清 华 大 学

综 合 论 文 训 练

题目：物联网分布式计算模型研究

系　　别：信息学院自动化系

专　　业：自动化

姓　　名：邹睿

指导教师：董炜 助理研究员

2013 年 6 月 6 日

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：学校有权保留学位论文的复印件，允许该论文被查阅和借阅；学校可以公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存该论文。

(涉密的学位论文在解密后应遵守此规定)

签　名：邹睿　导师签名：董炜　日　期：2013.6.24

中文摘要

随着互联网技术的不断发展，信息呈现爆炸式增长，很多应用都需要对大数据进行处理，例如智能电网是作为一个典型的物联网应用，每时每刻都有大量的数据需要处理，而智能电网中各个数据汇聚节点都有一定的计算能力，但节点间由于带宽的限制所产生的时延较大，如何利用所有的计算资源在大时延的环境下对分散的数据完成分布式计算就是我们所要解决的问题。

我们基于MapReduce模型提出了一种适用于MapReduce的改进的Min-Min算法，并在MapReduce的开源实现Hadoop中，根据其架构特点，对它的调度算法进行改进，并给出了两种性能优于 Hadoop 自带调度器的调度算法。最后在 Hadoop 集群中对设计的调度算法进行实验验证。

关键词：时延；MapReduce；Hadoop；任务调度

ABSTRACT

With the development of Internet technology, the amount of information grows fast. Many applications need to deal with big data.Take smart grid for example, as a typical application of Internet of things, a large amount of data needs to be processed constantly .Each data collecting node has its own computing ability and the network delay also exists because of the limit of bandwidth.How to make use of distributed computing resources under delay condition is the problem this paper aims to solve.

A modified Min-Min algorithm adapting to MapReduce model is proposed .Two task scheduling algorithms are developed to improve the performance of Hadoop. Finally the proposed scheduling algorithms are implemented and tested in Hadoop cluster.

Key words : time delay;MapReduce;Hadoop; Task scheduling

目录

第1章 引言

1.1. 研究背景

1999 年，物联网（The Internet of things）的概念被提出。它是在现有的互联网技术、网络无线射频识别系统、无线数据通信技术的基础之上，构建的一个将所有“物”连接到一起的网络，并用于进行资源和信息共享。随着物联网技术的开发，物联网更是成为了未来能够改变人类生活的十大技术之首。

在物联网中，数据以流（stream）的形式实时、高速、源源不断地产生，因此物联网数据具有海量性;物联网往往是由大量子节点组成,并且每个子节点都收集数据，物联网数据传输复杂性成为必然;只有刚收集到的数据才能展现各子节点感触到物的状态，因此物联网的数据具有时效性。

由于物联网的数据具有以上特点，同时由于物联网各子节点会源源不断的采集新的数据，大量的数据需要及时处理,同时因物联网数据海量性不可能长期保存，所以系统的反应速度或者响应时间就是系统可靠性和实用性的关键。

智能电网作为物联网一个典型案例，大量的数据采集节点和计算节点分布在各地电网中，图 1.1 为 2008 年南方电网图，由 1393 节点、2371 支路、244 发电机、620 负荷组成，由于数据量过大和节点地理位置分布较远，时延对传输和计算的影响较大，同时，电网的决策支持系统对数据的实时性要求较高，在时延较大的情况下，如何利用大量分布式计算节点较快的完成分散的大规模数据的计算就成了一个问题，这也就是本论文研究的内容。

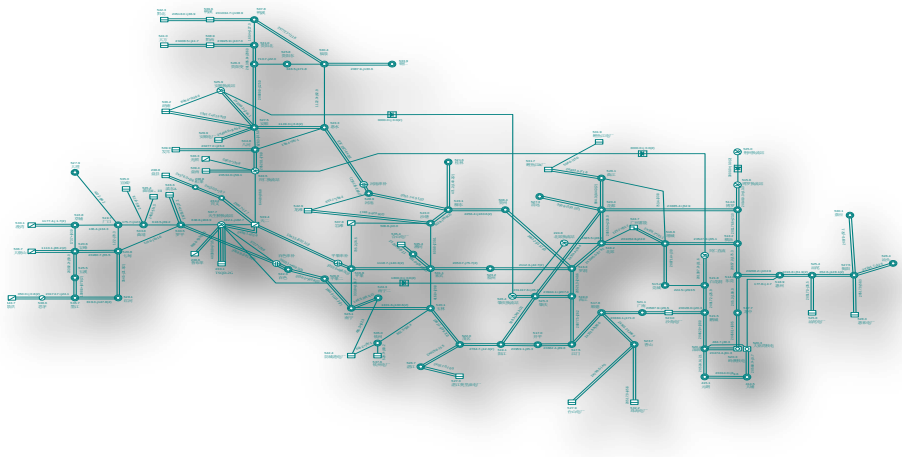

图 1.1 南方电网 2008 年节点图

这是典型的分布式计算问题，有两个问题需要解决，第一个问题是如何将大量计算转换成并行计算，从而将一个任务分为很多个子任务，并在大量子节点中同时运行各个子任务；第二个问题是用何种调度算法去对资源和任务进行统一管理，使得总任务完成时间尽量的短，且各计算子节点负载尽量均衡。

1.2. 研究现状

在现有的分布式计算模型中，网格计算和云计算模型是现在运用较多的两种分布式计算模型。云计算模型中的 MapReduce 将在第二章予以介绍，下面先对网格计算进行介绍：

网格计算是由网络中各子节点资源组成的一个统一的虚拟整体，利用这个整体的资源对网格中所有子节点用户提供服务[1]，是一种在大规模分布式资源中实现资源协调共享和解决问题的过程。

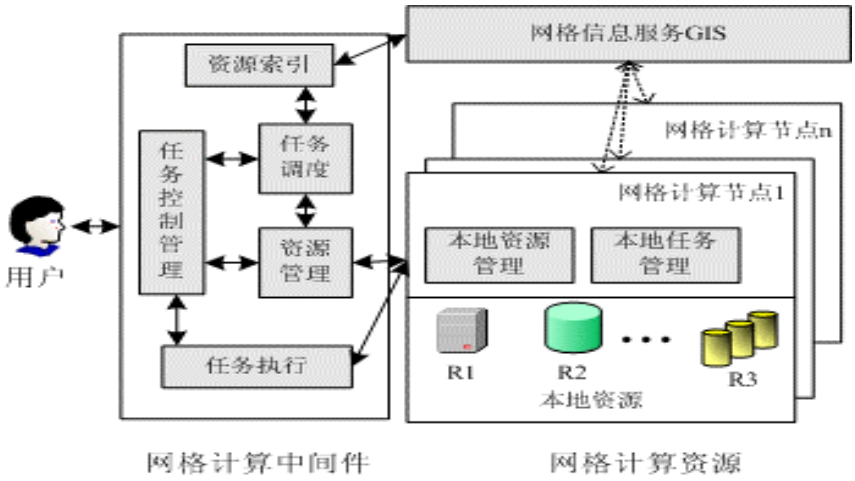

图 1.2 网格计算流程图

图 1 展示了网格计算的工作原理，网格的计算节点可以是服务器、集群，也可以是嵌入式设备以及工作站。在 GIS(Grid Information Server)中注册各计算节点，GIS 中存储有各计算节点的资源信息。每个计算节点上都有一个本地资源管理来管理本地资源，并通过通信与网格中间件中的资源管理器进行资源整合，共同调度资源，本地任务管理器则监督子任务在本地的执行情况。

用户向网格计算中间件的任务控制管理提交任务。然后任务控制管理组件将任务交给任务调度器，通过一定的方法，任务调度器将任务划分成许多能够无关联的子任务，同时将所有子任务的信息交给资源索引，通过资源定位，各子任务都获得适合各自任务的资源信息，并通过任务管理把所有子任务移交给相应的计算

节点的任务执行器。各子任务完成后，将任务结果传递给用户。

网格计算将所有计算资源构建成统一的整体，用户提交的任务可以方便地使用大量的资源来完成，而无需考虑所需资源的具体位置。它的任务多来自于单一子节点，然后通过大量节点来处理，输入的数据并不是分布式，使得它有一定的自治性。我们课题的大背景是基于电网，难免要对整体的数据进行处理，网格计算更多的考虑是单个任务如何高效的完成，而我们需要的是一个能缩短总体时间的分布式计算模型。

对于网格计算的任务调度算法，目前，已有很多科研组织对网格计算这种分布式计算模型的调度进行了研究。下面对一些较为经典的进行简单介绍：

AppleS[2]：即为 Application Level Scheduling，AppleS 的主要目标是将资源以最有效形式对任务进行调度，它设计了一个自适应算法来进行任务调度，在这个算法中，Max-Min,Work queue,Sufferage,Min-Min 等调度策略都可使用，对不同特性的任务选择最合适的的调度算法。

Nimrod[3]：此系统为在满足任务条件的同时提供经济计算模式，该系统根据任务在各节点预测的任务执行资源使用量和任务完成时间来进行一定的资源调度。

Condom[4][5][6]为了能够充分利用网格中的空闲资源来为用户服务，它的调度过程主要分为两个部分：匹配和声明，首先通过信息收集收集所有任务和子节点的信息，然后对每个任务进行资源配置，让每个任务选择适合处理该任务的节点资源，然后通知任务管理器和适合的节点资源，两者之间进行通信确认，确认完毕后开始进行数据传输处理。Condom 的优势在于能用于高吞吐率计算。

1.3. 本论文所做的工作

本文考虑时延对任务总时间的影响，基于 MapReduce 的分布式计算模型，通过参考网格计算的任务调度算法，设计了一种适用于 MapReduce 模型的调度算法，并在 MapReduce 的开源实现 Hadoop 上，提出了三种调度算法对其 Hadoop 任务调度进行改进，并在 hadoop 集群上予以验证。

1.4. 论文的组织结构

第二章介绍了 MapReduce 模型和 MapReduce 的开源实现 Hadoop，并对 Hadoop 的架构和调度机制进行了详细说明；第三章改进了 Min-Min 算法，使其适用于大

时延网络环境下的 MapReduce 任务调度，并通过仿真对 Min-Min 算法的调度性能进行分析；接着基于 Hadoop，对其调度算法进行分析，基于新增加的参数设计了三种 Hadoop 调度算法，然后对三种算法进行仿真；第四章通过在 Hadoop 集群进行实验，验证了由于带宽的限制而产生的时延对 MapReduce 调度的影响，并修改 Hadoop 调度算法进行验证性实验；第五章为总结和未来展望。

第2章 MapReduce 模型介绍

2.1. MapRedece 的基本原理

MapReduce 是由 Google 公司提出的一个编程模型[7]，它被设计用于大规模的数据处理，在 Google 公司内部，每天都有成千上万的 MapReduce 作业被执行。但 Google 只提出了这一模型，并没有对 MapReduce 进行开源。Hadoop 项目于 2006 年初开始，一直致力于 MapReduce 的开源实现，并且不断完善 MapReduce 的功能。经过 7 年的发展，Hadoop 已经成为了云计算领域的首选技术。

从 MapReduce 自身的命名特点可以看出，MapReduce 将要执行的问题拆解成 Map 和 Reduce 两个阶段，用户只需编写 map 和 reduce 两个函数，即可完成简单的分布式程序设计。MapReduce 将输入按照用户需求切割成大小相同的数据片段，且各个数据片段中的数据并不相关，然后将每个数据片段作为一个 Map 任务的输入，将所有 Map 任务分配个集群中的计算节点进行分布式计算，然后将 Map 任务的结果聚合起来，排序后用 Reduce 任务对中间结果进行迭代。

2.2. MapReduce 处理任务的具体步骤

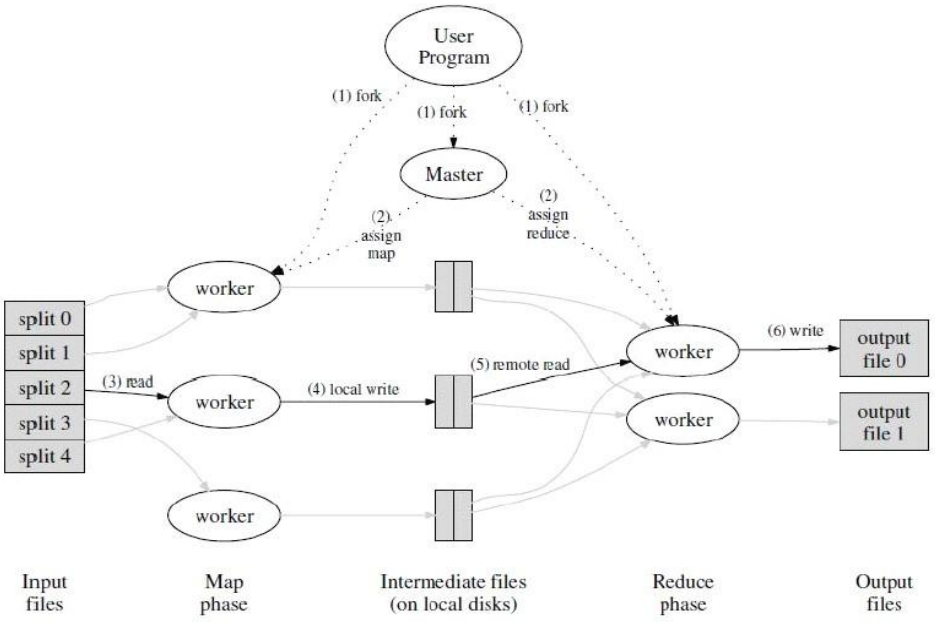

图 2.1 MapReduce 流程图

图 2.1 展示了我们的 MapReduce 接受作业后的全部流程。当有作业提交时：

1．程序首先调用 MapReduce 库中自带的函数或者自定义的 split 切割函数将输入的文件按照用户定义的大小切割成很多个 split，split 如果用户不进行配置，默认为 64MB。

2. 一个 split 作为一个 Map 任务的输入，将所有的 split 分别调度给所有的 Map Worker，Map 函数先将 split 分割成<key，value> pair，一般 key 值都是该数据的首字符相对于首字符的偏移量，然后对所有<key,value>pair 进行 Map 操作，产生新的<Key,value>pair。

3．对每个 Map 的输出进行 Combin 操作，即简单的合并操作，将具有相同的 key 值的数据合并，以减小中间数据的大小。

4．对所有的 map 输出进行排序，将具有相同 key 值的<key，value.>pair 聚合在一起，并一起传递给一个 Reduce worker 进行 Reduce 操作，值得注意的是，这个阶段将有很多不同的 key 值的数据给一个 Reduce 进行处理，所以先对输出进行排序就是十分必要的。

5． Reduce 任务执行完后，每个 Reduce 任务都有一个输出存在本地的分区文件上。直到所有子任务完成，一个 MapReduce 作业才全部完成。

2.3. MapReduce 的优点

MapReduce 模型能够解决的问题都有一个共同特点：作业可以被划分成多个子任务，而且每个子任务之间都相对独立不会有牵连，通过并行处理完所有子任务后，再通过迭代就能完成作业。在实际运用中，有大量的问题都能通过 MapReduce 解决，google 在其论文[7]中提出“贝叶斯分类，分布排序,文档聚类，K-means 聚类，web 连接图反转，web 访问日志分析，Top K 问题，反向索引构建”等都是 MapReduce 的典型运用。其他相关工作[12]表明,MapReduce 可以成功地用于图问题,如发现图形组件,质心聚类、枚举矩形和三角形列举。MapReduce 也进行过科学问题[13]。它在简单问题中表现良好，像组极方法产生随机变量和整数排序。

然而,MapReduce 也被表示有很大问题在更复杂的算法[14],如共轭梯度,快速傅里叶变换和块三对角线性系统求解。此外,这些问题大多是通过迭代的方法来解决,这表明 MapReduce 可能不适合迭代性质的算法。

MapReduce 不仅能解决很多实际问题，同时它也提供了以下几点优良特性：

（1）改变了传统的计算模式，移动计算是 MapReduce 的基本理念，MapReduce 的任务调度优先考虑本地的计算节点，即 master 优先将子任务分配给它数据块所

在的节点位置处理，使得计算尽量本地化，移动计算要比移动数据的更加的经济。

（2）MapReduce 设计了很好的容错机制，在 MapReduce 模型中，它认为机器出现故障或者网络中断是很正常的，所以 MapReduce 设计了完善了故障处理机制。它一方面对所有的子任务以及计算节点进行监控，另外一方面优先处理失效的任务。

（3）MapReduce 系统具有良好地扩展性。在 google 内部的 MapReduce 中，每个典型的 MapReduce 任务都执行在成百上千台计算机上。在 MapReduce 的开源实现中 Hadoop 中，3000 多个节点的 Hadoop 集群已经得以实现[11]，在这个集群上运行了 80000 多个 Map 任务和 20000 个 Reduce 任务。

2.4. Hadoop 平台

Google 共享了 MapReduce 的技术要点，但是并没有进行开源。Hadoop 作为 MapReduce 的开源实现，可以通过搭建 Hadoop 集群来运行 MapReduce 作业。

2.4.1 Hadoop 概述

2006 年年初，Hadoop 作为 Lucene 的子项目 Nutch 中的一部分被 Apache Software Foundation 基金会引进，它受到了 google 两篇论文的启发（一篇是 2003 年发表的关于 google 分布式系统(GFS)的论文，另一篇是 2004 年发表的关于 google 分布式计算框架 MapReduce 的论文），于 2006 年 2 月，Apache Hadoop 项目正式启动，支持 MapReduce 和 HDFS 的独立发展。经过七年的发展，hadoop 在所有开源云计算系统中稳居第一。很多高校和企业都在大规模使用 hadoop。[10]

2.4.2．Hadoop 基本构架

Hadoop 由两部分组成，它们分别是分布式文件管理系统 HDFS 和分布式计算框架 MapReduce。HDFS 主要用于大规模数据的分布式存储，而 MapReduce 则对存储在 HDFS 中的大规模数据进行分布式计算。

HDFS 构架:

HDFS 作为一个高容错的分布式文件系统，能提供高吞吐量的数据访问，适应于大规模数据的存储。

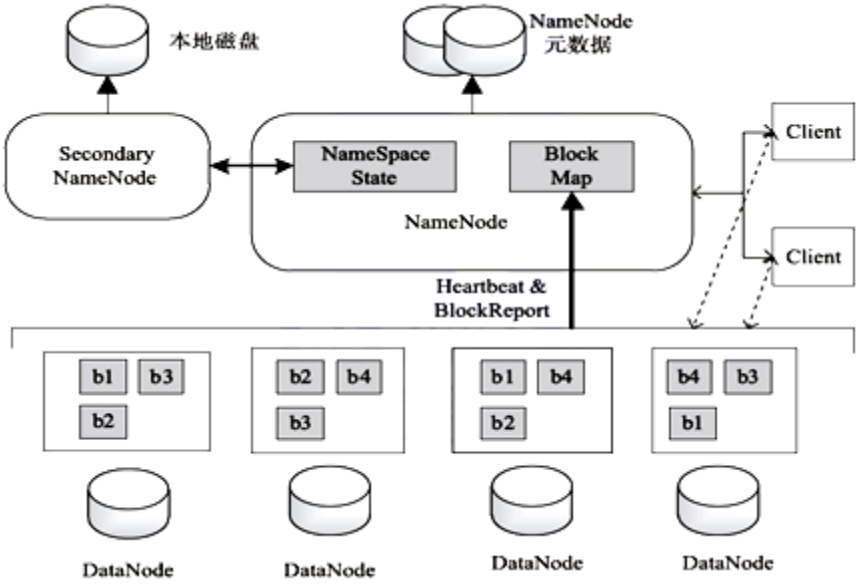

图 2.2 HDFS 架构图

HDFS 的构架如图 2.2 所示，它继承了 MapReduce 的 master/slave 架构，HDFS 中的 NameNode 对应于 master，DateNode 对应于 slave。Client 代表用户，Secondary NameNode 作为 NameNode 的一个辅助节点。

整个 Hadoop 集群只有一个 NameNode，它负责管理 HDFS 的目录树，含有所有存储任务的存储地址。此外，NameNode 还需监控各个子节点 DataNode 的健康状况，如果发现 DataNode 没有响应，则需重新备份存储在 DataNode 上面的数据。

DataNode 是 Hadoop 上的计算节点以及数据存储节点，它以固定大小的 block 存储文件，定期将数据信息汇报给 NameNode。当用户提交一个大文件到 HDFS 上时，该文件会被切割成若干个 block，按照一定的规则，分别存储到不同的 DataNode 上。

Hadoop MapReduce 构架:

Hadoop MapReduce 也采用了 Master/Slave 构架，其构架图如图 2.3。它由 Client，TaskTracker，JobTracker、Task 组成。下面对这几个组件进行介绍：

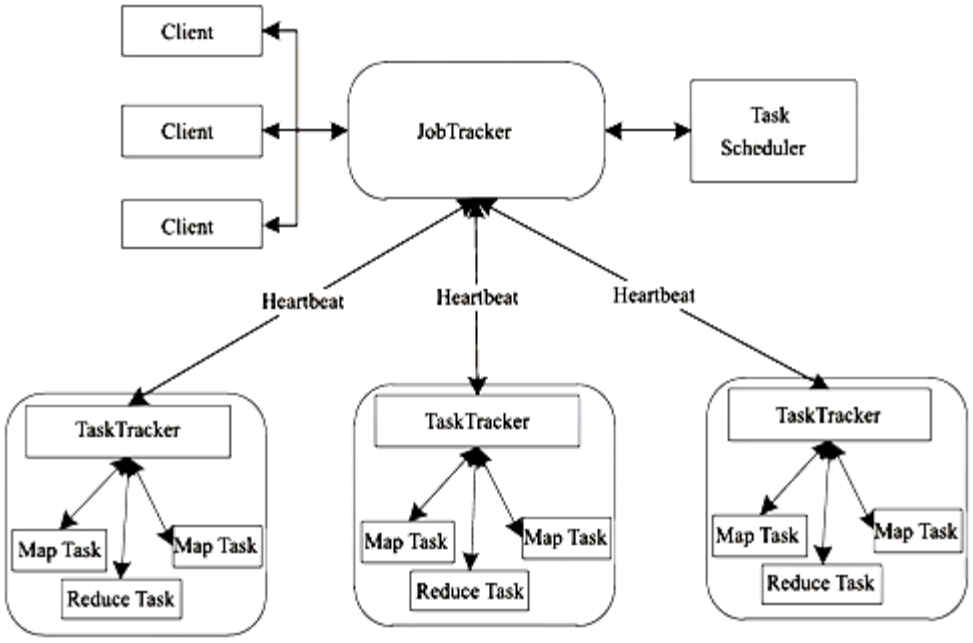

图 2.3 Hadoop MapReduce 架构图

在 Hadoop MapReduce 架构中，Client 将用户编写的 MapReduce 程序提交给 JobTracker。在 Hadoop 中，作业（Job）表示 MapReduce 程序。每个作业被分为若干个 Map/Reduce 任务（Task）。

JobTracker 在 NameNode 节点上，主要用于负责资源监控并通过调用 Task Scheduler 对作业和任务进行调度。所有 TaskTracker 的健康状况和作业的完成情况都被 JobTracker 所监控。一旦发现有任务失败或者 TaskTracker 出现故障后，JobTracker 会将其转移到其他节点上重新执行任务；同时，JobTracker 会跟踪所有任务的执行进度以及各 DataNode 资源使用量，并将这些信息告诉 Task Scheduler，当节点资源出现空闲时，TaskScheduler 会选择合适的任务使用这些资源。

TaskTracker 部署在 DataNode 上，它会主动的通过周期性的 heartbeat（心跳机制）将本节点上所有 slot 的使用情况和任务的完成进度汇报给 JobTracker，同时接受 JobTracker 的心跳应答，该应答由两部分构成，一部分为给 TaskTracker 的命令，另外一部分为下次汇报心跳的时间间隔。在 DataNode 上，资源被 TaskTracker 等量划分为若干个 slot。一个 slot 可以执行一个任务，当 slot 无任务时，TaskTracker 在 heartbeat 中向 JobTracker 反馈信息，Hadoop 调度器的就会给空闲的 slot 分配 Task。

Task 即为 Map Task 和 Reduce Task，这两种任务均由 TaskTracker 启动。Map

Task 的输入数据大小为 split，与 HDFS 的存储大小可以不同，可由用户定义。Split 与 block 的关系如图 2.4。split 个数即为 Map 的个数，所以 split 不能太大也不能太小，太大容易影响任务完成时间，太小容易使得 JobTracker 的任务量过大，数据读写次数过多。Map Task 和 Reduce Task 执行流程分别如图 2.5、图 2.6。

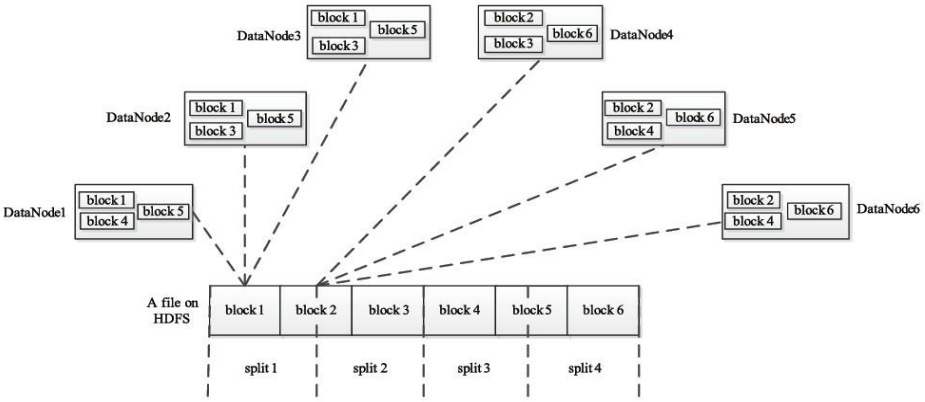

图 2.4 split 和 block 的关系

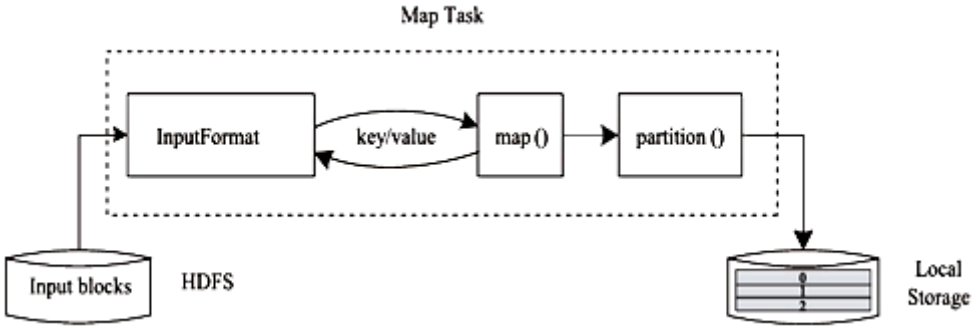

图 2.5 Map Task 执行流程

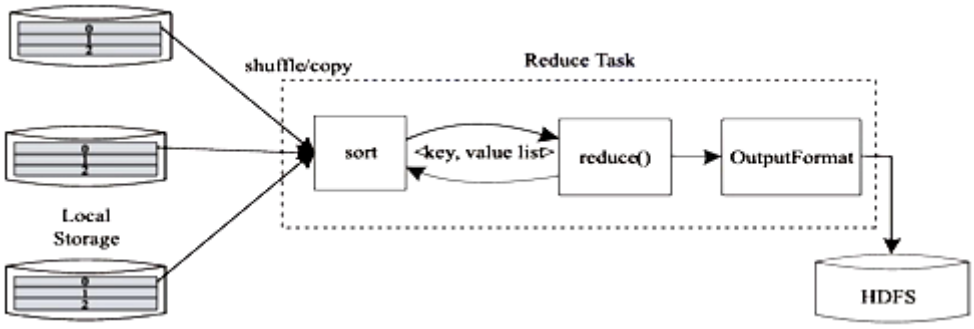

图 2.6 Reduce Task 执行流程

2.4.3. Hadoop 计算模型假设

Hadoop 项目主要目的是在实现 MapReduce 以及其分布式存储文件系统 HDFS，所以在设计之初，隐含了一些假设，这些假设都是根据负载均衡和同构集群建立的，而这些假设也影响了 Hadoop 推测执行设计算法。

假设主要有以下四点：

1.每个节点具有相同的计算能力

2.任务进度随时间线性增加

3.对失败的任务重新启动的代价忽略不计

4.同一作业所有 Map 任务计算量相同，所有 Reduce 任务计算量相同

2.4.4. Hadoop 任务选择策略

在 Hadoop 中，任务调度是一个可以拔除的模块，用户可以根据自己的 MapReduce 函数特点和 Hadoop 集群特点设计调度器。

目前的 Hadoop 版本自带了三种作业调度器：即 FIFO、Capacity Scheduler 和 FairScheduler。这三种调度器是不同的作业调度器，但是他们调用了相同的任务调度函数，并把任务调度函数存放在 JobInProgress 类中的 obtainNewMapTask 和 obtainNewReduceTask 函数中。

Hadoop 的任务调度将数据本地性当成最重要的考虑因素，尽量把最多的任务在本地进行计算，从而减少任务数据传输过程的网络传输开销。

对于目前的 Hadoop 而言，任务调度策略都采用了两层网络拓扑结构，如图 2.7 所示，所有的任务对每个 datanode 而言都划分为三种：rack-local（同机架）任务，node-local（同节点）任务以及 off-swith（跨机架）任务。

如果 H1 是计算节点，任务 Y 的数据在 H1 上，则 Y 是 node-local 任务。

如果 H1 是计算节点，任务 Y 的数据在 H2 上，则 Y 是 rack-local 任务。

如果 H1 是计算节点，任务 Y 的数据在 H3 上，则 Y 是 off-swich 任务。

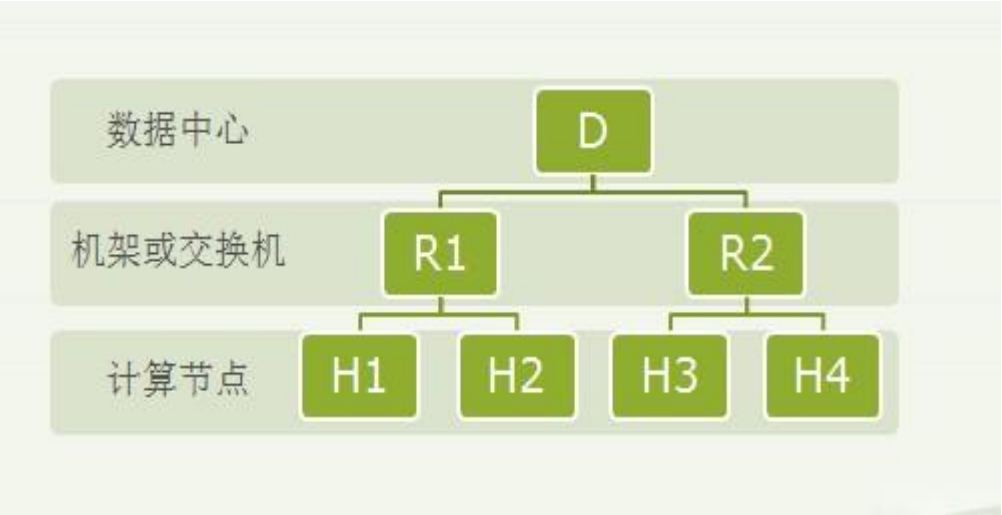

图 2.7 Hadoop 任务调度两层拓扑结构

当某个 DataNode 空闲时，调度器会调用任务调度器，任务调度器会优先调度运行失败的任务，如果这种任务列表为空，则优先将 node-local 任务即数据在本地的任务分配给该 DataNode，如果本地任务为空，则选择 rack-local 任务，若同机架任务也为空，则选择 off-swith 任务，当不在本地的任务也为空时，则检测是否有正在执行拖后腿的任务，并为该任务启动一个备份任务。当有任一节点将该拖后腿的任务执行完后，终止所有该任务的备份任务。

2.5. 本章小结

本章介绍了 MapReduce 的基本原理和执行 MapReduce 作业的基本步骤，MapReduce 能很好的将一个能够进行并行的任务分解的作业分解成很多个子任务，以此为基础对作业进行分布式任务调度，使单个作业可以利用集群中所有空闲资源进行处理，从而快速的完成作业，为大规模数据处理提供了可行的方法。本章还介绍了 MapReduce 的开源实现，通过介绍 Hadoop 的基本构架，分析了 Hadoop 自带的信息传输机制以及本身的任务调度器，为第三章利用 Hadoop 集群运行 MapReduce 作业和对传递的节点信息修改和调度算法可行性验证和改进提供基础。

第3章 异构环境下基于网络时延的 MapReduce 任务调度模型设计及仿真实验

第二章中介绍了 MapReduce 的基本思想，同时 MapReduce 具有良好将串行计算转换为并行计算能力，MapReduce 更多的研究在于如何将各种算法运用在 MapReduce 模型中，并研究他们的效率和伸缩度。但是 MapReduce 设计中对任务调度并没有过多的设计，主要是针对大量的作业调度进行了优化，防止出现由于某个作业占据大量的资源使得某些作业得不到应得资源的情况。这章将对在较大时延下的单个 MapReduce 作业进行任务调度研究。

3.1. 数学建模

已知条件：计算节点间时延、计算节点处理速度、任务规模

目标：任务求解总体时间最短

数学建模如下：

1. C$(c_1, c_2, \cdots c_n)$表示所有的计算节点
2. D $= \begin{bmatrix} d_{11} & \cdots & d_{1n} \\ \vdots & \ddots & \vdots \\ d_{n1} & \cdots & d_{nn} \end{bmatrix}$矩阵表示各计算节点间的网络时延(s)
3. P$(p_1, p_2, \cdots p_n)$表示所有的计算节点的计算能力（已每秒处理的数据数表示）(MB/s)
4. t$(t_1, t_2, \cdots t_n)$表示任务数据在所有节点上分布的情况(MB)
5. T $= \begin{bmatrix} T_{11} & \cdots & T_{1n} \\ \vdots & \ddots & \vdots \\ T_{n1} & \cdots & T_{nn} \end{bmatrix}$表示分配后的数据量在各个计算节点上的分布情况
6. b 表示经过处理后的数据规模
7. r $= (r_1, r_2, \cdots r_n)$表示各个计算节点上存储的中间节点规模情况
8. R= $\begin{bmatrix} R_{11} & \cdots & R_{1n} \\ \vdots & \ddots & \vdots \\ R_{n1} & \cdots & R_{nn} \end{bmatrix}$表示分配后的数据量在各个计算节点上的分布情况
9. 假设每个计算节点上都只部署单一 slot，即每个计算节点上不能并行处理多个任务
10. 假设需要调用的 Map worker 数为 m，Reduce worker 数为 r
11. 选取节点 i 为 Map 节点则记$x_i = 1$，否则$x_i = 0$

12. 选取节点 i 为 Reduce 节点则记$y_i = 1$，否则$y_i = 0$

13.不考虑调度算法的运行时间

则依据以上条件，可将问题建模如下：

$$\min_{j \le n} \max \left(\sum_{i=1}^{n} \frac{T_{ij}}{p_j} + \sum_{i=1, i \neq j}^{n} \frac{T_{ij}}{d_{ij}} + \sum_{i=1}^{n} \frac{R_{ij}}{p_j} + \sum_{i=1, i \neq j}^{n} \frac{R_{ij}}{d_{ij}} \right)$$

$$\sum_{j=1}^{n} (T_{ij} \times x_i) = t_i \quad (\forall\, i = 1,2 \cdots n)$$

$$\sum_{j=1}^{n} b \times T_{ij} = r_i \quad (\forall\, i = 1,2 \cdots n)$$

$$\sum_{j=1}^{n} (R_{ij} \times y_i) = r_i \quad (\forall\, i = 1,2 \cdots n)$$

$$\sum_{i=1}^{n} x_i = m$$

$$\sum_{i=1}^{n} \mathrm{y_i} = r$$

$$x_i \times \mathrm{y_i} = 0$$

$$T_{ij} \geq 0 \ (\forall\, \mathrm{i} > 0, j > 0)$$

3.2. 基于 MapReduce 的调度算法设计

在实际网络情况下，节点间延迟和各节点的负载情况随时间变化，网络情况下的任务调度问题已被证明是 NP 完全问题，很难通过调度获得最优解，因此大多数任务调度算法都被设计为启发式算法为主，很多这类算法都能近似与最优解。

静态调度算法在任务调度前就决策完成，静态任务调度算法的开销较低，而且静态调度算法有足够的时间利用所有信息，更能趋进于最优解，能够产生有效地调度策略，但在网络的情况下，很多参数现在都无法定量去预测，只能根据实测值去收集参数。所以并不具备较强的可用性。

较为典型的静态调度算法有遗传算法，蚁群算法和模拟退火算法。

动态任务调度算法分为两类，一类是在线模式启发式算法[8]，一类是批模式启发式算法[8]。在线模式启发式算法可以对所有已知的空闲 slot 会立刻做出决策，

分发任务，具有较强的实时性，但在线启发式算法多是根据某一特定的规则指派任务，只考虑到单一节点的信息，而没有考虑到其他节点的信息，最后的调度性能会受到很大的影响。批模式启发式处理算法则是将任务信息收集到一个队列中，将一定时间内的任务进行统一处理再调度。所以批模式启发式处理算法具有一定的实时性同时也具备静态算法的高效性[15]。

较为典型的在线模式启发式算法有 MCT(最小完成时间算法)，MET（最小执行时间算法），OLB（Opportunistic Load Balancing）等。

较为典型的批模式启发式算法有快速贪心（Fast Greedy）算法，Min-Min 算法、Max-Min 算法，Sufferage（损失度）算法等。

针对 MapReduce 中每个 Map 任务具有相同的输入数据大小的情况，所有 Map 任务所需的计算量都一样，任务完成时间差异具体在数据传输上，所以本地的任务必然在任务完成时间上更小，所以计算本地化能够大幅度的减少总的作业完成时间，同时计算优先本地化也能使得大规模数据网络传输中带宽的限制不再是问题。

Min-Min 算法即为根据任务完成时间来选择任务的一种算法，Tracy D 等人的论文结果表明[10],单一考虑作业完成时间而不考虑负载等问题情况下，Min-Min 能得到优于其他算法的调度性能。但是 Min-Min 在任务大小不一样的情况下，Min-Min 算法容易造成大量小任务优先在计算能力强的节点上执行，使得计算本地化较低，容易造成大量的数据传输，对于时延较大的系统，这点往往让人很难以接受。但是 Min-Min 算法对于具有相同计算量的数据任务，有着较好的本地化能力，下一小节将针对 MapReduce 特点及各节点时延较大的特点改进 Min-Min 算法，以适用于课题。

3.2.1. 改进 Min-Min 算法

Min-Min 算法根据命名含义可知，即为找最小值中的最小值，该算法先预测每个任务在所有计算节点上的最小完成时间，这个时间包括：任务计算时间，传输时间和存储时间等。然后再计算所有任务的最小完成时间中的最小值，将这个具有 Min-Min 值的任务从任务列表中删除，并分配给对应的计算节点。

记第 i 个子节点任务在第 j 台主机上的预测最小完成时间 MCT(i,j),若存在 m 个数据存储节点和 n 个计算节点，则 MCT 矩阵是一个 m*n 的矩阵。MCT 主要由以下几个参数决定：

1．数据节点 i 上的任务数量 n(i);

2．数据节点 i 上的任务在主机 j 上的预测执行时间 ETC(i,j)；

3．计算节点 j 的最早可用时间 TaskStart(j)（即上个任务完成时间）；

4．通过网络把任务节点 i 上一个任务所需的数据传输到计算节点 j 上的传输时间 Trans(i,j)；

MCT（i,j）的计算公式为：

MCT(i,j)=ETC(i,j)+TaskStart (j) +Trans(i,j).

该算法具体调度步骤：

1.当任务列表非空时，重复执行如下操作直至任务列表为空；

2.对于任务列表中所有待处理的任务，根据以上公式计算出所有任务到所有节点的任务完成时间，同时对所有任务找出它最小的预测完成时间，并将预测完成时间和计算节点记录下来，将最小预测完成时间存入数组 MT 中;

3.查找出 MT 数组中最小 MCT 的任务分配给产生这个最小 MCT 的计算节点；

4.从需要调度的任务列表中删除这个最小 MCT 的任务，并更新 MCT 矩阵。并重复以上操作。

3.2.2．改进 Min-Min 算法仿真

利用 Matlab 进行如下两个仿真：

（1）Min-Min 算法对所有任务计算量相同作业的计算本地化仿真

生成 100 个节点，每个节点随机生成 2~14MB/s 的数据处理速度，两两节点间的一个任务数据的传输时间 Trans 为均值 1S 方差为 2 的正态分布绝对值随机数，每个节点随机生成 4~100 的任务数 n（i）。

在 MapReduce 中每个任务数据默认大小为 64MB。为了将计算本地化程度是否由于任务的计算量相同而大量改进，将每个节点上任务生成[n（i）/2]个 2~126 之间的数值，再用 128 减去每个数值，得到另外一半任务大小，获得一个大小不相等，各节点总任务数据和原来大小一样的一个任务集。

将两个任务数据大小相同，分布状态相同，但任务划分不同的作业根据 Min-Min 算法进行调度仿真。通过十次仿真结果，实验平均值结果如表 3.1。

表 3.1 Min-Min 算法本地化实验结果

	总任务数	本地任务数	本地任务比	作业总时间（s）
相同的任务大小	5421	4078	0.7535	497
不同的任务大小	5421	273	0.0504	578

仿真实验可知，Min-Min 算法十分适用于任务输入数据大小相同的调度策略中，

具有较好的计算本地化能力。

（2）改进的 Min-Min 算法与快速贪心算法的作业总时间仿真

生成 100 个节点，每个节点随机生成 2~14MB/s 的数据处理速度，两两节点间的一个任务数据的传输时间 Trans 为均值 1S 方差为 2 的正态分布绝对值随机数，每个节点随机生成 4~100 的任务数 n（i），每个任务的数据大小默认为 64MB

快速贪心算法具体步骤：对所有 Map 任务进行随机排序，按照任务顺序进行选择，预测当前任务在所有节点执行完这个任务的时间，选出其中的最小值节点，将任务进行分配。然后刷新所有节点的可以接受任务的最早时间，重新对下一个任务进行选择，直至所有 Map 任务都被调度出去。

通过十次仿真结果，实验平均值如表 3.2

表 3.2 改进 Min-Min 算法仿真结果

	总任务数	本地任务数	本地任务比	作业总时间
改进的 Min-Min 算法	5107	3820	0.748	479
快速贪心算法	5107	298	0.058	608

仿真实验表明，在时延较大的 MapReduce 模型中，改进的 Min-Min 算法较之快速贪心算法有着较好的效率。

3.3. 基于 Hadoop 的调度算法改进

在 Hadoop 集群的信息管理中，任务调度机制是由计算节点向主节点申请任务，3.2 节中的 Min-Min 算法等批模式启发式算法无法在 Hadoop 中运用。Hadoop 的任务调度并没有考虑节点间时延情况，调度器调度主要依赖各节点数据信息和各节点的任务信息进行简单的调度。由于 Hadoop 源代码中已经实现将运行作业的架构完整实现，要修改 Hadoop 任务调度算法,只能是基于 Hadoop 框架进行改进。

3.3.1 Hadoop 任务调度流程

Hadoop 任务调度流程如图 5.1 所示，在 Hadoop 架构中，JobTracker 不会主动向 TaskTracker 分配任务，而是由 TaskTracker 每隔 3s 向 JobTracker 发送一次心跳（Heartbeat），Heartbeat 中包含了当前节点的运行信息，JobTracker 再给应答心跳

中给出任务。

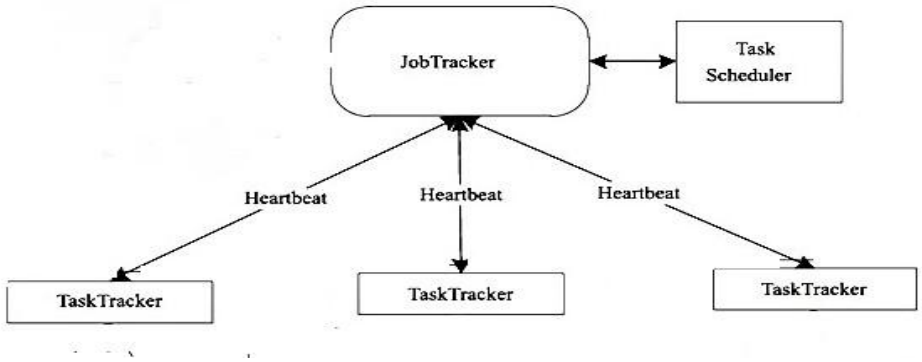

图 3.1 Hadoop 任务调度流程图

每个 Heartbeat 中的 TaskTrackerStatus 包含有以下信息：

String trackerName;//TaskTracker 名称

String host;//TaskTracker 的主机名

int httpPort;//对外的 HTTP 端口号

List<TaskStatus> taskReports;//当前 TaskTracker 上所有任务的运行状态

private TaskTrackerHealthStatus healthStatus;//TaskTracker 健康状态

private ResourceStatus reaStatus;//TaskTracker 的内存,CPU 情况

taskReport 存储了任务信息，其中包括每个任务的完成情况，开始时间和完成时间。resStatus 中保存了 TaskTracker 资源情况。

3.3.2. 增加和修改参数

为了提高任务调度效率，增加以下两个参数：两两节点时延参数和各计算节点完成一个 Map 任务所需的时间参数。由于在小规模集群下实验，通过参数配置 TaskTracker 的心跳时间间隔，将其调成 300ms。

各节点完成一个 Map 任务所需时间可以由 TaskReport 中的任务开始时间和完成时间相减得到，这两个参数都存在 taskReport 中，CountTime=finishTime-startTime。同时为了保证数据的实时性，需要先找到该 TaskTracker 上完成时间最晚的一个本地任务，计算它的任务完成时间，作为该 TaskTracker 上的计算能力，用于调度。

将时延参数加入 HeartBeat 的方法：在 Java 中 exec（”ping”+ip）即可获得每个节点到所有节点中的时延，并把它按照顺序存入一个文本中。当 MapReduce 运行时，同时在所有子节点上运行这个 Java 程序，并以 3S 为时间间隔反复调用以上函数。当 TaskTracker 要发送 HeartBeat 时，初始化 HeartBeat 时读入文本中时延，并

把它存在一个 Time Delay 数组中（各节点到本节点的时延为 0）。

JobInProgress 类主要用于监控作业运行状态，并为调度器提供底层的调度接口。在 JobInProgress 中加入二位数组 TimeDelay 存储延时，再加入所有子节点的计算能力 Count 数组。 当 JobTracker 通过心跳机制接收到 HeartBeat 时，将参数写入到 JobInProgress 中。

3.3.3. Hadoop 调度算法改进

Hadoop 采用的是当计算节点空闲后向主节点申请任务的机制，该机制更适合在线启发式算法的调度，而不适用 Min-Min 算法等批模式启发式算法调度。

通过上一节增加的时延参数和各节点计算能力参数，重新编写 Hadoop 的 Map 任务调度函数。Hadoop 的调度函数在 JobInProgress 类的 obtainNewMapTask 中。

在 Hadoop 中，任务本地化既可以缩短任务时间又可以减少网络传输开销，所以计算本地化依然是算法的核心思想。

对于基于 Hadoop 构架 MapReduce 调度算法，根据上一节添加的两个参数分别给出了两种算法：

第一种是较为常见的快速贪心算法，它根据各个任务到当前需要任务分配的计算节点的延时大小，选择延时最小的任务进行执行。该算法具有较好的作业完成速度，但是快速贪心算法容易造成大量计算的非本地化，使得网络开销较大。

第二种算法为：当有计算节点需要任务分配时，优先考虑本地数据任务，当本地数据任务处理完后，通过各节点计算能力和剩余任务量，优先考虑各节点完成剩余任务所需时间最多的节点任务进行分配。该算法没有用到时延条件，而是以尽可能的计算本地化为标准，所以该算法的计算本地化较好，但与此同时，任务执行时间与 Hadoop 原算法不会有太大的改进。

前两种算法都存在一定的缺陷，基于两种算法不足，本文提出第三种算法进行改进。算法三步骤如下：

（1）当有计算节点需要任务分配时，先遍历本地任务列表，如果不为空，则选取一个任务执行，任务分配结束

（2）当本地任务为空时，将其他所有剩余任务数不为 0 的节点（假设有 n 个），按照各节点剩余任务所需处理时间进行排序，选择剩余任务所需时间最长的 Z（Z 等于 n 乘以 m 向上取整）个节点（m 为一个百分比，可设置）。

（3）选取这 Z 个节点到计算节点最小的延时节点，若该延时小于给定值 t，则选择这个最小的延时节点上的任务。反之，则遍历 n 个节点到计算节点的延时，选

择其中的最小延时节点任务分配给该计算节点。

（4）当所有节点剩余未执行的任务数为 0 时，遍历所有正在执行的任务，找出其中预计时间最慢的任务，并与当前节点执行该任务预计结束时间进行比较，若能更快执行完该任务，且该任务备份数小于 3，则在这个节点启动该任务的一个备份，当任意节点完成该任务时，停止其他节点上该任务。

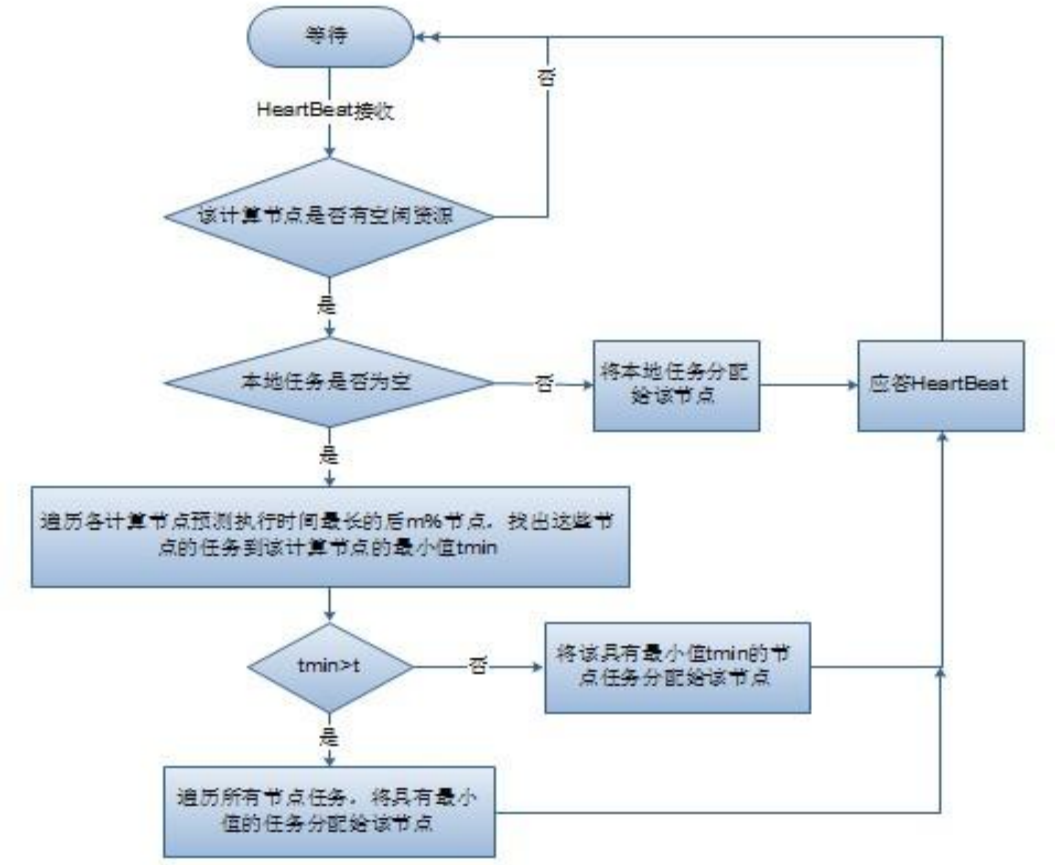

图 3.2 算法三流程图

根据网络情况调整 t 参数，m 可由实验调整，m 越大本地性越小，数据越快。

3.3.4. 设计算法的仿真实验

先对三种算法进行 Matlab 仿真实验：

现生成 100 个节点，每个节点随机生成 2~14MB/s 的数据处理速度，两两节点间的一个任务数据的传输时间 Trans 为均值 1S，方差为 2 的正态分布绝对值随机数，Trans 矩阵并每 3s 更新一次，每个节点随机生成 4~100 的任务数 n（i），每个

任务的数据大小默认为 64MB。

对相同参数和相同网络时延变化，对三种算法进行仿真，算法三取中 t=0.5S，m 取 30%。结果如下表 5.1 所示。

表 3.3 三种算法仿真结果

	总任务数	本地任务数	本地任务比	作业总时间（s）
快速贪心算法	5390	3246	0.602	513
算法二	5390	4096	0.760	628
算法三	5390	3931	0.731	547

由仿真实验结果可知，快速贪心算法具有较快的作业完成速度，但是任务本地性较差欠佳，不过由于贪心算法也会优先考虑本地任务，所以本地任务比仍保持较高的水平；算法二本地任务比较高，但是无法掩盖其作业完成速度较差的缺陷；

算法三的本地任务率较之算法二也相差不大，但是作业总时间有了显著的提高。

通过这个仿真实验可知，贪心算法和算法三都对 MapReduce 有较好的调度效果。

接下来，将通过实验对比选取合适的 m 值，从而达到更好的调度效果。

分别选取 m=0%，10%，20%，30% ，40%，50%，60%，80%，100% 。

t=10000S 对实验进行仿真，仿真结果如表 5.2 所示。

表 3.4 算法三参数选取实验结果

m 值（%）	总任务数	本地任务数	本地任务比	作业总时间（s）
1	4883	3706	0.759	595
10	4883	3682	0.754	545
20	4883	3653	0.748	528
30	4883	3618	0.741	517
40	4883	3511	0.719	511
50	4883	3373	0.691	502
60	4883	3143	0.644	503
80	4883	3005	0.615	500
100	4883	2988	0.612	498

当 m=1%时，该算法即为算法二，当 m=100%时，该算法即为贪心算法。

由实验可知当 m 取 20%~30%时，算法对本地任务比和作业总时间都有良好的效果。

3.4. 本章小结

本章通过对 MapReduce 进行建模，通过对各种任务调度算法进行分析，选择了较为适合的 Min-Min 算法，并对其进行改进以适应 MapReduce 架构。通过实验得出 Min-Min 算法的计算本地性和效率都十分理想。本章基于 Hadoop 的基础上，对 Hadoop 的调度参数进行增加和修改，并基于增加的参数对 Hadoop 任务调度提出三种算法，快速贪心算法和算法三通过实验都获得了较为优良的性能，下一章将进行硬件实验验证算法。

第4章 验证实验

4.1. 实验环境介绍

本次实验环境由实验室搭建的 Hadoop 集群和 Estinet 网络仿真环境共同组成。

4.1.1. Hadoop 集群安装

硬件环境：

虚拟机操作系统：Linux

Hadoop 集群中 Hadoop 代码版本号：Hadoop-1.0.1

Hadoop 集群由 5 台分布在不同服务器上的虚拟机（一个 namenode 节点，四个 datanode 节点）构成，每台虚拟机的 CPU 和内存参数将在具体实验中设置。

Hadoop 集群安装步骤如下：

1. 在各节点中建立统一用户名：hadoop
2. 将 Hadoop-1.0.1 代码解压缩到各个节点中。
3. 修改所有配置文件，将各主节点和计算节点进行申明，并创建 HDFS 存储输出和任务执行目录。
4. 修改所有节点中的 Hosts 文件中所有节点对应的 IP 地址，使得各节点能正常通信。
5. 修改 SSH，使得 Hadoop 集群运作中无需密码即可访问各节点。
6. 在 namenode 中 start-all.sh 启动所有 datanode 节点和 TaskTracker，利用 hadoop dfsadmin –report 命令检查各 datanode 节点存储数据情况，检验是否成功安装。

4.1.2. Estinet 仿真软件

Estinet 7.0 是一款由台湾思锐科技有限公司开发的通信网络仿真软件，它可以在实际的主机中运行应用程序并在仿真的网络环境上的虚拟主机和路由器上运行，它可以产生与实际网络中相近的流量模型。

Estinet 软件部署在另外一台主机的一个虚拟机上，由于必须所有节点都能够改变链路情况，所以将 Estinet 软件所在的虚拟机部署在所有节点传输数据的必经之路上。将五个 Hadoop 虚拟机通过修改 IP 设置成五个不同局域网，将 Estinet 部署的虚拟机多配置几个网卡，设置成这五个虚拟机的子网网关，并通过修改路由表，将五个 Hadoop 虚拟机节点间通信都经过 Estinet 部署的虚拟机，Estinet 仿真图如

图 6.1。然后通过 Estinet 可以修改任意两个节点之间的传播时延和带宽，如图 6.2 所示。

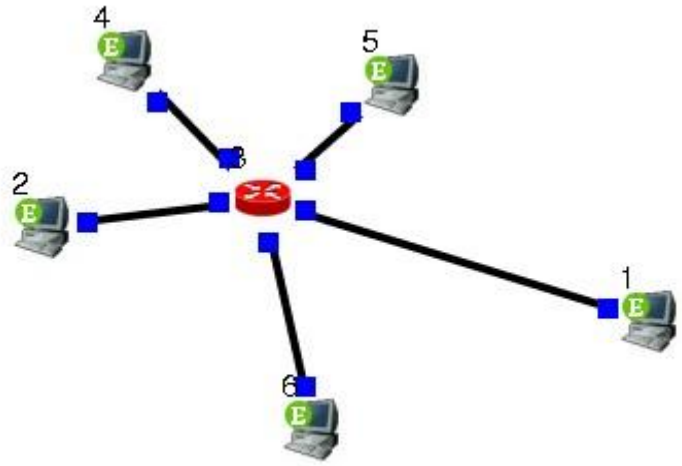

图 4.1 Estinet 仿真图

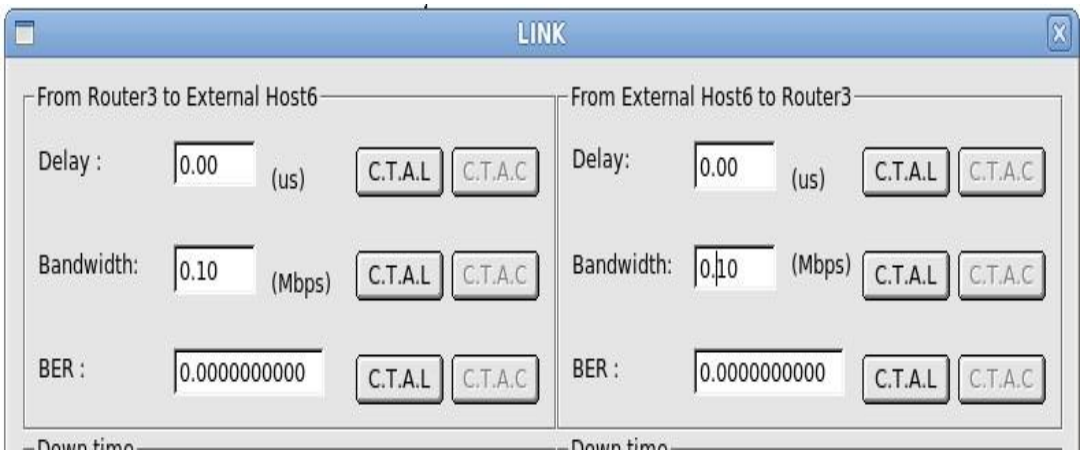

图 4.2 Estinet 修改网络参数图

4.2. 用于实验的 MapReduce 作业

本次用于实验的 MapReduce 作业采用最能体现 MapReduce 思想的词频统计作业，输入文件大小为 492MB，Splits 大小为 1M。

将文件按 Splits 大小分割成各个 splits，并将每个 splits 按行分割成<key，value> pair，key 值是存储该行首字符的偏移量，value 存储文本一行的数据。通过 Map 操作，生成一大串<word,number>的<key，value>，Map 部分实现代码如图 4.3。

```
public static class TokenizerMapper
     extends Mapper<Object, Text, Text, IntWritable>{

  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  public void map(Object key, Text value, Context context
                  ) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      context.write(word, one);
    }
  }
}
```

图 4.3 词频统计 Map 部分

并通过 Map 端排序和 combine 过程，对一个 Map 任务相同的 key 值进行合并，再将所有 Combine 输出结果排序后根据 Reduce 个数将结果分为若干个分片，每个 Reduce 读取自己所处理 Key 值范围读取相应的分片，并合并相同的 key 值，输出最后结果。 Reduce 部分代码如图 4.4。

```
public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

图 4.4 词频统计 Reduce 部分

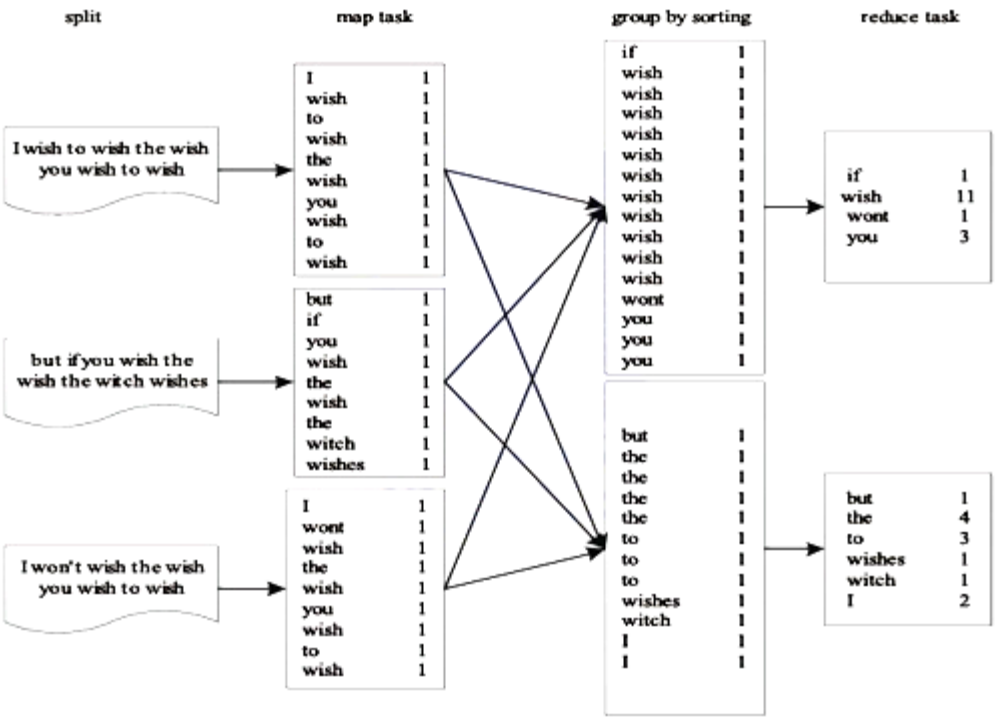

图 4.5 词频统计程序运行过程

4.3. 时延对 MapReduce 作业总完成时间影响实验

实验中参数如表 6-1，由于 Hadoop 对作业提交的数据利用 HDFS 进行负载均衡，较为平均的存储在各个 datanode 中。所有只有通过修改 CPU 和内存参数来修改节点的计算能力。

表 4.1 各节点 CPU、内存和存储数据表

	NameNode	DataNode-1	DataNode-2	DataNode-3	DataNode-4
CPU 内核总数	1	2	3	1	1
内存(GB)	1	2	3	1	1
数据（MB）	0	110	108	126	129

由于延时较大时，namenode 会认为 datanode 丢失，所以将任务大小改为 1MB，以便将任务计算时间和网络时延形成很好的比较。

分别用 estinet 对两两节点间添加时延和带宽参数，给各链路设计一个 ms 级别的传播时延，该时延对实验结果几乎没有影响，然后分别给各节点间设计相同的带宽，分别为 10MB/s,1MB/s,0.1MB/s。由于网络中传输的数据只有该 MapReduce

作业，所以实验结果无需通过多次实验取平均值来获得。

实验结果如表 6-2 所示。

表 4.2 Hadoop 在不同带宽环境下的调度结果

带宽 (MB/S)	非本地任务	本地任务	总时间（ms）
10MB	81	391	629120
1MB	67	405	678972
0.1MB	32	440	961810

由实验结果可知，每个任务的平均完成时间约为 4800ms，当带宽较大导致时延较之每个任务的平均完成时间过小时，时延变化对总作业完成时间并没有太大的影响，当带宽减小，时延增大到和每个任务的平均完成时间较为相近时，任务完成时间开始急剧增大，同时由于在非本地任务时间大幅提高后，本地任务增多。

4.4. Hadoop 自带任务调度器实验

当各虚拟机 CPU 和内存如表 6.3 所示时，在 1000ms 量级的时延下，实验结果如图 6.4 所示：

表 4.3 不同节点性能的实验结果

	非本地任务	本地任务	总时间（ms）
1000ms	41	431	1161810

为了改变计算节点性能，修改各虚拟机 CPU 和内存参数如表 6.3。

表 4.4 各节点 CPU、内存、数据存储信息

	NameNode	DataNode-1	DataNode-2	DataNode-3	DataNode-4
CPU 内核总数	1	1	1	1	1
内存(GB)	1	1	1	1	1
数据（MB）	0	110	108	126	129

在大时延下，即 1000ms 量级的时延下，实验结果如下：

表 4.5 不同节点性能的实验结果

	非本地任务	本地任务	总时间（ms）
1000ms	23	449	869980

两个实验结果比较可知，由于有节点计算能力突出，导致本地化程度下降，很多任务配给了计算能力较强的节点进行运算，然而，这反而导致总时间大幅增加。然而明显第一个实验中集群的计算能力较强，却产生了截然相反的结果，可以得出结论：Hadoop 自带的任务调度器在时延较大的情况下调度的效果不佳。

4.5. 设计的任务调度算法实验

第三章中已经为 Hadoop 调度器提出了三种改进算法，其中快速贪心算法和算法三通过仿真都得到了较好的性能，但是由于集群中 Hadoop 的 DataNode 节点数量有限，算法三的很难适应于这种小规模集群。本小节仅在 Hadoop 集群上对快速贪心算法进行实验。

将快速贪心算法写入 obtainNewMapTask 函数中，同时对原来调度函数中，对任务全部都在调度后进行备份调度的部分保留（防止有拖后腿的任务），通过修改配置文件将 split 大小设置为 1MB。实验中各节点参数如表 6.2 所示。

将 Estinet 仿真中每条链路进行带宽设计，对原调度器进行三次实验，为了保证各节点间带宽不相同，对每个节点间的带宽进行设计，带宽的量级分别为 10MB/S（节点到网关间带宽在 50~2MB/s 之间）,1MB/S（节点到网关带宽在 5~0.2 之间）,0.1MB/S（节点到网关带宽分别在 0.5~0.02 之间），在相同带宽环境下，对现在所用的调度器进行如上三次实验，实验结果如表 6.5。

表 4.6 快速贪心算法实验结果

调度算法	带宽 (MB/s)	非本地任务	本地任务	总时间（ms）
自带调度器	10	90	382	622776
自带调度器	1	73	399	670986
自带调度器	0.1	35	437	971293
新调度器	10	93	379	624168
新调度器	1	80	392	668812
新调度器	0.1	45	427	925380

通过以上实验结果，当带宽较大，各节点数据传输延时较小时，快速贪心算法的作业完成总时间和 Hadoop 自带调度器的作业完成总时间并没有多大差别。当带宽较小导致时延较大时，传输数据时间相对于计算时间大幅增加，快速贪心算法比之 Hadoop 自带的调度算法有着更快的任务完成时间和效率。但是由数据可知，

数据传输时间仅仅减少了 10%，可能的原因是由于 Estinet 构建的仿真图中，由于节点太少，节点间链接也过于简单，两两节点的带宽由于结构原因导致相同，同时节点过少导致了链路结构不够复杂。

4.6. 本章小结

本章在 Hadoop 集群中运行 MapReduce 作业，通过 estinet 给集群中两两节点加上时延，在时延较大的网络环境下，实验表明，Hadoop 任务调度器调度性能欠佳。将快速贪心算法加入 Hadoop 任务调度器中，计算本地化并没有下降，在时延较小时，新的调度算法并没有显示出较好的性能，当时延较大时，新的调度算法有很好的调度效果。

第5章 总结和展望

5.1 总结

随着互联网技术的不断发展，信息呈现爆炸式增长，很多应用都需要对大数据进行处理，这迫切的需要更加强大的计算能力，然而，在摩尔定律渐渐失效的今天，单一计算机的硬件发展开始减缓，单一计算机很难满足这种应用对计算能力的需求，利用集群来处理大规模数据的分布式计算模型就成为解决这种需求的唯一方法。

本文基于 MapReduce 模型上提出了一种改进的 Min-Min 算法，由仿真实验可知，这种算法很好的适用于 Map 任务计算量一样的 MapReduce 模型中。接着在 MapReduce 的开源实现 Hadoop 中，对 Hadoop 的任务调度器算法进行研究并改进，并基于 Hadoop 机制，在 Hadoop 中添加了时延参数和节点计算能力参数，并给出了在 Hadoop 集群的参数获得方法，同时根据添加的参数提出了两种典型的调度算法，一种是根据时延的快速贪心算法，一种是根据节点计算能力的自适应调度算法，第一种算法有较高的效率，但是计算本地化程度不高，第二种算法计算本地化程度较高，但调度效率较差。根据这两种算法的优缺点，对两种算法进行改进，并提出了算法三，该算法在调度效率和计算本地化上都有很好的效果。并通过实验对仿真中环境选择出较好的算法参数值。

为了验证大时延对 MapReduce 的影响和 Hadoop 本身调度算法不足以及新算法，我们使用了 Hadoop 集群和 estinet 软件仿真相结合的实验环境，用 Hadoop 集群执行 MapReduce 作业，用 estinet 仿真软件对节点间带宽进行限制，从而增大网络时延。通过大时延环境下的 MapReduce 实验，对比 Hadoop 原来的任务调度器，采用快速贪心算法的任务调度器有着较好的调度效率。

由于 Estinet 的修改时延本身有着较大的缺陷，无法模拟真正的大时延网络环境，同时 Hadoop 集群自带的 HDFS 存储机制使得实验中很难模拟任务数据的复杂分布式存储，而是根据负载进行一定策略的固定分配，也导致计算本地性在 Hadoop 集群中的效果要好于实际的 MapReduce 作业，以上都对实验结果有着较大影响。

5.2 展望

任务调度算法所解决的问题是一个 NP 完全问题，Hadoop 的开发者更关注于

当集群有多个 MapReduce 作业时如何保证各个作业间的合理调度，对 MapReduce 任务调度机制没有过多的设计。本文虽然对 Hadoop 的调度算法进行了优化设计，但由于 Hadoop 本身的机制所限，调度算法都是在线启发式算法。下一步工作是通过对 Hadoop 的任务调度机制进行修改，将 Min-Min 算法等批模式启发式算法加入到 Hadoop 调度器中，从而更好地优化调度性能。

图片索引

表格索引

参考文献

[1] 董征宇. 网格计算中任务调度算法研究[D]. 重庆大学, 2009.

[2] Berman F, Wolski R. The AppLeS project: A status report[C]//Proceedings of the 8th NEC Research Symposium. 1997, 16.

[3] 沈华,魏斐翡. ″网格资源计费模型的研究.″ 湖北工业大学学报 21.4 (2006).

[4] Basney J, Livny M, Tannenbaum T. Deploying a high throughput computing cluster[J]. High performance cluster computing, 1999, 1(5): 356-361.

[5] Litzkow M J, Livny M, Mutka M W. Condor-a hunter of idle workstations[C]//Distributed Computing Systems, 1988., 8th International Conference on. IEEE, 1988: 104-111.

[6] The Grid 2: Blueprint for a new computing infrastructure[M]. Morgan Kaufmann, 2003.

[7] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

[8] 罗红，慕德俊，邓志群等。网格计算中作业调度研究综述[J].计算机应用研究，2005，,2（5）：16-19

[9] 牛川川. 计算网格中任务调度算法和策略的研究 [D][D]. 南京: 南京理工大学, 2007.

[10] 刘鹏. 实战 Hadoop: 开启通向云计算的捷径[M]. 电子工业出版社, 2011.

[11]吴宝贵,丁振国.基于 Map/Reduce 的分布式搜索引擎研究[J].现代图书情报技术, 2007(8) : 52-55.

[12] Cohen, Jonathan. "Graph twiddling in a MapReduce world." Computing in Science & Engineering 11.4 (2009): 29-41.

[13] Bunch, Chris, Brian Drawert, and Matthew Norman. "Mapscale: a cloud environment for scientific computing." University of California, Computer Science Department, Tech. Rep (2009).

[14] Srirama, Satish Narayana, Pelle Jakovits, and Eero Vainikko. "Adapting scientific computing problems to clouds using MapReduce." Future Generation Computer Systems 28.1 (2012): 184-192.

[15]樊莎. 网格计算启发式任务调度算法的研究及在 GridSim 中的仿真. MS thesis. 西北大学, 2010.

[16]White T. Hadoop: The definitive guide[M]. O'Reilly Media, Inc., 2012.

致谢词

本论文是在我的导师董炜老师的耐心指导下完成的，董炜老师为此花费了大量的宝贵时间和精力并给予我很多指导，在此向导师表示衷心的感谢！导师高度的责任心和严谨的治学态度让我受益终生。

此外，曹军威老师常常对我的毕设进行指导，给予了我很大的帮助。万宇鑫博士给我的毕设提出了很多可行性建议，并指出我实验中的误区，让我能及时发现问题并顺利完成毕业设计。

还要感谢实验室的师兄师姐以及一同做毕业设计的几位同学共同营造了这个良好的毕设环境，是你们在我平时设计中和我一起探讨问题，给予了我很多启发。

没有以上老师和同学的帮助我不可能这样顺利地完成毕设，在此表示深深的谢意。

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：邹睿　　日　期：2013.6.24

附 录 A 外文资料书面翻译

适用于科学云计算问题的 MapReduce 框架

摘要：

云计算,它承诺几乎无限的资源,似乎相当适合解决需要大量资源的科学计算问题。为了研究这项内容，我们在我们内部的集群上建立了科学云计算 (SciCloud)项目和环境。项目的主要目标是研究在大学范围内建立私有云。通过这些云，学生和研究人员能够有效地使用大学计算机网络的现有资源去解决计算密集型科学,数学和学术问题。然而,要能够在云基础设施运行科学计算应用程序，应用程序必须降低到可以成功的利用云资源的框架，像 MapReduce 框架。本文总结了在 MapReduce 模型中关于换算迭代算法相关的挑战。科学计算的算法通过依赖 MapReduce 模型分为不同的类；在 MapReduce 模型上对每一个类性能进行测量和分析。这项研究主要侧重于研究 Hadoop MapReduce 框架并把它和一个替代 MapReduce 框架——Twister 相比,Twister 是专们为迭代算法设计的。分析表明,Hadoop MapReduce 在迭代问题方面具有显著的麻烦,但它适合高度平行问题,Twister 可以更有效率地处理迭代问题。这工作展示了在不同的问题类型中，适用于 MapReduce 模型的算法怎样影响了效率和可伸缩性,并允许我们通过比较两个框架的优缺点来判断哪个框架是更有效的。本研究是具有重要意义的科学计算问题,因为这经常不是一个轻松的任务去使用复杂的迭代方法来解决关键问题。

1.Introduction 介绍

科学计算是一个应用计算机科学解决典型科学问题的研究领域。它不应该被误解为只是计算机科学。科学计算通常是相关联的大型计算机建模和仿真,并且通常需要大量的计算机资源。 云计算[1]很适合解决这些科学计算问题,其承诺提供几乎无限的资源。

在适应资源密集型的应用程序的云,应用程序必须降低到可以成功地利用云资源的框架,这正是我们正在研究的云项目的科学计算方法。 一般来说,云基础设施是基于效率不高和经常注定失败（网络故障和硬件故障导致的失败）的商业电脑。

软件总是失败这可能会导致严重的问题。当在一个分布式系统中面临硬件或网络故障的时候，最好的办法通常是复制重要数据和重试失败的计算。也有分布式计算框架,提供容错机制， MapReduce 框架[3] 就是这样的框架。

MapReduce 作为在大量计算机上执行分布式计算的并行计算框架是由谷歌首次开发。自那时以来,它已经作为一个可执行自动可伸缩的分布式应用程序的云计算框架得以普及。谷歌 MapReduce 实现是专有的,这也就导致开源同行像 Hadoop MapReduce[4]的开发。Hadoop 是一个基于谷歌的 MapReduce 和谷歌文件系统[5](GFS)的 Java 软件框架。Hadoop 项目正在被 Apache 积极研制开发并且在商业和搜索方面被广泛应用,使得它有一个庞大的用户基础和足够多的作业。

MapReduce 应用程序的结构非常严格，同时在处理分布式应用程序其自动可伸缩性是很有吸引力的。对于 MapReduce 模型减小算法复杂度也不是轻而易举的事情并且并没有保证由此生成的 MapReduce 算法是有效的。先前的工作已经证明,MapReduce 非常适合简单和通常高度平行的问题。谷歌在他们的论文中展示,他们使用[3]MapReduce 对于各种各样的问题,如大规模索引、图计算、机器学习和从一个巨大的组索引网页提取特定的数据。其他相关工作[6]表明,MapReduce 可以成功地用于图问题,如发现图形组件,质心聚类、枚举矩形和三角形列举。MapReduce 也进行过科学问题[7]。它在简单问题中表现良好，像组极方法产生随机变量和整数排序。

然而,MapReduce 也被表示有很大问题在更复杂的算法,如共轭梯度,快速傅里叶变换和块三对角线性系统求解。此外,这些问题大多是用迭代的方法来解决这些问题,表明 MapReduce 可能不适合算法,迭代性质。然而,这里有超过一种类型的迭代算法。去研究如果 MapReduce 模型不适合所有迭代算法或者只是其中一个特定的子集,我们设计了一组类科学算法。算法根据他们如何难适应他们 MapReduce 模型和由此引起的结构来分类。为了比较这些分类，我们从每个分类选择算法去适应 MapReduce 模型,研究他们的效率和可伸缩性。这样的分类可以让我们准确判断哪些算法更容易适应 MapReduce 模型和对并行效率和可伸缩性的自适应算法，什么样的效应属于一个特定的分类。

接下来的文章组织如下。第二节简要介绍了 SciCloud 项目。第三节描述了在我们的云计算的基础设施上的 Hadoop MapReduce 模型和第四节描述了不同分类迭代算法。第五部分概述了算法实现和分析。第六节描述了一个替代 MapReduce 框架称为 Twister 和生产分析的算法框架。第七节提到相关工作和第 8 部分总结了论文,描述了在 SciCloud 项目中未来的研究方向。

2.科学云计算

科学云计算(SciCloud)项目的主要目标[2]是研究在大学范围内建立私有云。这些云、学生和研究人员在解决计算密集型科学,数学和学术问题上能够有效地使用现有的大学计算机网络资源。传统来说,这样的计算密集型问题针对性的面向批处理的模型的网格计算领域。SciCloud 试图用更多互动和面向服务的云计算模型实现去实现,这适合较大的应用程序。它的目标是开发框架,其中包括模型和为建立适当的选择、状态和数据管理的方法,自动伸缩功能和互操作性的私有云。一旦这样的云是可行的,可以用它们来为大学中有兴趣的团体之间的合作和在内部测试的试用、创新和社交网络提供更好的平台。SciCloud 也侧重于寻找新的分布式计算算法和试图在 MapReduce 算法中减少一些科学计算问题。

在市场上虽然有几个公共云,Google Apps(包括谷歌邮件、文档、网站、日历等)、Google App Engine[8](有限的提供一个可供 Java、Python 的弹性平台应用)和 Amazon EC2[9]可能是最众所周知和广泛应用的。Amazon EC2 是一种在操作系统上允许完全控制虚拟机。可以从许多可用的 Amazon Machine Images (AMI)和几个可能的虚拟机中选择一个合适的操作系统和平台(32 位和 64 位)。这一点不同于 CPU、内存和磁盘空间。此功能允许我们对于任何特定的任务自由选择合适的技术。对于 EC2, 服务价格取决于机器的大小,它的正常运行时间,以及使用云的中多少带宽。

这里也有几个自由实现的云基础结构如：Eucalyptus[10]。Eucalyptus 允许用 Amazon EC2 创建私有云。因此,云计算应用程序最初可以在私有云中开发,稍后可以推广到公共云。这是对于研究和学术有着巨大的帮助,作为实验的初始费用可以很大程度上降低了。这个主要目标我们已经建立了 SciCloud 集群上 8 节点组成的 SUN FireServer Blade 系统与 2-coreAMDOpteron 处理器,使用 Eucalyptus 技术。集群后来延长 2 个节点采用双四核处理器和 32 GB 的内存,每个节点用单一的四核处理器加上 4 个节点, 每个节点 8GB 的内存。

虽然一些应用程序是明显来自与这样一个私有云的设置,我们使用它在分布式计算和移动 web 服务域[2]上解决我们的一些研究问题。在移动 web 服务领域,我们尽可能在蜂窝网络扩展我们的 Mobile Ennterprise[11]的负载。一个 Mobile Ennterprise 可以通过参与 Mobile Hosts 建立一个手机网络,后者充当对智能手机和他们的客户 web 服务提供者。Mobile Hosts 通过遵守以下 web 服务标准[12]启用用于企业的特定用户服务的无缝集成, 同时也会用于收音机链接,通过资源约

束的智能手机中。几个应用程序通过 Mobile Host 被开发并展示了卫生保健系统,协同移动、社交网络和多媒体服务域[11]。我们改变一些 Mobile Enterprise 关于 SciCloud 的元件和负载平衡器并证明了移动 Web 服务中介框架[13]和组件是水平伸缩的。更详细的分析在[14]。除了以上在我们的研究中帮助我们,SciCloud 也有几个在数据挖掘、生物信息学领域的想法支持着我们的研究。

3.科学云计算 Hadoop 框架

为了有一个试验基于 MapReduce 应用的地方,我们已经建立了一个动态可配置 SciCloud Hadoop 框架。我们使用 Hadoop 集群去解决一些科学计算问题,如 CG MapReduce 算法。细节将在本节中被解决。

MapReduce 是一种编程模型和分布式计算框架。这是由谷歌率先开发出来用于处理非常大量日常生活中的每天都在增长的原始数据,如索引文件和 web 请求网络日志。谷歌使用 MapReduce 在数百或数千个普通计算机上处理数据。MapReduce 应用程序获取键值对列表作为输入,并由两个主要函数。Map 和 Reduce。Map 对输入列表中分开处理每个 key-value pair,并输出一个或多个 key-value pairs 结果。

map(key, value) ⇒ [(key, value)].

Reduce 函数聚合 Map 函数的输出。它得到一个 key 值和一个所有被分配给这个 key 值的一个列表所有值作为输入,执行用户定义的聚合并输出一个或多个 key-value pairs 值。

reduce(key, [value]) ⇒ [(key, value)].

用户只需要生产这两种方法来定义一个 MapReduce 应用程序;框架负责一切,包括数据分布、通信、同步和容错。这使得用 MapReduce 编写分布式应用程序更加容易,因为框架允许用户集中在算法和而它自己能够处理其他一切。并行化在 MapReduce 框架是在并行集群中通过在不同的机器上执行多个 Map 和 Reduce 任务。然而,谷歌的 MapReduce 实现是专有的。Apache Hadoop[4]是一个用 Java 编写的开源的 MapReduce 实现。除了 MapReduce,Hadoop 还提供 Hadoop 分布式文件系统[15](HDFS)可靠地在成百上千的计算机中存储数据。HDFS 是近似并基于谷歌文件系统(GFS)[5]。Hadoop MapReduce 框架用一种分布式样式使用 HDFS 既存储 MapReduce 应用程序的输入和输出。一个简单的 Hadoop 集群包含 n≥1 台机器运行 Hadoop 软件。集群是一个单一的主集群与不同数量的从节点。从节点可以同时作

为 MapReduce 的计算节点和 HDFS 的数据节点。Apache Hadoop 正在被大力发展用于商业和研究。

分析了 MapReduce 科学计算的性能,我们建立了一个小的 SciCloud Hadoop 集群。集群是由一个主节点和十六个从节点组成。只有从节点充当 MapReduce 任务节点,可以 16 平行 MapReduce 任务可以执行。每个节点都是一个虚拟机与 2.2 GHz 处理器,500 MB 的 RAM 和 10 GB 磁盘空间分配给 HDFS,使的总大小 HDFS 160 GB。更多的节点可以被添加到集群动态,当需要。我们的脚本,可以添加更多的从节点到框架和可以配置到主节点。我们也支持自动伸缩功能,这个功能能基于观察到大量的单个实例一开始就具备更多的从节点。细节将会在我们未来出版物上刊登。

4.算法类

我们已经设计了一组科学算法的类基于他们如何难适应 MapReduce 模型和需要哪些步骤。算法分为不同的类如下:

•可以适应作为一个单一的 MapReduce 模型执行的算法。

•可以适应作为一个顺序的固定数量 MapReduce 模型执行的算法。

•在一个迭代的内容都被表示为一个单一的 MapReduce 模型执行的算法。

•在一个迭代的内容都被表示为一个多个 MapReduce 模型执行的算法

第一个类可以被认为代表高度并行算法，第二种则被认为是简单的并行算法。第三和第四个代表迭代算法,一些类型的同步必须在每次迭代之间执行;例如检查结束条件或总计和广播上一次迭代的结果。第四类算法被认为是更复杂的迭代算法,在每个迭代中只有一些操作可以完全并行化。算法属于第 4 类通常很难有效地并行化,这是更难以实现适应他们 MapReduce 模型。研究属于一个特定的类是如何影响效率和可伸缩性的算法,我们从每个类 MapReduce 选择算法和分析结果。我们选择的算法将在下面章节介绍。

除了属于特定的类,并行效率和可伸缩性也受算法特征的影响。例如,它取决有多少在 MapReduce 模型外的计算。当它不能完成在 MapReduce 的 reduce 上，用迭代算法或聚合检查结束条件和处理最终结果,意味着经常有些部分的迭代算法必须在模型外执行并行性,从而降低了整个算法的并行效率。同时，属于第二个类的算法会变得不那么有效,如果MapReduce模型执行的数据很大。在不同的MapReduce模型间切换作为同步步骤，输入数据为每个不同的 MapReduce 的执行必须再次进

行处理,这意味着在第二和第三类当第二类步骤的数量比得上第三类迭代的次数时，可能没有实际的区别，。

此外,并行效率不仅取决于该算法对 MapReduce 模型的适应或继承特征的算法本身,还取决于执行环境。在不同的 MapReduce 框架中执行 MapReduce 应用程序可以在应用程序的运行时间上产生重大影响,还取决于此应用程序中运用的算法属于哪一类。

5.在 MapReduce 中减少迭代算法

我们从第 3 节谈到的每个算法类选择了一个算法来说明不同的设计选择和问题，在 Hadoop MapReduce 框架中出现的科学计算问题的。这些算法是:

•共轭梯度(CG)。

•两个不同的 k-medoid 聚类算法:

——分区在 Medoids(PAM)。

——聚类大型应用程序(CLARA)。

•整数的因子分解。

CG 属于第 4 类,PAM 属于第三类,CLARA 属于第二类,和整数因子分解是第一类的例子和高度并行算法。对于这里的每个算法，我们提供一个简短的描述、指出为什么他们属于给定的类的原因,去适应 MapReduce 模型的步骤和实验结果。

5.1 共轭梯度(CG)

共轭梯度法(CG)是一种求解矩阵形式的线性方程组的迭代算法：

Ax = b

在这个线性系统中，A 是一个已知的矩阵，b 是一个已知的向量，x 是向量的解。CG 的总体想法是起始一个不准确的解 x,然后通过不断的迭代提高其解的准确性。CG 是一个相对复杂的算法,不太可能直接用整个算法构建 MapReduce 模型。相反，使用 CG 的矩阵和向量操作在每个迭代都简化成 MapReduce 模型。因为这，它直接属于第四类算法。

•矩阵向量乘法。

•点积。

•两个向量加法。

•矢量和标量乘法。

Table 1
Run times for the CG implementation in MapReduce under varying cluster size.

Unknowns	24	500	1000	2000	4000	6000	8000
1 node	259	261	327	687	1938	3810	7619
2 nodes	255	259	298	507	1268	2495	4185
4 nodes	255	236	281	360	721	1374	2193
8 nodes	251	251	291	397	563	824	1246
16 nodes	236	240	278	297	338	511	809

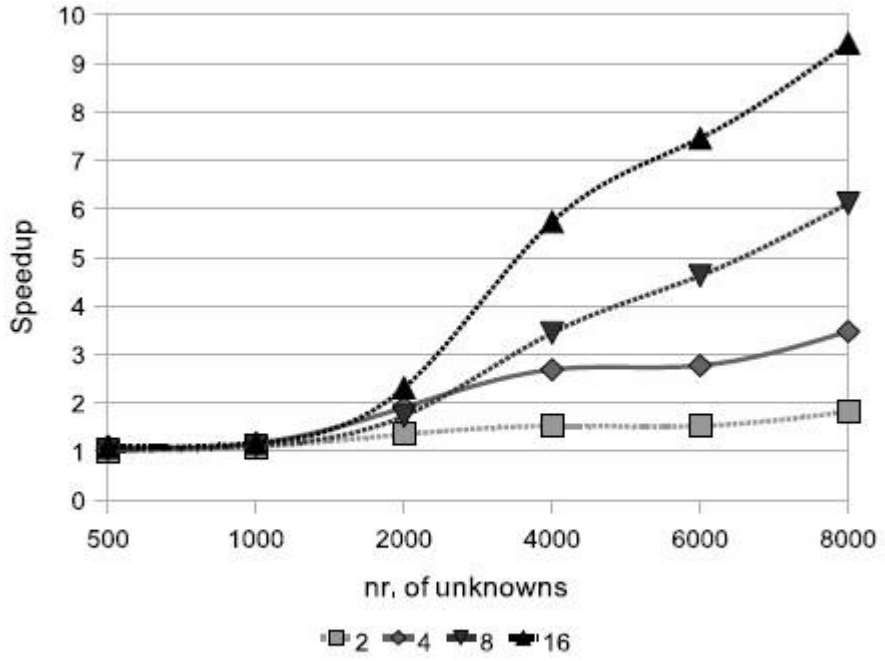

Fig. 1. Speedup for Conjugate Gradient algorithm with different number of nodes.

这些操作中的一个用于 CG,一个新的 MapReduce 工作就在运行。导致在每一次迭代时多个 MapReduce 工作在运行。这不是最有效的方式,因为 Hadoop 框架需要时间去安排,启动和完成，它可以被视为在 MapReduce 工作中每个迭代的延迟和执行多个作业构成了一个显著的开销。

此外,在 Hadoop, 矩阵 A 存储在 HDFS 并且被当做每一个迭代中输入的矩阵向量进行乘法操作。在 Hadoop MapReduce 框架中在不同的运行的 MapReduce 任务下，缓存输入是不可能,所以每次执行这个操作,输入必须再次从文件系统中读取。作

为一个矩阵的值在迭代之间永远不会改变，在每一个迭代重复同样的工作。这意味着一个重要的额外开销。实验为了能够并行加速计算运行在不同数量的并行节点。并行加速措施并行执行多少次是在一个节点上运行相同的 MapReduce 算法速度比。如果它是大于 1,这意味着至少有一些获得并行任务。这等于加速节点的数量被认为是理想的,意味着算法有一个完美的可伸缩性。CG 算法的运行时间都显示在表 1，计算速度显示在图 1。

它花了 220 秒去解决系统在只有 24 个未知的事物在 16 个节点的集群,这在这样一个小数量的计算下，无疑是非常缓慢的求解线性系统。不幸的是,这些解决更大的系统的测试也显示：CG MapReduce 算法并不能随着数据大小的增加而改善。例如,一个 8000 未知事物的线性系统,使用 MapReduce 算法花了近 2 小时解决。这些结果表明,大部分的时间都花在 MapReduce CG 的后台任务,而不是实际的计算中。

5.2 分割环绕物件法

分割环绕物件法[17](PAM)是一个迭代 kmedoid 聚类算法，在这领域具有重要的价值。k-medoid 聚类的一般的想法是,每个集群被最核心的元素所代表,medoid 和在所有对象和集群之间的比较减少到对象和集群中 medoids 的比较。

为了聚集一组对象到 k 不同集群,PAM 算法首先选择 k 随机对象作为初始 medoids。在第二步,对于每个在数据集的对象,距离每个 k medoids 都被计算并且对象都被分配到与 medoid 最接近的集群。因此,数据集分为 k 不同集群。在下一步的，PAM 算法重新计算每个 medoid 的位置,选择最中央对象作为新 medoid。这个将对象划分到集群,并重新计算集群 medoid 的位置的过程被重复,直到与之前的迭代结果没有变化,这才意味着集群变得稳定。

类似于 CG,PAM 制定了初步解决问题的方法,在这种情况下,聚类,在每一个迭代后,提高了解决方案的准确性。同时,和 CG 一样,（PAM）不可能将整个算法用于 MapReduce 模型。然而,整个迭代的内容可以概括为 MapReduce 模型,表明 PAM 算法属于第三类。由此产生的 MapReduce 工作可以表示为:

•Map:

——找到最接近的 medoid 和在其集群分配对象,。

——输入:(集群 id、对象)。

——输出:(新集群 id、对象)。

•Reduce:

——找到哪个对象是最中央,把它指定为集群中一个新的 medoid。

——输入:(集群 id(所有集群对象列表))。

——输出:(集群 id、新 medoid)。

Map 函数重新计算每个对象属于哪个集群,Reduce 函数为每个生成的集群找到一个新中心。MapReduce 任务重复执行直到集群中 Medoid 位置的不再变化。类似于 CG, 因为在每个时间都有一个新的 MapReduce 作业执行，PAM 也存在工作滞后和在每一个迭代从文件系统重读输入的问题。

5.3 聚类大型应用程序

聚类大型应用程序[17](CLARA)也是一个迭代 k-medoid 聚类算法,但是相比 PAM,它只集群数据集的随机子集的小找到整个数据集 medoids 候选人。这个过程被重复多次，最好的 medoids 组候选人被选择作为最终结果。不同于 PAM,迭代的结果是相互独立的,并且不需要执行在一个序列上。

因此,同时在不同的任务中执行内容的和消除迭代结构的迭代算法是可能的。无论是执行多少不同的任务，一切都可以减少到两个不同的 MapReduce 工作。第一步工作是从输入数据集选择一定数量的随机子集, 同时使用 PAM 的集群和输出结果。第二个 MapReduce 工作是计算第一个任务的每个结果的质量检测,通过在整个数据集的一个并发的 MapReduce 任务中检查他们。由于只有两个 MapReduce 工作,工作延迟保持最小并且输入数据集只读两次。这两个 MapReduce 工作概述如下:

第一个 CLARA MapReduce 任务:

•Map:

——分配给每个对象一个随机的 key 值。

——输入:(key、对象)。

——输出:(随机 key,对象)。

•Reduce:

——读前 n 的对象,这是升序排序的关键。因为钥匙被随机指定,在排序后对象的顺序是随机的。在 n 个对象上运行 PAM 的聚类找到 k 不同候选 medoids。

——输入:(key,对象的列表)。

——输出:(关键、k medoids 的列表)。

第二个 CLARA MapReduce 任务:

•Map:

——每个对象,找到最接近的距离 medoid 并计算它们的距离。对于每一个对象,有多少候选集 medoids 就进行多少次操作,并且为每个对象生成一个输出。

——输入:(集群、对象)。

——输出:(候选集合 id, 最近的 medoid 的距离)

[每个候选集都有一个输出]。

•Reduce:

-用相同的候选集合 id 计算距离之和。

——输入:(候选集合 id,距离的列表)。

——输出:(候选集合 id、总数(距离的列表))。

Table 2
Run times for the PAM algorithm.

Objects	10 000	25 000	50 000	75 000	100 000
1 node	1389	1347	2014	3620	6959
2 nodes	1133	1697	1826	2011	6130
4 nodes	803	782	1156	2562	2563
8 nodes	635	627	1513	1084	1851
16 nodes	297	497	432	761	1029

Table 3
Run times for the CLARA algorithm.

Objects (thousands)	25	50	100	500	1000	5000	10 000
1 node	117	118	125	183	261	819	1517
2 nodes	79	84	89	150	215	476	832
4 nodes	61	66	72	120	127	316	486
8 nodes	52	56	61	114	124	218	320
16 nodes	44	50	58	99	98	104	156

第二个任务的结果是一个列表的计算总数,每个代表和所有对象距离他们最近的medoids的总和,一个用于每个候选集。对象之间最小的距离和的候选组medoids并且他们最近的 medoids 作为最好的聚类。MapReduce 任务被执行的数量始终是两个,这个算法属于第二类。

从实验结果(表 2、3 和图 2、3)可以看到,CLARA MapReduce 算法速度远远超过 PAM,尤其是当数据集中对象的数量增加的时候。PAM 是不能够处理数据集大于 100000 对象而 CLARA 可以处理包含数百万甚至数千万对象的集群集。还应指出的是,最小数据集时间对于 CLARA 和 PAM 已经相当大了。这是因为 MapReduce 框架的后台任务开始的相对缓慢,所以每个单独的 MapReduce 任务开始减慢了算法。这对

PAM 的影响比对 CLARA 大得多，因为 PAM 中有太多了的 MapReduce 迭代、,CLARA 只使用两个 MapReduce 任务。

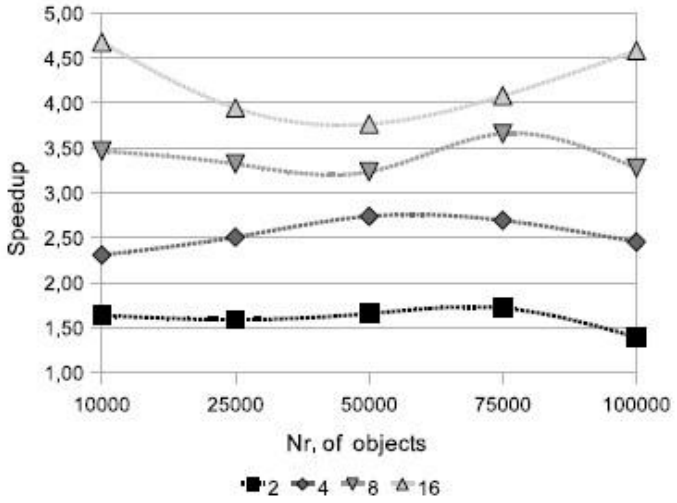

Fig. 2. Parallel speedup for PAM with different numbers of nodes.

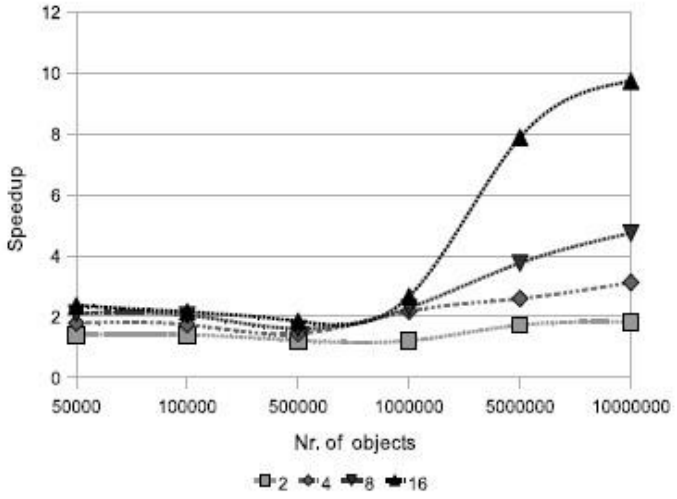

Fig. 3. Parallel speedup for the CLARA algorithm with different number of nodes.

5.4.整数的因子分解

整数的因子分解是一种将整数划分成一组相乘得到原来整数的素数的方法。例如一个数 21 的因子是 3 和 7。整数的因子分解被使用与例如打破 RSA 密码体制。

在这种情况下我们选择整数的因子分解的最基本的方法，试除法。这个方法不

用于实践,因为它相对缓慢，并且存在更快方法像一般数域筛网[18]。但我们选择这个方法纯粹是为了说明它适应一个高度平行问题，相较于其他三个算法，它属于 MapReduce 模型的第一类。

用试除法去分解一个整数,这个整数所有可能的分解都会被检查是否均匀划分。如果每个都通过检查,那么它是一个分解。这可以采取 MapReduce 模型,通过划分多个可能的分解在不同的分组中,并同时对每个分组用单独的 Map 或 Reduce 任务检查:

•Map:

——得到一个数字被分解作为输入,发现数量的平方根并从 2 到数字的平方根分为 n 个更小的范围,输出每一个。

——输入:(key、数字)。

——输出:(id,(开始、结束、数字))(每个范围的一个输出,总共 n 个]。

•Reduce:

——得到在哪里验证分解一个数字和一个范围， ,作为输入,并发现如果任何数字在这个范围均匀划分数字。

——输入:(id,(开始、结束、数字))。

——输出:(id、因素)。

因此,不同于先前的算法,这个算法是减少到一个单一的 MapReduce 任务,这意味着在序列运行多个任务没有开销，和为什么该算法属于第一个算法类。

Table 4
Run times for the integer factorization with different numbers of nodes.

Digits	17	18	19	20	21
1 node	51	142	361	2058	6767
2 nodes	37	67	188	1117	3271
4 nodes	30	50	120	512	1622
8 nodes	27	36	70	299	887
16 nodes	27	38	59	215	566

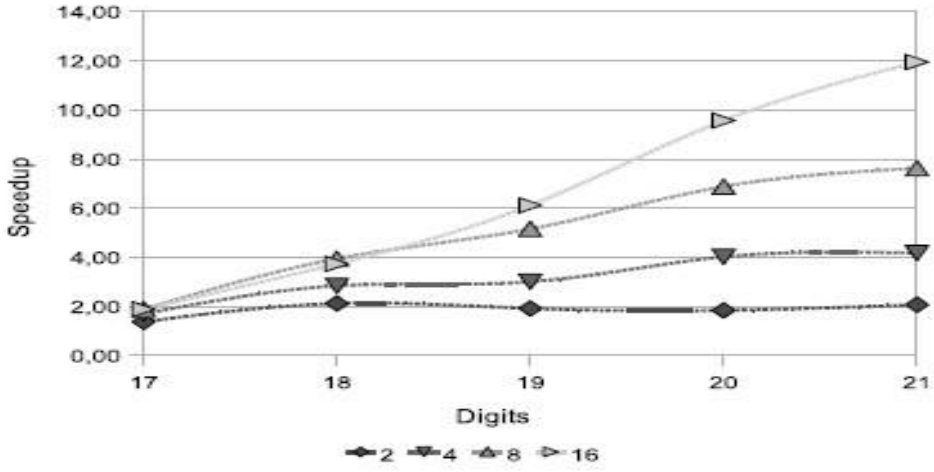

Fig. 4. Parallel speedup for the integer factorization with different numbers of nodes.

整数分解的运行时间在表 4 中给出，速度在图 4 显示。从图 4 可以看到,当因子分解的数字很小的时候, 并行使用多个 worker 只有很小的优势。相对于 2 节点集群加速比略高于 1,只有在 16 个节点集群才达到 2.22。这是因为计算数量相对框架的后台任务更小。然而,随着输入数字规模的增大,加速开始显著增长。当有 21 位的输入时候,两个和四个节点执行的速度是 2.07 和 4.17,表明使用多个节点有一个理想的结果发现分解的大小什么时候输入足够大。用大量的节点增长的速度达不到节点增长数量,表明计算不够长去对于使用 16 个节点得到完整的收益。计算的增长速度数据表明,这种算法具有良好的可伸缩性,算法属于第一个类可以非常适合 Hadoop MapReduce 框架。

5.5. 总结分析

从我们的实验结果中,我们发现Hadoop MapReduce在迭代算法中有几个问题。第 4 类复杂的迭代算法在每一次迭代中可能需要一个或多个执行 MapReduce 任务。然而,如果迭代次数过大,然后在一个序列中执行许多 MapReduce 任务，因为工作

延迟而导致低效率。任务的延迟时间就是 MapReduce 框架的安排、开始和结束 MapReduce 工作,不包括花在做实际的计算上的时间。

此外, Hadoop MapReduce 任务的输入到存储在 HDFS,HDFS 的数据是按分布式方式保存得。每次执行 MapReduce 任务时都要从读这个输入,即使很大一部分的输入在任务运行之间的是不会改变,因为它不可能在 Hadoop MapReduce 框架缓存输入。对于如 CG 和 PAM 算法,大量的输入数据保持不变,这意味从文件系统中很多次读入相同的输入数据, 在每一个迭代做重复的工作和最后导致结果效率降低。

对于算法类 3 和 4, 在每个迭代中一个或多个 MapReduce 任务运行,工作延迟,和无法缓存 MapReduce 应用程序输入可以显著增加运行时间。这意味着大量的时间都花在 MapReduce 框架的后台任务管理上,和更少的时间花在执行实际的计算。当有大量的迭代时，这大大影响了迭代算法。

不管遇到的问题,所有实现算法都能够利用多个节点实现加速,如图 5 所知,在我们的测试中 RSA 有最好的加速和 PAM 有最坏加速。然而,很难充分比较不同算法之间的加速数据,因为他们强烈地依赖于算法特点,输入尺寸,计算的时间和后台任务等。对于需要更多后台任务的迭代 MapReduce 算法,达到一个理想的加速是更加困难的。同时增加问题的大小,因此花在实际的计算时间,通常改善结果,也增加了花在后台任务的时间,增加输入大小意味着在每一个迭代有更多的时间花在阅读来自文件系统的输入数据上。

6.Twister MapReduce 框架

在认识了 Hadoop 在每个算法类的问题后，我们对与其他 MapReduce 框架的比较结果非常有兴趣，去判断哪个是 Hadoop 框架本身的问题和一般 MapReduce 固有的问题。去看看什么影响了在每个算法类中 MapReduce 框架实现在算法效率和可伸缩性的方面的选择。

Table 5
Run times for the CG implementation in Twister.

Unknowns	500	1000	2000	4000	6000	8000	10 000	20 000
1 node	3.19	3.40	3.00	4.96	7.69	11.27	16.22	56.01
2 nodes	3.33	3.40	2.82	3.99	5.72	6.98	9.51	28.15
4 nodes	3.27	3.29	2.76	3.54	4.03	5.29	6.54	16.33
8 nodes	3.38	3.30	2.81	3.56	3.79	4.76	5.44	14.75
16 nodes	3.40	3.42	2.75	3.50	3.56	4.11	4.86	10.05

Table 6
Run times for the PAM algorithm in Twister.

Objects	10 000	25 000	50 000	75 000	100 000	200 000	300 000
1 node	5.45	20.55	25.00	96.61	204.55	638.56	1888.71
2 nodes	2.93	10.06	22.85	51.19	93.06	359.88	808.96
4 nodes	3.91	7.99	14.63	15.51	91.78	197.15	343.64
8 nodes	4.04	4.93	15.11	31.84	38.13	131.41	355.77
16 nodes	4.25	6.63	11.55	22.26	24.87	85.76	237.43

Wechose Twister[19]作为替代 MapReduce 框架,因为它被宣传成迭代 MapReduce 框架,因此它应该在 Hadoop 中对于 Hadoop 已经显示出了麻烦的算法类 3 和 4 提供一个更好的方案。Twister MapReduce 框架区分迭代过程中不会改变的静态数据和在每一个迭代中可能会改变的正常数据。它还提供了长期运行 Map 和 Reduce 任务而这些任务作为 Hadoop 中必须的,不需要在 MapReduce 运行迭代之间被终止。这两个是主要的特点,单独从 Hadoop 中分离 Twister 和提供迭代算法更好的支持。Jaliya Ekanayake et al。[20]相比 Hadoop MapReduce, Twister 和的 MPI 在不同的数据和计算中加强应用。他们的研究结果表明,Twister 可以大大减少 MapReduce 应用程序迭代的开销。

我们决定测试类 3 和 4 的 Twister MapReduce 框架,看看 Twister 是否可以更快的管理迭代算法。我们在 SciCloud 建立了一个小的 Twister 集群。集群是一直于 Hadoop 集群非常类似去为了更精确的比较。它是由一个主节点和十五从节点。主节点和从节点都充当 MapReduce 任务节点,结果就是 MapReduce 任务可以在 16 个平行节点上执行。每个节点都是一个 2.2 GHz 处理器,500 MBRAM 的虚拟机。Twister 没有分布式文件系统,和所有输入文件仅仅是分配到本地硬盘的节点。

我们在 Twister 中实现的算法是 CG 和 PAM,代表了类 3 和 4。表 5 和图 6 显示了这些实验的运行时间。6 和 7 展示了计算速度。

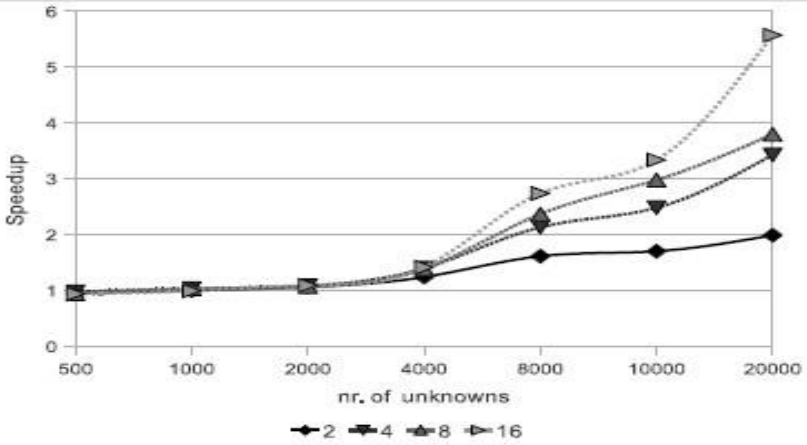

Fig. 6. Parallel speedup for CG in Twister.

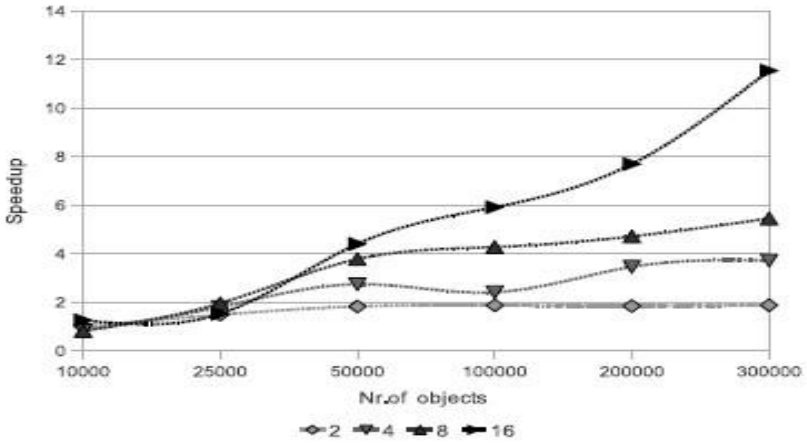

Fig. 7. Parallel speedup for PAM in Twister.

比较了 Twister 和 Hadoop(表 1 和 2)对这些算法运行时间清楚地表明 Twister 对于类 3 和 4 是更高效。Twister 在相同大小的问题上可以用更少的时间去解决更大的问题并且当有 16 个节点的时候，这是在 Hadoop 上运行时的 50 - 100 倍的速度,。算法结构的方式是适应 MapReduce 模型的，在 Twister 和 Hadoop 中是保持完全相同的,然而 Twister 更高效处理 MapReduce 应用程序迭代后台任务。在 Hadoop 工作每次迭代滞后约 19 - 20s,而在 Twister 中不管它迭代的次数都是低于 3 s。Twister 还存储大量的输入到内存,不需要在每一个迭代从文件系统读一遍。在 CG 中,这意味着能够在集体记忆的集群和 inPAMit 中存储整个矩阵和意味着能够存储所有的集群对象。

然而,Twister 也在分布式应用程序具有一定的局限性。使用 Twister 的明显

的优势在它跨迭代在内存中保持静态输入数据的能力。但它意味着,这种静态的数据必须在 Twister 机器配置融入集体记忆。在数据密集型任务中这可能是一个相当不合理的需求。例如,用 twister 处理 1 TB 的数据将需要超过 128 台机器与每个 8 GB 内存作为只是存储数据到内存中,更不用说在应用程序的其余部分中需要的内存,框架本身和他运行的操作系统。Twister 还没有一个能和 Hadoop 提供的容错机制相比的合适的容错机制,这在运行在一个公共云是一个非常严重的问题,Twister 机器容易相对频繁出现问题。

比较 Twister 和 Hadoop 的属于第三和第四类的算法已经表明,Twister 更适合这些类。同时, 由于 Hadoop 提供的容错,Hadoop 更适合第一个类的算法以及在一般的数据密集型算法,Twister 在拟合数据到内存中有问题。结果不是太明确对于减少数据密集型算法属于第二类。当不同的 MapReduce 程序的数量并不大时,Hadoop 可以执行良好并且考虑到容错,它应该被认为是更合适的。但因为在 Hadoop 中短时间运行的任务, 每次一 MapReduce 周期结束都被终止,当随着 MapReduce 任务增加，Hadoop 渐渐失去效率，Twister 则代替成为更好的选择。因此, 在用第二类算法选择框架应该更取决于需要 MapReduce 步骤数量和任务需要的数据大小多少。

7.相关的工作

除了 Hapoop 和 Twister,还有多种基于 Mapreduce 模型的分布式计算框架的实现。Yingyi Buetal 提出了通过支持迭代 MapReduce 应用,扩展了 MapReduce 框架的 HoLoop。它添加了多张数据缓存机制，并使得任务调度器 loop-aware。他们声称自己和 Twister 不同，因为 HaLoop 更适合于迭代算法因为使用内存缓存和长时间运行 MapReduce 任务使得 Twister 与商品硬件相适应，而且更容易失败。

Matei Zahariaetal. 提出了 Spark，这是一个支持迭代应用同时保留了 MapReduce 的可伸缩性和容错机制的框架.Spark 关注的是不同的类似 MapReduce 的任务的执行时它们之间的数据的缓存。这些执行通过弹性分布式数据集（RDDs）可以在整个计算机群的内存中显示地保存。

但是，Spark 不支持简化的组操作，只能使用一个任务来收集结果，这将严重影响算法的可伸缩性。不过对于并行的 Reduce 任务，每个任务都能处理不同的子群的数据，这也有利于算法的实用性。

Google 解决 MapReduce 问题使用的是图迭代算法----Pregel.Grzegorz Malewiczetal.介绍 Pregel 算法时认为对于图迭代算法来说这是一个可伸缩性和容错性的平台。

与其他以前的相关工作相比,Pregel 不是基于 MapReduce 模型,而是基于 Bulk Synchronous 并行模型。在 Pregel 中，计算是由 supersteps 构成的，这些用户定义的方法之和每个图的当前节点有关。每一节点都有一个状态并且能接受从在前一个 super-step 中的另一个的节点上发送来的消息。尽管节点的核心方法是和在每个项目当地的执行的 MapReduce 的图操作是相似的。能够保留两个 super-step 之间的每个节点的状态为迭代算法提供了支持。类似的，Phoenix 为共享内存系统实现了 MapReduce。它的目标是在没有并行管理程序负担的前提下支持高效执行。由于它是在共享内存上使用的，它就能不容以发生问题。只要数据可以放进内存，我们就鼓励继续使用迭代算法。这个想法很有趣，但是我们不能认为共享内存模型是 SciCloud 项目的解决方案，因为我们对使用现有的大学资源和商业硬件来解决这个问题更感兴趣。

Saurabh Sehgal et al 用 SAGA(用于网格应用的的简单 API)实现了 MapReduce 模型。这是一个支持独立分布式编程的平台，它旨在提供分布式科学用的互操作性。他们认为 MapReduce 模型非常适合用来实现应用程序的互操作性而且它展现了分布式应用的三种不同层次互操作性。首先，该应用可以在对应用不做任何修改的情况下使用不同的分布式平台；第二，应用可以在不同的编程模型之间无缝切换；第三。多个编程模型可以同时用来解决同一个任务。他们得出结论说，应用级互操作性的影响是迈向理解通用编程模型的重要一步

8.结论以及未来工作指导

云计算及其近乎无限的资源，似乎很适合解决消耗很多资源的科学计算问题。本文研究的工作适用于 MapReduce 模型的科学计算。研究制定了 4 种不同的算法类，并在这类之间分为科学方法如下：高度并行算法，简易并行算法，迭代算法和复杂的迭代算法。本文研究了每类的影响并行效率和可扩展性的算法，使之适应 Hadoop MapReduce 框架和分析结果。

从此次分析可以得到，Hadoop MapReduce 框架有几个迭代算法的问题，在每次迭代时需要执行一个或多个 MapReduce 工作。对于每个被执行的 MapReduce 工

作，有时花费到了背景任务上，而无论输入大小如何，这都可以被看做是 MapReduce 的工作延迟。如果迭代次数较大，那么这种延迟将会在 Hadoop 中占有很大的比例，并降低了这种算法的效率。更进一步，输入 Hadoop MapReduce 作业将被存储在 HDFS。如果一个 Hadoop MapReduce 作业被执行了一次以上，这意味着每次输入都被再次读取，无论输入从先前的迭代改变了多少次。对于 CG 和 PAM 等算法，其中大部分输入不会再迭代和迭代次数较大时改变，这是一个很严重的问题。

后面的研究试图实现将最后的 2 级迭代升级为一个交替的 MapReduce 框架，又称 Tweister 的算法，使之能够判断哪些我们遇到的问题是具体的 Hadoop 框架问题和 MapReduce 模型本身固有的问题。实验结果，Twister 比对 Hadoop 的运行时间能对 3 级到 4 级进行更好地迭代算法。然而，Twister 不是没有缺点。因为 Twister 的主要优势来自于它能够将在迭代时输入数据存储到内存中，并且它也需要将这些数据融入集合内存群中才能有效。处理数百 TB 的数据是不可执行的任务。Twister 的另一个缺点就是相比于 Hadoop 的脆弱的容错能力。容错对于长期的云计算迭代任务平台有着重要影响，通常倾向于硬件和网络故障。

由于这些原因，我们得出结论，Hadoop 和 MapReduce 框架更适合于数据密集型算法，属于第一和第二级，其中包括高度并行和简易并行算法。然而，当大量的 MapReduce 工作需要执行第二级算法时，Hadoop 将失去其效率 Twister 则需要考虑被替代。像包含迭代和复杂迭代算法的第三第四级算法，Twister 已被证明是比 Hadoop 更有效的。结果表明只要适当地考虑算法特点并根据这些特点选择一个合适的 MapReduce 框架，MapReduce 就可以成功地用于解决科学计算问题。

除了 Hadoop 和 Twister，我们也考虑了其他利用云计算资源来解决科学计算问题的框架。因此，今后的工作将包括研究其他的迭代 MapReduce 框架，如火花或 HaLoop，还有交替分布式计算模型，如 Bulk Synchronous Parallel。除了评估现有的云计算解决方案，我们也有兴趣设计一个科学计算的原始分布式云计算框架，这将满足我们的需求，并提供自动并行化、容错并将适用于所有本文所描述的算法类。很明显 Hadoop 和 MapReduce 框架非常适合于高度并行算法，我们今后的工作也将包括用 Hadoop MapReduce 框架实施其他高度并行科学计算算法，比如基于 Monte Carlo 的方法[27,28]。

Srirama, Satish Narayana, Pelle Jakovits, and Eero Vainikko. "Adapting scientific computing problems to clouds using MapReduce." Future Generation Computer Systems 28.1 (2012): 184-192.

综合论文训练记录表

学生姓名	邹睿	学号	2009011526	班级	自91
论文题目	物联网分布式计算模型研究				
主要内容以及进度安排	主要内容： 对现有的分布式计算模型进行调研，结合智能电网自身的特点，将网络环境和计算联合考虑，构建一个合适的分布式计算模型，使数据传输和分布式计算时间尽量的短。 进度安排： 1. 根据已有模型，对模型各参数进行初步的设计，并对云计算开（5周）。 2. 编写控制器的调度函数和主节点对分布式计算工作进行划分。（6～10周）。 3. 对一个具体的例子编写Map函数和Reduce函数，对程序进行调试（11～14周） 指导教师签字：董建 考核组组长签字：王小纲 2013年 3月21日				
中期考核意见	该同学工作进展顺利，符合计划安排。 考核组组长签字：王小纲 2013年 4月24日				

指导教师评语	网络时延对分布式并行算法性能具有重要影响，论文基于现有分布式并行模型设计了适用于物联网环境的调度方法，并在 Hadoop 集群上进行了编程实现，具有较好的应用价值。 指导教师签字： 2013 年 6 月 13 日
评阅教师评语	论文方向新颖，表述清楚，写作符合格式，按时完成前期工作计划，符合答辩要求。 评阅教师签字：陈雯 2013 年 6 月 13 日
答辩小组评语	论文研究内容新颖，研究工作进展顺利，写作符合格式，课题及创新性达到本科生毕设要求。 答辩小组组长签字： 2013 年 6 月 14 日

总成绩：83

教学负责人签字：

2013 年 6 月 24 日