等数据，按照用户消耗存储的多少进行收费；EC2（Elastic Compute Cloud）弹性云计算，弹性的可扩展的云计算服务器，根据不同的需求用户可以调用一个主频 1.71 GHz 内存 1.93G 的 Athlon 级机器，也可以调用拥有 16 个 GB 内存的的 64 位的多核系统，收费也是根据虚拟系统的功能来区分的；Simple Queuing Service 简单队列服务，一种简单的消息队列；Simple DB 简单数据库管理，这一服务尚未成熟仍然在测试阶段。

2) Google 的 GAE[6]（Google App Engine）

Google 在全球网络搜索领域占据着最广泛的用户，拥有领先的互联网技术和世界上最强大的数据中心。

GAE 采用让技术人员在 Google 的基础框架上开发 Python 应用程序的方式向用户提供云服务，用户使用这些应用程序不收取费用，Google 可以向用户提供一定量的免费的存储空间和页面访问量（最高每月 500M 和 500 万次）。Google 也在不断扩展 GAE 所支持的语言种类，2009 年 4 月 GAE 已经可以支持 JAVA，并且为企业业务也提供了一些新的服务。GAE 提供的服务包括基于 Web 的文档、电子数据表、数据将存储、Google 账户以及其他生产性应用服务。GAE 有免费的版本，也有付费的高级版本。

3) Sun 的 Network.com 和 Caroline 项目[7]

Sun 公司从用户的角度出发，旨在让用户更方便更容易的使用云计算服务，它把当下的主要工作放在以下的两个事情上：第一个是 Network.com，它提供了一种按使用量收费的模式，用户可以通过 Sun 的网格计算工具来使用它的计算基础设施。现在 Network.com 已经演变成了一个“按需提供服务的虚拟数据中心（virtual on-demand data center）”，用户可以依照其业务需求的改变随时使用相应的服务；第二个是 Caroline，该项目的目的是通过减少网络服务接口的数量，以使开发者或者网络管理员提高服务开发和服务部署的速度。它一部分用作数据中心的虚拟化方面，另一方面开发者可以使用 Java、Phthon、Perl、Ruby 这样的编程语言编写代码，并把它们存在 MySQL 或者文件系统中，然后快速的部署到网络的虚拟机上，通过 Caroline 可以使 Web 应用和开发人员更容易获取基于云的资源。

4) 微软的软件加服务[8]

微软的宏伟计划是"在企业级软件、合作伙伴托管服务以及云服务三者之间保持一种对称与平衡"。更简洁的说法，微软称之为"软件加服务"。今年微软已经推出了其第一批针对商业市场的 SaaS 产品，包括 Dynamics CRM Online、Exchange Online、Office Communications Online 以及 SharePoint Online。这里的每一个都提供多客户共享的版本，主要针对的是中小型企业用户；而单用户版本则针对需要5000个以上软件授权的大公司。针对个人消费者，微软的在线服务则包括 Windows Live、Office Live 和 Xbox Live。

5) 云计算安全[9]

2009 年 9 月瑞星公司宣布首个"云安全网站联盟"成立，标志着中国 IT 企业对云安全充满了期望和重视。目前在云安全实力比较强的两家公司是瑞星和趋势科技。瑞星的云安全（Cloud Security）计划是网络时代信息安全的全新体现，云安全技术融合了网格计算、并行处理、未知病毒行为判断等新兴技术和概念，通过互联网上的大量客户端（瑞星卡卡用户）自主的对网络中应用软件的异常行为进行检测，并把病毒和木马的解决方案发送到每一个终端客户。趋势科技 Secure Cloud 云安全网络防护方案可以在最新威胁达到用户计算机或者企业网络之前对其进行拦截，从而让网络安全智能化。旨在降低对客户端耗时的特征码下载的依赖性，转而借助威胁信息汇总的全球网络，该网络采用了趋势科技的云安全技术，在 web 威胁到达网络或计算机之前即可通过云端计算对其予以拦截。

2.2. 并行计算概述

2.2.1. 并行计算的基本概念

并行计算[10]简单来说就是通过将一个计算任务分配给多个处理器并行完成的一种策略，它的目的就是为了尽可能的减少计算任务花费的挂钟时间（此时间是计算任务从提交到完成的实际时间，而不是计算所占用的 CPU 时间，并行计算由于需要处理一些并行指令往往会增大计算任务的 CPU 使用时间）。它的基本思想是用多个处理器去处理单个计算任务可以获得用单处理器计算速度 N 倍的效果。这种加速效果我们成为线性加速，它仅是并行计算的理想状态，在实际应用中很难达到。

在单处理器下，一个计算任务被分解成多条执行，按顺序获得 CPU 执行时间，如图 2-3 所示：

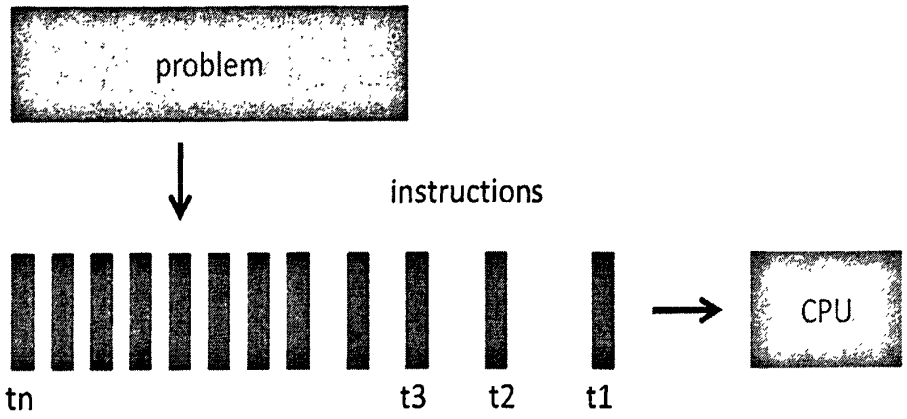


图 2-3 单处理器计算模式

在多处理器情况下，计算任务被分解成多个相互独立的子任务，各个子任务之间可以并行执行，每一部分被分解成多条指令，各个指令可以在不同的处理器上面执行，如图 2-4 所示：

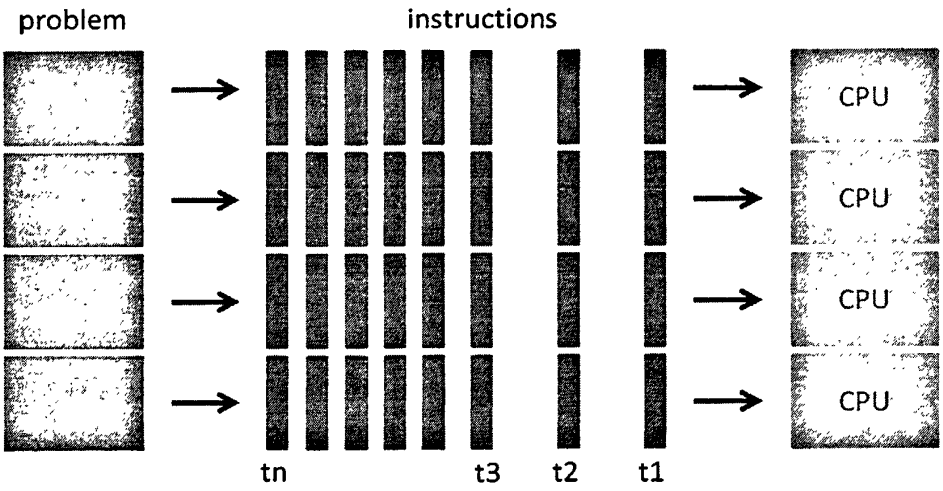


图 2-4 多处理器并行计算模式

2.2.2. 并行计算中并行机模式

并行计算技术可分为时间上和空间上的并行计算，并行计算科学所研究的是空间上的并行计算。按照 M.J.Flynn 的方法，并行机可以分为下列两种模式[10]：单指令多数据流（SIMD）和多指令多数据流（MIMD）。常见的多指令多数据流主要有以下五类：并行向量处理机（PVP）、对称多处理机（SMP）、大规模并行处理机（MPP）、工作站集群（COW）和分布式共享存储处理机（DSM）。下面对并行机

常见的集中模式作简要的介绍。

1) 对称多处理机：顾名思义他强调系统中资源访问的的对称性，在 SMP 中各个处理器访问系统资源如共享存储器花费同样的代价，它是基于均匀访存模型（UMA 的）。SMP 有着很好的并行度，但受到系统总线宽度的限制，所以其处理器数量是有限的。
2) 分布式共享存储处理机：DSM 也被看成一种分散的全域地址空间，它主要应用于丛集电脑上。在其中的每一个节点都将内存空间分为共享的和费贡献的两类，且在丛集电脑中所有节点上的共享内存空间是一样的。它代表了，对于共享内存空间的任何一个地址，丛集电脑中每个节点都存放相同的内容。DSM 改善了 SMP 的可扩展性，已成为了当前并行计算的重要发展分支。
3) 大规模并行处理机：在 MMP 模式下，每一个节点都是有微处理器、分布式存储器和网络接口电路组成，各节点之间通过约定的高性能网络互相连接。MPP 是并行处理机发展过程的中坚力量，目前一个系统已经可以支持上万个处理机。
4) 工作站集群：COW 中每个节点都是一个完整的工作站，工作站之间通过松耦合的网络接口互相连接，即他们连接到 I/O 总线上。

2.2.3. 并行计算模型

并行计算机并不像串行计算机一样有着统一的计算模型（串行计算机基本都使用冯·诺依曼的计算模型），常见的并行计算模型有以下几种，PRAM 模型[10]、BSP 模型[10]、LogP 模型[10]和 C^3 模型[10]。

PRAM 模型（Parallel Random Access Machine）：它假设模型中存在一个无限容量的共享存储器和若干个功能相同的处理器。处理器可以根据需要在任意时刻访问共享存储器。在 PRAM 中全部操作都是同步执行的，系统使用一个同步时钟进行控制。PRAM 的优点是设计简单，但其假设的无限大的共享存储器在实际中是不存在的，因此由于种种原因，它的全局访问速度通常比预想要慢。它几个变体在不同程度上弥补了 PRAM 的缺陷。带有局部存储的 LPRAM 模型和带有异步时钟的 APRAM 模型。

BSP 模型（Bulk Synchronous Parallel Copmuting Model）：它是由 Viliant 和 Bill MCColl 一起提出的。BSP 模型放弃了局部性原理，使得程序的设计和程序的实现

都得到了简化，而且 BSP 察觉在计算数据的规模很大的情况下，系统往往需要使用多个处理器，而现实中处理器的个数往往不能满足计算的需求，因此 BSP 将一个处理器映射到多个虚拟进程来解决计算对处理器上的需求。

LogP 模型：由大卫·卡勒等提出的。它使用了下面四个参数对 LogP 进行阐述，分别是 L、O、G、P。其中 L 代表 Latency，即信息从传送者到接受者传输花费的时间；O 代表 Overhead，即处理器在处理一条消息时所需要的额外消耗，处理器在做额外开销这段时间内不能做其他操作。G 代表 Gap，即相邻的发送和接受消息之间的最小时间间隔。P 代表 Processer，即系统中处理器的数量。LogP 讨论了网络通信的特征，却没有对网络拓扑进行讨论，而且 LogP 中是消息同步的。

2.2.4. 并行算法

目前最常用的并行算法是 PCAM 方法，即划分、通信、组合和映射。首先系统将所要处理的问题分割成数据量大小相同的小任务块，这些小任务块可以独立在各处理器上执行；通信阶段要考虑系统处理过程中需要交换的数据以及协调任务的执行情况；组合式为了提高并行计算的性能和减少资源消耗，将一些小的问题组合到一起执行；映射主要处理任务的分发，按某种策略将任务分配给处理器执行。

2.3. MapReduce 模型概述

在过去的几年中，Google 设计了上百个用于处理海量数据的的应用程序，用来对不同数据进行计算，例如蠕虫对每个主机的网页量的采集、限定时间的常用查询等等。这种业务在概念上都很简洁，但应用程序面对的是巨大的数据量，为了能在有效的时间内完成工作，这些计算必须分不到上千台计算机集群上去并发执行。如何控制并发、如何分配任务以及如何处理失败等相关问题，使得原本简单的计算被掩埋在了处理上述问题而带来的复杂代码中。

Google 针对上述问题提出了一种模式——MapReduce[11][12]。MapReduce 编程模型是云计算的核心技术之一，它把简单的业务处理逻辑从复杂的实现细节中提取出来，提供了一系列简单强大的接口，通过这些接口可以把大规模的计算的自发的并行和分布执行。使得开发人员不需要大量的并行计算或者分布式的开发经验就可以高效的利用分布式资源。

MapReduce 是一种并行计算的模型，上面提到的云计算的绝大多数应用大都

是采用的这种编程模型。它同时也高效的的处理了任务调度和负载均衡的问题。但是目前 MapReduce 仅适用于内部松耦合且可以高度并行化的应用程序，如何使这种模式高效的运行在紧耦合应用程序的情况，是这种编程模式以后的发展趋势。

2.3.1. MapReduce 的编程模型

MapReduce 借鉴了函数式程序设计语言的设计思想，把处理并发、容错、数据分布等的细节抽象到一个库里面，这种想法源自 Lisp 语言中所包含的 Map 和 Reduce 功能。这些 Map 操作处理输入记录的每个逻辑块，并且产生一组中间的键-值对集，在中间键-值对集具有相同键的结果中使用 Reduce 操作，来合并相关数据。下面对这个模型的两个关键函数进行简单介绍。

Map 函数[11][12]：通常是根据不同的业务需要由用户提供的。它用来处理作为输入的一组键-值对，生成另外一组键-值对作为中间结果。MapReduce 函数库会将中间结果中具有相同键的部分聚集起来，发送给 Reduce 函数。

Reduce 函数[11][12]：它和 Map 函数一样也是用户自定义的。它依据传递的中间结果的键值来处理相关的中间结果集，他通过合并中间键值相同的的结果集最终产生一个更小的结果值集合。Reduce 函数处理的中间值是一个迭代器，这样用户就不必担心传递的中间键值会超过内存的容量。

从理论上讲，和 MapReduce 操作的相关类型如下：

$$map(key_{in}, value_{in}) \rightarrow list(key_{out}, value_{intermediate})$$

$$reduce(key_{out}, list(value_{intermediate})) \rightarrow list(value_{out})$$

可以看出，输入的键-值对集合输出的键-值对集是在不同域上的，而输出的键-值对和中间的键-值对是在相同域上的（例如 Map 的输入正是用作 Reduce 的输出）。

2.3.2. MapReduce 的典型应用

MapReduce 的最简单且最典型的应用就是在海量的存档文件中统计各个单字的出现次数。具体流程可以用一下伪代码表示：

```
Map(String key , String value)
  //key：文件名
  //value：文件内容
  For each word in value ;
    EmitIntermediate(w , "1")
```

```
Reduce(String key , Iterator values)
  //key：某个单词
  //value：值列表
  For each v in values ;
    Result += ParseInt(v) ;
    Emit(AsString(Result)) ;
```

在这个应用中，Map 检查文件中的每一个单字，并将这个单词的出现数赋值 1，产生了（单词，出现数）的中间结果集，Reduce 根据中间结果的键（单词）进行分组，最终将所有文档中的每个单词的出现次数进行合并。

除此之外，还有一些经典问题可以用 MapReduce 轻松的描述出来[11]：

1. 分布式 Grep：Map 函数检察输入行，如果满足模式匹配将该行传递给 Reduce 函数，Reduce 函数简单的把中间数据输出就可以了。
2. 主机关键词向量指标：关键词向量（Term-Vector）为单个或一组文档中关键词及其出现次数的向量表示，形式如<word，frequency>。Map 对计算每一个文件的关键词向量，并输出主机关键词向量指标（hostname，Term-Vector），Reduce 将具有相同 hostname 的的中间键-值对进行合并并去除不常用关键词，得到最终输出<hostname,Term-Vector>。
3. URL 访问频率统计：和 WordCount 类似，Map 计算网页请求应答日志，产生<URL，1>的中间结果集，Reduce 合并 URL 相同的中间结果集，输出一个成对的<URL，count>。

2.3.3. MapReduce 模型的实现方法

MapReduce 提供了丰富的接口，而且接口的实现方法也是多种多样，因此 MapReduce 的实现也可以分为很多种，应该根据硬件环境的差异来选择不同的实现方式。下面以 Google 计算环境为例介绍一下 MapReduce 的设计和关键技术实现。

2.3.3.1. Google 计算环境

Google 广泛采用由普通计算机组成的大型集群作为其计算环境，集群中的计算机通过交换机网络互相连接[11]。在 Google 的环境里：

1. 每个节点通常运行 Linux 操作系统，配置双 x86 处理器和 2~4GB 的内存；
2. 网络连接设备都是常用的，一般在节点上使用 100M 或者 1000M 的网络，带宽消耗量比较小，通常情况下占用不到一半；
3. 每个集群中都有大量的计算机，一般规模在几百到几千不等，在这种规模的集群中，机械故障是很自然的。
4. 存储设备选取性价比高的 IDE 硬盘，设备直接放在节点上，底层有分布式的文件系统来管理这些节点上的硬盘。为了保证文件系统的可靠性和安全性，采取复制的方法。
5. 用户向调度程序提出应用请求，这些请求通常包含一组应用任务，调度程

序将这些任务分发到一组几点上执行。

2.3.3.2. MapReduce 实现框架

Map 操作首先对输入数据进行分区（例如分割成 M 块），拆分后的输入块可以在不同的节点上并行执行。Reduce 操作的分布式以中间产生的 key 的分布为基础的，通常根据某种分区函数（例如 *Hash(key)%R* ）将最终输出分割成 R 份。分区数 R 和分区函数都是用户指定的。

图 2-5 展示了 MapReduce 模型的结构框架[11]：

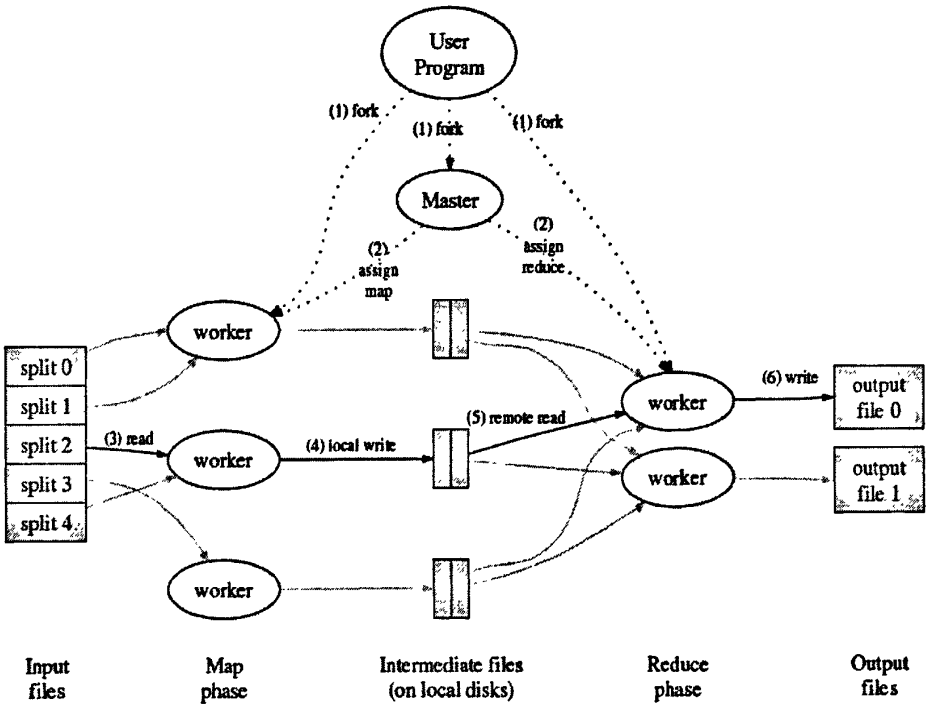


图 2-5 MapReduce 体系结构图

下面详细介绍一下当 MapReduce 操作被调用时的详细过程：

1. MapReduce 函数库将输入文件分成大小在 16~64M 之间的块，块的大小一般通过参数设定，假设共分为 M 块。然后在不同的节点上执行处理程序的副本；
2. 在这些分派的执行程序中存在一个叫做 Master 的特殊程序，即主控制程序。其他的执行程序都是作为主控制程序分配任务的工作程序（Worker）。主控制程序需要将上面的 M 个 Map 作业和 R 个 Reduce 作业分配到空闲的工作程序上执行；

3. 一个被分配 Map 作业的工作程序以文件分割后的一个小块为输入，并处理输入数据，分析产生 Key/Value 对传递给 Map 函数，这些中间结果会被暂时缓存在内存中；
4. 暂时保存的中间结果集会按照一定时间间隔写入工作节点硬盘，利用分区函数把这些中间结果分成 R 个区。写入工作节点硬盘的数据位置信息会被传递到主控制程序，主控制程序将传递的数据信息分发给负责 Reduce 任务的工作节点；
5. 主控制程序接收到数据的位置信息后，通过远程调用从 Map 节点硬盘上读取中间结果集。Reduce 节点在获得全部中间结果集后，按照中间结果集的 key 排序。排序是必要的，输入数据 key 值不同在经过 Map 操作后所的中间结果的 key 值可能相同，排序保证中间结果按 key 值连续存放；
6. Reduce 处理有序的中间结果集，对中间 key 值相同的结果作合并处理，并将中间数据传递给用户提供的 Reduce 函数。Reduce 函数将该区块追加到最终输出文件中；
7. 当 Map 作业和 Reduce 作业全部完成之后，主控制程序唤醒用户应用程序，进程返回到用户程序调用点。

根据上面的详细步骤，图 2-6 展示了在 MapReduce 在被调用过程中的一个虚拟实例和数据的流程，图中的 M 和 R 都表示工作节点，整个过程是在主控制程序的调度下完成的[22]。

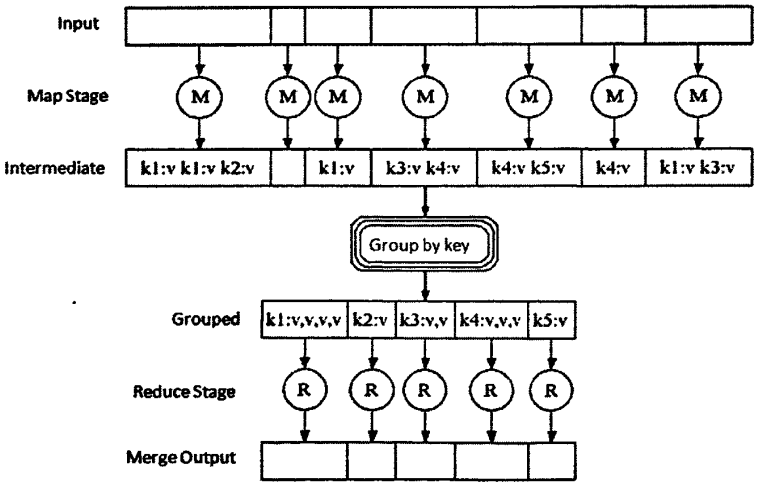


图 2-6 MapReduce 调用详细步骤及数据流程图

2.3.3.3. MapReduce 的任务颗粒度和并行

网络带宽一直是 MapReduce 模型的一个瓶颈，因此如何减少计算过程中带宽的消耗可以使提升整体性能和系统的吞吐量。基于以上的原因，我们采取将计算向存储转移的策略。Google 的 GFS 将文件分成 64M 大小的块，每一块都有几个拷贝（通常是 3 个），分布在不同的机器上。主控制程序在分配 Map 任务时尽量把任务放在其对应的输入数据所在的机器上执行，如果不能的话就把任务分到距离输入数据尽可能近的机器上（两台机器在同一个交换机下）。

在上面的实现中，我们提到 MapReduce 会把任务分成小块来执行，在理想状态下，Map 任务和 Reduce 任务要比计算机节点数量多得多，每一个节点要执行多个 M 和 R 任务来提高动态的负载均衡能力，并且能够加快故障恢复的速度（例如在失效节点上执行的 Map 任务都可以重新分配到所有其他的节点上进行执行）[12]。

然而，实际情况下 Map 和 Reduce 的取值是由一定限制的，因为主控制程序必须执行 O（M+R）次调度，并且在内存中保存 O（M*R）个状态。进一步讲，用户通常会指定 Reduce 值，因为每一个 Reduce 都将最终产生一个独立的输出文件。通常我们习惯调整 Map 的值，使得每一个独立的 Map 任务处理大约 16M~64M 的输入数据（这样上面所讲的计算向存储转移的策略会最有效），而且 Reduce 通常指定比较小的值，这样 Reduce 任务就不会占用太多的节点。一般情况下 Map、Reduce 和节点是这样的比例[23]：Map=200000，Reduce=5000，使用 2000 台工作节点。这种任务颗粒度的划分使得 MapReduce 各阶段可以更好的并行执行，且能提供更好的动态复杂平衡，其并行过程如图 2-7 所示：

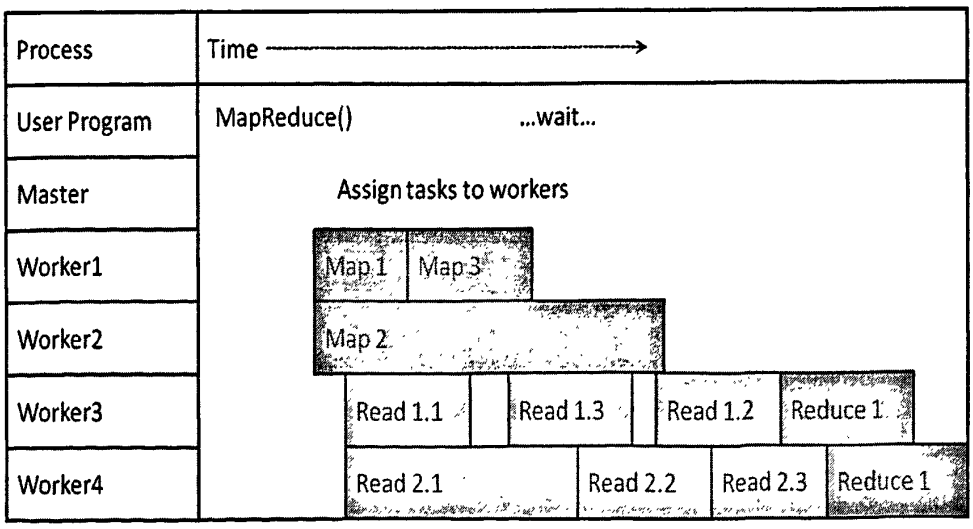


图 2-7 MapReducede 执行次序

2.3.3.4. MapReduce 的容错考虑

MapReduce 是被设计用来处理成千上百台机器上的海量数据的，因此它的函数库必须还要考虑机器故障的容错考虑。

首先考虑工作节点的失效考虑。主控制程序会定时 ping 各个工作节点，如果某工作节点没有在指定时间内返回相关信息，主控制程序认定该工作节点失效，所有这个节点已完成的 Map 任务和正在处理的 Map 任务和 Reduce 任务都将被重置成初始空闲状态，其它任何工作节点都可以重新执行该任务。之所以失效节点完成的 Map 任务需要重新执行，是因为再详细过程 3 中，Map 任务产生的中间结果会被缓存在本地磁盘中，导致中间结果无法访问。在 Reduce 任务完成后，最终结果已经被保存在了全局文件系统中，所以失效节点完成的 Reduce 任务不需要再被重新执行。当 Map 任务切换到其它几点上执行时，所有执行 Reduce 任务的节点都被告知这一事情，这样的话，还没有来得及从失效节点读取数据的 Reduce 工作节点会从新的节点读取数据。这种容错机制在失效工作节点数量比较大的情况下也是适用的。主控制程序只需要把所有节点上的任务重新执行一次，就可以完成 MapReduce 操作。

第二种情况是主节点失效。主节点会定期写出其数据结构，我们称之为检查点，从而在主控制程序失效后可以从最新的检查点恢复执行操作。但是，一个系统中只运行一个主控制程序，所以它失效了就比较麻烦了。在目前这种实现架构上，如果主节点时效，就需要终止 MapReduce 的执行。客户端可以检测到主节点的失效，并按需重新执行 MapReduce 操作。

2.3.4. Hadoop 中调度算法的研究

Hadoop[15]是开源社区 Apache 旗下的项目，使用 Java 进行开发，目前的版本号是 0.20。它是对 Google 公司 MapReduce 模型的开源实现，虽然该框架的实现并不是非常完善，但由于 Google 对 MapReduce 的保密性使得它仍然是我们现在研究云计算基础框架的重要样本。

它需要运行在大型集群上，并对成千上万的任务进行调度的一个海量数据的并行处理系统，如何能够满足各种作业的不同需求，给用户提供更便捷的服务，是 Hadoop 平台必须重视和解决的难题。Hadoop 的任务调度是一个主从式的模式，一个叫做 JobTracker 的主节点控制整个系统的作业调度，其余节点为 TaskTracker，在他们资源空闲时向 JobTracker 请求分配任务。如何选择合适的调度程序运用于 Hadoop 平台对 Hadoop 的执行能力和交互能力有着非常大的影响，因此业界也提

出了许多针对 Hadoop 的调度模式，但是目前 Hadoop 系统中使用最多的仍然是 FIFO 调度算法[13]，其中雅虎和 Facebook 的开发者也分别提出了 Hadoop 作业调度算法，并且已经投入到新版本的 Hadoop 实践中去。

1. 先进先出调度算法（First In First Out）

在 FIFO 调度算法中，所有的作业按照用户的提交时间顺序执行，Hadoop 使用一个 JobQueue 维护用户提交的作业，作业的分配有 JobTracker 完成。FIFO 算法的思想非常简单，但是它在对用户提交作业不做区别，而现实应用中，不同作业往往根据用户类别而有各种各样的差异。例如用户支付费用的不同，会导致系统对不同用户的请求给予分等级的服务。而且，在一个生产型作业之后，他将长期使用系统资源，在这个作业提交之后的交互型作业由于得不到系统资源而不能及时执行，从而影响了系统的交互能力。

2. 公平份额调度算法（Fair Scheduling）[14]

它是由 Facebook 提出的一种新的调度算法，Facebook 的初衷是让 Hadoop 更好的处理不同类型的作业需求。

公平份额调度算法的基本思想是最大化的保证系统中作业平均分配系统资源。假如在系统中仅存在一个执行的作业，那么它会占用全部的系统资源；当有新的作业进入系统后，一些 TaskTracker 会被释放并执行新作业的任务，而且保证各作业之间能够获得大体一致的计算机资源。这样，交互型作业不会等待太多时间，而生产型作业也会在合理的时间内完成。

在这种调度算法中，系统使用作业池来维护用户提交的所有作业，在作业池中的作业可以平等的使用系统资源。在缺省情况下，公平份额调度算法为每个用户建立一个单独的作业池。这种做法的好处是使得系统可以根据用户的数量平均分配系统资源，而不管每个用户实际提交了多少个任务。在实际应用中，公平份额调度算法通常会给不同的作业和作业池赋予不同的权值，这使得该算法不再是绝对的公平分配，但却更符合现实中的应用环境。它使得系统可以根据任务的重要程度等各种因素合理的为不同用户的不同作业分配系统资源。

在用户资源池中的全部作业都可以被公平份额调度算法调度执行，但是通常它会限制每个用户作业池中运行作业的上限，对作业数量进行限制非常有必要，因为过量的作业同时执行会产生巨量的中间记录信息，更会带来过度频繁的上下文切换，从而影响了系统的性能。对作业数量的限制并不会导致用户晚提交的作业失败，它们只是需要等待系统中更早提交的任务执行完毕。

公平份额调度算法根据作业赤字来决定各作业之间获得资源的水平。作业赤

字就是作业在理论上应该获得的系统资源和作业实际获得的系统资源的差值，这、里系统资源主要指作业的执行时间。作业赤字越大表示作业当前获得的资源水平越低，那么当又空闲 TaskTracker 来请求任务时，应该优先分配赤字大的作业的任务。

3. 计算能力调度算法（Capacity Scheduling）[15]

计算能力调度是由雅虎提出的作业调度算法。它在功能上和公平份额调度算法相似，但在设计和实现上却有着很大的差异。

在计算能力调度算法使用多个 JobQueue 维护用户提交的作业，每个 JobQueue 都可以根据配置获取一定的 TaskTracker 执行任务，计算能力调度算法按照配置文件为不同的队列分配合理的系统资源。一个任务被提交给系统时被随机放入某个队列中。计算能力调度算法采取下面的措施来提高系统的资源利用率：当某个系统资源已经被分配给某 JobQueue 但未被使用时，每个 JobQueue 公平的分享这些资源；当没有按照配置的设定数获得充足的资源的 JobQueue中的作业处理量增大时，之前分配给该 JobQueue 又被其他 JobQueue 占用的系统资源在完成当前任务后立刻返还给它应属的 JobQueue。可见，计算能力调度算法将每一个 JobQueue 模拟成为了一个具有特定处理能力的独立的 Hadoop 集群资源。

计算能力调度算法在每个 JobQueue 中采取基于优先级的 FIFO 算法。每个 JobQueue 中的 Job 按照优先级排列按顺序获得系统资源。计算能力调度算法是非抢占式的，即当一个任务开始执行之后，它不会因为 JobQueue 中有进来一个优先级更好的任务而放弃已占用的系统资源。新加入的作业只能等待正在执行的作业执行完成，再按照优先级访问系统资源。

此外，计算能力调度算法还会根据作业的不同类型进行按需调度。如果一个作业需求大量的内存资源，那么调度算法就要保证该作业会被分配到内存资源充分的 TaskTracker 上去执行，避免由于资源不足造成任务无法执行或者执行速度缓慢。

以上是三种 Hadoop 平台中常见到的调度算法。其中 FIFO 调度算法仍然是 Hadoop 平台中最常用的调度算法，因为其算法简单明了，适合海量数据处理时大规模的任务调度。调度算法现在已经成为了 Hadoop 业内最常讨论的话题之一。在现实中，调度算法已经成为 Hadoop 平台的一个可插拔的组件，这更有利于广大用户了解讨论，并对 Hadoop 调度算法提出更新更有创造性的建议。另外，主从式的调度方式也成为激烈的讨论话题，一旦 JobTracker 宕机，整个系统将停止工作。因此在未来 Hadoop 的调度算法的发展可能会由多个 JobTracker 协同完成，这将会

是 Hadoop 作业调度和资源管理的一个重要的研究方向。

2.4. 本章小结

本章是介绍性的章节，主要介绍了云计算的基本概念、模型、特性以及在业内的典型应用。并对云计算的并行处理模式 MapReduce 作了详细的介绍，包括 MapReduce 编程模型、实现框架等方面的内容。MapReduce 的开源实现 Hadoop 的调度算法的发展。阐述了调度算法对于 Hadoop 平台的重要性，以及未来可能的发展方向。

第三章　MapReduce 在 Hadoop 中的性能评估及分析

Hadoop 是开源组织 Apache 对 Google 的 MapReduce 的开源实现，对其系统结构的性能优劣进行评估是一个困难且任务繁重的工作，它需要考虑到系统的实现架构、应用环境、主要业务特点等多方面的因素。联想到 MapReduce 是运行在大规模集群上处理海量数据的并行计算并且还需要提高与用户的交互性等特点，本文在这里设定了如下的评估指标、测试环境搭建、基准测试程序集以及评估目标。

3.1. Hadoop 平台的研究

Hadoop 设计时基于以下的几点假设上：将服务器失效看做正常现象，存储和处理的数据是海量的，文件不会被频繁写入和修改，机柜内的数据传输速度大于机柜间的数据传输速度，海量数据的情况下移动计算比移动数据更有效。

支撑 Hadoop 框架的两个核心技术是 HDFS（Hadoop Distributed File System）[17]和 MapReduce，其中 HDFS 是源自 GFS[18]，是 Hadoop 框架中底层的文件存储系统，MapReduce 是在 Hadoop 的并行数据处理模型。

Hadoop 从架构上来看是一种主从式结构，表现在一个主从式的文件系统 HDFS 在底层支撑其主从式的 MapReduce 数据处理功能。在 HDFS 的支持下，Hadoop 可以轻松实现计算项存储转移这一策略，减少了数据传输给并行计算带来的时间和资源开销，使计算速度大幅度提升。主从式的基础存储系统和主从式的数据处理最终决定的 Hadoop 的基本架构模型。

3.1.1. 主从式的 HDFS

HDFS 有一个叫做 NameNode 的主节点和若干叫做 DataNode 的子节点组成。NameNode 负责存储文件系统的元数据，诸如文件系统的名字空间等，向用户映射文件系统并负责管理文件的存储等服务。DataNode 存放实际的数据，一个文件被切割成若干 16M-64M 的小块，这些数据块分散在各个 DataNode 上，同时 HDFS 还将为数据块在不同的机架上设置副本用来保证数据的安全性（默认拷贝数为 3）。用户获取数据并需要通过主节点，而是直接与数据块所在的 DataNode 直接通信[17]。其系统的架构如图 3-1 所示：

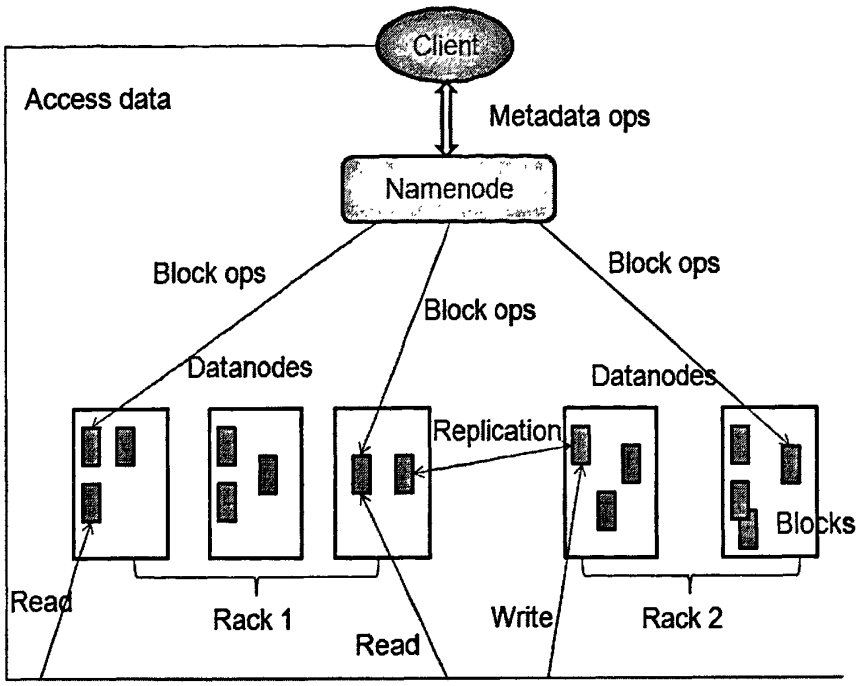


图 3-1 HDFS 架构

HDFS 的工作过程是这样的，用户请求操作文件的指令由 NameNode 接受，NameNode 将存储数据的 DataNode 的 IP 返回给用户，并通知其他接受副本的 DataNode，有用户直接与 DataNode 进行数据传送，与此同时，NameNode 更新其存储的元数据内容。整个文件系统采用标准的 TCP/IP 协议通信，实际上是架构在 Linux 的一个上层文件系统。其典型的文件大小一般都在 GB 值 TB 的数量级。

3.1.2. 主从式计算系统 MapReduce

MapReduce 模式在 Hadoop 中的实现流程如图 3-2 所示：

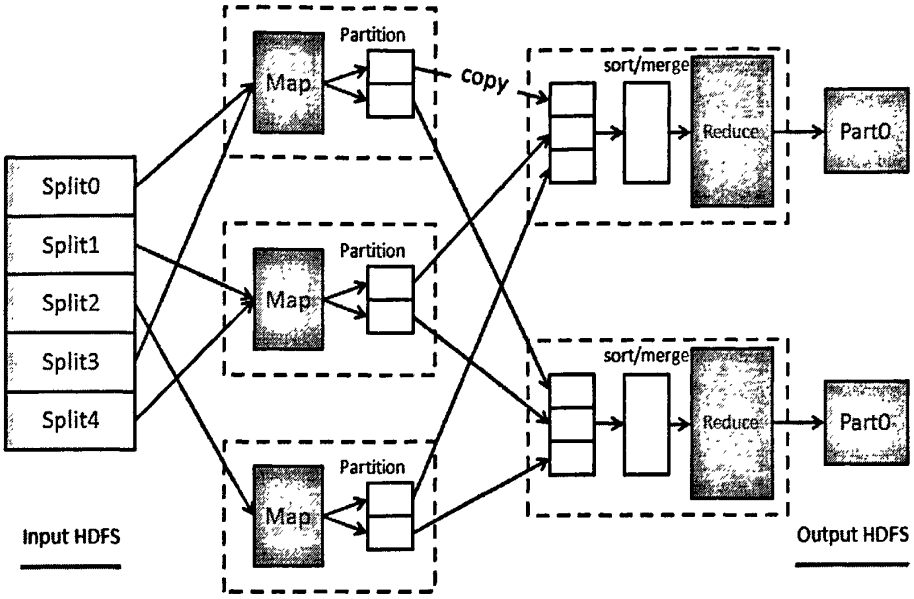


图 3-2 Hadoop 中 MapReduce 流程图

在 Hadoop 中，同样对输入文件进行分割成若干较小的相互独立的任务块供 Map 函数处理，输出一个中间的键-值对。这个中间结果可以先用 Hadoop 自带的合并器（通过调用 combiner 类）做初步的合并，这一个过程是在 Map 的节点上进行的，其目的是为了减少中间数据的传输而带来的网络流量。再将 Map 处理的中间结果传送给 Reduce 之前还需要将中间结果进行分区（通过调用 pationer 类），来确保中间键-值对中键值相同的会被同一个 Reduce 所执行。

Hadoop 中 MapReduce 实现的任务流程图如图 3-3 所示：

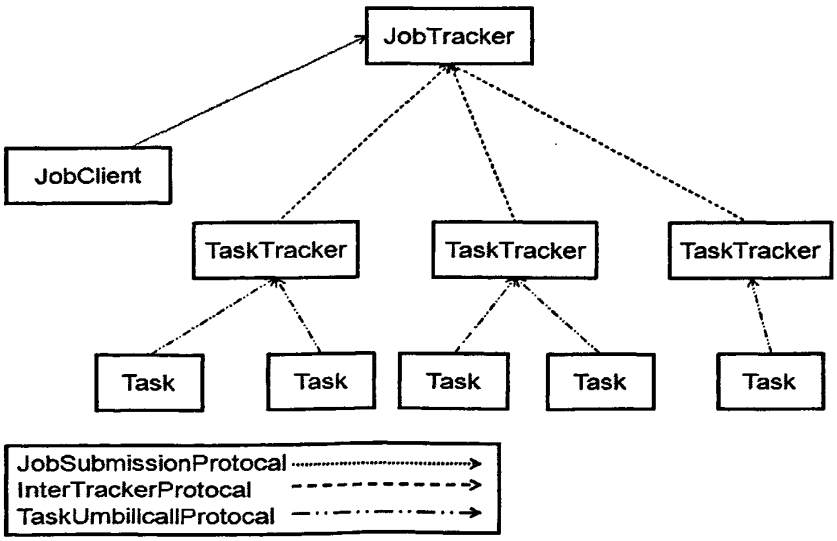


图 3-3 Hadoop 中 MapReduce 的任务流程图

与 HDFS 类似，在 MapReduce 中有一个被称为 JobTracker 的主程序，他负责整个 MapReduce 中任务的控制管理工作，JobTracker 需要读取文件块的存储信息，因此 JobTracker 通常和 NameNode 在同一个节点上，并负责创建子节点的 TaskTracker（从属任务），TaskTracker 直接在子节点上进行处理，完成了计算到存储的迁移。TaskTracker 将状态和完成的信息向 JobTracker 汇报。

3.2. MapReduce 性能评估指标的设计

MapReduce 已经被广泛运用到了现实中的服务程序当中，例如搜索引擎应用、信息统计、科学计算、网络数据挖掘等。结合上述应用的特征，我们制订了以下的几个指标来评估 MapReduce 在实际应用中的性能，总分为以下 5 各方面：

1. 任务独立响应时间与任务总响应时间

任务独立响应时间是指运行单个任务的时候，系统从提交到返回最终处理结果的时间，不管是正常完成、失败或者取消。这是一个很重要的指标，它反映了 MapReduce 模型和用户的交互能力，响应时间越短证明系统与用户的交互能力就越强，用户可以在很短的时间内就知道自己提交的作业的运行情况，从而使用户获得更好的服务和交互性。

总响应时间在交付系统定量的任务时所有机器完成任务所消耗的时间，它反映了整个系统的计算能力和吞吐量，小的总响应时间说明该模型可以使用少量的计算机资源去完成同样的任务。

2. 平均响应时间

这个指标是在多个任务并行进行下统计的。MapReduce 是一种并行计算的模型，他就是被设计来在并行的处理海量数据的，大量任务并行处理下的性能是反映该模型好坏的重要衡量标准。因此这个指标的优劣直接反映了它在大量任务并行执行时的处理能力和系统的吞吐量，是一个非常重要的指标。

3. 加速比

加速比是指对于相同的数据和任务处理量，当增加计算机数量和集群规模的情况下，对并行计算能力提升的比率。它主要反映了 MapReduce 模型在可扩展方面的性能。如果通过增加并行处理节点的数量，可以大大的提升相同任务下的响应时间则可以说明 MapReduce 是一种具有高可扩展性的并行计算模型。

4. 公平性

公平性是指在多个任务同时执行的情况下，系统任务的调度策略是否满足某

些公平性原则。比方说对于短作业和提交失败需要重新执行的作业应该赋以较高的优先权，优先执行，对于长作业则可以酌情延后。具有良好的公平性可以对计算模型的其他指标也有一定的提升。

5. 容错性

考虑到 MapReduce 的使用环境——超大集群，计算节点的失效是非常正常的正常，它在实现上将节点故障视作正常现象，也是分布式计算系统必须处理的正常问题，因此在节点出现故障后的任务恢复能力和系统的稳定性成为 MapReduce 的一个重要的测试指标。

3.3. 设计基准测试程序集

3.3.1. 基准测试程序的设计

基准测试程序的选择是决定我们性能评估正确性和真实性的一个决定性因素。它应该能够代表 MapReduce 的典型应用，最典型的就是正在运行中的应用程序的集合。因此本文挑选基准测试程序都代表了 MapReduce 实际中典型应用的关键程序，使得本文的分析结果最具代表性。

本文选择的基准测试程序在应用上覆盖了诸如搜索引擎（网页级别、倒排索引）、常务分析统计（字数统计，模式匹配）、网页挖掘（Kmeans、线性回归）以及代表性的两阶段处理计算 PennySort。下面对本文选择的 3 个基准测试程序做简要介绍。

3.3.1.1. 字数统计

在前面的 2.3.3 节已经提到了它代表了 MapReduce 的典型处理过程。本问中的统计是在 CWT200G 中文网页数据集上进行的。Map 任务的主要工作是对每个静态网页作分析，除去网页上的标签，并将网页上的语句段落按照逻辑拆分成常用的词组，Map 操作的关键字就是这些才分出来的词组，并将键-值对的值赋值 1；Reduce 任务将中间键相同的键-值对进行合并，把每个键的值相加得出静态网页中的各个词的出现频率。它测试了 MapReduce 在处理大规模数据使时的稳定性和可靠性。

3.3.1.2. 网页级别

网页级别（PageRank）是 Google 用来决定一个网页的等级或者说重要性的一

个方法，它揉合了标题标识和关键词标识等一些其他的因素，经过 PageRank 值（PR 值）做调成，成为了 Google 判断一个网站好坏的唯一方法。它使得 Google 可以让重要的网页在用户的搜索结果中占据靠前的位置，从而使用户可以获得权威的信息，提高了搜索的结果的相关性和质量。影响一个网页 PR 值的影响因素有很多，例如与 PR 值高的网站做链接、提高本网站链接内容的质量、加入搜索引擎分类目录等。具体的 PR 值可以通过下面的公式计算得到：

$$PR(A) = (1-d) + d(PR(T_1)/C(T_1) + \cdots + PR(T_n)/C(T_n))$$

上式中 d 表示阻尼因数，它表示你投票或者链接到其他网站是获得的实际的 PR 值，阻尼因数是介于 0 到 1 之间的一个数，通常情况下去 0.85。T_i 表示指向网页 A 的某个网页，$C(T_i)$ 表示网页 T_i 向外的链接数，可见一个网页的 PR 值是由所有指向该网页的 PR 值所决定的，这个值经过若干轮的迭代最终产生，可以证明 PR 值的迭代式收敛的。

使用 MapReduce 计算一个网页的 PR 值分为以下两个部分：

第一阶段，构建链接图。Map 函数通过分析每一个网页，将网页的 URL 设置为键，给网页赋值一个初始的 PR 值，并且将网页所包含的全部指出的 URL 合并在一起作为输出值，可以表示为 $URL \rightarrow (PR(URL), URL_1, URL_2, \cdots, URL_3)$。Reduce 只需要把 Map 过程的输出值作为最终结果输出即可，不需要做其他任何特别的处理。

第二阶段，迭代阶段。它以上一个阶段产生的链接图为输入，用下面的 MapReduce 计算，进行多次迭代直到数据收敛为止。Map 函数将输入记录做成如下的映射 $(URL \rightarrow URL_1, URL_2, \cdots, URL_3)$ 和 $(URL_1 \rightarrow PR(URL)/C(URL))$，$(URL_2 \rightarrow PR(URL)/C(RUL))$ 等。Reduce 函数负责把相同的 URL 的值进行汇总计算出网页的新的 PR 值，和原输入按照相同的格式输出。然后使用一个简单程序来检测前次的迭代 PR 值和本轮得到的新的 PR 值是否收敛，如果收敛则把这个 PR 值输出作为最终结果，如果未收敛则重复第二阶段。

在本文中，我们模拟网页的链接分布，自动生成若干个 URL，这些 URL 的分布根据 Zip-f 分布决定。

3.3.1.3. PennySort

PennySort 是由图灵奖得住 Jim Gray 提出的 Benchmark 排序的一种形式，Benchmark 排序要求待排记录长度必须是 100 字节，并且前 10 个字节是一个随机的 key。它可以用 MapReduce 来完成，首先 Map 函数提取待排记录的前 10 个字节

作为键，后面的字节作为值与之组成键-值对。分区函数将键的范围作为分区的标准，Reduce 函数将 Map 函数分割后的数据对待排记录进行排序，输出排序后的结果。

3.3.2. 基准测试程序集的衡量指标

考虑到 MapReduce 的实现方式和实际应用，本文选择的基准测试程序主要考虑到了如下的衡量指标：

1) 任务规模：处理数据的字节数，全部 Map 任务和 Reduce 任务的个数。
2) Map 任务的可选择性：在所有的 Map 任务中，数据的输出字节数同数据的输入字节数比的平均值。
3) Reduce 任务的可选择性：在所有的 Reduce 任务中，数据的输出字节数同数据的输入字节数比的平均值。
4) Map 任务的字节计算速度：即 Map 任务处理数据的总字节量和总花费时间的比值。
5) Reduce 任务的字节计算速度：即 Reduce 任务处理数据的总字节量和总花费时间的比值。
6) Map 的形式：是读入分割块的一部分数据还是将分割块作为整体按照顺序读入。
7) Map 的复杂度
8) Reduce 的复杂度

3.4. 实验平台的搭建

3.4.1. 集群配置方案

本文的实验都是在 IBM 技术中心实验室的硬件环境上进行的。整个集群共由 9 台普通 PC 机组成，其中 1 台作为主控制节点，另外 8 台是工作节点。我们将工作节点分为 2 组，每组 4 台 PC 机构成一个机柜，在同一个机柜内的工作节点通过全双工 1Gbps 以太网卡与 IBM SNA2005-B16 1Gbps 交换机相连接，连接两个机柜的是一个千兆路由器。

在集群中的每台计算机都是 IBM ThinkCenter M4000t 的 PC 机，配置 2G 内存，CPU 配置为 Intel 酷睿 2 双核 E840，硬盘为 7200 rpm 的 SCSI 硬盘。每台机器上

都装有相同的 Linux RedHat Fedora 10 作为操作系统。除此之外，我们还将 4 台机器的硬盘做成一个 RAID-0 的逻辑卷。

集群的结构示意如图 3-4 所示：

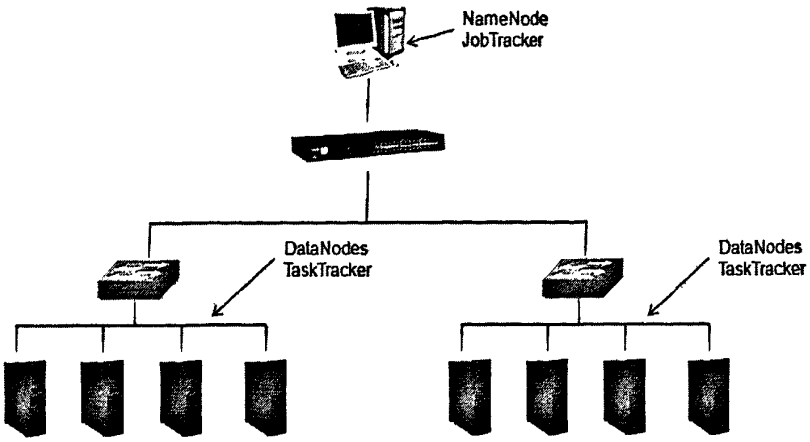


图 3-4 平台结构示意图

3.4.2. Hadoop 的配置与安装

Hadoop 是基于 Java 编写运行于大型主机群上的应用程序，因此对 Hadoop 进行安装首先要配置好 ssh 和搭建好 Java 运行平台，配置 ssh 和安装 JDK 没有先后顺序之分[19]。

3.4.2.1. 配置 ssh 和 JDK

Hadoop 需要在各个工作节点上传递数据或者控制信息，因此需要实现节点之间的无密码访问，否则系统会不停地向用户请求密码验证，为了叙述方便，本文仅以两台机器的 Hadoop 搭建作为示例，多台机器的环境搭建与两台机器类似，且在具体区别的地方，本文也给出了注解，在本文设计的平台上的配置如下[19]：

1) 向每个节点的/etc/hosts 文件中增加节点 IP 地址和机器名的对应关系，并在各个节点上建立相同的用户，我们修改后的/etc/hosts 如下所示：

   121.48.165.176　PC1

   121.48.165.177　PC2

2) 去除节点间用户的 ssh 访问密码。在 NameNode 的/root 目录下建立.ssh 目录，并进行 ssh 配置，具体如下所示：

   ssh –keygen –t dsa

   cd /root/.ssh

cp id_dsa.pub anthorized_keys

chmod go –rwx anthorized_keys

scp –r /root/.ssh PC2: /root

在多个节点的情况下需要把PC1的.ssh文件夹中的内容复制到每一个工作节点上。

本文选择 JDK1.6.0_14 作为 Java 的运行版本，安装完成后对相应的环境变量进行设置。

3.4.2.2. Hadoop 的安装配置

本文使用 Hadoop v 0.20 版本，安装完成后，需要对它以下几个配置文件做适当的修改[19]：

1) 修改 Hadoop-env.sh，master 和 slaves

修改 Hadoop-env.sh 里面的 JDK 路径，在本文的平台下为：

export JAVA_HOME=/usr/java/jdk1.6.0_14

修改 master 文件，它对应 NameNode 所在的机器，本例中为 PC1，slaves 文件对应 DataNode 所在的机器，分别是 PC2~PC9。

2) 修改 core-site.xml 配置文件，配置 NameNode 的 IP 和端口号，具体配置信息如下所示：

```
<configuration>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://PC1:9000</value>
    </property>
</configuration>
```

3) 修改 hdfs-site.xml 配置文件，配置数据备份数量。具体配置文件信息如下所示：

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```

4) 修改 mapred-site.xml 配置文件，配置 JobTracker 的 IP 和端口。具体配置文件信息如下所示：

```
<configuration>
    <property>
        <name>mapred.job.tracker</name>
        <value>PC1:9001</value>
    </property>
</configuration>
```

在把所有机器上的配置文件修改完成后，仍需要格式化 NameNode 才能够完成对 Hadoop 的安装过程，格式化可以使用 Hadoop 的 bin 文件夹下的命令来完成，使用格式如下：

```
hadoop  namenode  –formate
```

经过以上的步骤后我们完成了对 Hadoop 平台在一个 9 台 PC 机群上的搭建，此时我们可以通过运行 bin 目录下的 start-all.sh 来启动 Hadoop。本文对 MapReduce 模型性能的测试和分析以及下面一个章节对调度算法的改进及测试都将在这个平台上进行。

3.5. 实验方案设计

3.5.1. 数据结构的设计

为了对系统运行时对各个子任务性能指标进行统计，我们用下面的数据结构保存实时的数据信息：

```
Class ProfileInfo
{
    // Statistics Info for Map
    int   mapDataIn;
    int   mapDataOut;
    int   mapRecNum;

    //Statistics Info for Local Combine
    int   localCombDataIn;
```

```
    int   localCombDataOut;
    int   localCombRecNum;

    //Statistics Info for Reduce
    int   reduceDataIn;
    int   reduceDataOut;
    int   reduceRecNum;

    //Statistics Info for Transfer
    int   tranDataInOut;
    int   tranRecNum;

    //Statistics time : by seconds
    int   taskCostTime;
};
```

该数据结构主要负责统计以下几个阶段的数据：在 Map 阶段，统计 Map 任务处理数据的大小、输出数据的大小、处理了多少条记录；以及做本地结合的的数据的输入和输出值、处理的记录个数；在 Reduce 阶段，统计 Reduce 任务处理的数据的大小、输出数据的大小、处理了多少条记录；以及传输阶段传输的数据（包括输入和输出）；最后对记录下每个任务所花费的时间。

3.5.2. 对统计信息进行分析

工作节点执行任务后记录下所统计的性能的对应数据后，由本地文件管道传递给工作节点的心跳程序，心跳程序通过远程过程调用（RPC）[20]将性能信息传送给主控制节点。

远程过程调用是指调用过程的代码并不在发起调用的机器上，它允许通过 TCP 或 UDP 使调用者和被调用者可以建立连接和通信，从而可以发起调用。它有点类似 C/S Socket 编程，RPC 的基本通信模型可以说是 C/S Socket 编程的一种同步通信方式。其调用协议如图 3-5 所示：

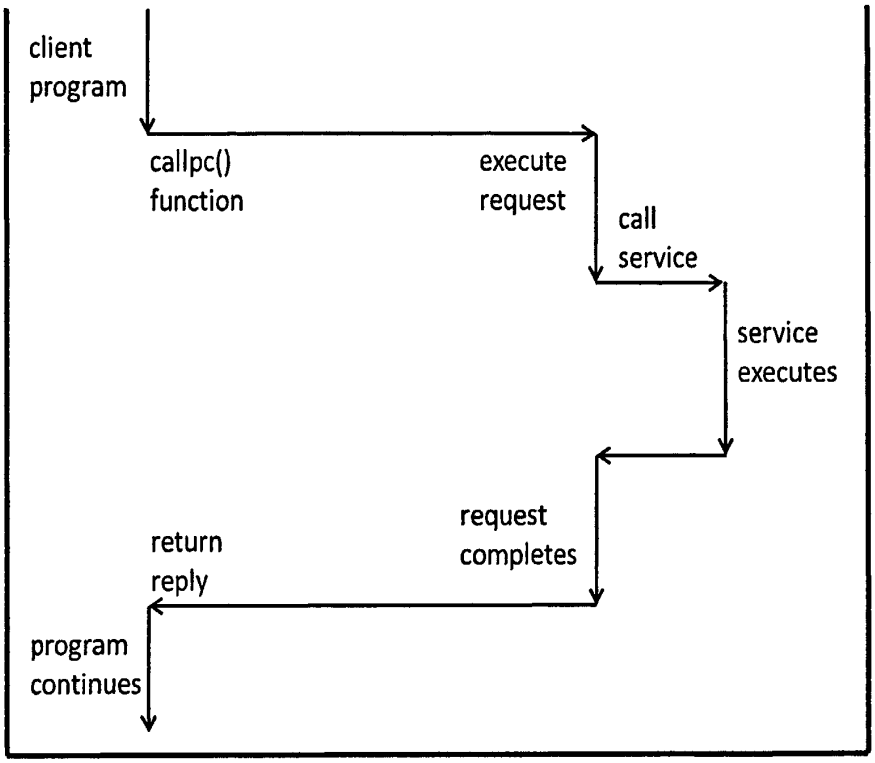


图 3-5 RPC 调用协议图

下面对 RPC 的调用协议做简单的解释。首先要建立 RPC 服务，规定底层的传输通道使用的协议（TCP 或 UDP）。客户端通过之前提供的目的 IP 地址和 RPC 上的应用程序号，经过传输通道传递给相应的服务器。此后客户端进入等待状态，在额定的时间内等待服务器作出答复直至超时（Time-Out）发出超市信号。服务器端在接受到服务请求之后，会找出传送的应用程序号所对应的程序执行的入口地址，服务器转入执行请求的服务，执行完成后将结果反馈给客户端。

3.6. 实验结果及分析

本文的采用字数统计、PennySort 和网页级别作为 MapReduce 性能测试的 3 个基准程序，对于每个基准程序执行时数据的负载规模如下：

1. 字数统计采用总计 1.2GB 的数据量，并且使用本地结合；
2. PennySrot 采用总计五千万条记录，每条记录 100K，合计 5GB 的数据量；
3. 网页级别采用总计 150 万条 URL 记录，合计 1.5GB 的数据量。

3.6.1. 任务独立响应时间与任务总响应时间

表 3-1 展示了我们实验所得到的任务独立响应时间和任务总响应时间：

表 3-1 独立响应时间和总响应时间

程序/时间（秒）	独立响应时间	总响应时间
字数统计	252	2128
网页级别	227	1975
PennySort	419	4387

独立响应时间是每个基准程序独立执行的情况下，从用户的作业提交到作业全部结束所用的时间，总响应时间表示某一个作业的全部任务（包括 Map 任务、Reduce 任务和 Transfer 任务）的完成时间的总和，它从整体反映了某一个作业对于资源的消耗和需求量。

对任务独立响应时间和任务总响应时间做统计主要是为了后面平均响应时间等指标做基准，从而测试 MapReduce 在扩展性和公平性等方面的表现。

3.6.2. 平均响应时间

对平均响应时间的评估我们从两个方面来考虑——同构机群和异构机群。

3.6.2.1. 同构机群下 MapReduce 的平均响应时间

同构集群下的测试平台既我们之前搭建的 1 台主控制节点和 8 台工作节点组成的机群，每台机器的硬件配置和安装的操作系统都是一样的。在这种情况下，我们同时向系统提交字数统计、PennySrot 和网页级别这三个作业，每个作业处理的数据规模和 3.51 节中是一致的。通过实验测得三个任务的平均响应时间是 304 秒。

3.6.2.2. 异构机群下 MapReduce 的平均响应时间

为了搭建一个异构的平台，我们在每个机柜上随机选择一台工作节点，用实验室内较老的计算机替换他们，配置为内存 512MB，硬盘 5400rpm 串口。在这个平台下我们测试得到 3 个任务从提交到全部结束共花费 323 秒。

理论上讲，在并行计算中，并行任务的平均响应时间应该是每个任务的独立响应时间的平均值，按照我们 3.5.1 中的数据，平均响应时间应该是 299.3 秒。在同构机群下 MapReduce 的并行计算的性能其实与独立运行的计算能力所差无几，然而在异构机群中，并行计算的性能却并不尽如人意，大大的打了折扣。

上述数据显示在同构环境下 MapReduce 对并行的各任务的进行调度使得系统

可以在一个可接受的时间范围内完成对一批任务的优化处理。而对于异构的机群，其调度使得计算能力有比较大的下降。我们通过对系统进行实时监控，发现字数统计和网页级别作业的计算中，某些 Map 任务在调度上没有获得足够的优先权，而落后于其他的 Reduce 任务或 Transfer 任务，依赖于上述 Map 任务的其他任务必须等待该 Map 任务完成才能开始执行，即使当前的系统中仍然存在大量的空闲的工作节点。其次，我们发现 MapReduce 将忽略异构环境中机器的性能差异，平均的将分割后的任务分配给每一个工作节点，在一些性能好的工作节点上的任务全部完成后，性能差的节点的任务还在等待前面任务的执行，不能及时的将这些等待中的任务分发给已经空闲的节点。

从上面的数据我们得出这样的结论，当前的 MapReduce 模型中的调度算法还不能非常适用于异构机群平台上面的应用，本文后一个章节也会针对这种问题对 MapReduce 模型提出一种新的调度算法，使得 MapReduce 在异构平台上仍然可以有非常好的表现。

3.6.3. 加速比

加速比主要是衡量一个系统在扩展性方面的优劣，同时可扩展性也是衡量 MapReduce 这种并行计算模型性能的非常重要的一个指标。优异的可扩展性才使得 MapReduce 可以区分于一般的分布式系统，并从众多并行计算模型中突出出来，可以被用来广泛的使用在超大规模的集群上。

可扩展性分为两个层面，第一个是指当系统的硬件资源增加时，对于相同规模的作业，系统的处理能力；第二个是指当系统和硬件资源和处理作业的规模保持相当的水平增长时，系统的处理能力。

对于第一种情况，我们选择计算规模和 3.5.1 中的一样，系统处理数据的保持在这个规模，而系统中的工作节点从 1~8 个不等，看在逐渐增加工作节点的情况下，系统中各个任务的响应时间的变化。如图 3-6 所示：

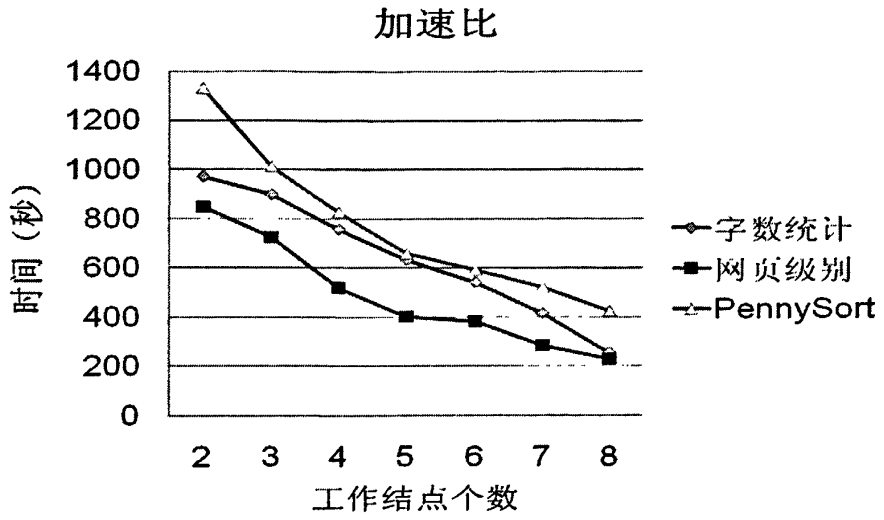


图 3-6 MapReduce 加速比实验结果

从上图可以看出，增加工作节点可以显著地提高系统对同规模数据的处理能力。每个作业运行的时间都随着工作节点的增加而降低，这说明了 MapReduce 在并行计算的应用上具有非常好的可扩展性。

对于第二种情况，我们设定不同机群的工作结点数分别为 2 台、4 台和 8 台，对应不同规模的机群系统处理的数据量也按照相应的水平进行增减。具体的数据规模分别是：

1. 数字统计：1.2GB；2.4GB；4.8GB。
2. 网页级别：150 万条 URL 记录，合计 1.5GB；300 万条 URL 记录，合计 3.0GB；600 万条记录，合计 6.0GB。
3. PennySort：5000 万条记录，合计 5.0GB；1 亿条记录，合计 10.0GB；2 亿条记录，合计 20.0GB。

实验结果如图 3-7 所示：

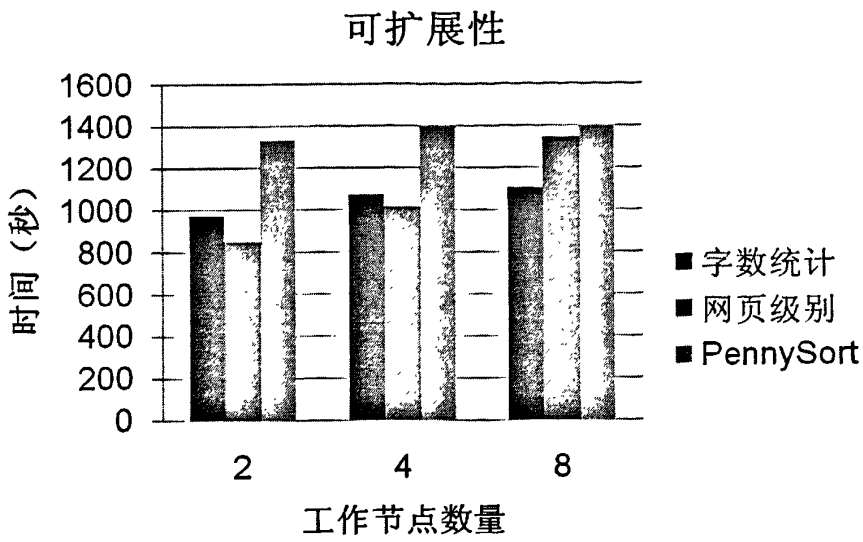


图 3-7 MapReduce 的可扩展性实验图

如图所示，当机群节点数量和处理数据的规模按照同等速度同时增长时，MapReduce 在处理数据上基本保持了相同的水平，这在另一个方面证明了 MapReduce 具有良好的可扩展性。上图中网页级别的响应时间随着规模扩大而呈现出较大的增长是因为随着链接的增加数据规模的增加速率大于线性增加速率。

3.6.4. 公平性

公平性可以从很多不同的场合来考虑，本文主要从任务调度的场合来考虑 MapReduce 的公平性。我们设计这样一种情况：实验涉及到两个任务，任务 A 和任务 B。A 是一个数据处理量很大的长任务，被首先提交给系统，B 是一个轻型的短任务，在 A 提交过一段时间后提交，考虑 A 和 B 的处理情况来评估 MapReduce 对短作业的公平性。

具体的任务的规模如下：任务 A，5 亿条记录的 PennySort 任务，处理的数据量为 500GB；任务 B，1 千万条记录的 PennySort 任务，处理的数据量为 1GB。两个任务的计算类型相同，都是排序。在独立运行的情况下，A 任务需要 2900 秒才能完成，B 任务只需要 40 秒便可以完成。

通过实验我们发现，当 A 任务首先提交之后，B 任务的很多子任务都没有被及时的调度，在 B 提交之后经过了 357 秒才最终完成，而 A 任务的计算延迟为 2831 秒，基本上没有受到 B 任务的任何影响。这也就是说，目前 Hadoop 的调度算法存

在着不合理性，这种调度算法对短任务是不公平的，在短任务提交之后经过很长一段时间猜得到处理。

3.7. 对实验结果的分析

根据对 3.5 节的实验结果的观察和分析，本文总结出以下的几个方面可能成为影响 MapReduce 模型性能的瓶颈。总结如下：

1. 在工作节点性能存在差异的时候，某些配置低的机器工作的处理时间相对较长，这些低性能机器上处理的任务我们称之为落后者，它们会在整体上增加系统对一批作业的响应时间。如图 3-8 所示：

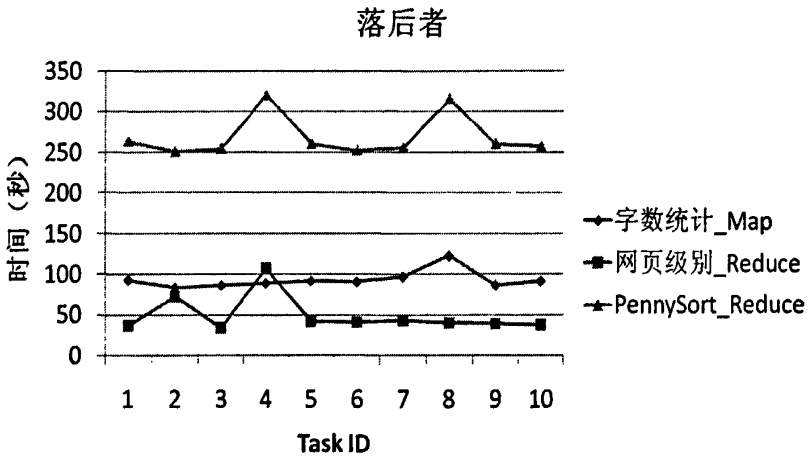


图 3-8 各任务响应时间图

由上图可以看出，在每个应用相对应的不同阶段，其子任务的响应时间大致在一个相当的时间范围内，而又少数子任务的响应时间会超出平均值比较多的数量，这就说明了少数性能差的机器对现有的 MapReduce 模型的计算性能有着非常大的影响，如果合理的修改 MapReduce 的调度算法，来提高落后者的响应时间，则可以优化现有 MapReduce 模型的性能。

2. 网络带宽成为 MapReduce 模型的另一个瓶颈，对贷款的要求和消耗比较大。

我们将提交的作业分成 Map、Transfer 和 Reduce 三个阶段，来考虑每个阶段所占响应时间的比例。我们发现在网页级别的作业中，Transfer 任务所占的比例要比 Map 和 Reduce 阶段要更为突出，占用了任务总处理时间的一半以上，这是由于分块的不平衡和对中间数据的转移所造成的，使得 MapReduce 将大量的时间花费

在了数据转移上，从而降低了系统的计算能力。如何改进传输方式，将计算更好的向存储转移会提高 MapReduce 在并行计算上面的处理能力。实验数据如图 3-9 所示：

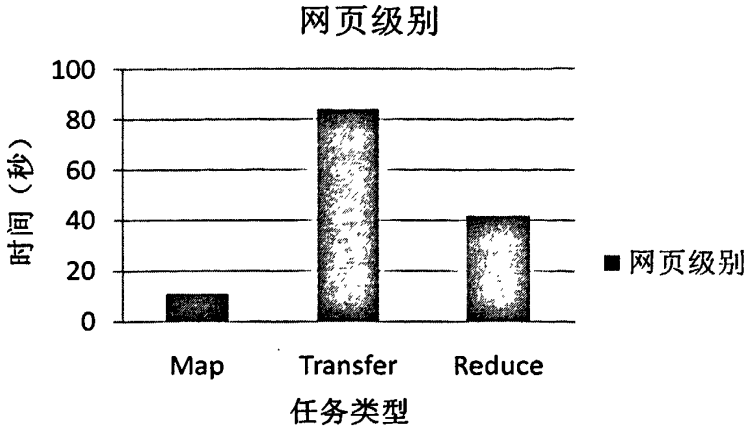


图 3-9 网页级别计算中各阶段响应时间

3. 对任务不合理的分割导致每台工作节点上负载差异大，最终导致 MapReduce 的响应时间。各工作节点的负载如下所示：

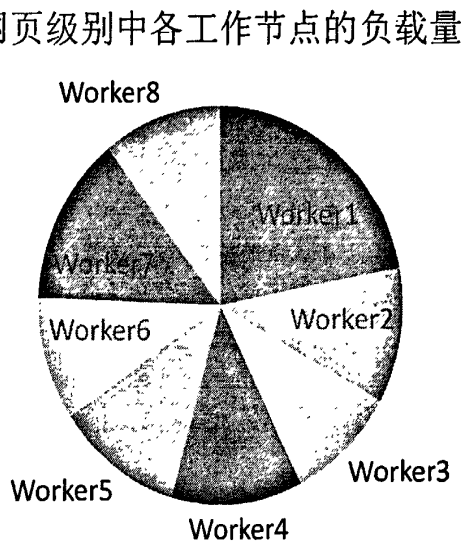


图 3-10 网页级别中各节点负载分配

由图 3-10 可以看出，在对子任务进行分配时，1 号工作节点的被分到得数据量明显大于其他工作节点，任务的分配在出 1 号节点外的其他节点上比较平均，最终导致了 1 号节点成为了系统性能的瓶颈，改善这种任务分配上的不均衡也可以提高现有模型的计算性能。

3.8. 本章小结

本章是论文的重点，主要研究了 Hadoop 的主从式 HDFS 文件系统和 MapReduce 的实现方式。为评估 MapReduce 在典型应用上的响应时间、加速比等性能制订了一系列的基准程序，验证了 MapReduce 并行计算模型的具有强大的计算能力和良好的可扩展性，并根据实验分析出影响目前 Hadoop 平台中 MapReduce 编程模型计算能力的一些因素，例如掉队者问题。为下面第四章提出 Hadoop 的新的调度算法做了实验依据。

第四章 对 Hadoop 调度算法的改进优化

通过上一章对 MapReduce 模型性能进行的实验测试和分析，我们了解到由于部分的掉队者影响到了系统的处理并行任务的响应时间。不管是在 Map 阶段还是 Reduce 阶段，位于某些工作节点上的子任务的运行进度明显要比同阶段同类型的子任务要慢，从而导致整个作业的响应时间增加，影响了 MapReduce 模型并行处理的能力。

响应时间是短作业来说是最重要的一个指标，因为用户往往需要非常迅速的知道程序运行的结果来决定下一步操作，例如用于调试的日志查询分析、系统监控、商务智能等应用程序中，短作业是这这些应用程序的主要用例。同时响应时间对于 SQL_like 的查询系统中也是非常重要的，在这类系统中主要的编程语言有 Pig 和 SCOPE，用这类语言所编写的程序强调了即时查询的重要性，而即时查询需要有良好的用户响应时间。对于按小时付费的网络服务，单任务的响应时间显得尤为重要，因为用户的支出是根据使用资源的时间决定的，响应时间越短则会有更好的经济效益。

本章节将会以如何提高短作业的响应时间为重点目标，在现有的 Hadoop 的调度程序的基础上设计一种新的调度策略，从而改善在异构环境中 MapReduce 的性能。

4.1. Hadoop 中调度程序的研究

4.1.1. 推测执行任务（Speculative Executing Task）

MapReduce 模型给我们带来的重要的利益就是它会自动处理故障情况，即它有非常好的容错性。如果一个工作节点故障后，它会随机的选择另一个在另一个工作节点上重新执行故障节点的任务。同样重要的，当一个工作节点没有瘫痪，但是运行在该节点上的任务的执行速度明显比其它节点慢，这种节点我们称之为掉队者（Straggler）。MapReduce 同样会挑选另外一个工作节点执行掉队者的任务的拷贝，我们称这种机制叫做推测执行[39]（Speculative Executing），这些推测执行的任务我们称之为后备任务。掉队者可能是由各种原因造成的，例如硬件故障或

者配置错误等。Google 指出通过预测执行任务可以在当前系统的基础上提高 44% 的响应时间[12]。

对于推测执行任务对性能上的提升对于机群同构的情况下是成立的，但是在异构的环境中，例如虚拟数据中心 Hadoop 现有的调度程序并不足以胜任，例如 Amazon 的弹性云计算。在虚拟效用计算的环境中，应用运行在虚拟的资源上，而虚拟机的在性能上的表现是不能准确控制的。同样即使是私有的实体数据中心，异构性也是普遍存在的。公司或者企业往往会将新旧硬件设备混合使用，而且虚拟化技术也会给数据中心带来管理方便和整合服务器的好处。根据调查[39]，在异构机群环境中将导致不正确的、过度的推测执行的任务，过度的重复执行掉队者任务经常会比在没有推测执行的情况下性能还差，在某些实验中，大约 80%的任务都被认为是掉队者而从其他的节点重新执行了一次。

在实际应用中，如何确定一个节点或任务是不是掉队者是一件非常困难的问题，主要有以下几个方面的原因[46]：

1. 执行后备任务需要消耗系统资源，他也需要同正在运行中的其他任务竞争系统资源，如带宽等；
2. 如何选择后备任务执行的工作节点和如何选择后备任务是同等重要的，如果将后备任务分配到性能差的节点上执行，不但不会提高响应时间，反而会增加该节点的工作量，浪费系统资源；
3. 在异构环境中确定掉队者节点是非常困难的；
4. 应该及时发现掉队者并执行它的后备任务，从而尽可能的减少系统的响应时间。

4.1.2. Hadoop 中的推测执行

当一个工作节点为空时，他会向主控制节点发送一条消息，在接收到这条消息后，Hadoop 的调度程序会选择一个就绪任务分配给空闲的工作节点去执行。就绪任务按照以下三种情况进行分类。首先，任何失败的任务给予最高的优先权。如果这个任务重复失败多次，Hadoop 将停止对该任务的执行，把这个失败的任务归结为某个程序错误；其次，当并不存在失败的任务时，优先执行未被执行过的任务，对于 Map 阶段，输入数据在本地的将会被优先考虑；最后，Hadoop 会根据它的推测算法推测出一个需要执行的任务。

Hadoop 对于后备任务的选择是根据进度值[28]（Progress Score）确定的，下文

记为 PS。Hadoop 为每一个任务的进度赋予一个 0 到 1 之间的数值来标识其任务的进度。对于 Map 阶段，这个进度值就是 Map 已经处理的数据和输入数据的比值，对于 Reduce 阶段，这个数值是由 3 部分组成的，每部分占总进度值的 1/3。据此，Reduce 被分为下面的三个阶段[28]：

- 复制阶段：此时获取所有 Map 的输出；
- 排序阶段：此时对 Map 的输出按照中间键进行排序；
- Reduce 阶段：对每个具有相同中间键的结果集调用用户自定义的 Reduce 函数。

对于每个阶段其进度值即为该阶段处理数据量和输入数据量的比值。下面举例说明：如果一个任务处理了复制阶段的一半的数据量，则它的进度值为 $PS=1/2*1/3=1/6$；又如一个任务处理了 Reduce 阶段的一半的数据量，则它的进度值为 $PS=1/3+1/3+1/2*1/3=5/6$，以此类推。

Hadoop 为了确定掉队者定义了一个阈值（Threshold）[39]，当一个任务的进度值低于阈值且已经执行了至少一分钟时，那么将把这个任务标记为落后者，Hadoop 将会在另一个节点上执行该任务的后备任务。阈值的设定是根据当前任务（包括 Map 任务和 Reduce 任务）的平均进度决定的，它的值为平均进度值减去 0.2。所有低于阈值的任务都被认为是执行进度慢的任务。Hadoop 的调度程序同时保证同一时刻同一任务的后备程序执行的副本不会超过一个。在同构的环境中，Hadoop 对阈值的认定是合理的，因为任务开始和结束的时间大体上是一致的，而对后备任务的推测也仅在最后一批任务上执行。

最后一点需要说明的是，当 Hadoop 运行批量的任务时，它通常采用先进先出的调度方式（First In First Out），提交的任务按照提交时间被顺序执行。

4.1.3. Hadoop 调度程序中的几点假设

Hadoop 的调度程序设计时隐含着对下面几个方面的假设[28]：

1. 每个工作节点处理任务的能力大体上是一致的；
2. 在整体的时间内任务的执行保持恒定的速率；
3. 将后备任务分发给其它节点执行不需要消耗资源和时间
4. 一个任务的进度值是该任务所处理的数据量和它所需处理数据总量的比值，在 Reduce 阶段，每个子阶段占总进度值的 1/3；
5. 任务是按照一批一批的形式完成的，所以当一个任务的进度值较小时，可

以认定它为掉队者；

6. 同一种类型的任务（Map 任务或 Reduce 任务）在处理的数据量上大致相同。

4.1.4. 异构性使得 Hadoop 中的假设失效

4.1.4.1. 机群的异构性

4.1.3 中提到的假设 1 和假设 2 在异构的环境中显然是不成立的，而且异构环境在现实使用中是非常常见的。在 4.1.1 中我们提到无论是虚拟数据中心还是非虚拟数据中心，机群总是存在着各种各样性能上的差异。这种异构性严重的影响了 Hadoop 任务调度的性能。因为调度程序使用一个固定的阈值来确定某一个任务是否是掉队者，这可能导致在同一时刻运行中的大多数任务都低于这个阈值，而这些任务都需要根据推测被重新执行。执行这些后备任务同样需要和正在执行的正常的任务竞争资源（上一节提到的假设 3 是不成立的），这样会造成不必要的任务的重新执行和资源的过度浪费。

同样，由于Hadoop选择输入数据与计算任务在同一个节点上的任务优先执行，会使调度程序选择了执行不适当的任务。例如，假设当前的平均进度值是 70%，A 任务的进度值是当前平均进度之的一半为 35%，B 任务的进度值为当前平均进度之的 1/10 为 7%，很有可能因为 A 任务所要处理的数据就在它所在的工作节点上从而优先执行 A 任务，而不是执行更为落后的 B 任务。

4.1.4.2. 异构行推翻 Hadoop 的其他假设

4.1.3 中提到的假设 3、4 和 5 不管是在同构还是异构的环境中都是不成立的。对于假设 3 由于各个任务的执行是以资源共享为基础的，所以执行后备任务同样需要竞争资源。例如我们在上一章提到网络带宽已经成为了 MapReduce 海量处理时的一个瓶颈，同样，执行后备任务也会同 I/O 作业竞争磁盘读写。而且在有多个任务并发执行的情况下，执行不必要的后备任务会占用空闲的工作节点，从而使得这些工作节点不能再分配给新的任务。

典型的 MapReduce 的作业中，Reduce 阶段的复制子阶段是运行时间最长的一部分操作，因为他需要等待所有相关的 Map 任务处理的中间结果。从而假设 4 也不成立。通常情况下，一旦 Reduce 任务获得了所有相关的 Map 任务输出的中间结果集，完成复制阶段后另外的排序和 Reduce 阶段也会迅速完成。显然复制阶段占 Reduce 阶段工作量的 1/3 的设定是不合理的。假设一部分 Reduce 任务率先完成了

复制阶段的工作，这些任务的进度值会迅速的从 1/3 增加到 1。这些任务会使得平均进度值突然增加到一个比较大的数值。我们假设 30%的任务完成了复制阶段后，现阶段的平均任务进展度为30%*1+70%*1/3≈53%。此时，大量的未完成复制阶段的 Reduce 任务就会被当成掉队者，他们的后备任务会分配到其他的节点上面执行，而且网络带宽会被复制这些不必要的后备任务所填满，这种推测执行带有一定的随意性，工作节点上的任务队列马上被填满了，而真正的掉队者则没有执行其后备任务的机会。在实际应用中，据统计在一个拥有 900 个工作节点的弹性云计算模型中，80%的 Reduce 任务都被重复执行了后备任务[28]，这显然是不必要的，给系统带了了巨大的资源浪费。

对于假设 5，由于 MapReduce 模型中，每一个工作节点同时执行若干的 Map 任务，这些任务是按照一批一批的顺序进行执行的，在异构的环境下，节点在性能上存在着差异，所以随着时间的推移，不同批次的任务有可能同时执行。在这种情况下，由于较早批次的任务会比新加入的任务具有较高的进度值，所以 Hadoop 调度程序会将一个新的、执行速度快的任务重新执行，而真正的掉队者的后备任务却没有执行。

最后，由于阈值设置为平均进度值减去 0.2，因为任务的平均进度值不会超过 1.0，所以但凡某个任务的进度值超过了 80%，那么这个任务就再也不会被推测为掉队者而执行其后备任务。

综上所述，Hadoop 的推测执行后备任务的策略虽然可以比较高效的适用于同构环境，但在异构环境下，它对于执行条件的一些假设使得 Hadoop 的调度程序常常会选择不适合的任务执行后备任务，而且频繁的推测执行也会导致系统发生任务抖动，使系统的主要工作变成了调度众多的后备任务[39]。在实际应用中，雅虎在某些作业中已经禁止调度程序推测执行后备任务[28]，FaceBook 也禁止在 Reduce 阶段执行后备任务，因为它会影响 MapReduce 的性能。下面的一节将会对现有的 Hadoop 的推测算法进行改进，并设计出一个基于优先权的作业调度算法，并测试这种调度算法比之前的 Hadoop 算法在短作业处理上面具有更好的性能。

4.2. Hadoop 中与任务调度有关的类

Hadoop 实现一个作业调度一般分为下面三个阶段：Job 创建过程、Job 初始化过程、Task 的执行过程，分别研究各阶段相关的类及作用。

4.2.1. Job 创建过程

在 MapReduce 中与 Job 创建过程相关类和方法之间的关系如下：

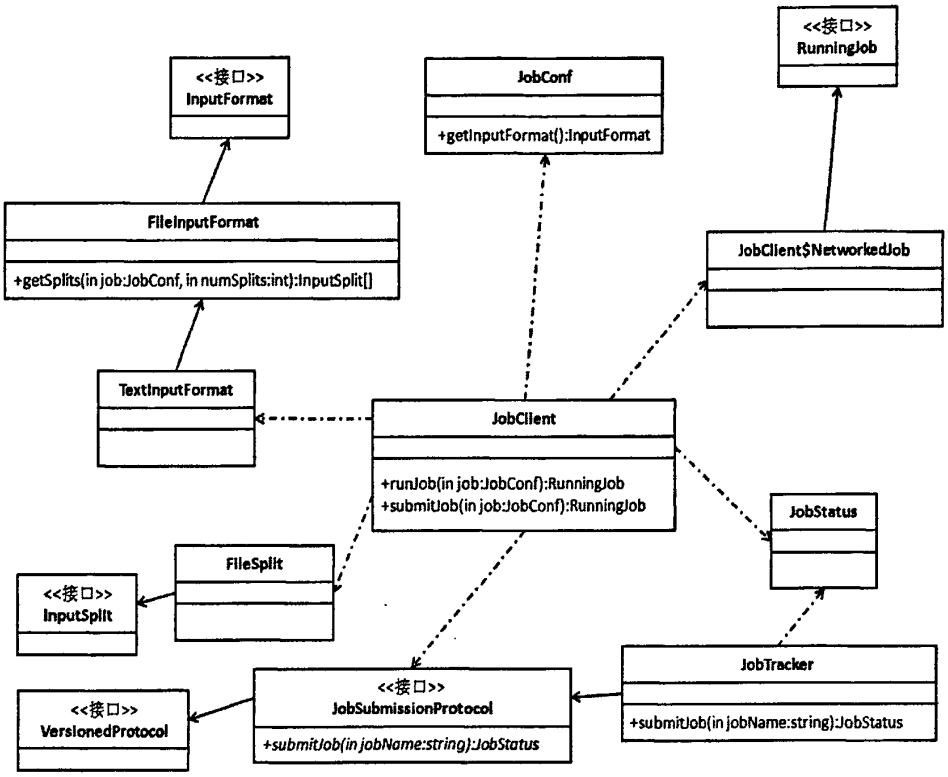


图 4-1 与创建 Job 有关的类和方法

MapReduce 的 Job 会通过 JobClient.runJob(job)开始运行 Job 并将输入数据切割成小块，该方法产生了一个 JobClient 的实例。它根据用户在 JobConfig 类中定义的 InputFormat 类的具体实现将输入数据集分割成一批小的数据集，每一个分割后的数据集对应一个 Map 子任务。若没有用户自定义的 InputFormat 类的实现，则 JobClient 会调用缺省的 FileInputFromat.getSplit()方法将输入文件分割成小的 FileSplit，分割后的输入文件在 HDFS 中的路径、偏移量、Split 块大小等一些信息一同被打包存放在 JobFile.jar 中并存放在 HDFS 中，然后再将 JobFile.jar 的路径提交给 JobTracker 去调度和执行。

通过上面产生的 JobClient 的实例调用方法 JobClient.submitJob()将作业提交给 JobTracker，JobTracker 根据获得的 JobFile.jar 路径创建与 Job 有关的一系列对象（JobInProgress 类和 TaskInProgress 类）用来调度执行 Job。JobTracker 创建 Job 成功后会将一个 JobStatus 对象回传给 JobClient，这个对象记录了该 Job 的状态信息，包括执行时间、Map 任务和 Reduce 任务执行的进度等等。JobClient 根据回传

的 JobStatus 对象创建一个 RunningJob 对象，该对象定时从 JobTracker 获取运行过程中的数据统计来监控并将监控信息打印到用户控制台。

4.2.2. Job 初始化过程

在 MapReduce 中与 Job 初始化过程相关的类和方法之间的关系如下：

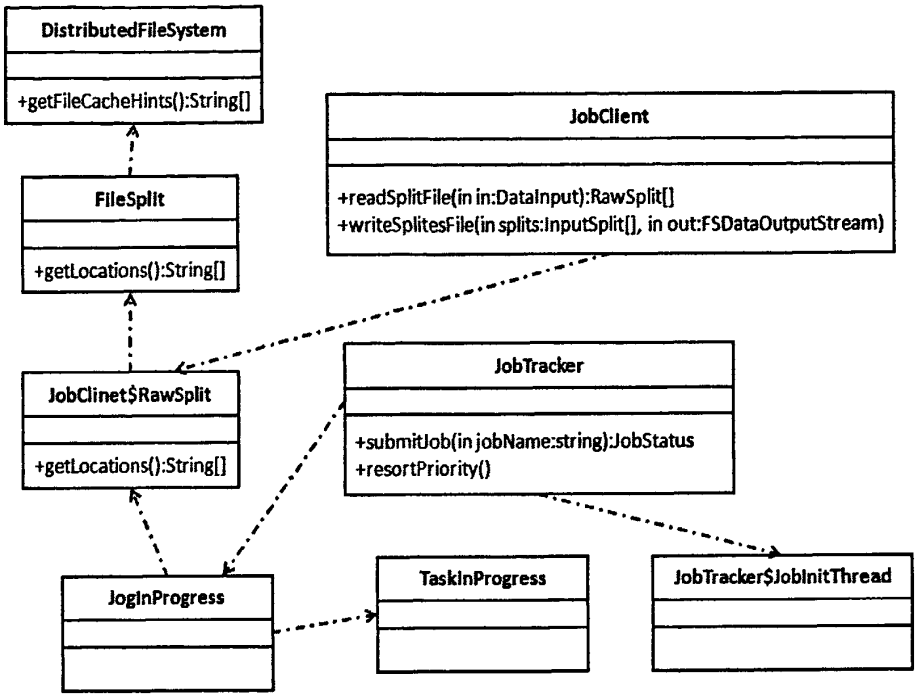


图 4-2 与初始化 Job 有关的类和方法

当 JobTracker 接收到新的 Job 提交的请求后（即 JobClient 调用 submitJob()函数），它会创建一个 JobInProgress 对象用来对任务进行管理和调度。在创建 JobInProgress 时会初始化和该任务相关的若干参数，比如 Job.jar 的位置，Map 任务和 Reduce 任务要处理的数据，该 Job 初始优先级以及做统计信息的对象等等。JobTracker 利用两个变量管理这些被提交的任务，首先用一个 map 的成员变量 jobs 保存所有被提交的 Job 对象，还有一个 list 的成员变量 jobsByPriority 来按照提交任务的优先级来维护提交的 Job。随后会按照 jobs 的优先级别对 list 的成员进行重新排序，通过调用 JobTracker.resortPriority()函数，在此基础上在对 list 的成员按照 job 的提交时间进行排序，以确保先提交的 job 会被优先执行。然后 JobTracker 会调用初始化线程 JobTracker.JobInitThread()对该 Job 进行初始化（JobTracker 内部会有几个线程来控制 jobs 队列），JobInitThread 在接收到信号之后找出优先级最高的 Job，调用 JobInProgress.initTasks()对该 Job 执行真正的初始化。

Task的初始化相对复杂一些。首先JobInProgress创建对Map的监控对象，Map监控对象的多少是根据分割数据块的多少确定的，通过在 JobInProgress.initTask()函数里面调用 JobClient.readSplitFile()来获得输入文件的 RawSplit 列表可以得到分割块的数量。之后，JobInProgress 会创建对 Reduce 的监控对象，Reduce 监控对象的多少是由 JobCofing 类决定的，如果 JobConfig 未指定 Reduce 的个数，则取缺省值为 1。监控 Map 和 Reduce 的对象都是 TaskInProgress，但是这两个不同类型的监控对象的构造方法也不同，他会根据不同的参数创建具体的 Map 任务和 Reduce 任务。

当所有子任务的监控对象创建完成后，JobInProgress 构造 JobStatus 用来记录 Job 的执行状态和执行进度等信息。至此完成了初始化 Job 的全部过程。

4.2.3. Task 执行过程

各种 Task 的实际执行是通过 TaskTracker 发起的。在经过初始化一系列参数和服务之后，TaskTracker 会试图同 JobTracker 连接，连接成功后，它会每隔 10 秒调用 JobTracker.heartBeat()来愈 JobTracker 进行通信，汇报本节点上的任务执行状态并接收 JobTracker 发送来的指令同时执行新任务，如果需要的话。TaskTracker 通过 transmitHeartBeat()获取 JobTracker 发送来的心跳程序的响应结果，然后调用 HeartbeatResponse.getActions()获取有 JobTracker 传送过来的指令数组，包括新任务的分发和终止任务的执行等。在发送心跳程序之前，TaskTracker 会监控该节点当前执行的任务数和本地磁盘的使用情况等信息，如果满足接受新任务的条件，则将心跳程序中的 askForNewTask 设置为 true。

TaskTracker 会根据任务种类的不同分别创建相应的 MapRunner 和 ReduceRunner 对象分别用来启动 Map 子任务和 Reduce 子任务。通过 TaskRunner.start()真正的启动并执行任务。它的主要工作就是初始化 Java 子进程的相关环境变量、装载 job.jar 等。

与 Job 执行的相关类和方法的作用如图 4-3 所示：

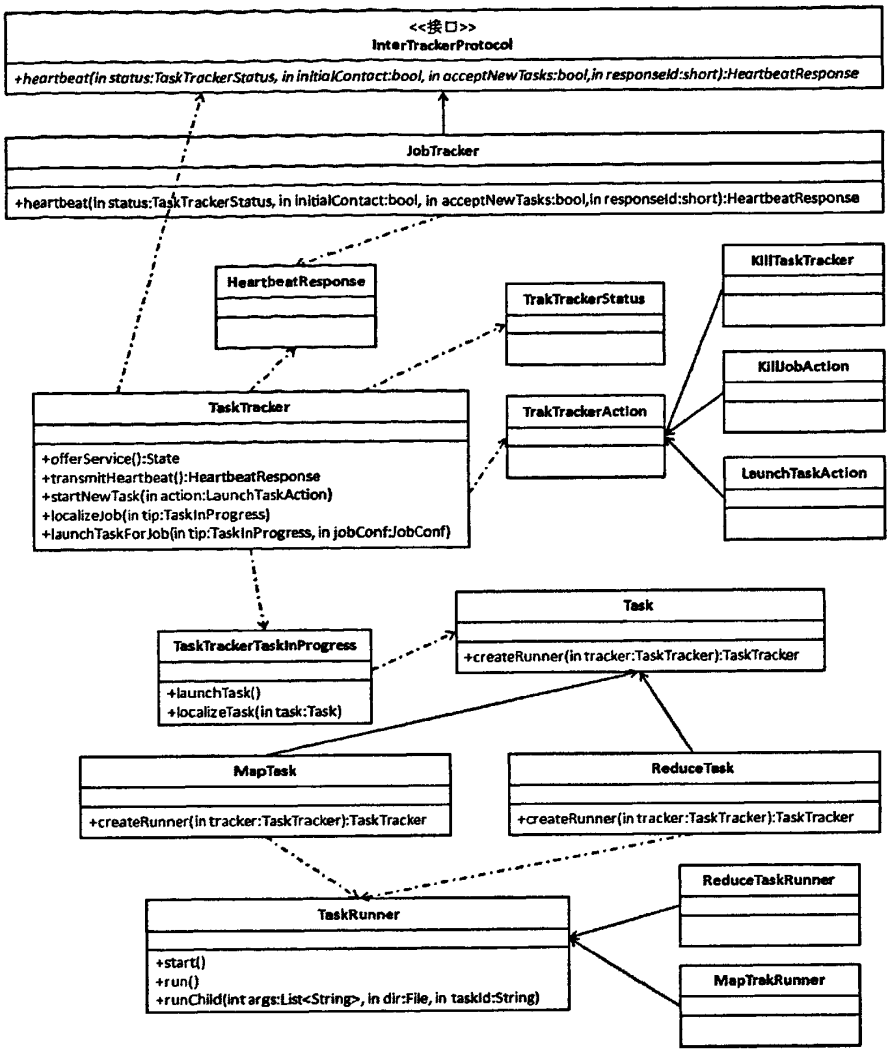


图 4-3 与执行 Job 有关的类和方法

4.3. 基于优先级加权的滑动窗口调度算法

本节将考虑机群的异构性并在 MapReduce 推测执行的方法上进行改进，提出一种全新的作业调度算法，该调度算法能够根据当前系统中的负载量自动调节可执行作业调度窗口的大小动态调节系统的负载均衡，使用更准确的方法推测掉队者任务，并将它分配到高性能的计算节点执行，提高系统响应时间，并且控制后备任务的数量避免引起任务抖动。我们称之为给予优先级加权的滑动窗口算法（PWSW 算法）。

设计该调度算法时主要有以下几个基本原则：

1. 自适应的负载调节。系统根据当前各节点的负载水平动态调成可执行任务的数量；
2. 作业调度是基于优先权的。在系统执行过程中，作业可以根据优先权的高低分配相应或多或少的系统资源和执行时间；
3. 作业调度考虑机器的异构性。该算法会根据系统中每个节点的性能好坏决定分配合理的工作量给不同节点；
4. 真正意义的推测执行。采取改进的方法确定掉队者任务，并将任务分配给性能好的工作节点执行。
5. 控制后备任务数量以防止任务抖动。

PWSW 算法的基本思想如下：所有提交的 Job 首先被放入就绪队列中，根据系统的负载量及 Job 执行情况将 Job 调入滑动窗口或增大滑动窗口添加新的 Job。滑动窗口内的 Job 会轮流获得执行 Task 的机会，每轮执行 Task 的数量由 Job 的权重决定，权重大的可以获得相应多的执行机会。系统定时评估每个运行任务的预计结束时间，并将任务根据预计结束时间排序，选择预计结束时间比较大的几个任务执行其后备任务。

图 4-4 展示了 PWSW 算法的执行过程：

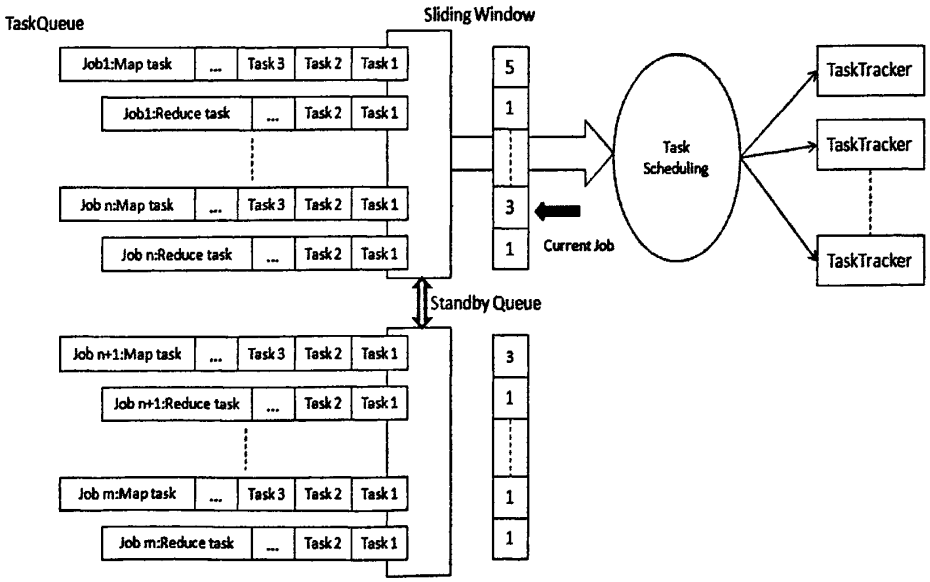


图 4-4 PWSW 算法示意图

算法执行的基本过程如下：

1. TaskTracker 检测当前节点状态，发现存在空闲资源的情况下，向 JobTracker 发送消息，主动请求 JobTracker 分配任务；

2. JobTracker 判断申请任务的阶段是快节点或慢节点。如果是快节点且满足当前后备任务队列非空和后备任务数未超过上限时，将后备任务分配给该节点，否则执行下一步；
3. 当 JobTracker 接收到请求分配任务的消息后，将滑动窗口中当前 Job 的一个 Task 分配给请求任务的 TaskTracker 执行，然后修改 Job 的相关信息，将分配的 Task 从 Job 的 TaskQueue 中移除，将 Job 在本轮中的剩余调度次数减 1，若此后 Job 的剩余调度次数小于 1，则将指针移向滑动窗口的下一个 Job，否则保持当前指针位置，等待分配此 Job 的下一个 Task；
4. 若指针指向 Job 队列末端时，JobTracker 统计各 TaskTracker 的平均工作负载量，如果平均负载量低于某个阈值则增大滑动窗口调度新的作业进滑动窗口，如果高于某个阈值则缩小滑动窗口大小，将某个正在执行的 Job 放到就绪队列，并保存其执行状态；
5. JobTracker 计算所有执行的 Task 预计结束的时间，在不超过后备任务上限的情况下执行掉队者的后备任务；
6. 对当前滑动窗口中 Job 队列，更新其执行信息。如果滑动窗口增加新任务或者移除正在执行的任务，则更新每个 Job 的本轮执行次数，更新后备任务队列的信息；
7. 将滑动窗口指针移向新的队首重新执行调度。

PWSW 调度算法主要涉及到三个队列，slidingWindow[]、standbyQuque[]和 taskQueue[]。其中 slidingWindow[]和 standbyQueue[]中元素的数据结构相同，它们都是用来维护用户提交的作业的，区别在于只有 slidingWindow[]中的 Job 才会被调度程序所调度，这两个队列中的数据元素 jobInfo 的结构如下：

```
Class jobInfo
{
    int jobIdentifier;
    int jobSize;
    int taskNumber;
    int avgTaskSize;
    int priority;
    float weight;
    int currentTaskLeft;
    boolean executedOrNot;
```

```
};
```

当 Job 进入 slidingWindow 中 exectuingOrNot 的值被赋值为 TRUE，standbyQueue 将该值初始化为 FALSE。taskQueue 里面的元素对应某个 Job 的一个 task，可能是 Map Task 也可能是 Reduce Task。在本文所设计的调度算法中，以上三个队列全部使用 List 实现。

4.3.1. 权重的计算方法及任务分配策略

4.3.1.1. 权重的计算方法

PWSW 是一个基于 priority 的算法，每一个任务 priority 的是根据多种因素在运行时决定的。由于云计算为用户提供了丰富多彩的应用服务，而且根据用户的付费情况也会分等级提供不同质量的服务，因此作业 priority 是由多种多样的原因决定的。它如何取值根据用户和服务提供商之间的协议确定的，与本文的研究重点无关，本文的算法只需要按照不同的 priority 实现给作业提供运行时间和资源分配的差别即可，本文只将 priority 简单的取做 1、2、3 等自然数即可。

作业 priority 的差别将体现在作业的 weight 上面，weight 是根据作业 priority 确定的一个实数，他确定了每轮 Job 可以执行 Task 的数量，本文中 currentTaskLeft 取 weight 的下整数，若 Job 对应的 taskQueue 剩余 Task 数量少于 weight 的下整数，则取 TaskQueue 中剩余 Task 的数量作为 currentTaskLeft 的值。

所以更新滑动窗口中 Job 的 weight 和 currentTaskLeft 的方法实现如下：

```
updateSlidingWindow(SlidingWindow queue)
{
    for(each job in queue)
    {
        weight=caclWeight(priority);
        currentTaskLeft=Min(Floor(weight),leftTaskNumOfJob));
    }
};
```

下面推导一下 weight 的计算方法：

上面提到 weight 决定了一轮调度中每一个 Job 可以执行多少个 Task，而每个 Job 的 avgTaskSize=jobSize/taskNumber，所以在一轮中 Job[i]所处理的数据量为 $S[i]=avgTaskSize[i]*weight[i]$，从而在一轮调度中系统处理的总数据量为：

$$S=\sum_{i=1}^{n} avgTaskSize[i]*weight[i] \tag{4.1}$$

Job[i]对应的优先级为 Priority[i]，意味着 Job[i]在一轮调度中所处理数据同系统调度一轮所处理的数据的比值，应该与 Job[i]的优先级的数值占当前滑动窗口中所有 Job 优先级数值的比值相同，用公式表达为：

$$\frac{avgTaskSize[i]*weight[i]}{S}=\frac{priority[i]}{\sum_{i=1}^{n} priority[i]} \tag{4.2}$$

上式中，priority[i]与 avgTaskSize[i]都是已知的，下面我们估算每一个轮转周期系统所处理的数据量 S。一般情况下 Map Task 的数量要比 Reduce Task 的数量要多很多，我们用系统能够同时运行的 Map Task 的能力来表示系统的所能处理 Task 的能力，记为 taskAbility。而且每个 Map Task 所处理的数据量为输入数据的分块，大小即为 HDFS 中数据块的大小记为 blockSize，所以每一个轮转周期系统所处理的数据量可以表示为：

$$S=Max(num*blockSize*jobNum,TaskAbility*blockSize) \tag{4.3}$$

其中 num 为一个轮转周期滑动窗口内 Job 平均执行的 Task 数量，为了算法的简单，我们在这里 num 值取 2，jobNum 为窗口中 Job 数量。因此，根据公式 4.2 和公式 4.3 可以将权重表示为：

$$weight[i]=\frac{S*priority[i]}{avgTaskSize[i]*\sum_{j=1}^{n} priority[j]} \tag{4.4}$$

4.3.1.2. 一个轮转周期内的任务分配策略

滑动窗口内的所有 Job 按照优先级和提交时间排列，JobTracker 按照一定的规则将其中一个任务分配给请求 Task 的 TaskTracker。具体步骤如下：

1. 查看 sidingWindow 中指针指向的当前 Job，取出其 jobInfo 信息，由 jobIdentifier 来确定该 Job 对应的 taskQueue[i]；
2. 查看当前的 taskQueue 是的类型，若当前 taskQueue 是 Reduce Task Queue，则要考虑队列中当前的 Reduce task 所对应的 Map Task 是否执行完毕，假如没有执行完成，则将 slidingWindow 的指针向后移动一位，继续执行步骤 1。若当前 taskQueue 是 Map Task Queue 或者 Reduce Task Queue 的当前 Task 所对应的 Map Task 已经执行完毕，则将 taskQueue 中当前 task 分配给发出请求消息的 TaskTracker；

3. 将步骤 2 中分配的 task 从 taskQueue 中移除，同时修改 taskQueue 的相关参数，修改当前的 jobInfo 相关参数（将 currentTaskLeft 减 1、已完成任务数等）。若修改后的 currentTaskLeft 小于 1，则需要把 SlidingWindow 队列的指针向后移动一位，否则指针位置不变。

任务分配流程示意如图 4-5 所示：

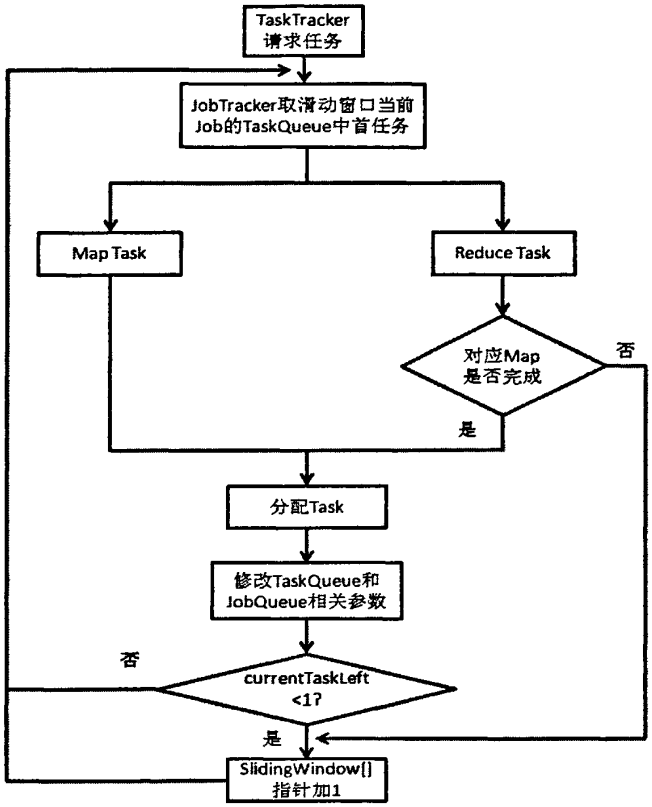


图 4-5 一个轮转周期内的任务分配示意图

4.3.2. 自适应调整滑动窗口的大小

Hadoop 自身所带的调度算法并没有针对系统负载量对 Job 调度队列中的 Job 数量进行动态调整。在系统的负载已经过量的情况下，如果用户继续通过 JobClient 提交 Job 的话，所提交的 Job 仍然会被增加到 Job 调度队列中。超过系统承受量的 Job 队列长度并不会提高系统的总吞吐量，反而会使系统中的工作节点都处在一个超负荷的状态，降低了作业的响应时间和系统的处理能力。再加上 Hadoop 的推测执行策略，每个节点上的 Task 因为相互竞争资源，最终 Task 获得的系统资源将不足以快速的完成任务，而这些进度慢的 Task 会被 Hadoop 视为掉队者，进而执行

其后备任务，这种策略在负载过重的情况下进步一加剧了系统的负载量，系统花费大量时间和资源区执行太多不必要的后备任务。

4.3.2.1. 调整滑动窗口大小的基本思想和流程

在 PWSW 算法中，当一个调度周期结束 SlidingWindow 的指针指向队尾时，系统会根据当前负载量的水平，动态的调整 SlidingWindow 的大小，从而使得系统负载量维持在一个合理的水平。

为了实现上述目标，我们将用户提交的作业分成两个队列存放，slidingWindow 和 standbyQueue。当有作业被用户提交后，经过预处理（将输入文件分块，读取 job.jar 等配置信息）后放入 standbyQueue 等待调度，standbyQueue 中的 Job 按照优先级和提交时间排序。当一个轮转周期结束后，JobTracker 统计所有节点的负载水平并计算出系统的平均负载水平（avgLoadRate），同时对系统的负载水平定义两个阈值，高负载阈值（HLoadRate）和低负载阈值（LLoadRate）。若平均负载水平超过高负载阈值则减小滑动窗口大小，根据算法将正在执行的作业放入就绪队列并保存该 Job 的执行环境；反之如果平均负载水平低于低负载阈值则增大滑动窗口大小，根据算法将就绪队列中的某些作业放入调度窗口中执行。具体流程如下：

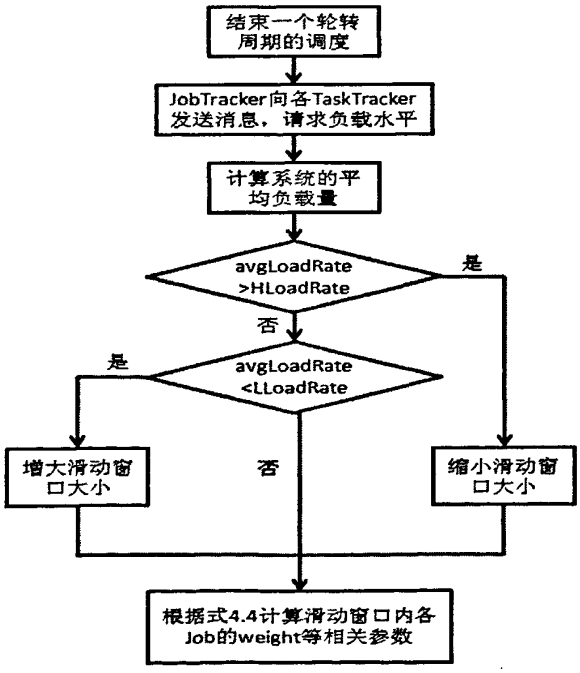


图 4-6 调整滑动窗口流程示意图

4.3.2.2. 滑动窗口调整算法

当 JobTracker 发现当前系统的负载过高或过低时，它将采取一定策略减小或者增大滑动窗口的大小，将 slidingWindow 和 standbyQueue 中的作业做相应的调整。调整滑动窗口可以是整个运行过程中系统负载基本保持平衡的状态，但是调整幅度的的大小应该科学选择。例如系统检测到负载水平低于负载清闲阈值时调度大量的 Job 进入滑动窗口，使得下一个轮转周期内系统超负荷运行，继而刚刚调入的 Job 又会被移入就绪队列。

所以调整滑动窗口的原则即增加和减少一定数量的 Job 使得改变后滑动窗口内的 Job 数量是系统的负载维持在一个合理的水平。

当前系统的平均负载水平可以表示为：

$$avgLoadRate = \frac{num * jobNum}{taskAbility} \tag{4.5}$$

num 和 jobNum 和 5.3.1.1 节中表意一样。下面以扩大滑动窗口为例推导拟增加 Job 数量。假设当前系统的平均负载水平较低，即 $avgLoadRate < LLoadRate$，则在添加 Job 后系统的平均负载水平不应该超过 HLoadRate，可用公式表达为：

$$\frac{num(jobNum + increment)}{taskAbility} \leq HLoadRate \tag{4.6}$$

根据公式（4.5）和（4.6）可以计算得到拟增加的 Job 数为：

$$increment = \frac{taskAbility(HLoadRate - avgLoadRate)}{num} \tag{4.7}$$

调度 Job 进入 slidingWindow 的规则有以下三个方面：

1. 优先调度执行过且被换出的 Job（通过 executedOrNot 标识）；
2. 调度优先级高的 Job；
3. Job 按照提交顺序调度。

同理我们可以得到减小滑动窗口时拟减少 Job 数为：

$$decrement = \frac{taskAbility(avgLoadRate - LLoadRate)}{num} \tag{4.8}$$

在将 Job 从 slidingWindow 移动到 standbyQueue 时，系统保存该 Job 的执行环境，包括已完成 Task，Map Task 处理的中介结果位置等，以使再次调度时可以而重复执行。

根据多次试验，我们得出将系统的 HLoadRate 和 LLoadRate 分别设置在 85% 和 65%时，系统的平均负载均衡，且稳定性也比较高。

4.3.3. 更效率的推测执行

Hadoop 提出推测执行的思想，目的就是为了让执行速度慢的任务可以获得多一次的执行机会，从而使得掉队者的后备任务可以比原始任务更快的执行结束，以缩短任务响应时间。

考虑 Hadoop 现有的推测执行的策略，它考量任务的进度值本身就不能准确的反映系统中任务的进展度，使得系统会执行不必要的后备任务。

即使 Hadoop 推测的任务确实为掉队者任务，它简单的将其分配给任一个请求任务的工作节点，而不考虑该工作节点在异构性上的差异。那么对于这种情况，假设一个任务 T，进度值为 40%且被推测为掉队者，T 在原始的节点上预计 50 秒可完成，此时机群中一个性能差的节点 A 在资源空闲的情况下请求任务，而 Hadoop 的调度程序将任务 T 分配给节点 A，而 A 节点处理能力差，重新运行任务 T 可能需要超过 50 秒的时间，比方说 70 秒。那么此时对后备任务的执行将是毫无意义的，它不仅没有提高系统的响应时间，反而增加了节点 A 的处理负担。

可以通过执行多个掉队者任务的副本来使得后备任务有较高的概率替换原始任务的执行，但是这样对系统的资源需求更大。针对 Hadoop 推测执行策略的不足，本文将会做以下两个方面的改进。

4.3.3.1. 掉队者判定策略

为了使推测执行更有意义，本算法中提出了一种新的评估标准——近似结束时间（Approximate End Time，下文简称 AET）。推测执行基于下面的一条原则：执行在未来预计执行最久的 Task 的后备任务。因为推测执行这种 Task 使得其后备任务最后可能覆盖原始任务，即其后备任务最有可能比原始任务更快的执行完成，从而减少了任务的响应时间。下面我们提出 AET 的计算公式。Hadoop 使用 ProgressScore 来判定 Task 之间的快慢程度有一定的盲目性。首先本文使用 ProgressRate 来表示任务执行的平均速度，其值为 ProgressScore 与该 Task 执行的时间 T 的比值，即 $ProgressRate = ProgressScore/T$。我们假设 Task 大致按照相同的速度执行，则其近似结束时间可表示为：

$$AET = \frac{1 - ProgressSocre}{ProgressRate} \tag{4.9}$$

AET 以当前 Task 执行速度为基准估算出 Task 的预计会在多久执行完成。使得即使当前系统存在着不同批次的 Task，因为新调度的 Task 执行时间 T 比较小，

所以其 AET 也不至于过大而去执行一个新调度 Task 的后备任务。

同时为了防止后备任务的执行给系统带来任务抖动，PWSW 算法仅对在系统中运行 1 分钟以上的任务计算其推测结束时间，且算法对系统中同时执行的后备任务设置一个上限，上限值记为 BackupTaskCap。此上限值不宜过大以至于影响系统对作业的正常调度，根据实验测试，当 $BackupTaskCap = 10\% * taskNum$ 时，系统推测执行会得到比较好的效果，其中 taskNum 为系统正在执行的任务数。

4.3.3.2. 慢节点判定策略

前面已经介绍了 Hadoop 对后备任务的分配具有一定的盲目性，使得推测执行并不能按照其设计思想优化系统处理能力。本文中 PWSW 算法采取一个简单的优化策略，在实验中证明这个简单的策略具有良好的改善效用。即我们只将后备任务分配给性能好计算速度快的节点执行。

为了实现这个规则，PWSW 规定了一个判节点是快节点还是慢节点的阈值（SlowNodeThreshold），低于此阈值的节点被认为是慢节点（SlowNode），否则认为是快节点（QuickNode）。当快节点请求任务且系统中有需要执行的后备任务时，优先将后备任务分配给快节点执行，慢节点请求任务时则不予分配后备任务。在本算法中 SlowNodeThreshold 约定为节点平均进度的 25%。

节点的进度水平不仅仅是根据当前节点上运行的任务的进度值确定的，它是由节点从系统初始化开始执行成功的任务和当前正在执行的任务进度值的总和。执行成功的任务进度值记为 1，否则记为其进度值。因为这样才能更好的确定节点的处理能力和计算速度，更准确的判定节点的快慢。SlowNodeThreshold 可以用公式表示为：

$$SlowNodeThreshold = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} ProgressScore[i](Task[j])}{n} * 25\% \qquad (4.10)$$

其中 n 表示系统中结点个数，m 表示每个节点执行的任务数量（完成的和正在执行的总数）。

4.3.3.3. 推测执行的实现

为了实现 PWSW 中对后备任务的调度，我们在 slidingWindow 队列中维护一个 BackupTask 的队列，记为 BackupTaskQueue，该队列中 Task 的数量每个轮转周期根据 BackupTaskCap 和系统中正在执行的 BackupTaskNum 所决定，维持队列长度和正在执行 Task 任务数等于后备任务上限。

PWSW 的后备任务分配策略如下：

1. TaskTracker 检查到当前系统空闲，向 JobTracker 发出请求任务的消息；
2. JobTracker 接收到任务后，判断该节点的执行状况，若节点是快节点，BackupTaskQueue 非空且 $BaskupTaskNum < BackupTaskCap$，将当前 BackupTask 分配给该节点，修改 BackupTaskQueue 相关信息；
3. 否则按照 4.3.1.2 节中策略分配任务。

因此 PWSW 算法的整体流程图可以表示为：

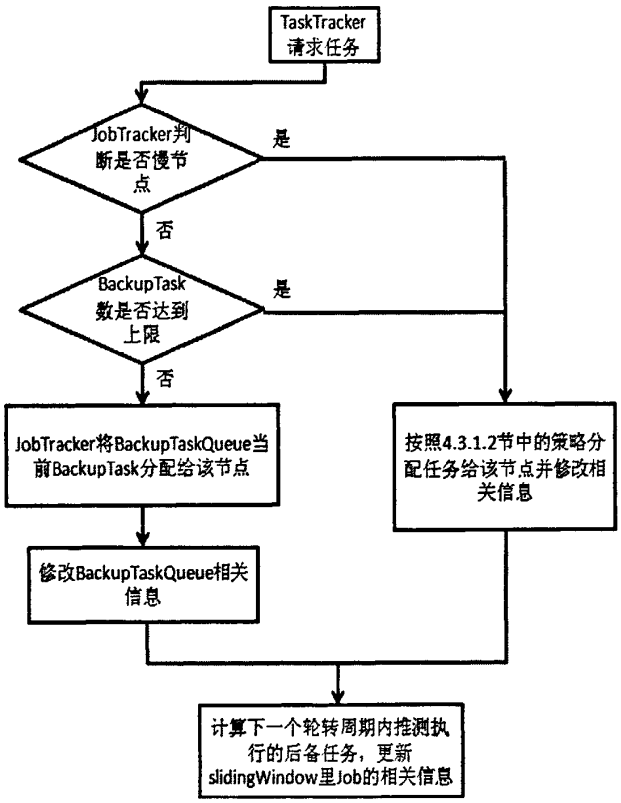


图 4-7 PWSW 任务调度总体流程图

4.3.4. PWSW 算法较 Hadoop 调度算法的优势

基于以上几点的改动，PWSW 算法比原始 Hadoop 调度算法有以下几个方面的优势：

1. 自适应的负载平衡，PWSW 会根据当前系统中各节点的负载水平动态的调节滑动窗口的大小以平均整个执行过程中的负载均衡；
2. 对掉队者判定更准确，预计结束时间比 Task 进度值更能反映运行任务之间的快慢程度，BackupTaskCap 的设定不至于使后备任务数量过多，影响系

统的正常处理能力。而 Hadoop 的规定一个固定的进度值阈值，这种判定落后者方法会使调度程序执行大量不必要的后备任务；

3. PWSW 考虑到系统的异构性，将后备任务分配给性能好的节点执行。而 Hadoop 简单的将任何请求任务的节点认为是快速节点，使得后备任务往往不能比原始任务执行的更迅速；
4. PWSW 考量任务的预计结束时间，而不是进度值或进度速度，这种方式使得 PWSW 的推测执行通常都可以减少系统的响应时间。例如以下两个任务：任务 A 执行速度是平均任务执行速度的 1/5，而进度是 80%，任务 B 执行速度是平均执行速度的 1/2，而执行进度是 10%。根据 Hadoop 调度策略，由于 A 的执行速度比 B 的执行速度慢所以要执行任务 A 的后备任务。然而，实际中 B 需要花费更久的执行时间，因此执行 B 的后备任务往往会更显著的提高系统的响应时间。

4.3.5. 本章小结

本章是本文的重中之重，在第三章实验分析的基础上进而对 Hadoop 调度算法进行研究分析，并对其不合理的地方做出了改进。

首先调研了 Hadoop 平台先进先出的调度方式，发现其在负载均衡上的不足，然后提出了 Hadoop 任务调度和推测执行后备任务对于异构平台的不适用性。然后研究 Hadoop 在调度过程中有关的类、方法及其作用和相互关系。最后本文提出了一个自适应负载均衡的适用于异构环境的调度算法。

第五章 实验及结果分析

5.1. 实验平台选择及配置

为了检测PWSW在异构集群上的优越性，我们使用3.5.2.2中提到的异构平台，即并行计算系统中有1个Master节点和8个计算节点，每4个计算节点通过交换机相连接组成一个机柜，每个机柜上有3台IBM ThinkCenter M4000t和一台配置较低端的机器组成，各工作节点配置参数与第三章相同。

同时为了使实验简单快速，我们通过对典型应用字数统计来测试PWSW算法比Hadoop原始的先进先出调度算法在处理相同的数据量是的优越性，以及PWSW算法是否达到了实验设计的要求。

为了用本文编写的PWSW调度算法替换Hadoop原始的先进先出调度算法，我们还需要做一下几方面的调整。首先，将编写打包好的PWSW程序包PWSWScheduler.jar复制到HADOOP_HOME/LIB目录下，并修改相应的hadoop-site.conf。替换掉原始先进先出导读算法。修改后的内容如下：

```
<property>
    <name>mapred.jobtracker.taskScheduler</name>
    <value>org.apache.hadoop.mapred.PWSWScheduler</value>
</property>
```

5.2. 实验结果及分析

(一)实验一，相同优先级和任务粒度下测验PWSW算法的公平性、响应能力和对掉队者任务的推测执行能力。

在这个实验中，本文一共执行了4个字数统计的Job，每个Job的规模如下：

1. Job1和Job3处理数据量为521M，HDFS每个物理块的大小为64M，这两个任务被分割成8个Map子任务；
2. Job2和Job4处理数据量为1GB，同理每个任务共有16个Map子任务。

所有Job的优先级在PWSW调度过程中取值为1，Job按照1、2、3、4的顺序依次提交，大致同时提交，Job提交的时间差不会超过一秒。表5-1列出了每个

Job 的大小、任务书、优先级等信息：

表 5-1 PWSW 实验一中 Job 的执行信息

	Job1	Job2	Job3	Job4
JobSize	512M	1GB	512M	1GB
Blocks	8	16	8	16
Map Tasks	8	16	8	16
Reduce Tasks	1	1	1	1
Priority(PWSW)	1	1	1	1
weight(PWSW)	4	4	4	4
currentTaskLeft（PWSW）	4	4	4	4

表 5-1 中，JobSize、Blocks、Map Tasks 和 Reduce Tasks 在 FIFO 和 PWSW 调度中取值一样，后面的几个执行参数仅存在于 PWSW 调度中。根据公式（4.4），因为各 Job 的 Priority 和 avgTaskSize 都相同，所以计算所得 weight 和 currentTaskLeft 也都相同。值得注意的是，在每一个轮转周期内，若有新 Job 假如或者有 Job 完成的情况下，各 Job 的 weight 和 currentTaskLeft 都需要重新计算一次。图 5-1 显示了 PWSW 和 FIFO 调度算法在处理上面相同规模的数据时的处理结果：

图 5-1 PWSW 实验一数据

由上图计算得到作业的平均执行时间分别是 avgJobTimeCost_FIFO=695(s)，avgJobTimeCost_PWSW=620.5(s)。可见 PWSW 调度算法提高了系统的处理能力，

而且，每个 Job 的执行时间基本和作业大小成正比，具有了更好的公平性。例如对于 Job1 和 Job3，FIFO 分别花费 203 秒和 653 秒，Job3 多花费了进 2 倍的时间，这显然是不公平的，而PWSW算法则更好的处理了这方面的问题。这是因为PWSW更好的处理了 Map Task 的并发性，Map 任务更早的完成会使得 Reduce 得以更早的处理数据，从而提高了系统的性能。

下面观察以下在真个系统的执行过程中，各 Job 的每个 Task 所消耗的时间，如图 5-2 所示：

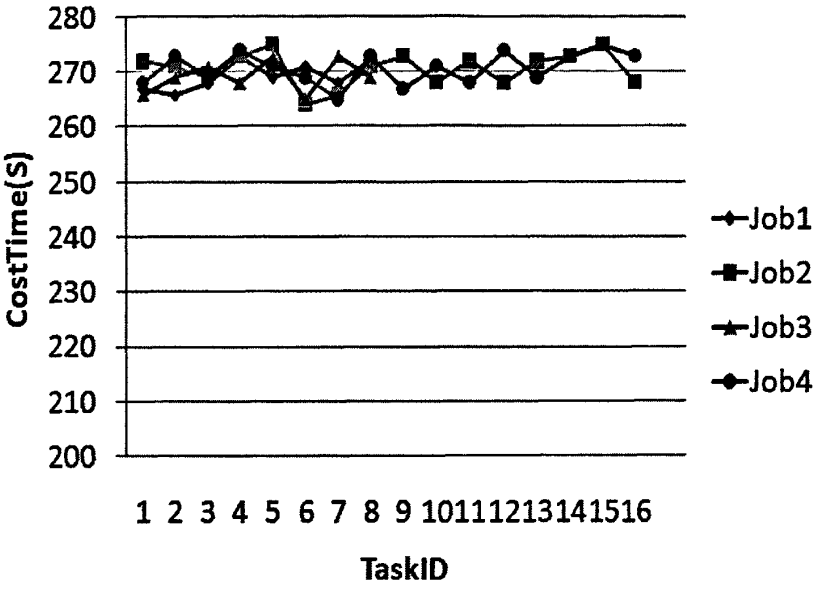


图 5-2 每个 Job 各 Task 消耗时间图

通过上图可以看出，在系统总的执行过程中，每个 Job 各 Task 基本维持在一个相对平均的水平，对于掉队者任务，系统能够及时发现并执行器后备任务，从而提高了系统的响应时间，提高了系统和用户之间的交互能力。因此，PWSW 调度算法更能够适应机群异构的环境。

(二)优先级不同的 Job 队列的调度性能

为了考察 PWSW 对优先级不同的 Job 队列的调度性能，本实验设计了下面 4 个 Job：

1. Job1 和 Job3 处理的数据量大小均为 512M，priority 分别赋值为 1 和 2；
2. Job2 和 Job4 处理的数据量均为 1GB，priority 也分别赋值为 1 和 2。

Job 的提交顺序依然是按照序列号提交，提交时间间隔误差也在一秒之内。表 5-2 展示了各 Job 执行时的一些信息：

表 5-2 PWSW 实验二中 Job 的执行信息

	Job1	Job2	Job3	Job4
JobSize	512M	1GB	512M	1GB
Blocks	8	16	8	16
Map Tasks	8	8	8	16
Reduce Tasks	1	1	1	1
priority(PWSW)	1	1	2	2
Weight(PWSW)	2.67	2.67	5.33	5.33
currentTaskLeft (PWSW)	2	2	5	5

每个 Job 在 FIFO 和 PWSW 调度算法中的响应时间分别为：

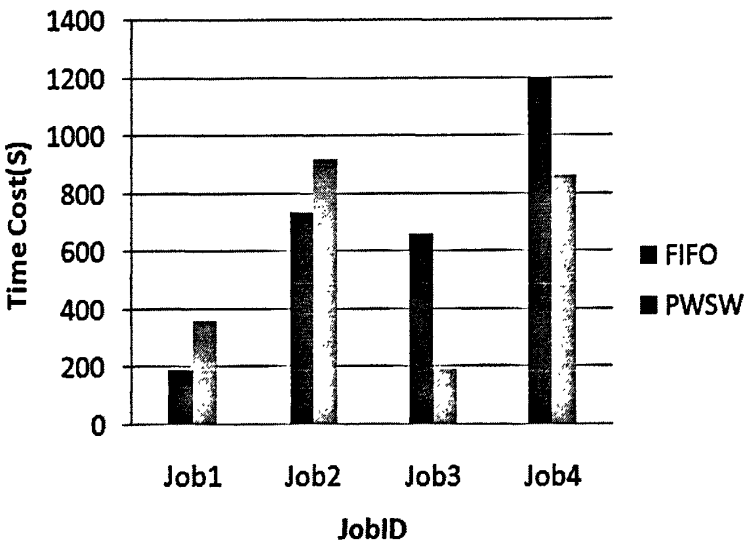


图 5-3 PWSW 实验二数据

由上图可以看出，PWSW 很好的执行了按照优先级对 Job 进行调度的原则。在 PWSW 中 Job3 的响应时间大致比 Job1 缩短了一半的时间，而不像在 FIFO 中 Job3 的响应时间超过 Job1 响应时间的 3 倍多。Job4 的响应时间也比 Job2 的响应时间要少，但是不是太明显，这主要是因为在早期各工作节点被短任务（Job1 和 Job3 的）所占据，而没有及时的调度 Job4 的任务，如果系统执行一段时间处于稳定阶段后，则 PWSW 对优先级的响应会更好。

5.3. 实验结果分析

通过我们对 Hadoop 原始的 FIFO 调度算法的研究和改进，提出新的 PWSW 调度算法更适用于云计算的运行模式。首先云计算所执行的环境通常以异构集群为主，而且云计算提供的服务方式也是根据用户付出不同的费用来提供不同等级的应用服务。而 PWSW 调度算法基于优先级，可以据此为用户提供各种等级的服务，且其针对异构环境而对推测执行的改进，使得系统在异构环境中的处理能力得到了很大的改善。并且可以通过控制滑动窗口的大小动态的动态的对负载平衡进行监控。它继承了 Hadoop 原始调度算法的透明性，是对它的一个比较好的改善方案。当然，PWSW 算法中仍有许多不足，需要在以后的工作中慢慢改善。

5.4. 本章小结

本章是本篇论文工作得到验收和证明的一章，介绍了 PWSW 算法的运行平台，并通过 MapReduce 的一个典型应用检测了 PWSW 算法对 FIFO 算法在响应时间、公平性、推测执行的方面的优越性，最后分析总结了实验结果。

第六章　总结和展望

MapReduce 并行计算模型已经在众多应用领域表现出了非常好的性能，如何在这种编程模式上进一步的提高系统的处理能力已成为当前业界非常关注的问题。本文通过实验验证了 MapReduce 在 Hadoop 实现中对于典型应用的性能评估，验证了其公平性、加速比等重要性能指标。并在第三章试验的基础上分析得出当前 Hadoop 调度算法存在的问题，并发现它在设计上并没有针对异构集群中会出现的问题提出相应的处理措施。

本文重点对其 FIFO 调度算法和推测执行的模式进行了研究，并分析了和 Hadoop 作业调度有关的类和方法。在此基础上设计了一个具有更好的系统处理能力、公平性且更适合异构环境的调度算法 PWSW。使得该进的 Hadoop 调度算法在对掉队者任务的判定上更准确，并且实现了根据用户提供不同等级的服务的功能。最后，验证了 PWSW 算法达到了我们设计预定的需求。

Hadoop 的调度算法的研究已经成为企业和社区中关注量最大的话题之一，各企业和开源社区也展开了对 Hadoop 调度算法的设计。本文只是针对目前其调度算法上的不足，提出了一些简单的解决方案。希望可以为云计算和 Hadoop 的发展带来一些帮助。而且 Hadoop 这种主从式的设计模式，使得一旦 JobTracker 宕机系统就无法执行，这种模式也成为了业内讨论的重点。因此，在未来 Hadoop 的调度算法应该是会由多个 JobTracker 系统工作，共同完成对系统作业的调度，这将成为 Hadoop 调度策略和资源管理的一个可行的发展方向。

总之，云计算正处在飞速发展的阶段，在这个过程中定会出现大量的问题等待我们去研究，我们需要运用自己所学的知识，创造性的解决这些问题，努力为云计算的发展做出自己的一份贡献。

致 谢

追求卓越，拒绝平庸，在硕士就读的三年中，这句话无时无刻不再影响着我。在这短短的三年中，我得到了身边老师、同学、朋友无私的帮助和热忱的关怀，此刻，我要郑重的向你们表示感谢。

首先我要感谢我的导师刘玓副教授，刘老师是 IBM 技术中心的常务主任，老师知识渊博，其严谨的治学态度、勤奋的工作风格、坚韧的人格魅力和平易近人的态度在这三年中深深地影响着我。在就读期间，刘老师无论是在生活上、学习上或是工作上都给了我无私的帮助和关爱，在刘老师身上学到的点点滴滴都将使我受益终生。在这里，我由衷的感谢刘老师，谢谢您的教导。

感谢三年中所有给我传输过知识的老师，他们无私的传授让我学到非常多的知识。IBM 技术中心的文军老师、许毅老师、郝晓清老师以及我的同门师兄师姐们。他们在我的课题研究过程中给了我许多关键的意见，并且提供了我论文研究所需要的实验环境，感谢你们。

感谢杨锴、李劲秋、李桂花和李琴同学，和你们相处的三年是愉快的。怀念和你们一起读书和共事的时间。感谢我的舍友赵见和孙建华同学，你们在生活上都给我了我无私的帮助，并在我实习期间帮助我办理了许多校内的手续。

感谢 IBM BI Cognos 一起工作的同事们，在你们身上我学到了许多为人处世的方式以及善始善终的工作态度。能够与你们共事是我的荣幸。

感谢百忙之中抽出时间评阅本论文的专家组学者和专家，感谢你们对本论文提出的宝贵的意见。

最后，我要感谢我的学校电子科技大学，是它给了我自由的发展空间，其雄厚的文化底蕴，刻苦勤奋的学风以及各种学术交流使得我在就读期间获得了很大的进步。

参考文献

[1]. CNNIC.第 25 次互联网络发展状况统计报告.
http://www.cnnic.cn/html/Dir/2010/01/15/5767.htm

[2]. Wikipedia.Colud Computing. http://en.wikipedia.org/wiki/Coluld_computing/

[3]. 埃森哲咨询公司云计算定义. http://www.accenture.com/Countries/China/

[4]. IDC：公共云计算服务潜力无限.http://www.chinaidaa.org/index.asp

[5]. Amazon Elastic Compute Cloud.http://aws.amazon.con/ec2/

[6]. Google App Engine.http://appengine.google.com/

[7]. Sun 中国技术区.http://developer.sun.com.cn/

[8]. Microsoft Azure.http://www.microsoft.com/azure/windowsazure.mspx

[9]. 瑞星云安全联盟. http://union.rising.com.cn/index/index.aspx

[10]. Ananth Grama,Anshul Gupta,et al.Introduction to Parallel Coputing,Second Edition.2003

[11]. R.Lämmel.Google's MapReduce Programming Model-Revisited .Draft Online since 2 January,2006.22-24

[12]. Jeffery Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM,vol.51 no 1(2008).107-113.

[13]. 王峰.Hadoop 集群作业的调度算法.程序员.2009.1-3

[14]. Fair Scheduler Guide.
http://hadoop.apache.org/common/docs/current/fair_scheduler.html

[15]. Capacity Scheduler Guide.
Guide.http://hadoop.apache.org/common/docs/current/capacity_scheduler.html

[16]. Apache Hadoop.http://hadoop.apache.org/

[17]. The Hadoop Distributed File System.Architecture and Design.
http://hadoop.apache.org/core/docs/current/hdfs_deisgn.html

[18]. Ghemawat, S., Gobioff, H., Leung, S.-T. The Google file system. In Proc. of the 19th ACM SOSP (Dec. 2003), 29-43

[19]. 王鹏.云计算的关键技术和应用实例.北京.人民邮政出版社.2010：91-96

[20]. Remote Procedure Call. http://en.wikipedia.org/wiki/Remote_procedure_call

[21]. Lizhe Wang, Jie Tao, Marcel Knuze.Scientific Cloud Computing:Early Definition and

Experience. The 10th IEEE International Conference on High Performance Computing and Communications.

[22].C. Ranger ,R. Raghuraman and A. Penmetsa. Evaluation MapReduce for multi-core and muti-processor system. In HPCA '07:Proceedings of the 13th International Symposium on High-Performance Computer Architecture.2007.

[23].Jeffrey Dean .Experiences with MapReduce ,an abstraction for large-scale computation . Proc.15th International Conference on Parallel Architectures and Compilation Techniques.2006.

[24].陈全,邓倩妮.云计算及其关键技术.高性能发展与应用.2009,1(26).2-6

[25].孙光中,肖锋,熊曦.MapReduce 模型的调度算法机容错机制研究.微电子学与计算机.2007.178-180.

[26].冯大辉.云计算中的存储.2008 年第 11 期.62-64.

[27].周峰,李旭伟.一种改进的 MapReduce 并行编程模型.计算机技术与信息发展.2009.

[28].Matei Zaharia Andy Konwinski Anthony D. Joseph Randy Katz Ion Stoica. Improving MapReduce Performance in Heterogeneous Environments, proceedings of OSDI, 2008.

[29].王鹏,吕爽,聂治.并行计算应用及实战.机械工业出版社.2009.12-20

[30].Brian Hayes. Cloud Computing. Communication of the ACM. Vol.51 No.7.2009.9-11

[31].Daryl C. Plummer. David W. Cearley, David Mitchell Smith. Cloud Computing Confusion Leads to Opportunity. Gartner Research. 2008

[32].Apache Hadoop Core.http://hadoop.apache.org/core/

[33].Hadoop. The hadoop project. http://lucene.apache.org/hadoop/, 2006

[34].M. A. Vouk. Cloud Computing – Issues ,Research and Implementations. Proceedings of the ITI 2008 30th International Conference on Information Technology Interfaces.

[35].Miclal Miller. Cloud Computing: Web-based Applications That Change the Way You Work and Collaborate Online. Pearson Education. 2009.

[36].Rob Pike, Sean Dorward, Robert Griesemer. Interpreting the data: Parallel analysis with sawzall. Scientific Programming 13, 4 (2005), 277–298.

[37].Michael Isard, Mihai Budiu, Yuan Yu et al. Dryad. Distributed data-parallel programs from sequential building blocks. proceedings of the ACM SIGOPS, 2007

[38].Chris Olston, Benjamin Reed, Utkarsh Srivastava, et al. Pig Latin: A Not-So-Foreign Language for Data Processing. ACM SIGMOD. 2008

[39].Edmund B. Nightingale, Peter.M Chen, Jason Flinn. Speculative execution in a distributed

file system. ACM Trans. Compute System 24(4). 2006. 361-392

[40]. J. Bernardin, P. Lee, J. Lewis. DataSynapse, Inc. Using Execution statistics to select tasks for redundant assignment in a distributed computing platform. Patent number 7093004. 2006-8-15.

[41]. Mor Harchol-Balter, Task Assignment with Unknow Duration. Journal of the ACM, 49(2). 2002.260-288

[42]. M. Croella, M. Harchol-Balter, C. D. Murta. Task assignment in a distributed system: Improving performance by unbalancing load. In Measurement and Modeling of Computer Systems. 1998. 268-270

[43]. B. Ucar, C. Aykanat, K. Kaya. Task assignment in heterogeneous computing systems. Journal of Parallel and Distributed Computing 66(1). 2006 32-46.

[44]. S. Manoharan. Effect of task duplication on the assignment of dependency graphs. Parallel Compute. 27(3).2001.256-270

[45]. Y. Su, M, Attariyan, J. Flinn. AutoBash: improving configuration management with operating system causality analysis. ACM SOSP 2007.

[46]. G. Barish. Speculative plan execution for information agents. PhD dissertation, University of Southernt California. 2003

[47]. D. K. Madathil, R. B. Thota, P. Paul et al. A static data placement strategy towards perfect load-balancing for distributed storage clusters Parallel and Distributed Processing. IEEE International Symposiun on 14 -18 April 2008.2008.1-8.

[48]. C. J. Patten, K. A. Hawick. Flexible high-performance access to distributed storage resources High-Performance distributed Computing.The Ninth International Symposiun on 1-4 Aug.2000.175-181

[49]. G. Lin, G. Dasmalchi, J. Zhu et al. Cloud Computing and IT as a Service: Opportunities and Challenges Web Services.ICWS '08.IEEE International Conference on 23-26 Sep. 2008.4-5

[50]. Voas. Jeffrey, Zhang Jia. Cloud Computing: New Wine or Just a New Bottle? Volume 11, Issue 2. 2009.15-17

硕士期间经历及取得的成果

[1]. 2008 年 7 月在 IBM 技术中心获得 AIX180 认证（IBM AIX Basic Operations V5 Certification）。

[2]. 2008 年 12 月 25 日，IBM 全国主机竞赛参赛作品“职工医疗保险电子交易系统”获得优胜奖。

[3]. 2009 年 10 月获得电子科技大学三等奖学金。