****

南京大学申请硕士学位论文

**面向高性能计算的YARN平台关键技术与应用研究**

作 者： **杨晨**

专 业： **计算机科学与技术**

研究方向： **云计算、并行计算**

指导教师： **唐杰 副教授**

**南京大学计算机科学与技术系**

**2016年5月**

­

****

**Big Data platform development research based on YARN**

Presented By

**Yang Chen**

Supervised by

**Associate Professor TANG Jie**

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE AND TECHNOLOGY OF NANJING UNIVERSITY

IN CANDIDACY FOR THE DEGREE OF MASTER

May 2016

**南京大学研究生毕业论文中文摘要首页用纸**

毕业论文题目： 基于YARN的大数据平台开发研究

计算机科学与技术 专业 2013级硕士生姓名： 杨晨

指导教师（姓名、职称）： 唐杰 副教授

**摘要**

近年来，随着信息技术的飞速发展，各种网络应用带来了数据规模的高速增长，使得大数据迅速发展成为科技界和企业界甚至世界各国政府关注的热点。为了满足海量数据存储和分析需求，需l 要使用大量计算机协同工作共同完成空前的复杂任务。相较于传统的数据，人们将大数据的特征总结为5个V，即体积大（Volume）、速度快（Velocity）、模态多（Variety）、难辨识（Veracity）和价值密度低（Value）l。由此诞生了Hadoop 和Spark 等大数据处理框架。Hadoop 作为一个开源的分布式计算系统，具有高容错性、高扩展性和高可靠性，它允许用户在廉价的机器上部署Hadoop 集群。Hadoop2.0 中将资源管理和处理组件分开，形成以一个通用的资源处理框架: YARN。YARN 在可伸缩性、效率和灵活性上提供了明显的优势。Spark 是一个围绕速度、易用性和复杂分析构建的大数据处理框架。Spark 把MapReduce 的中间结果保存在内存中，大大的提高了集群的性能，Spark也可部署在YARN 上。

本文主要的目的是更改源码来提高集群的性能，主要从两个方面来实现：一、YARN 的资源调度器更改，我们实现了一个插拔式的资源调度器EASY （Efficient Adaptive Scheduler on YARN）；二、Shuffle 过程更改，Shuffle 过程主要是reduce 收集数据的过程，我们使用RDMA （Remote Direct Memory Access）来加速网络传输速率，并对Shuffle做了一些改进。最后我们通过PKTM （Pre-Stack Kirchhoff Time Migration）实际应用来验证集群的性能，PKTM 已经逐渐成为地震勘探数据处理的常用成像手段，该方法的突出优点是成像效率高，但数据量较大，运行速率低，因此需要大数据并行计算处理。

**关键词：**大数据; Hadoop; Spark; PKTM; RDMA; Shuffle

**南京大学研究生毕业论文英文摘要首页用纸**

THESIS： Big Data platform development research based on YARN

SPECIALIZATION: Computer science and technology

POSTGRADUATE: Yang Chen

MENTOR: Associate Professor TANG Jie

**Abstract**

In recent years, Big Data quickly becomes the focus of attention in the scientific community and the business community, even in governments around the world. Compared to the traditional data pattern, today's Big Data pattern has the following characteristics: Volume, Velocity, Variety, Veracity and Value. Thus was born the big data processing framework: Hadoop, Spark and so on. Hadoop, as an open-source distributed computing platform, has high fault tolerance and scalability. It allows users to deploy Hadoop on inexpensive servers. Hadoop2.0 separates the resource management and processing module into two parts, forming a common resource processing framework: YARN. YARN provides significant advantages in scalability, efficiency and flexibility. Spark is a speed, ease of use and built around a complex analysis of large data processing framework. Spark stores the MapReduce intermediate results in memory, greatly improve the performance of the cluster, Spark can also be deployed on YARN.

The main purpose of this paper is to change the source code to improve the performance of cluster, it mainly from two aspects to achieve the goal: (1)change the resource scheduler of YARN, we implemented a pluggable resource scheduler EASY(Efficient Adaptive Scheduler on YARN); (2)change the process of Shuffle which is mainly to collect remote data from other nodes in reduce phase, we use RDMA(Remote Direct Memory Access) to speed network transmission rate and do some improvements in Shuffle phase. Finally, we verified cluster performance by PKTM in pratical, PKTM has become common image processing methods in seismic data, the outstanding advantage of this method is the high imaging efficiency, but it has a large amount of data and low run rate, so it needs large data parallel computing.

**Key words:** Big Data, Hadoop, Spark, PKTM, RDMA, Shuffle

**目录**

[第一章 绪论 1](#_Toc450478290)

[1.1 研究背景与意义 1](#_Toc450478291)

[1.1.1 Hadoop背景 2](#_Toc450478292)

[1.1.1.1 MapReduce介绍 2](#_Toc450478293)

[1.1.2 YARN平台背景 3](#_Toc450478294)

[1.1.3 Spark背景 4](#_Toc450478295)

[1.1.3.1 RDD介绍 5](#_Toc450478296)

[1.2 研究现状 6](#_Toc450478297)

[1.2.1 YARN研究现状 6](#_Toc450478298)

[1.2.2 Shuffle研究现状 7](#_Toc450478299)

[1.2.3 PKTM研究现状 10](#_Toc450478300)

[1.1 本文主要工作和内容组织 11](#_Toc450478301)

[1.3.1 本文工作 11](#_Toc450478302)

[1.3.2 章节安排 11](#_Toc450478303)

[第二章 一种基于人工鱼群算法的YARN资源调度器 13](#_Toc450478304)

[2.1 YARN调度器概述 13](#_Toc450478305)

[2.3 总体框架 13](#_Toc450478306)

[2.2 问题陈述 14](#_Toc450478307)

[2.3 EASY架构 16](#_Toc450478308)

[2.3.1 Memory-based分配 16](#_Toc450478309)

[2.3.2 Time-based分配 18](#_Toc450478310)

[2.3.2.1 Dependency 19](#_Toc450478311)

[2.3.2.2 Adaptive 20](#_Toc450478312)

[2.3.3 AFSA介绍 21](#_Toc450478313)

[2.3.4 实验环境和结果 27](#_Toc450478314)

[2.3.4.1 实验配置 28](#_Toc450478315)

[2.3.4.2 实验结果和分析 29](#_Toc450478316)

[2.4 本章小结 33](#_Toc450478317)

[第三章 基于RDMA的Shuffle算法改进 34](#_Toc450478318)

[3.1 Shuffle介绍 34](#_Toc450478319)

[3.2 RDMA介绍 36](#_Toc450478320)

[3.3 Shuffle改进 37](#_Toc450478321)

[3.3.1 Shuffle设计 38](#_Toc450478322)

[3.3.2 Shuffle实现 39](#_Toc450478323)

[3.4 实验结果与分析 41](#_Toc450478324)

[3.5 本章小结 43](#_Toc450478325)

[第四章 基于改进的YARN平台的PKTM系统 44](#_Toc450478326)

[4.1 PKTM介绍 44](#_Toc450478327)

[4.2 PKTM分布式算法改写 45](#_Toc450478328)

[4.2.1 Hadoop上的Kirchhoff算法实现 45](#_Toc450478329)

[4.2.2 Spark上的Kirchhoff算法实现 48](#_Toc450478330)

[4.3 实验结果 51](#_Toc450478331)

[4.3.1 数据准备 51](#_Toc450478332)

[4.3.2 实验结果 51](#_Toc450478333)

[4.4本章小结 56](#_Toc450478334)

[第五章 总结与展望 57](#_Toc450478335)

[5.1 总结 57](#_Toc450478336)

[5.2 展望 57](#_Toc450478337)

[参考文献 58](#_Toc450478338)

[科研成果 64](#_Toc450478339)

[在校参加的研究工作 64](#_Toc450478340)

[在校发表论文 64](#_Toc450478341)

[致谢 65](#_Toc450478342)

# 绪论

## 研究背景与意义

随着互联网的飞速发展，特别是近年来随着社交网络、物联网、云计算以及多种传感器的广泛应用，以数量庞大，种类众多，时效性强为特征的非结构化数据不断涌现，数据的重要性愈发凸显，传统的数据存储、分析技术难以实时处理大量的非结构化信息，大数据概念应运而生[1-2]。为了对新时代大数据进行有效的处理，应运而生了一系列大数据计算处理框架，主要有Hadoop，Spark，Storm等。大数据的处理主要有：复杂的批量数据处理；基于历史数据的交互式查询；基于实时数据流的数据处理。不同的框架通过不同的实现方式来处理不同类型的数据。这些大数据处理框架中最为流行的当属Hadoop和Spark两种框架，Hadoop和Spark经过不断的发展，衍生出一个通用的资源管理框架YARN。本文主要工作是基于YARN框架的更改。

高性能计算HPC（High Performance Computing）是计算机科学中的一个分支，主要是研究并行算法和开发相关的软件，致力于研究高性能的超级计算机。高性能计算方法的基本原理就是将问题分为多个部分，然后由计算机的各个处理器（或者多个计算机连接）同时解决各个问题，最终大大的缩短了解决问题的时间。高性能计算能有效、可靠、快速的并行运行大型的程序。高性能计算主要关注的重点是性能，这里的性能通常定义为一种计算率（比如每小时工作负载、每秒浮点运算速度等）。高性能计算是超级计算的同义词，尽管在技术上超级计算机系统接近当前电脑最高的操作速率。高性能计算的大部分用户是科研工作者，工程师和学术工作者，应用于很多领域，如地震勘探。地震勘探是高性能计算最重要的应用领域之一[42]。

大数据处理和高性能计算在实际中都有很广泛的应用，很多工作者在致力于这两个方向的研究开发。在高性能计算中，主要有单机多处理器的大型机和普通计算机集群。目前大数据处理和高性能计算都部署在集群上，通过大量性价比较高的普通计算机相互连接，把大型程序分成不同的部分运行在各个计算机上，从而达到较高的效率。集群的方式具有单机不具备的成本，可靠性，容错性等优点。在YARN框架上，地震勘探等高性能计算有诸多不足，比如并行部分的资源调度不合理，中间数据传输速度慢等。因此，本文主要针对这些不足做出改进，下面首先简要介绍Hadoop，Spark，YARN等背景知识。

### 1.1.1 Hadoop背景

Apache Hadoop是MapReduce计算框架的一个开源实现，专注于应对大规模互联网数据计算[3]。Hadoop作为一个基于Java 语言的分布式计算框架，它可以部署在廉价的机器上[4-5]，适用于不同种类的数据密集型处理[6]。Hadoop主要有两个组件[4]：MapReduce和HDFS。MapReduce 是一个分布式计算编程范式，而HDFS则是分布式数据文件系统。Hadoop框架透明地为应用提供可靠性和数据移动。在MapReduce 阶段中，应用程序被分割成许多小部分，而每个部分都能在集群中的任意节点上执行或重新执行；HDFS用来存储所有计算节点的数据，这为整个集群带来了非常高的带宽。MapReduce 和HDFS 的设计，使得整个框架能够自动处理节点故障，它使应用程序运用成千上万的独立计算的电脑来处理PB级的数据。Hadoop还包括了很多子项目，包括HBase，Hive，Mahout，Sqoop，Zookeeper，Avro等。Hadoop的主要架构如图1.1所示。下面我们主要介绍MapReduce的计算模型。



图1.1 Hadoop家族架构

#### 1.1.1.1 MapReduce介绍

MapReduce是一个最先由Google提出的分布式计算软件架构，它可支持大数据量的分布式处理[3]。MapReduce架构的一个重要特点是自动处理错误，对用户隐藏容错性的复杂性[7]。这个架构最初起源于函数式程序的Map和Reduce两个函数组成，但它们在MapReduce框架中的应用和原来的使用上大相径庭。MapReduce框架中，用户的程序总是被分成Map端和Reduce 端，用户只需要填写Map和Reduce 函数，提交应用程序到Hadoop系统端，系统会自动切分数据，分布式运行提交的程序。 Hadoop上的MapReduce 程序运行程序如图1.2 所示。



图1.2 MapReduce程序流程

MapReduce是一个高性能的批处理分布式计算框架，用于对海量数据进行并行分析和处理。MapReduce适合于解决对整个数据集的，易于分批处理的问题，数据一次写多次读，本质上是一种线性编程，适用于顺序处理数据，而高性能计算的并行算法大多不适合用MapReduce模型解决。

### 1.1.2 YARN平台背景

YARN是下一代MapReduce 框架，该框架主要从MRv1资源管理框架中解耦出来，并为每个应用组件提供调度功能[8]。YARN 主要由三部分组成：ResourceManager(RM)、NodeManager(NM)、ApplicationMaster(AM)。RM是一个全局的资源管理器，负责整个系统的资源管理和分配，它是YARN框架最核心的模块。它主要由调度器和应用管理器组成。NM是每个节点上的资源和任务管理器。一方面，它会定时向RM 汇报本节点的资源使用情况和各个Container（资源分配单位的抽象概念）的运行状态；另一方面，它会接收并处理来自AM 的任务启动/停止等各种请求。用户提交的每个应用程序均包含一个AM，它实际上是一个简化版的JobTracker，主要功能包括：与RM 调度器协商以获取资源、与NM 通信以启动/停止任务、监控所有任务的运行状态，并在任务运行失败时重新为任务申请资源以重启任务。YARN框架如图1.3所示。



图1.3 YARN框架

### 1.1.3 Spark背景

Spark是UC Berkeley AMP lab所开源的类Hadoop MapReduce的通用的并行计算框架，它支持Java，Scala，Python等语言[11]。Spark 是基于MapReduce算法实现的分布式计算[12]，拥有Hadoop 所具有的优点；但不同于MapReduce的是Job 的中间输出结果可以保存在内存中，从而不再需要读写HDFS，因此Spark 能更好的适用于数据挖掘、机器学习等迭代算法。Spark 的核心组件是RDD（Resilient Distributed Datasets）。Spark架构如图1.4所示。



图1.4 Spark架构

#### 1.1.3.1 RDD介绍

RDD是一个容错的、并行的数据结构，可以让用户显式地将数据存储到磁盘和内存中，并能控制数据的分区[12]。同时，RDD还提供了一组一组丰富的操作来操作这些数据，它提供了粗粒度的转换操作，而不是细粒度的更新操作[13]。RDD作为数据结构，本质上是一个只读的分区记录集合。一个RDD可以包含多个分区，每个分区就是一个数据集片段。RDD可以相互依赖。如果RDD的每个分区最多只能被一个子RDD 的一个分区使用，则称之为narrow dependency；若依赖多个子RDD 分区，则称之为wide dependency[13]。RDD 的依赖模型如图1.5所示。



图1.5 RDD依赖关系

RDD是Spark最核心的部分，它表示已被分区、不可变的并能够被并行操作的数据集合，不同的数据集格式对应不同的RDD实现。RDD必须是可序列化的，RDD可以读入到内存的缓存中，每次对RDD数据集的操作之后的结构都可以存放到内存中，下一个操作可以直接从内存中输入，省去了MapReduce大量的磁盘I/O 操作。这对迭代运算比较常见的机器学习算法来说，运行效率大大提高。

## 1.2 研究现状

### 1.2.1 YARN研究现状

2002~2004年Doug Cutting联合了Mike Cafarella开发了一个开源的搜索引擎Nutch，经过了一年的努力，该Nutch能够 1亿网页的抓取索引[44]。

2004~2006年Google发表论文，向全世界提出MapReduce概念[43]，详细介绍了MapReduce的计算模型以及GFS概念[45]，介绍了文件系统，于是Doug和Mike根据这两篇文章用Java语言重构了Nutch。

2006年以后，Dog Cutting把Nutch底层的GFS/MapReduce剥离出来做成系统，并命名为Hadoop给了Yahoo公司，至此Hadoop逐渐发展了起来。

Hadoop是一个共享的计算平台，扩展性、多租户技术、隔离性，安全性等都是Hadoop考虑的问题。Hadoop计算层在其演变过程中涉及了很多这些问题，它的架构经历了很多进化阶段。下面我们主要介绍四个主要的阶段：

1. Ad Hoc集群时代：集群以Ad Hoc和基于用户的形式搭建。
2. Hadoop on Demand：是一个能在大型物理集群上供应虚拟hadoop集群的系统。
3. 共享集群：共享的MapReduce框架和共享的HDFS 文件系统。
4. YARN框架：通用资源框架，解决了以前阶段的需求和缺点。

目前YARN已经是一个弹性计算平台，它的目标已经不再局限于支持MapReduce一种计算框架，而是朝着对多种框架（如Spark）进行统一管理的方向发展。共享集群有很多优点：资源利用率、运维成本低和数据共享。YARN共享集群示意图如图1.6所示。



图1.6 共享集群框架

YARN的核心模块为RM（ResourceManager），该模块主要负责资源管理和分配。它主要由两个组件构成：调度器和应用程序管理器。调度器为RM的核心功能，它根据容量、队列等限制条件，将系统中的资源分配给各个正在运行的应用程序。

YARN中的资源调度模块是插拔式的，用户可以根据自己的需要更改调度器模块。为了提高YARN在job完成时间，本地性或者其他性能标准上提高性能，许多研究者都实现了自己的调度器。Zaharia等提出了一种delay-scheduling算法[46]用来在保证公平性的前提下提高数据本地化性能，作者通过比较默认的FIFO和Fair调度器证明了delay-scheduling算法数据本地性的性能；Kumar等提出了基于内容感知的调度器[47]，通过收集Hadoop中的一些运行信息来调度资源；Gupta等人则开发了ThroughputScheduler调度器[48]，该调度器通过使用贝叶斯学习算法找到和节点容量最佳匹配的Job需求来调度资源；Lee引入了JoSS调度器[49]，在map和reduce任务同时提高了数据本地性，避免Job饥饿，提高Job运行时间；Yingjie Guo等[50]提出了用贝叶斯分类算法来判断Job的类型，然后从中选择Job来执行；Xiaotong Zhang针对reduce端提出了改进一个贪婪的算法[51]来解决小Job饥饿的问题，从而提高集群的性能。目前YARN资源调度器对高性能计算程序的调度依然有提升的空间。

### 1.2.2 Shuffle研究现状

在Hadoop的job运行过程中，Shuffle阶段一直是一个核心的阶段[14]。Shuffle的主要目标是完整地从Map端拉取数据到Reduce端，并尽可能地减少对带宽的不必要消耗，减少磁盘I/O对任务执行的影响。Shuffle 阶段是隶属于Reduce过程的子过程，但是Shuffle的整个过程在Map Reduce过程中起到了一个数据过渡的作用。该模块是连接Map端和Reduce 端的核心桥梁，该模块是插拔式的，用户可根据自己的需求更改Shuffle的实现。

Hadoop的Shuffle发展在0.10.0版本[43]之前采用了“每个Map任务产生R个文件（R个分区）的方案”，该方案会产生大量的随机读写I/O，对于大数据的处理非常不利；为了避免Map任务产生大量的文件，HADOOP-331尝试对该方案进行优化[52]，优化方法：为每个Shuffle加一个环buffer，一旦buffer满了，则将内存数据spill到磁盘上（一个索引文件，保存偏移量），最终合并这些spill文件；上述阶段并没有对Shuffle做架构调整，在Hadoop2.0之前，buffer被分为两个部分，对buffer的调优很繁琐，MAPREDUCE-64解决了这个问题[53]，把两个buffer合成了一个；在Hadoop2.0后，Shuffle的架构做了调整，它被当成一个子阶段嵌入到Reduce阶段中，Map和Reduce阶段可以同时运行，Reduce负责收集数据，在Map未完成时一直占用这资源，为了继续提高资源利用率，需要把Shuffle从Reduce独立出来变成服务，这个问题MAPREDUCE-2354正在解决[54]。

文章[56-57]中介绍了Shuffle的性能问题，由于Shuffle是Hadoop中非常核心的模块，这就意味着Shuffle的改进很有意义。Condie等提出instant shuffling的概念[55]，直接将Map中间数据发送到Reduce端，节省创建中间数据的时间；Camdoop的文章[58]在数据转发中使用了分层聚合的策略来设计Shuffle过程，减少了中间数据的网络传输；Seo [57]虽然通过减少I/O冗余提高了性能，但是他并没有改进由Shuffle引起的问题；Zaharia在设计的新的调度器中[46]，通过数据本地性改进了Map端的Shuffle过程；Wang等[59]通过一个全局的顺序读策略来读取Map数据提高性能；Yanfei等[60]把Shuffle和Reduce结合，并用了一些延迟策略来提高Shuffle性能；MaRCO等[61]重叠了Shuffle和Reduce阶段，减少了Reduce等待时间，提高了效率。尽管已经有了很多对Shuffle改进的工作，但Shuffle对于高性能计算程序仍有大的提升空间。

Shuffle过程是Reduce 阶段的初始操作阶段，过程简单的理解就是“远程拷贝数据”的过程，拷贝的目标数据源就是map端的中间输出结果[15]。Reduce过程要先获取这些数据，然后才能执行本身的操作。一般在Map 产生的中间数据会写在当前节点的磁盘上，而reduce task可能会运行在其他节点上，这就意味着shuffle阶段的拷贝数据会经过网络传输，目前使用的是Java的TCP/IP协议传输数据，速率相对较低，如果中间结果数据量较大的话，会消耗很多的网络带宽。从整体上看，Map是映射，负责数据的过滤分发；Reduce是规约，负责数据的计算归并。Reduce 的数据来源于Map，Map的输出即是Reduce的输入，在Reduce端包括copy和sort 过程。

Shuffle阶段横跨Map和Reduce端[16]，Shuffle 阶段主要分为6个阶段：

1. Collect阶段，将中间结果输出到内部的环形数据缓冲区。
2. Spill阶段，当缓冲区的使用率达到一定阀值后，触发一次Spill操作，将环形缓冲区的部分数据写到本地磁盘。在将数据写磁盘之前，先要对写磁盘的数据进行一次排序操作。
3. Combiner阶段，待Map Task 任务的所有数据都处理完后，会对任务产生的所有中间数据文件做一次合并操作，以确保一个Map Task 最终只生成一个中间数据。
4. Shuffle阶段，也成为Copy 阶段，Reduce Task 从各个Map Task上远程复制一片数据，并针对某一片数据，如果其大小超过一定阈值，则写到磁盘上，否则直接放到内存中。
5. Merge阶段，在远程复制数据的同时，Reduce Task 启动了两个后台线程对内存和磁盘上的文件进行合并，以防止内存使用过多或磁盘上文件过多。
6. Sort阶段，按照MapReduce 语义，用户编写的reduce函数输入的是按key 进行聚集的一组数据。为了将key相同的数据聚在一起，hadoop采用了排序的策略，由于各个Map Task已经实现对自己的处理结果进行了局部排序，因此Reduce Task只需对所有数据进行一次归并排序即可。

Shuffle的整体结构图如1.10所示，图中Spill 阶段包含了Collect、Spill 和Combiner阶段，该结构如图1.11 所示。



图1.10 Shuffle过程



图1.11 Spill过程

### 1.2.3 PKTM研究现状

石油和天然气行业是分布式计算和并行计算的重要消费者[62]，因为在该行业大部分程序都具有海量的输入数据。地质勘探是寻找石油的重要途径，而地质成像算法则是地质勘探中非常重要的环节。

PKTM（Prestack Kirchhoff Time Migration）被认为在处理地质数据中最有效的成像偏移算法[62]。PKTM是计算密集型的算法，有着非常复杂的浮点运算，这就导致了PKTM算法有着很长的运行时间，当需要迭代使用速度模型时，PKTM算法将更加耗时。地质成像是指通过在地面不同的区域放置炮点和检波器，通过炮点爆炸传输的波被地层反射回检波器来收集数据（都是浮点数据），这过程会产生大量的数据，然后通过PKTM算法来不断的计算这些浮点数，最后产生输出文件，用特定的工具（ximage）可以生成地质图像。从PKTM过程可以看出PKTM不仅数据量极大，计算也很复杂，导致单机算法根本满足不了现有的需求，这就需要并行的PKTM算法。

由于PKTM在地质成像上的广泛应用，很多地质工作者以及计算机工作者从事PKTM算法的优化和改进工作。Rizvandi文章[62]中介绍了PKTM在Hadoop上的改进，该文章只是单纯的实现了一个分布式版本，还有很大的提升空间；Dai和Li提出了一个通过旅行时间获取偏移的速度模型的实际模型[63]；Li和Wang改进了PKTM算法[64]，减少了I/O带宽，提高了效率；Zhang等人提供了在GPU上实现的KPSDM算法[65]（该算法也是地质成像算法），使用并行计算，大大的缩短了运行时间；同时，Shi和Li也实现了用GPU的CUDA语言实现PKTM算法[66]；文章[67-69]中也是实现了GPU版的PKTM算法；而文章[70-71]则是在MPI上实现了PKTM算法。

上述文章可以看出，PKTM算法有着数据量大，运行时间长等缺点，目前主要有两种改进，一种是在Hadoop上的分布式并行算法改进，这种改进只是单纯的更改了PKTM算法，但是没有让Hadoop来适应PKTM算法；而GPU版的算法改进容错性不足，当有一个任务失败时，整个运行程序结果就会失败。因此PKTM算法还有很多的改进空间。

## 1.3 本文主要工作和内容组织

### 1.3.1 本文工作

本文对大数据主流平台Hadoop和Spark进行了调研，并修改了YARN框架的源码，从两个方面对其进行了改进，以提升集群的整体性能，最后我们实现了PKTM地质成像算法的改进，并用其证明了我们对YARN框架改进的有效性，具体工作如下：

* 首先介绍YARN调度器的功能，针对调度器资源利用率不高以及Job运行时间长的问题，将Job的调度问题抽象为一个多背包模型，然后分析该模型并提取出目标函数，并把该目标函数应用到人工鱼群算法上，最后在YARN框架中实现了插拔式调度器以及实验验证该改进的有效性。
* 首先深入分析了MapReduce的Shuffle 过程，调研了Infiniband等硬件新进展，并介绍了RDMA（Remote Direct Memory Access）远程直接内存拷贝技术，根据Shuffle过程中数据传输较慢的问题，提出了一种基于RDMA 技术的Shuffle算法，替代了传统的TCP/IP 传输协议，并优化了Shuffle中的关键算法，用Hadoop基准程序验证Shuffle改进的性能。
* 介绍PKTM算法，实现了PKTM的Hadoop和Spark版并行化算法，并用这两个算法来验证YARN平台改进的性能。

### 1.3.2 章节安排

本文一共有五个章节，各章节安排如下：

1. 第一章：绪论。分别介绍了Hadoop、Spark、YARN的背景知识和研究现状，以及存在的一些问题，本文的研究工作和文章整体框架。
2. 第二章：YARN平台改进。该章首先介绍YARN调度器功能，并分析了YARN调度器中存在的问题，然后抽象出多背包问题，应用人工鱼群算法来解决多背包问题，并实现了插拔式的人工鱼群调度器，最后用实验验证该调度器的可行性。
3. 第三章：Shuffle过程改进。首先分析了MapReduce的Shuffle 过程，分析Shuffle中的问题，使用RDMA 技术替代了传统的TCP/IP 传输协议，并改进了Shuffle的算法，最后用实验验证性能提高。
4. 第四章：PKTM算法改进。首先实现了PKTM的Hadoop和Spark版并行化算法，并做出一定的改进，最后用此算法来验证YARN框架改进后的性能。
5. 第五章：总结与展望。总结本文所阐述的所有算法，并对下一步工作做出展望。

# 一种基于人工鱼群算法的YARN资源调度器

## 2.1 YARN调度器概述

本文替换了YARN中自带的插拔式调度器模块，用人工鱼群算法实现了新的调度器。下面从几个方面来介绍EASY 调度器。1、介绍Yarn调度器；2、Yarn调度器的不足；3、改进的可能性；4、本章总结

* 1. 概述

阐述问题，

2.2 相关工作

## 2.3 总体框架

EASY调度器的设计目标是提高资源利用率和缩短job运行时间。EASY作为RM的里的一个组件，主要有两个部分构成：Memory-based 分配和Time-based分配。EASY的架构图如图2.1 所示。

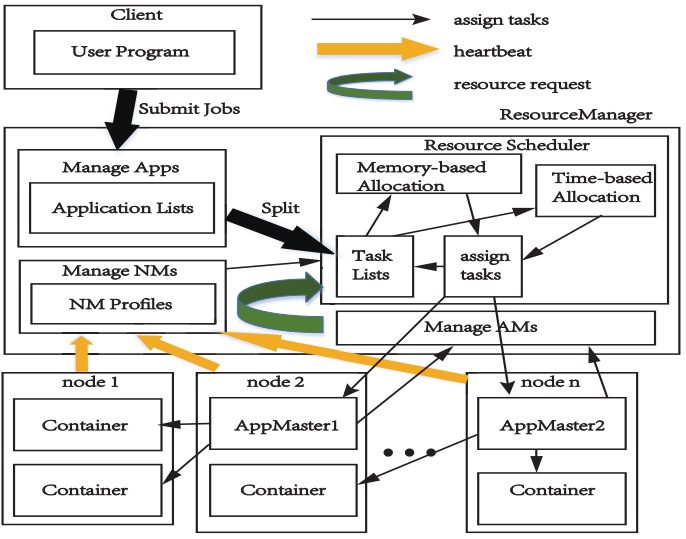


图2.1 EASY架构

从图中更可以看出，EASY调度器组合了两个部分来调度tasks。通过这两个部分选择最优的tasks来运行。从细节上来看，client向YARN中提交jobs，然后启动程序对应的AM，AM传输tasks到EASY，并且EASY根据NM心跳机制传来的信息决定tasks分配到哪个节点上。在图2.1 中有两个job 在运行，每个job对应一个AM，然后EASY 模块不断重复的分配tasks 到节点上运行。

## 2.2 问题陈述

在YARN调度器模块中，tasks被分配到各个节点上。由此产生的问题就是：哪些tasks 被分配到哪些节点上。这个问题就是调度器的重点。问题的模型如下图2.2所示。



图2.2 问题模型

我们通过图中模型来分析问题。我们定义一组Jobs为{J1, J2, J3, …}, m个节点{S1, S2, S3, … Sm}. 每个job由一系列map tasks 和reduce tasks。在YARN 中，当一个job 提交时，RM会分配一个Container来运行程序对应的AM。然后AM会接管这个job的生命周期，并向RM 请求资源，每个资源请求都是一个元组<>，代表优先级，资源列表（如<1024MB,1core>），代表相同tasks 的个数，表示数据存放的位置，表示task是否本地化。我们将jobs 拆分成tasks，T={}，这样在集群中我们便没有了jobs 的概念，只有tasks 的概念。

从YARN的源码上来看，所有task都运行在Container中，没有map和reduce 区分，但是AM 在向RM提交请求时，不同种类的task有不同的优先级，可以通过该优先级来判断task的种类。另外在YARN中有多种资源存在，如内存、cpu等，我们定义资源向量为k 种资源，目前YARN 中只考虑2种资源：内存、cpu。另外，我们定义两个二维数组W, V。W[i, j]表示在节点中的剩余可用资源。调度器可以根据每次心跳机制获得的信息来获取W数组。而V[i, j] 数组代表所有task 的资源请求，j代表第j种资源。

YARN中所有的调度器的目标主要有两点：最大化资源利用率和缩减运行时间。我们定义代表任务的开始时间（设置job开始时间为0），表示任务的总的运行时间。所以最理想化的目标函数为：

公式中表示当前在中分配的任务。这个公式阐述了从job 开始到最后一个task执行完毕所消耗的所有时间，该时间越小表示集群效率越高。但由于task运行时间不能准确得到，所有这只是最理想的目标函数。本文使用人工鱼群算法实现YARN调度器模块，用人工鱼群的目标函数来替代理想的目标函数以尽量实现最佳效率。下面将分别介绍EASY 调度器的两个部分、人工鱼群算法以及实验验证。

## EASY架构

EASY使用管理人工鱼群算法，我们通过设计目标函数来计算每个task的值，根据这些值的大小来选择运行在节点上的tasks。因此，本文的关键在于目标函数的设计，从Memory和Time两方面来分析目标函数的构建，。YARN的资源管理器对cpu 资源隔离不够，因此，本文主要讨论Memory资源管理。

### 2.3.1 Memory-based分配

在YARN中，所有节点并行的发送心跳信息到RM 中。这意味着在一个时刻可能会有很多节点同时向RM 发送心跳。因此，我们可以抽象出一个多背包（MKP）问题。每个节点相当于一个背包，tasks相当于背包中的物品，EASY把tasks 分配到各个节点上就相当于把物品放到背包中，这就是MKP问题。因此，MKP模型非常符合YARN的运行环境。下面用一个例子说明MKP 模型的好处（此处只考虑Memory）：如果每个节点都有100容量的Memory，并且有6个待分配的tasks，它们的Memory需求分别是82、43、42、15、12、6。MKP 模型会将<82,12,6> 放入第一个节点，<43,42,15> 放入第二个节点，在tasks很多的情况下，该分配方法能最大化Memory利用率。我们通过图2.3 来对比MKP模型和YARN中现存调度器的分配模型。



图2.3 模型对比

在图中我们看出，FIFO调度器的模型按照先进先出的方式来调度tasks；而Capacity和Fair 调度器根据公平性和容量调度tasks，达到了相对较大的资源利用率；然而MKP模型得到了最大的资源利用率。

从源码角度来看，整个EASY调度过程就是一个RPC（Remote Procedure Call）方法调用。当节点发送一个心态信息时，其实它在调用一个RPC函数，通过参数把信息传入到调度器中，然后调度器分析这些信息，求出最佳tasks并分配到节点上。在YARN中，所有心跳信息保存在队列中，然后调度器每次都从队列中取出一个信息进行处理，而在EASY中，我们用一个后台线程一次处理多个信息，这样就对应了MKP模型。MKP模型是一个NP-hard问题[17-18]。我们使用AFSA来解决该问题，AFSA是人工智能中一种典型的行为程序，适合求解组合优化问题，寻找最优解。MKP的目标函数如下所示：

是每个task的价值，MKP模型的目标是最大化资源利用率，所以我们根据资源来定义task的价值。我们用加权和来定义该价值，。在Hadoop 中，reduce task 会在map task没有结束前就开始运行，并一直占着资源，影响集群性能，所以我们对每个task定义一个常数来区分map和reduce，同种类的task 具有相同的。所以该阶段的目标函数为：。该目标函数的值和资源利用率成正相关，值越大，资源利用率越高，因此作为Memory 阶段的目标函数。

### 2.3.2 Time-based分配

Time-based分配是EASY调度器的另一个组件，该组件的目标在于最大化的减少job运行时间。现在考虑下面两种情况：

1. reduce task滞留在内存中，影响内存利用率。
2. 两个同种类的tasks运行不同的时间。

在第一种情况下，过早的启动reduce task将会影响系统性能，所以我们定义一些原则来寻找最佳的启动时间。而在第二种情况下，相同种类的task理论上运行相同的时间，但是我们不能准确预测task的运行时间。在EASY中，我们使用理论的模型，假设同种类的task运行相同的时间，因此我们保存已经运行完成的task的时间来预测同种类还没运行的其他tasks。我们命名第一种情况为“Dependency”，第二种情况为“Adaptive”。所以该阶段的目标函数为：。 在该公式中a,b 为权重，代表目标函数注重哪个方面。在“Dependency” 中，同一个job 中的map task 将在reduce task之前运行；而“Adaptive” 中，虽然不一定会提高效率，但会首先运行短的任务。

#### 2.3.2.1 Dependency

在Hadoop中，reduce tasks会在map tasks结束运行之前就运行，reduce tasks会一直占着资源，直到所有map tasks运行完毕。在其他的文章[19-22]中，也都使用了reduce 和map 的依赖关系，但是他们解决方法不能直接应用到我们的问题中。从源码的角度来看，YARN 中没有map和reduce的概念，但是按照上面所说，AM提交的不同tasks的资源请求的优先级不同，因此我们通过优先级来确定map和reduce之间的依赖关系。针对依赖关系问题，我们设计了以下几种情况：

C1：如果tasks数组T中含有AM tasks，这些tasks 会被首先运行。因为AM任务会负责整个job的生命周期，所以我们定义AM的dependency为一个较大的常数：。

C2：如果一个job中已经运行了更多的map tasks，那么更适合先运行该job 剩余的map tasks。 此外，当这个job的reduce task需要更多的资源时，先运行map tasks 将会提高资源利用率。因此，map tasks的dependency 为：。

C3：如果一个job几乎运行完了所有的map tasks，那这个时候就适合运行reduce tasks来收集数据。这个规则是针对于同一个job中的tasks的紧急性，上两个规则针对的是不同jobs之间的tasks 的优先级。因此reduce tasks 的dependency 定义为：。

在这里，代表job 中已经被分配的map和reduce tasks的数目；而表示task 的价值；显示了job 中所有map tasks的数目；最后代表job 中正在运行的map和reduce数目。所以针对Dependency 的目标函数为，P代表被分配在节点上的所有tasks 的集合。通过该Dependency目标函数，EASY可以得到符合所有规则的最优tasks，这样就缩短了整个集群的运行时间。

#### 2.3.2.2 Adaptive

在Hadoop集群运行过程中，会出现各种各样的问题，例如，同种tasks 的运行时间不一致。本文抽象出一个装箱问题来尽量减少整体运行时间。首先，本文假设整个集群只有一个节点，这个节点就相当于一个箱子，而所有tasks 则相当于物品。假设所有tasks的运行时间是一样的，当集群运行时，调度器分配尽量多的tasks到集群中，所有tasks运行一样的时间，这时集群又空了，调度器再次分配tasks到集群中，这样依次运行，直到所有tasks运行完毕。我们假想每次集群运行一波tasks 当成使用了一个箱子，所以要使运行时间越短就是箱子越少，这就是装箱问题，示意图如下图2.4 所示。



图2.4 装箱问题

装箱问题是一个理想的模型，在同种tasks运行时间大致相同的情况下，该模型会很有效；在糟糕的情况下也不会降低性能。装箱问题的目标函数如下所示，其中A代表分配在中的所有tasks：

要有公式编号

在装箱问题中，一个重点因素是tasks的运行时间，我们设置来表示运行时间。该参数的求解如下所示：

1. 当集群才启动时，所有的都初始化为0。当一些tasks运行完毕后，其他同种类的tasks 将会被设置为已经结束的tasks的运行时间值。
2. 在同一个节点上，所有分配在该节点上的tasks都会并行运行，所以该节点上的装箱问题只要考虑最大的那个。
3. 在不同的节点上，运行在上面的tasks也是并行的，这里我们简单的考虑用叠加值来代表整个集群的装箱问题，即。

综合多方面的因素，该阶段的目标函数为：

代表分配在节点上的task 的运行时间。这公式意味着分配在不同节点上的tasks运行时间的叠加和，前面的负号代表最优解和叠加和成负相关，tasks运行时间越短代表集群性能越好。通过该目标函数我们可以尽量找到运行时间上的最优解。由此我们得到了整个调度器的目标函数，下面我们将介绍人工鱼群算法。

### 2.3.3 AFSA介绍

鱼群往往能自行或尾随其他鱼找到营养物质多的地方，因而鱼生存数目最多的地方一般就是本水域中营养物质最多的地方。人工鱼倾向于靠近鱼群来保护自己，并且寻找食物和避免冲突[23]。人工鱼群算法就是通过构造人工鱼来模仿鱼群的各种行为。人工鱼群算法在组合优化问题中有这广泛的应用[24]，并且该算法也是人工智能算法中非常典型的一个[25]。 人工鱼有几种典型的行为：觅食行为、聚群行为、追尾行为等[26]。人工鱼群有以下几个特点：

1. 具有较快的收敛速度，可以用于解决有实时性要求的问题。
2. 对于一些精度要求不高的场合，可以用此算法快速的得到一个可行解。
3. 不需要问题的严格机理模型，甚至不需要问题的精确描述，这使得它的应用范围得以延伸。

下面将详细介绍适合EASY调度器的人工鱼群算法。首先，我们将MKP模型对应到AFSA算法。我们有n 个tasks，所有tasks 的分配状态构成一条人工鱼，可以被表示成，表示task 被分配到哪个节点上，该向量代表最优解[24]。我们可以把人工鱼当成是n维空间的一个点。例如，X=(0,2,1,2,3,0,1,3,2) 表示第2,4,9 个任务分配到第二个节点；第3,7个任务分配到第一个节点；第5,8个任务分配到第三个节点；而第1,6个任务没有分配。鱼希望靠近鱼群来保护它们和寻找食物，基于这些天然的行为，AFSA 算法模拟鱼群行为，本文主要介绍的行为有：追尾，聚群，觅食，跳跃和随机游走。在该算法中，上面介绍的目标函数就是鱼群的食物浓度。人工鱼群算法的模型如图2.5所示。



图2.5 AFSA模型

AFSA算法有很多参数，人工鱼之间的距离定义为，这里使用Hamming 距离来求解；visual用来定义人工鱼的可视范围；是拥挤因子，，代表人工鱼可视范围的拥挤程度；trynumber表示人工鱼移动的最大尝试次数；还有定义一个公告板board，用来记录每次迭代的最优解。下面介绍AFSA算法的各种行为。

**初始化：**

首先我们先随机初始化N条人工鱼，，每个就是一条人工鱼。在初始化过程中，我们边随机化边根据资源容量来检查人工鱼的可行性。这样就基本保证了每条鱼的可行性，然后我们重复N次初始化N条鱼。当两条鱼的距离，则表示人工鱼在鱼的可视范围内，我们规定可视范围内的所有鱼的个数为，可视范围的拥挤程度阈值定义为，。基于这个设定，有以下几种可能：

1. 当visual为空了，这意味着，人工鱼会随机游走选择更好的区域。
2. 当visual可视范围不拥挤时，，人工鱼向可视范围内的最佳点移动，如果最佳移动点不能提高目标函数值，则向中心点移动。
3. 当visual非常拥挤时，，人工鱼向着可视范围外的更好的聚群移动。

**追尾行为：**

我们知道鱼会向着更多食物的地方移动，在该行为内，我们在可视范围内选择一个最近的人工鱼，如果该鱼的目标函数大于当前鱼，，则向该鱼方向移动；否则执行随机游走行为。

**聚群行为：**

在该行为中，我们考虑几种情况：当visual可视范围内寻找的最优点不能提高目标函数的值，，这意味着追尾行为不可行。这种情况下，我们求出可视范围内的中心点，如果，则向聚群中心移动，否则执行随机游走行为。中心点的求解算法如算法1所示。



**觅食行为：**

该行为存在两种情况：

1. 当visual可视范围不拥挤时，并且最佳点和中心点都不能提高目标函数。
2. 当visual可视范围拥挤时，人工鱼会选择一个更优的区域。

在该行为内，我们随机选择一条鱼，并且trynumber减一，如果目标函数大于当前值，，则该行为结束，并且人工鱼向该鱼移动；如果该随机鱼不能提升目标函数的值，则继续该过程，直到trynumber变为0，然后随机选择一条鱼进行移动。

**随机游走：**

当visual可视范围为空时，这就意味着没有其他鱼在当前鱼的周围或者其他行为都失效时，我们使用该行为。在该行为中，我们只是随机的选择一条鱼，然后向着该条鱼移动。

在人工鱼的各种行为中，我们可能会得到各种不可行解，例如，分配的资源总量大于集群中的总量（）。基于这种情况，我们应用一个Decoding算法来使不可行解变的可行，该算法如算法2 所示。



在该Decoding算法之后，我们又使用了一个Add\\_Item算法来进一步提高可行解。通过该算法，我们可以把没有分配的tasks进一步分配到有足够资源的集群上。该算法如算法3所示。



人工鱼群算法是一个迭代的算法，所以下一条鱼表示为，其中t表示迭代次数，所以迭代的公式如下：

**跳跃行为:**

在人工鱼群算法中，当L次迭代时，我们可能得到相同的最大值，但这并不一定是最优解，可能是局部最优。基于此种情况，我们使用跳跃行为，当L次迭代最优目标函数值没有改变时，我们随机初始化其中u条鱼，然后再使用其他行为。

**终止行为:**

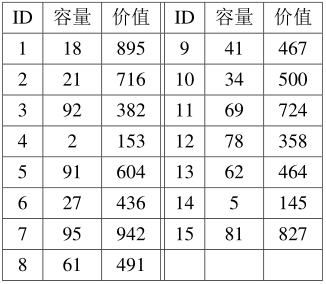
一般的人工鱼群算法有两个终止条件：达到最大的迭代次数和达到最优解。 但在我们的调度器中，最优解是难以预测的，因此我们使用 迭代次数来终止人工鱼群算法。整个AFSA算法如算法4-5所示。





### 2.3.4 实验环境和结果

在进行实际的调度器实验之前，我们先用一个模拟实验来测试下人工鱼群算法在多背包问题上的效率。我们定义三个背包，每个背包容量分别为：110,150,190，而15个物品的体积和价值分别从[1,99]和[100,999] 两个区间中随机得到，具体值如表2.1所示。而AFSA算法参数如表2.2所示。





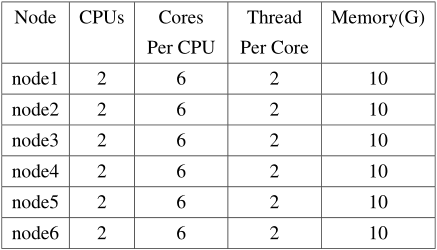
我们运行该程序10次，然后发现每次都是在迭代15次之前就得到了最优解：6151，平均耗时0.134s。运行示意图如图2.6所示。



图2.6 AFSA模拟实验

#### 2.3.4.1 实验配置

我们在YARN上做实验，一共有6个节点，每个节点配置10G内存和24 个CPU 虚拟核，具体配置如表2.3所示。而AFSA参数为：N=20, L=10, =50, =0.7, step=2, trynumber=20, m=3, visual=5。

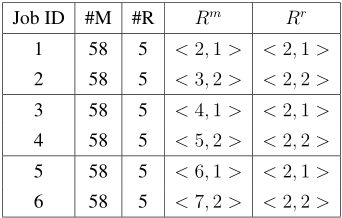


#### 2.3.4.2 实验结果和分析

在YARN运行过程中，我们使用两种情况来验证EASY调度器的效率：

（1）同种类型的jobs

在该实验中，我们使用简单工作负载的WordCount程序，一共有6个job，每个job有7.18G 的输入文件，该文件由RandomTextWriter程序产生，具体配置如下表2.4 所示。



该实验结果如图2.7所示。在图中，我们看出EASY调度器比Hadoop 目前的调度器有更高的性能。例如，EASY的运行时间比FIFO和Fair调度器缩短了33.578% 和46.059%。在文章[27]中，HaSTE调度器在相似的实验环境下，运行时间比FIFO和Fair 缩短了27%和44.6%。可以看出EASY 的调度器比HaSTE 效率高。同样的，可以看出EASY在内存利用率上也有相当的提高。

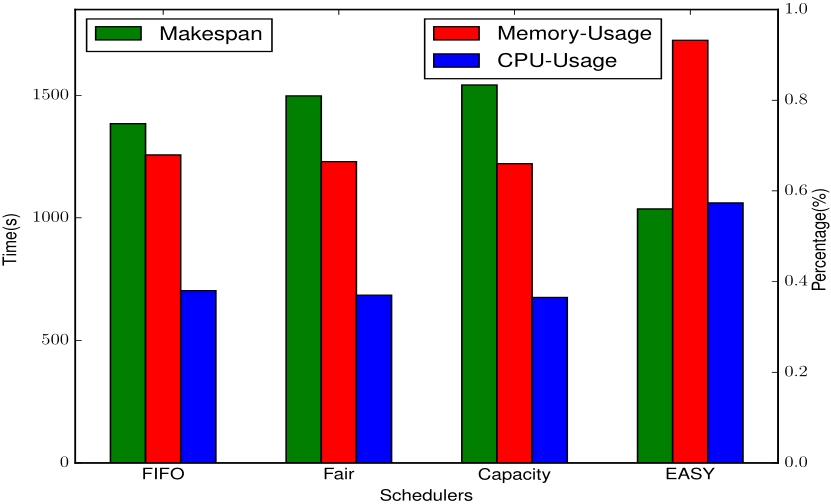
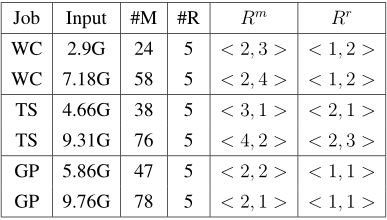


图2.7 同种类型的jobs的运行时间和内存利用率对比

（2）不同种类型的jobs：

在此种情况下，我们考虑复合负载的jobs，例如，WordCount、TeraSort、Grep。TreaSort 的数据来源于TeraGen，WordCount的数据来源于RandomTextWriter，实验配置如表2.5所示。



该实验结果如图2.8所示。图中我们可以看出EASY调度器运行时间比FIFO和Fair缩短了43.255%和40.54%，相比于HaSTE的36.3% 和33.9%提高了更多，这就意味着EASY调度器比HaSTE调度器效率更高。同样的，EASY的内存利用率也比HaSTE的高。

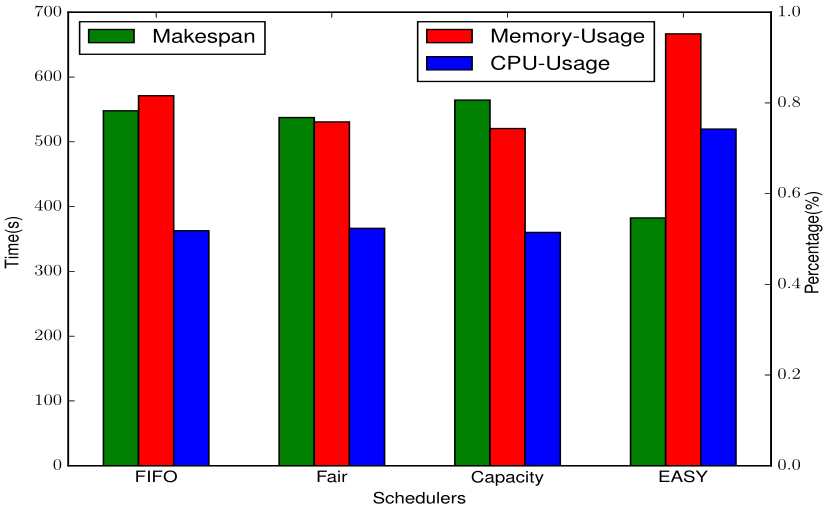


图2.8 不同种类型的jobs的运行时间和内存利用率对比

在该实验下，我们继续分析不同job的内存使用情况。在EASY调度器中，我们通过改写代码，添加线程，让调度器每隔150毫秒统计一次所有job的内存使用和cpu使用。通过该程序，我们可以统计正在运行的jobs的内存和cpu使用情况。具体的实验图如图2.9和2.10所示。从两幅图中看出，EASY调度器的内存利用率和cpu利用率要高于其他调度器。我们主要分析下内存利用率，在FIFO 调度器中，jobs按照提交顺序依次执行，一开始达到了最大的内存容量，但是当前面的jobs执行完成后，最后的job中的tasks将孤立的执行，这会大大的影响总的内存利用率；而在Fair和Capacity 中，jobs基本上是并行执行，占用差不多的内存，但很难适配的利用最大的内存，从图中可以看出很难达到60G，所以利用率也不会很高；而在EASY调度器中，我们可以看到jobs中的tasks 组合在一起达到了最大的内存，而且一直保持着接近最大内存的内存利用率，所以EASY调度器的内存利用率最大。在内存图中，我们看到每张图的左边都有几G的分配，这是所有jobs的AM tasks正在运行。同理我们从cpu 图中可以看出，EASY 调度器的cpu 利用率要大于其他调度器。

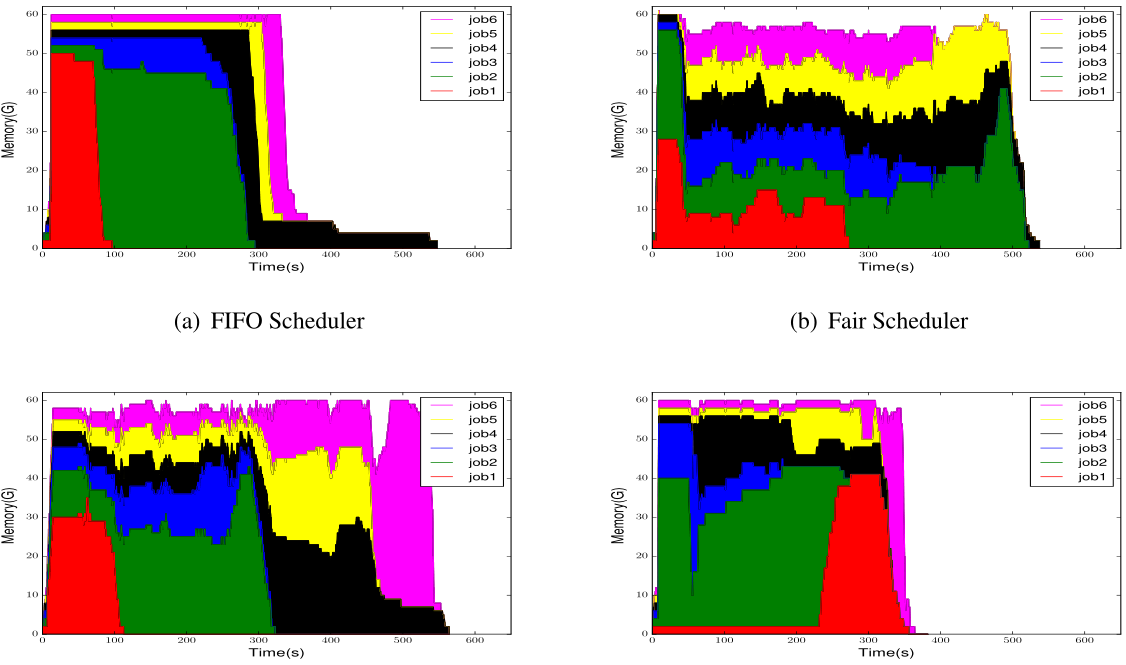


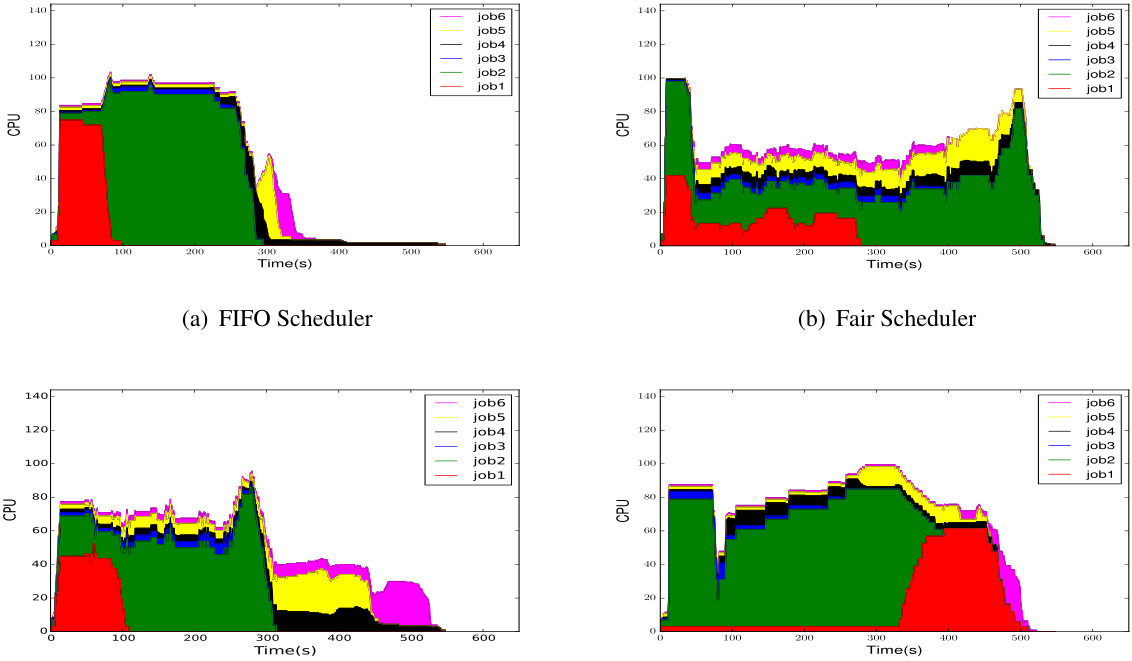
图2.9 四种调度器的运行时内存

图2.10 四种调度器的运行时cpu

总体上，EASY在缩短jobs运行时间和提高资源利用率两方面都有了提升，这两方面是相辅相成的，当内存利用率提高了，并行的tasks就会增多，最终会缩短运行时间。此外，AFSA算法对参数的不同取值不敏感。

## 2.4 本章小结

本章主要介绍了YARN的调度器更改，我们从提高内存利用率和减少job运行时间两个方面来提出更改方案，从调度器的过程中，我们抽象出了一个MKP多背包问题，并用AFSA算法来解决该问题，我们通过分析不同目标场景下的情况，不断的更改AFSA算法的目标函数，以达到每次调度都选取了最优的tasks运行，此外我们还从cpu，内存利用率等方面的实验验证了更改后的YARN框架性能提高。

# 基于RDMA的Shuffle算法改进

该章首先介绍了Shuffle的特点，然后分析Shuffle的缺点，并用RDMA提高网络传输性能，并改进Shuffle算法来适应RDMA技术。

## Shuffle介绍

如第一章所示，我们已经介绍了Shuffle的过程，一共分为6个过程。Shuffle横跨map 端和reduce 端，不过一般我们只考虑reduce 端的shuffle过程，主要的就是copy过程，负责从map 端的中间结果中拷贝数据，该过程具有很高的随机I/O读写请求，该阶段在数据量较大的情况下很消耗时间。下面我们通过图3.1 来介绍WordCount 的Shuffle过程。从图中我们可以看到WordCount 的主要流程。



图3.1 WordCount Shuffle过程

我们用Ganglia工具[28]来监控Hadoop集群执行WordCount程序，数据来源用RandomTextWriter程序自动生成50G 数据。集群运行新能如图3.2所示。



图3.2 集群运行jobs性能示意图

从图中我们可以看出当集群具有很高的带宽利用率时，CPU利用率变的比其他时间更低，内存也没有有效的利用。这就意味着浪费了集群的物理资源：带宽，内存和cpu。 通过Ganglia监控工具以及其他一些linux 工具，执行一系列jobs，我们分析了影响Hadoop性能的瓶颈主要有：

1. **Reduce端合成了太多的阶段**

Reduce端主要包含两个独立的阶段，Shuffle阶段和Reduce阶段。Shuffle 阶段请求map中间数据，是I/O密集型和网络密集型阶段。而reduce 函数的执行时cpu密集型和内存密集型的，因为它需要分配足够的内存来缓冲并排序数据，输出最后文件[29]。 此外，reduce会一直等到Shuffle结束才开始执行，这意味着在Shuffle阶段浪费了cpu资源；而在reduce阶段会浪费网络资源，因为没有Shuffle 过程。如果数据量特别大的情况下，Shuffle过程的时间接近于reduce的时间，这样会大大浪费集群的资源。

1. **随机I/O请求导致Shuffle阶段的低效率**

每个reduce任务都要先拷贝数据通过HTTP协议，来自reduce任务的请求大部分是随机的，并且map中获取数据的偏移地址也是随机的。对于一个map任务而言，需要从不同的磁盘上的中间数据读取块数据，发送特定的请求到特定的reduce任务，每个请求都会触发大量的I/O读写时间。总的来说，shuffle阶段会导致大量的磁盘数据随机I/O读写请求。

总而言之，Shuffle阶段带来了以上多方面的缺点，我们通过更改Shuffle阶段来提高集群性能。

## 3.2 RDMA介绍

RDMA（Remote Direct Memory Access）连接线在高性能系统中被广泛使用[30]。RDMA 通过网络把数据直接传入到计算机的内存区，将数据从一个系统快速传输到远程系统内存中，而不对操作系统造成任何影响，这样就不需要利用多少计算机的处理能力。它消除了外部存储器复制和文本交换操作，因而能腾出总线空间和CPU周期用于改进应用系统性能。

RDMA的工作过程如下所示：

1. 当一个应用执行RDMA读或写请求时，不执行任何数据复制。在不需要任何内核参与的条件下，RDMA请求从运行在用户空间中的应用中发送本地网卡。
2. 网卡读取缓冲的内容，并通过网络传送到远程网卡。
3. 在网络上传输的RDMA信息包含目标虚拟地址、内存钥匙和数据本身。请求完成既可以完全在用户空间中处理（通过轮询用户级完成排序），或者在应用一直睡眠到请求完成时的情况下通过内核内存处理。RDMA操作使应用可以从一个远程应用的内存中读数据或向这个内存写数据。
4. 目标网卡确认内存钥匙，直接将数据写入应用缓存中。用于操作远程虚拟内存地址包含在RDMA 信息中。

RDMA一般都是搭建在InfiniBand上的，InfiniBand架构是一个新的工业标准架构的I/O 服务器和服务器内的通信[31]。InfiniBand是一种基于交换的串行I/O 互连体系结构，具有高带宽，低延时的优点[32]。RDMA直接内存交换示意图如图3.3 所示。



图3.3 RDMA过程图

## 3.3 Shuffle改进

基于上面分析的Shuffle缺点，本文改进了Shuffle过程。Shuffle 过程的主要问题是：频繁的磁盘随机I/O读写请求。我们把Shuffle 独立出来做成一个service，该service 集成了有效的读策略并且保持map任务，API 等等不变。本文针对第二代Hadoop做的改进，而文章[33]则是在MRv1上改进。本文的改进主要有以下贡献：

1. 把Shuffle阶段从reduce端提取出来，并且把它实现为一个通用的service。这个service和reduce端异步执行，这样更能有效的利用集群的物理资源。
2. 在Shuffle service中集成顺序读策略，这个改进会大大减少随机I/O请求的数量，从而提高shuffle中的数据传输比率。

随机I/O请求问题有很多解决方法，文章[34]介绍了用HBase 作为后台存储系统来提高性能，但该文章不能提高MapReduce 的性能。另外，文章[35-36]使用其他方法取代对中间结果按键值排序，这些方法依然不能解决Shuffle的频繁随机I/O读写请求。下面我们将介绍我们的Shuffle 设计以及实现。

### 3.3.1 Shuffle设计

我们的Shuffle设计主要通过三个特点来提高性能。

**（1）从reduce中划分出shuffle阶段：**

这个阶段把shuffle的一系列过程，如copy，merge从reduce中切分出来，这就意味着老的reduce任务将分成一个shuffle任务和一个reduce任务。分出来的shuffle依然负责为reduce任务准备输入数据。shuffle任务的设计如图3.4所示。从图中我们可以看出，shuffle任务和reduce任务位于同一个NodeManager中，我们可以并行的运行这两个tasks，从而可以传输更多的数据以及提高了资源利用率。虽然把shuffle分裂出来可以提高网络和资源利用，但是reduce的随机请求不会变，因为从map端到shuffle task再到reduce task的网络传输并没有改变。



图3.4 新老Shuffle对比

（2）**把Shuffle阶段作为service：**

一个NodeManager上通常会运行多个reduce tasks，每个reduce task 对应一个shuffle task，这样会大大的占用资源，使reduce tasks的并行度降低，基于上一阶段，我们把shuffle 阶段合成一个shuffle service，用来出来同一NodeManager 上的所有reduce tasks。该结构如图3.5 所示。该service组件包含了shuffle 的各个子过程，并为该NodeManager上所有reduce tasks 提供服务。由于同一个NodeManager 上的所有reduce tasks共用一个节点，这就意味着从map端拷贝数据到该NM上的所有reduce tasks 的随机I/O请求可以合并到一个上，减少了请求的数目。Shuffle service作为一个轻量级的service，只在reduce 需求时传输数据。此外在shuffle service 中组合了有效的读写策略来减少磁盘随机读写。基于上述的优点，shuffle service具有了更加复杂的连接和buffer管理。该阶段在NodeManager中开启了Shuffle服务，但是多个Shuffle服务也需要进行通信，因此我们在Master节点启动一个ShuffleManager服务，用于Shuffle之间的通信。



图3.5 Shuffle Service

**（3）Shuffle service结合的I/O读写策略：**

这阶段我们的策略是当一个节点上的reduce task 有I/O请求时，保存该请求，并设定间隔时间，该间隔时间用来等待该节点上其他reduce tasks发送I/O请求，然后更加map输出文件的偏移量和reduce tasks 的序号来排序并组合这些请求，最后NodeManager读取信息并顺序的返回给reduce tasks。在该阶段的传输数据我们使用RDMA协议来直接内存拷贝，加快了数据传输速度。我们只需要把涉及TCP/IP Socket 传输的代码改成RDMA传输的代码就行。

### 3.3.2 Shuffle实现

从源码的角度来看，Hadoop的Shuffle主要有两个类：一个是ShuffleHandler类，位于NodeManager 内，该类继承了AuxiliaryService 服务类，作为一个服务启动在NM内；另一个是Shuffle类，位于reduce task端，用于获取数据并运行reduce tasks。当集群上有job运行时，shuffle service将被启动，然后它作为一个后台进程位于NodeManager 内，处理上面所有的reduce tasks请求。在ShuffleHandler类中，我们用一个SequenceShuffle 类来保存一个reduce请求，SequenceShuffleManager类用来保存所有的SequenceShuffle请求，然后该类对这些请求进行合并，并构造返回信息给reduce。该结构的示意图如图3.6所示。此外，还有一个ShuffleManager 服务，用于各个NodeManager之间的Shuffle 通信。



图3.6 Shuffle Service I/O示意图

总的来说，我们的Shuffle改进主要分为两个方面：（1）ResourceManager中的ShuffleManager服务；（2）NodeManager中的Shuffle服务。整个Shuffle改进的架构如图3.7 所示。



图3.7 Shuffle改进架构

## 3.4 实验结果与分析

该实验依然搭建在第二章介绍的集群上，集群配置如表2.3所示。本实验中用了RDMA技术，具有40GB 的速度。同样的，我们通过两种jobs：WordCount cpu型程序和TeraSort I/O型程序来分别测试Shuffle 的性能。具体实验如下所示：

1. **WordCount**

该实验我们使用WordCount基准程序来测试，如图3.1中所示的WordCount示例，我们知道WordCount的具体过程。为了产生尽量多的远程拷贝数据，我们配置Hadoop的副本为1。WordCount程序的输入文件由RandomTextWriter程序自动生成，一共50G的数据，分为50个文件，每个文件1G。我们针对不同的reduce tasks的数目来测试性能，结果如图3.8所示。从图中我们看出，经过改进后的Shuffle运行的WordCount比没有改进前的提高了性能。



图3.8 WordCount性能比较

上面我们使用了单个WordCount程序来验证性能，现在我们同时运行5个jobs 来验证集群整体性能，运行结果如图3.9所示。图中显示了针对同种WordCount，我们的资源利用率有了提升。



图3.9 WordCount集群运行性能

1. **TeraSort**：

该实验我们使用TeraSort来测试Shuffle 性能。TeraSort 的数据来源于TeraGen程序。该程序专门用来生成适应于TeraSort 程序的数据，一共20G（因为TeraSort运行较慢，所以数据量小点）。因为我们只关注Shuffle 的过程，所以我们同样针对不同的reduce tasks的数目改变来测试性能，结果如图3.10 所示。图中可以看出我们的Shuffle过程对TeraSort程序运行时间有很大的改进。



图3.10 TeraSort性能比较

我们已经使用了5个同种的jobs来测试集群的性能，在该阶段，我们使用2个TeraSort和3个WordCount 来测试集群，结果如图3.11所示。图中显示了不同种jobs：WordCount和TeraSort同时运行，集群的性能。从图中我们看出，同时运行的jobs中TeraSort首先运行，该程序是基于I/O的，所以I/O读写性能很高，而后面的WordCount是基于CPU的，所以CPU利用率比较高，其I/O读写性能要比TeraSort要低很多。同时，我们也可以看出在I/O较高的地方CPU利用率也相应有所提高。



图3.11 混合jobs运行性能

## 3.5 本章小结

在本章，我们首先论述了Shuffle的缺点，并介绍了RDMA技术，然后我们通过对Shuffle做一些更改，以及应用RDMA技术来提高数据传输速率，我们主要从把Shuffle独立成一个service和优化了读写策略两个方面来提高Shuffle性能，最后我们通过WordCount和TeraSort两个程序以及Ganglia监控软件来检测集群性能。

# 基于改进的YARN平台的PKTM系统

## PKTM介绍

Kirchhoff叠前时间偏移是地震数据处理中最耗时的常用模块之一，该算法是地震数据处理行业中最广泛采用的成像方法[37-38]。Kirchhoff 叠前时间偏移假定已知地震道数据，对于所有的反射点，从激发点到接收点的传输时间，已知地震道数据的激发点和接收点的坐标，在传输时间内，反射点的轨迹如图4.1 所示。利用叠前时间偏移可以有效地解决交叉地层速度的矛盾性问题，通过比较图中常规速度谱和叠前时间偏移速度谱，可以看到经过处理后，速度谱能量更加集中，速度拾取矛盾也可以得到有效解决。Kirchhoff 算法如算法6所示。



图4.1 地震成像示意图



## 4.2 PKTM分布式算法改写

本文对kirchhoff算法在Hadoop和Spark两个平台上都进行了改进，集群中Hadoop和Spark两个框架都是搭建在Yarn平台上的。下面我们分别介绍两个框架的更改。

### 4.2.1 Hadoop上的Kirchhoff算法实现

Hadoop算法的设计主要是Map和Reduce的过程，算法主要流程如下所示：

1. 分析系统环境：根据Hadoop的官方介绍[4]，一个节点上的mapper 数目在10~100之间的效果最好，我们可以获得集群的环境配置，如：集群节点个数，每个节点的内存，cpu总数，每个cpu的核数，每个核的线程数。
2. 自定义输入阶段：在Hadoop中，每个输入文件块对应一个mapper。Hadoop提供了一个默认的分块机制，128M 分为一个块。我们通过中心点文件对输入道进行划分，由于中心点文件大小比较小，所以我们要自定义mapper数量。Hadoop提供了重写FileInputFormat类来进行逻辑划分输入文件为split，每个split对应一个mapper。因此我们只要设置了每个split的大小就可以确定mapper的数量。首先在FileInputFormat中获取所有split的总大小：，。在Hadoop 中，Reduce 任务的个数是由用户设置的，在mapper 任务没有结束时，reduce 就会被启动，因此，reduce占用资源会影响mapper的数量，基于这些考虑设置split大小：

是一个可控参数，保证mapper的数量是集群资源容纳mapper数量的倍数，这样每次mappers都可以并行执行，不会剩余很少的mappers最后运行。最后我们通过设置mapreduce.input.fileinputformat.split.maxsize为来划分输入文件。另一个限制是，保证split的大小为一道数据大小的倍数，防止切分一道数据。通过输入split，我们读取输入道数据，形成<key,value> 键值对，并发送到每个mapper 中进行处理。key 代表输入道在文件中的道号，value代表每个输入道对应的中心点坐标。

1. Map阶段：对split传过来的输入道数据进行处理，读取HDFS上的速度数据，炮点坐标数据和检波器坐标数据，每个mapper都是并行的进行处理，这加快了运算的速度。由于每个输入道都会产生一定量的输出道，每个不同的输入道产生的输出道有很多重合，因此，我们再mapper 中用一个HashMap 保存输出道的数据，相同的输出道被叠加到同一个value 上，节省了数据传输时间。另一点是，每个输入道产生输出道会进行多次循环，大大的浪费了时间，因此在每个mapper 中我们用多线程来处理输入道数据，我们设置每个mapper数据的线程数为(threads-2)，如果系统支持超线程，则设置每个mapper 中的线程数为2\*(threads-2)。最后一个策略是将输出道的数据写入本地文件，减少数据的发送量，减轻网络带宽。将一个mapper中的输出道写入二进制文件，保存输出道的道号和文件名、偏移量，格式如：<key,filename#offset>，只发送这些键值对到Reduce 任务。
2. Combine阶段：在Combine阶段，对同一节点上的mappers 产生的数据进行聚合，同一个key的value 值进行聚合，这将会大大减少一个节点上的网络传输量，从而提高程序性能。
3. Partition阶段：由于mapper产生的输出道的key 值为输出道道号，Hadoop会自动按文本来排序，而不是按照数字大小排序，所以，在mapper发送到reduce 端时，进行一个映射，把道号按照reduce 任务的大小进行划分，reduce id小的对应道号小的输出道。映射公式为：

这个策略会减少最后整个输出道的排序时间，只需要在每个reduce中进行排序，这同时也进行了并行的排序。

1. Reduce阶段：对mapper中传来的输出道<key,filename#offset>键值对，我们根据键值对读取mapper生成的文件，进行聚合相同的key值，对key 值相同的value 进行叠加。
2. Output阶段：在Hadoop中，每个reduce 会对应一个输出文件，Hadoop 中默认的是把reduce的输出键值对按文本的形式输出到文件，这里我们对reduce 重写输出文件类：FileOutputFormat，在每个输出类中，我们获取键值对并排序，按照排好的序列将value 值写入二进制文件，并把文件名用最小的key 值命名。
3. 成像步骤：根据reduce阶段生成输出文件，我们根据文件名中的最小key 值进行排序，然后依次把所以的二进制文件聚合到一个image二进制文件中。最后删除所有的中间文件以节省空间。

Hadoop上Kirchhoff算法结构图如图4.2所示。



图4.2 Hadoop上Kirchhoff算法的结构图

### 4.2.2 Spark上的Kirchhoff算法实现

Spark算法的设计主要是Map和Reduce的过程，算法主要流程如下所示：

1. 获取集群环境阶段：Spark提供了newAPIHadoopFile方法来获取HFDS 上的输入文件，该方法读取HDFS 文件产生的<key,value> 键值对生成一个RDD数据集， key代表的是输入道道号， value代表的是输入道对应的中心点坐标。一个RDD记录就代表一系列输入道，该RDD的分区会默认是HDFS 上输入文件的分块数。RDD 的一个分区就对应了一个executor （相当于mapper），多个executor分布在不同节点上并行执行。此外，Spark 提供了一个spark-submit 脚本来提交用户程序。该脚本可以通过命令行设置executor的个数N，每个executor 使用的内存量以及每个executor使用的CPU核数，因此我们只需要读取这些参数就可以知道集群的环境。这种设置非常方便，从这些参数中我们就可以知道RDD 的分区可以设置为：

是一个可变参数，为一个整数，表示分区数为executor 数的倍数，这样就保证了每次都并行处理分区，不会遗留剩余的分区被executor 单独处理。N代表这个集群上的executor个数，一般1~2 CPU 核一个executor，如果内存特别小，则会影响executor数量，因此选择CPU核心数和内存划分数最小的一个作为executor的数量。

1. 输入阶段：Spark提供的newAPIHadoopFile方法可以读取HDFS 文件，并且能通过自定义的输入格式来读取文件，我们定义一个输入文件类，把输入道文件读取成<key,value>键值对，并且保证每道数据都是一个整体，不能切分一道数据，方便计算。key代表输入道道号， value代表输入道道号对应的中心点坐标。通过读取HDFS，Spark返回一个RDD数据集，RDD 的每条记录包含一系列输入道，因为RDD支持手动分区，根据上面的公式，我们把RDD 分区以在executor 上并行执行。另外，RDD还支持persist操作，可以把中间数据缓存在内存中， persist提供了多级缓存机制，可以把数据完全缓存在内存中，也可以缓存在硬盘上，或者两种都使用。不仅如此， persist操作还支持缓存双份在内存或硬盘中，加速了并行读写速度。persist在第一次计算后就缓存了数据，等下次计算时直接读取缓存，加快了执行速度。因此，我们把从HDFS中读取的输入道数据缓存在内存和硬盘中（防止过大，内存不足），当再次读取数据时加快速度，并且能在后续RDD失败时，能快速重建该RDD。
2. FlatMap阶段：在Spark中，RDD的每个分区都会进行FlatMap操作，一次一个executor计算一个分区，当分区计算完成后继续计算其他分区，多个executor 并行执行。在FlatMap阶段，获取<key,value>键值对中的输入道道号和中心点坐标，对每个输入道读取HDFS上的速度文件，炮点坐标文件和检波器坐标文件来计算出输出道的数据。每个输入道对应一个孔径，每个孔径中有一系列输出道，相邻的输入道产生的输出道有很多交集，因此我们在FlatMap中使用一个HashMap 来保存输出道，相同的输出道叠加到一起。这有利于减少节点之间和节点内的通信量。因为每个输入道会产生大量的输出道，一个FlatMap包含了大量的输入道，因此我们用个多线程来并行加速一个RDD分区，如果机器系统开启了Hyper-Threading，则设置线程数为2\*(threads-2)，如果Hyper-Threading关闭，则设置线程数为 (threads-2)。在Hadoop中，当mapper 任务运行到一个合适的百分比时，系统会启动reduce任务来收集mapper任务产生的中间结果，reduce 任务会一直占用资源，这会影响后期mapper的数量。使得程序性能下降。而在Spark 中，当mapper 任务结束后会返回一个RDD，然后这个RDD 被应用到reduce 操作，因此，系统不会在mapper 没执行完时进行reduce操作，这使得mapper 可以完全并行执行，提高了程序的性能。另外，在Spark中，一旦RDD分区后，分区信息会一直保留着，也就是说，在执行reduce操作时，RDD 的分区数依然是mapper规定的分区数。如果mapper 产生的分区数不适合reduce 任务，则可以使用repartition函数对RDD 进行重新分区来适应reduce任务。在这阶段，对输出道进行聚合生成<key,value>键值对， key代表输出道道号， value代表输出道对应的成像数据。返回这些键值对生成的RDD。
3. Partition阶段：首先，我们获取输出道的总大小onx，然后根据输出道的key值进行划分，保证小的key值对应同一个reduce 任务，这样就避免了最后成像数据的排序时间，只需要在每个reduce 任务中并行排序就行。我们根据reduce任务的个数来确定每个key的映射情况，公式如下：
4. ReduceByKey阶段：该阶段接收来自映射的FlatMap 的<key,value>键值对，对每个相同的key值的value进行叠加，生成图像数据。 reduceByKey方法的另一个优化就是：它会先把同一个节点的相同输出道道号\emph{key}进行聚合，然后再传递到reduce任务中，这减少了节点间的通信量。reduceByKey方法将返回一个RDD，保存了输出道道号和成像数据组成的键值对。每个reduce任务会并行的运行在不同的executor上，一个reduce任务完成后就会执行下一个reduce任务，只要保证reduce分区是reduce 任务的倍数，这样才能保证所有的reduce 任务并行执行，不会浪费资源。
5. SortByKey阶段：在Reduce阶段生成的RDD进行排序操作，因为生成的成像文件是按照输出道道号的大小进行排序的，运用Spark的sortByKey函数可以对每个RDD 分区中的输出道进行聚合。所有的分区都是并行排序，加快了排序时间。最后返回排序好的RDD。
6. 成像阶段：根据排序好的RDD进行写文件到HDFS，Spark提供了saveAsNewAPIHadoopFile函数对RDD 分区分别写入文件，产生的文件名由该分区中最小的输出道道号的值来命名，最后根据文件名来合并所有的成像文件，生成一个二进制成像文件，并删除中间文件。

Spark上Kirchhoff算法结构图如图4.3所示。



图4.3 Spark上的Kirchhoff算法实现

## 4.3 实验结果

在该实验中，我们使用了上述的集群，集群配置如表2.3所示。PKTM是地质方向的一个成像算法，所需的数据量庞大，下面我们首先介绍PKTM的数据文件。

### 4.3.1 数据准备

PKTM主要包括三个方面：数据预处理，数据迁移，输出成像。PKTM 使用两种输入数据文件格式，包括“meta” 文件和“data”文件[38]。在本程序中输入文件主要包括：输入道meta 文件（shot.meta），输入道震源坐标文件（fsxy.meta），输入道检波器坐标文件（fgxy.meta），输入道中心点坐标文件（fcxy.meta），速度信息文件（rmsv.meta）。每个meta 信息文件都对应一个data文件，用于保存数据的文件：shot.data, fsxy.data, fgxy.data, fcxy.data, rmsv.data。

### 4.3.2 实验结果

在该实验中，我们主要测试两个方面：第一个方面，PKTM算法的改进在Hadoop和Spark 上的效率提高，该实验主要为了验证我们改进的分布式PKTM算法的效率，我们实验了Hadoop和Spark两个版本（为了方便第二方面实验）；第二个方面，我们的资源调度器和Shuffle 改进对PKTM 算法的性能提高，该实验验证了我们对YARN 调度器和Shuffle 的改进对不同框架（Hadoop 和Spark）的性能。下面分别介绍这两个方面的实验。

**1.PKTM算法在Hadoop和Spark两种框架上的改进：**

因为我们实现了两种框架的PKTM算法，因此我们分别测试Hadoop和Spark 的性能。通过我们的测试发现分布式PKTM算法比cpu版本算法效率提高了20倍。因此我们只测试一些参数对算法的影响，在实验中我们使用小数据集进行测试。

**在Hadoop上的测试：**

1. 我们首先测试mapper任务的container内存量变化给程序性能带来影响，测试结果如图4.4(a)所示。我们可以看出当内存扩展到一个阀值后，mapper的数量就由于内存的增大而变少，因此执行时间就会变长。
2. 最后，我们测试reduce任务的个数对程序性能的影响，结果如图4.4(b) 所示。当reduce任务过多时，程序性能将趋向于平稳，因为reduce过多的话，有些reduce任务中将没有键值对处理，会很快结束。



图4.4 Hadoop上的测试

**在Spark上的测试：**

在Spark中，我们只测试两个方面，每个executor的内存量和RDD的分区数，该Spark实验使用的是Spark on YARN 的模式。

1. 我们首先改变container内存的配置来测试程序的性能，测试结果如图4.5(a) 所示。图中显示，container的内存量变大，性能会变差，因为大内存导致了并行度的降低。
2. 我们测试输入道数据的RDD分区，每个分区对应一个mapper，因此可以看成是mapper 的数量变化，测试结果如图4.5(b)所示，当RDD的分区数是集群所能容纳的倍数时，再增加分区，运行时间将趋向平稳。



图4.5 Spark上的测试

**2.调度器和Shuffle改进后对PKTM的性能：**

该阶段我们主要比较改进后的集群和原先集群性能的差别，主要从Container内存和Reduce数目来测试，我们首先规定Container内存不变，然后测试不同Reduce数目的性能比较，然后再改变Container内存的大小，如下图4.6所示。从图中看，在第一幅图中，我们设置Container的内存为0.5G，在不同的Reduce数目下kirchhoff程序运行的时间对比。依次类推，第二幅图中，设置Container内存为1G，第三、四幅图则为1.5G、2G。



图4.6 Kirchhoff程序在不同平台的性能对比

此外，我们继续测试Container的内存直到3.5G，测试结果如下图4.7所示。从图4.7左所示，我们对不同的Continer内存(0.5,1,1.5,2,2.5,3,3.5)G进行测试，图左中绿色三角代表的是在原始YARN环境下，Kirchhoff程序在Hadoop框架下的运行时间；而红色圆点则代表的是Hadoop框架下Kirchhoff在更改后的YARN上的运行时间；最后黄色的点表示在Spark框架下Kirchhoff在更改后的YARN上的执行时间。图4.7右图是左图的右侧视图，从图中可以明显的看出Spark和Hadoop框架下的Kirchhoff程序比原始框架下的程序运行时间大大的减少了。





图4.7 性能比较

最后，我们通过Ganglia监测工具来测试5个Kirchhoff程序运行时的内存，CPU，I/O使用情况。测试结果如图4.8所示，从图中我们可以看出，Kirchhoff程序的CPU、内存和I/O等方面都有不错的性能。



图4.8 Kirchhoff程序运行情况

## 4.4本章小结

在本章中，我们主要实现了PKTM算法，首先我们相比于CPU算法实现了基于Hadoop框架和Spark框架的两种分布式PKTM算法，然后为了验证我们更改后YARN的性能，我们分别搭建Hadoop框架和Spark框架在YARN上，分别运行各自对应的PKTM算法版本，然后通过对不同的Container内存和不同的Reduce数目来验证性能，从实验中我们可以看出我们的改进有一定的性能提高。

# 总结与展望

## 总结

本论文中，我们主要对大数据平台YARN框架做了更改以提高集群性能并用实际应用来测试更改后集群的性能。本文主要做了一下几个工作：

1. 更改YARN框架的资源调度器，构建了一个MKP多背包问题来解决YARN资源调度的问题，然后在调度器中用人工鱼群算法来分配tasks，提高了集群的资源利用率和减少了jobs 的运行时间。
2. 改进了Reduce端的Shuffle阶段，把Shuffle分裂出来，做成服务，并用顺序读的策略，减少磁盘随机读写，并且用上RDMA的技术加速网络数据的传输，减少了Shuffle过程的时间。
3. 将PKTM算法由CPU版改为分布式版本，在Hadoop和Spark上都进行了实践，然后将该算法运行在改进的YARN 上，Spark使用的也是YARN模式，从不同的框架（Hadoop 和Spark）上来测试YARN的改进性能。

## 展望

论文中通过一些改进提升了集群的性能，但是还有很大的改进空间，如下所示：

1. 在YARN的资源调度器中，实际的模型是多资源的多背包问题，而我们的模型只是简单的多背包问题，这就意味着我们可以用其他启发式的算法来解决该模型，此外，我们还可以用一些学习的算法来不断的预测tasks的运行时间，以此来达到最佳的优化结果。
2. 在RDMA的应用方面，不仅可以更改Shuffle阶段，还可以把整个RPC通信框架改为RDMA 传输[39]，理论上会大大缩短网络传输时间。
3. PKTM算法是cpu密集型的算法，该算法包含了大量的计算，我们可以结合Hadoop和GPU 来进行高速计算，把GPU 计算模块打包成动态链接库来在Hadoop上分布式计算。
4. 将Hdfs替换为Alluxio（由Tachyon改名而来）[40-41]等内存数据集，用来提高运算效率，大大的缩短集群的数据传输时间。

# 参考文献

1. 孟小峰, ., 2013. 大数据管理: 概念, 技术与挑战[j]. 计算机研究与发展50(1), 146–169.
2. S, L., 2012. The age of big data[j]. New York Times 11.
3. Dean, J., Ghemawat., S., 2008. Mapreduce: simplified data processing on large clusters. Commun. ACM 51(1), 107–113.
4. Apache, b. Apache hadoop. http://hadoop.apache.org.
5. ChengXueQi, JinXiaoLong, W.e., 2014. Summary of the big data systems and analysis techniques[j]. Journal of Software in chinese 25.
6. Gu L, L.H., 2013. Memory or time: Performance evaluation for iterative operation on hadoop and spark[c]. High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC EUC) , 721–727.
7. Zaharia M, Konwinski A, J.A.D.e.a., 2008. Improving mapreduce performance in hetero-geneous environments[c]. OSDI 8(4), 7.
8. Vavilapalli V K, Murthy A C, D.C.e.a., 2013. Apache hadoop yarn: Yet another resource negotiator[c]. Proceedings of the 4th annual Symposium on Cloud Computing. ACM 5.
9. Ghemawat S, Gobioff H, L.S.T., 2003. The google file system[c]. ACM SIGOPS operating systems review. ACM 37(5), 29–43.
10. D, B., 2008. Hdfs architecture guide[j]. Hadoop Apache Project 53.
11. Apache, c. Apache spark. <http://spark.apache.org>.
12. M. Zaharia, M. Chowdhury, S.S.M.J.F., Stoica, I., 2010. “spark: Cluster computing with working sets,”. In HotCloud.
13. Zaharia M, Chowdhury M, D.T.e.a., 2012. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[c]. Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association 2, 2.
14. Wang Y, Xu C, L.X.e.a., 2013. Jvm-bypass for efficient hadoop shuffling[c]. Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on. IEEE , 569–578.
15. H, H., 2011. Hadoop performance models[j]. arXiv preprint arXiv 1106, 0940.
16. 彭辅权, 金苍宏, .., 2012. Mapreduce 中shuffle 优化与重构[j]. 中国科技论文7(4), 241–246.
17. Assmann S F, Johnson D S, K.D.J.e.a., 1984. On a dual version of the one-dimensional bin packing problem[j]. Journal of algorithms 5(4), 502–525.
18. R, R.G., 1998. An improved genetic algorithm for the multiconstrained 0-1 knapsack problem[c]. Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on. IEEE , 207–211.
19. Browning, T.R., Yassine, A.A., 2010. “resource-constrained multi-project scheduling: Priority rule performance revisited,”. International Journal of Production Economics 126, 212–228.
20. J.Blazewicz, J.K.L., Kan, A., .“scheduling subject to resource constraints: classification and complexity”. Discrete Applied Mathematics.
21. Kolisch, R., Hartmann, S., 1999. Heuristic algorithms for the resourceconstrained project scheduling problem: Classification and computational analysis. Springer.
22. P. Brucker, A. Drexl, R.M.K.N., Pesch, E., 1999. “resource-constrained project scheduling: Notation, classification, models, and methods,”. European Journal of Operational Research 112, 3–41.
23. Azad M A K, Rocha A M A C, F.E.M.G.P., 2014. Improved binary artificial fish swarm algorithm for the 0╟1 multidimensional knapsack problems[j]. Swarm and Evolutionary Computation 14, 66–75.
24. Qin L, Z.K., 2015. Study on multi-knapsack problem based on improved artificial fish school algorithm[c]. First International Conference on Information Sciences, Machinery, Materials and Energy. Atlantis Press.
25. Wang C R, Zhou C L, M.J.W., 2005. An improved artificial fish-swarm algorithm and its application in feed-forward neural networks[c]. Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on. IEEE 5, 2890–2894.
26. Ma X, L.Q., 2010. Artificial fish swarm algorithm for multiple knapsack problem[j]. Journal of Computer Applications 30(2), 469–471.
27. Yao Y, Wang J, S.B.e.a., 2014. Haste: Hadoop yarn scheduling based on task-dependency and resource-demand[c]. Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on. IEEE, 184–191.
28. Ganglia. Ganglia. <http://ganglia.info/>.
29. Zaharia, M., B.D.S.S.J.E.K.S.S.S.I., 2009. Job scheduling for multi-user mapreduce clusters. UCBerkeley Technical Report UCB/EECS-2009-55.
30. Woodall T S, Shipman G M, B.G.e.a., 2006. High performance rdma protocols in hpc[m]. Recent Advances in Parallel Virtual Machine and Message Passing Interface. Springer Berlin Heidelberg, 76–85.
31. F, P.G., 2001. An introduction to the infiniband architecture[j]. High Performance Mass Storage and Parallel I/O 42, 617–632.
32. Lu X, Islam N S, W.u.R.M.e.a., 2013a. High-performance design of hadoop rpc with rdma over infiniband[c]. Parallel Processing (ICPP), 2013 42nd International Conference on. IEEE, 641–650.
33. Li J, Lin X, C.X.e.a., 2013. Improving the shuffle of hadoop mapreduce[c]. Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on. IEEE 1, 266–273.
34. Dhruba Borthakur, Jonathan Gray, J.S.S.K.M.N.S.H.K.K.R.D.M.A.M.S.R.R.S.A.A., 2011. Apache hadoop goes realtime at facebook. Proceedings of the 2011 ACM SIGMOD International Conference on Management of data.
35. Abhishek Verma, Nicolas Zea, B.C.I.G.R.H.C., 2010. Breaking the mapreduce stage barrier. Proceedings of the 2010 IEEE International Conference on Cluster Computing, 235–244.
36. E.Mazur, B.Li, Y.D., Shenoy, P., 2010. Towards scalable one-pass analytics using mapreduce. Technical report, Technical report, UMass Amherst.
37. Shi X, Wang X, Z.C.e.a., 2009. Practical pre-stack kirchhoff time migration of seismic processing on general purpose gpu[c]. CSIE 2, 461–465.
38. WangGang, TangJie, W., 2014. Gpu-based cluster framework[j] in chinese. Computer Science and Development 24(1), 9–13.
39. Lu X, Islam N S, W.U.R.M.e.a., 2013b. High-performance design of hadoop rpc with rdma over infiniband[c]. Parallel Processing (ICPP), 2013 42nd International Conference on. IEEE, 641–650.
40. Apache, a. Apache alluxio. <http://tachyon-project.org/>.
41. Li H, Ghodsi A, Z.M.e.a., 2013. Tachyon: Memory throughput i/o for cluster computing frameworks[j]. memory. cluster computing frameworks[J] 18, 1.
42. 张军华, 臧胜涛, 单联瑜, 等. 高性能计算的发展现状及趋势[J]. 石油地球物理勘探, 2010, 45(6): 918-925.
43. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.
44. Cafarella M, Cutting D. Building nutch: Open source search[J]. Queue, 2004, 2(2): 54.
45. Ghemawat S, Gobioff H, Leung S T. The Google file system[C]//ACM SIGOPS operating systems review. ACM, 2003, 37(5): 29-43.
46. Zaharia M, Borthakur D, Sen Sarma J, et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling[C]//Proceedings of the 5th European conference on Computer systems. ACM, 2010: 265-278.
47. Kumar K A, Konishetty V K, Voruganti K, et al. CASH: Context aware scheduler for Hadoop[C]//Proceedings of the International Conference on Advances in Computing, Communications and Informatics. ACM, 2012: 52-61.
48. Gupta S, Fritz C, Price B, et al. ThroughputScheduler: Learning to Schedule on Heterogeneous Hadoop Clusters[C]//ICAC. 2013: 159-165.
49. Lee M C, Lin J C, Yahyapour R. Hybrid Job-driven Scheduling for Virtual MapReduce Clusters[J].
50. Guo Y, Wu L, Yu W, et al. The Improved Job Scheduling Algorithm of Hadoop Platform[J]. arXiv preprint arXiv:1506.03004, 2015.
51. Zhang X, Hu B, Jiang J. An optimized algorithm for reduce task scheduling[J]. Journal of Computers, 2014, 9(4): 794-801.
52. <https://issues.apache.org/jira/browse/HADOOP-331>
53. <https://issues.apache.org/jira/browse/MAPREDUCE-64>
54. <https://issues.apache.org/jira/browse/MAPREDUCE-2354>
55. Condie T, Conway N, Alvaro P, et al. MapReduce Online[C]//NSDI. 2010, 10(4): 20.
56. Pavlo A, Paulson E, Rasin A, et al. A comparison of approaches to large-scale data analysis[C]//Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. ACM, 2009: 165-178.
57. Seo S, Jang I, Woo K, et al. HPMR: Prefetching and pre-shuffling in shared MapReduce computation environment[C]//Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on. IEEE, 2009: 1-8.
58. Costa P, Donnelly A, Rowstron A, et al. Camdoop: exploiting in-network aggregation for big data applications[C]//Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). 2012: 29-42.
59. Wang Y, Que X, Yu W, et al. Hadoop acceleration through network levitated merge[C]//Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2011: 57.
60. Guo Y, Rao J, Zhou X. iShuffle: Improving Hadoop Performance with Shuffle-on-Write[C]//ICAC. 2013: 107-117.
61. Ahmad F, Lee S, Thottethodi M, et al. MapReduce with communication overlap (MaRCO)[J]. Journal of Parallel and Distributed Computing, 2013, 73(5): 608-620.
62. Rizvandi N B, Boloori A J, Kamyabpour N, et al. MapReduce implementation of prestack Kirchhoff time migration (PKTM) on seismic data[C]//Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2011 12th International Conference on. IEEE, 2011: 86-91.
63. Dai H, Li X Y. Migration velocity analysis of C-wave using INMO-CIP gather of PKTM: A case study from the Gulf of Mexico[J]. Computers and Geosciences, 2003, 31(7): 891-899.
64. Li C, Wang Y, Yan H, et al. High performance kirchhoff pre-stack depth: migration on hadoop[C]//Proceedings of the Symposium on High Performance Computing. Society for Computer Simulation International, 2015: 158-165.
65. Zhang J, Xu W, Meng X, et al. A Data-ware Scheduling Framework on GPU Grid for Kirchhoff Prestack Depth Migration[C]//75th EAGE Conference & Exhibition incorporating SPE EUROPEC 2013. 2013.
66. Shi X, Li C, Wang S, et al. Computing prestack Kirchhoff time migration on general purpose GPU[J]. Computers & Geosciences, 2011, 37(10): 1702-1710.
67. Shi X, Wang X, Zhao C, et al. Practical pre-stack kirchhoff time migration of seismic processing on general purpose gpu[C]//2009 World Congress on Computer Science and Information Engineering. IEEE, 2009: 461-465.
68. Shi X, Li C, Wang X, et al. A practical approach of curved ray prestack Kirchhoff time migration on GPGPU[M]//Advanced Parallel Processing Technologies. Springer Berlin Heidelberg, 2009: 165-176.
69. Sun P, Shi X. An OpenCL Approach of Prestack Kirchhoff Time Migration Algorithm on General Purpose GPU[C]//Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2012 13th International Conference on. IEEE, 2012: 179-183.
70. Zhao C, Yan H, Shi X, et al. Decf: A coarse-grained data-parallel programming framework for seismic processing[C]//Computer Science and Software Engineering, 2008 International Conference on. IEEE, 2008, 3: 454-460.
71. Dai H. Parallel processing of prestack Kirchhoff time migration on a PC cluster[J]. Computers & geosciences, 2005, 31(7): 891-899.

# 科研成果

## 在校参加的研究工作

1．“十二五”国家科技重大专项专题“煤层气地震数据处理算法并行化及高效数据组织技术研究”（ZX05035-004-004HZ）

## 在校发表论文

2. Yang C, Tang J, Gao H, et al. Pre-stack Kirchhoff Time Migration on Hadoop and Spark[M]//Algorithms and Architectures for Parallel Processing. Springer International Publishing, 2015: 190-202.

# 致谢

时光荏苒，转眼间，研究生三年已过。在这临走之际，回首这三年的努力与欢笑，短暂而充实，这三年带给我的，不仅仅是学业上的进步，更是愉快的回忆和生活上的历练。回首往昔，我衷心感谢在这三年里指导和帮助过我的老师和同学们，伴随着他们的支持和鼓励，我才能顺利地完成研究生阶段的学习任务和研究工作。

首先我要感谢唐杰老师，唐老师温和谦逊，平易近人，在学术方面有着独到的见解、经验丰富，知识渊博，在课题的研究过程中给了我很多宝贵的建议和指导，在研究遇到困难时，不断的给予我帮助和鼓励。唐老师一向严以律己，宽以待人，他对待工作一丝不苟，潜移默化的影响着我们对待学习生活的态度。在论文的写作过程中，唐老师也给了我很多的指导，感谢唐老师对我的帮助。

在此，我还要感谢武港山老师，武老师是多媒体教研室主任，他学识渊博、治学严谨，将多媒体信息教研室管理的井井有条，为我们提供良好的环境。武老师还具有丰富的科研经验，给了很多帮助。

在研究生阶段的学习过程中，不仅两位老师给了我们很多的帮助和鼓励，同学们还给了我很过关心和支持。感谢1010、1011实验室的各位同学。有幸与师兄、师姐、师弟们共同学习与生活，让我的研究生生活如此多彩。

衷心祝两位老师工作顺利，身体健康，万事如意！也祝各位同学学业有成，前程似锦！

最后要感谢我的家人，感谢他们给了我无私的关怀和鼓励。在我的人生道路上给了我很多指点。祝愿他们永远健康快乐！