学校代号 10532　　学　　号 S141000896
分 类 号 TN393　　密　　级 普通

湖南大學
HUNAN UNIVERSITY

硕士学位论文

Hadoop 平台下音视频转码与优化

学位申请人姓名 杨竞
培 养 单 位 信息科学与工程学院
导师姓名及职称 李仁发 教授
学 科 专 业 计算机科学与技术
研 究 方 向 云计算
论文提交日期 2017 年 5 月 9 日

学校代号：10532

学　　号：S141000896

密　　级：普通

湖南大学硕士学位论文

Hadoop 平台下音视频转码与优化

学位申请人姓名：　杨竞

导师姓名及职称：　李仁发 教授

培 养 单 位：　信息科学与工程学院

专 业 名 称：　计算机科学与技术

论文提交日期：　2017 年 5 月 9 日

论文答辩日期：　2017 年 5 月 21 日

答辩委员会主席：　邝继顺 教授

Transcoding and Optimization of Audio and Video in Hadoop

by

Jing Yang

B.E.(Hunan University)2014

A thesis submitted in partial satisfaction of the

Requirements for the degree of

Master of Engineering

in

Computer Science and Technology

in the

Graduate School

of

Hunan University

Supervisor

Professor Renfa Li

May, 2017

湖 南 大 学

学位论文原创性声明

本人郑重声明：所呈交的论文是本人在导师的指导下独立进行研究所取得的研究成果。除了文中特别加以标注引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写的成果作品。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律后果由本人承担。

作者签名：　　日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解学校有关保留、使用学位论文的规定，同意学校保留并向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅。本人授权湖南大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。

本学位论文属于

1、保密□，在_____年解密后适用本授权书。

2、不保密☑。

（请在以上相应方框内打“√”）

作者签名：　　日期：　　年　　月　　日

导师签名：　　日期：　　年　　月　　日

摘　要

近年来随着物联网的高速发展，它产生的数据量也急剧增长，网络流媒体传输作为物联网行业的重要应用之一，其过程中产生的数据量是非常可观的。终端的异构性和网络的不稳定性，会导致对视频编码格式多样化的需求。海量多格式流媒体的存储与分析一体化是一个研究难点。传统的存储方式无法满足其需求，转码又是计算密集型任务，考虑到集中式系统难以扩展，分布式存储与计算方式从技术上克服了这个难题。Hadoop作为一种开源的分布式大数据存储与分析工具，凭借其易用性和稳定性得到了广泛的使用。然而基于流媒体视频数据的非结构化特性，直接将Hadoop分布式框架应用到网络流媒体的处理中将会遇到很多问题。

本文通过深入研究Hadoop框架和流媒体数据的特点，实现了Hadoop平台下海量音视频数据的转码，利用HDFS分布式文件系统存储海量多格式流媒体文件，通过MapReduce分布式计算框架对转码过程进行并行加速，结合container资源的分配和调度方式，提高资源的利用率，优化转码时间。主要工作及创新点如下：

提出了一种Hadoop下基于位置信息的视频分割方法，目前集群加速转码相关研究都是借助开源工具对视频进行预分割，增加了额外的文件读写开销。而且视频分割处理是固定在一个节点上完成的，当视频处理服务请求量过大时，该节点会成为系统性能的瓶颈。本文通过修改MapReduce getsplit函数中的FileSplit方法，读取数据包中的位置信息，通过在分割点位置前多读数据包的方式，避免了分割在关键帧处导致的转码后视频跳帧的情况。本方法实现了在Hadoop上的快速视频分割，避免了系统的性能瓶颈，优化了系统的转码时间。

提出了一种基于样本资源需求的容器资源配置方法，该方法解决了在集群变更的情况下，以最小的开销找到container最优配置的问题。基于样本资源需求的容器资源配置方法通过在样本集群中确定内存配置因子和CPU核配置因子，当集群发生变更的时候，只需在新集群上对样本split音视频分片进行转码，分析转码单个split分片内存和CPU资源的使用情况，通过基于样本资源需求的容器资源配置算法即可计算出新集群的最优并行度，从而对container进行配置。本方法可以很好地应用于虚拟化集群，同样也适用于物理机集群。实验结果表明，相较于直接使用Hadoop进行分布式转码，优化后的系统转码时间减少了25%左右，缓解了虚拟化集群内存过载的情况，CPU的资源利用率也有所提高。

关键词：媒体转码；Hadoop；FFmpeg；容器

Abstract

In recent years, with the high-speed development of the Internet of things, the amount of data produced is growing sharply, and network streaming media transmission as one of the important application of Internet of things industry, produce very considerable amount of data. The heterogeneity of the terminal and the instability of the network will lead to the need for diversification of the video encoding format. The storage and analysis integration of mass and multi-format streaming media is a research challenge. The traditional storage can not meet their needs, while transcoding is computationally intensive work, and centralized storage is difficult to extend. Distributed storage and computing technology is to overcome the problem. Hadoop, as open source distributed data storage and analysis tool, with its ease of use and stability has been widely used. Based on unstructured characteristics of streaming media video data, directly applying the Hadoop distributed framework to network streaming media processing will encounter many problems.

This paper thoroughly studies the Hadoop framework and the characteristics of streaming media video data, and implements the massive audio and video data transcoding in Hadoop platform. Using HDFS to store mass streaming media, using MapReduce to parallelly accelerate the process of transcoding, combined with the container resource allocation and scheduling, improves the utilization rate of resources and optimizes the transcoding time. The main work and innovation points as follow:

The paper proposes a video segmentation method based on the location information in Hadoop. At present most of the studies in cluster transcoding acceleration are using open source tools for audio and video pre-segmentation, which increases the additional file read and write overhead. And video segmentation processing is fixed on a node to complete, when the video processing service request is too large, the node will become a bottleneck in the system. The paper modifies the FileSplit method in getsplit function of MapReduce, obtaining the pos information in AVPacket. Through reading more data packet before the split point, avoid the segmentation in the key frames and frame jumping after video transcoding. The realization of the video segmentation in Hadoop, avoiding the performance bottlenecks, saving the overall transcoding time.

The paper proposes a container resource configuration method in Hadoop Transcoding cluster based on requirements of a sample split. The proposed method

solves the problem of finding the optimal configuration of container with the least expense when the cluster changes. The container resource configuration method in Hadoop Transcoding cluster based on requirements of a sample split determines the memory configuration factor and the CPU-vcore configuration factor in the sample cluster. Transcoding only one sample split and analyzing its memory and CPU requirements, and then calculating the optimal container resource configuration through the container resource configuration algorithm. The method applies both virtual cluster and physical machine cluster. The experiment results show that the optimized system , compared with the default one, is approximately 25% better in the transcoding time, mitigating the memory overload in the virtual cluster, and has a improvement in CPU usage in a certain extent.

Key Words: Transcoding; Hadoop; FFmpeg; container

目　录

插图索引

附表索引

第 1 章 绪 论

1.1 选题背景及意义

1991年，美国麻省理工学院的 Kevin Ash-ton 教授首次提出了“物联网”的概念[1]。物联网就是物物相连的互联网，是在互联网的基础上向物理世界的延伸和扩展，被称为继计算机、互联网之后，世界信息产业发展的第三次浪潮。物联网被视为下一个推动世界高速发展的重要生产力，引起了学术界和工业界的广泛重视，很多国家都斥巨资对其进行深入研究。2009年，IBM 首席执行官彭明盛首次提出了“智慧地球”这一概念[2]，建议政府投资新一代的智慧型基础设施，当年即被列为美国振兴经济的重点战略决策。欧盟、日本等国都十分重视物联网的发展，进行物联网相关技术和产业的整体布局。同年8月，温家宝提出了“感知中国”的概念，把我国物联网领域的研究和应用开发推向了高潮，将物联网作为新兴战略性产业写入政府工作报告，予以重点关注和推进。然而，物联网作为当下最热门的技术之一，在给社会带来巨大变革的同时，也给整个信息技术行业带来了前所未有的新挑战。

物联网是通过各种信息传感设备，实时采集任何需要监控、连接、互动的物体的各种信息。数据的采集和传输无时无刻不在进行，最终产生的数据量也将非常可观[3]。在整个物联网传输系统中，对于海量数据的集中管理是非常重要的，这样方便我们对历史数据以及当前数据进行对比分析，获取有用信息。如何有效地存储和管理这些来自物联网的数据，并对这些数据进行分析处理，获取其中潜在的价值变得尤为关键。

我国早在“十二五”期间就部署了物联网、云计算的相关专项，“十三五”期间持续推动大数据、云计算、物联网的广泛应用。其中视频监控尤其是网络流媒体的传输被列为重点建设项目[4]。网络流媒体传输作为物联网行业的重要一部分，其过程中产生的数据量是非常可观的，这些流媒体数据除了具备大数据的通用特征（5V，即数量（Volume）、多样性（Variety）、速度（Velocity）、价值（Value）以及真实性（Veracity））外，流媒体大数据还具有自身独特的属性，一是媒体大数据的多样性更加充分，媒体数据的来源，媒体格式，展示形式都表现出多样性[5]；二是媒体数据的非结构化特征，绝大多数媒体数据都是非结构化的。视频监控领域的存储设备需要具备容量大、可靠性高、持续性好等特点，而普通的关系型数据库根本无法存储来自监控设备的音视频数据。以磁盘阵列为主的集中式存储方式虽然安全性高、稳定性好，但是价格昂贵。集中存储只满足了

普通的存储需求，无法实现存储与分析的一体化。而且在实际应用中，集中式系统难以扩展，随着数据量的线性增长不足以满足处理能力的需求。分布式的存储与计算方式，从技术上解决了这个难题。

当下最流行的分布式计算方式有网格计算和高性能计算等，主要应用于电子政务、分子材料研究、石油物探、金融服务等众多需要数据密集型计算的领域。但是这些分布式的计算方式易用性低，需要开发者对整个框架有深入的理解，熟悉底层的通信机制。Google 公司对分布式存储与计算的研究做出了重大贡献，其两位创始人 Page 和 Brin 在创业初期时就设计了一个名为“Bigfdes”的文件系统，随后演变成为 GFS 分布式文件系统[6]，以满足大数据的存储需求，与 GFS 一起被提出的是 MapReduce[7]分布式计算模型。它们与用来存储结构化数据的 BigTable 一起，被并称为 Google 的“三驾马车”。为了应对大规模数据的并行处理，Google 设计了一个全新的抽象模型，使用 MapReduce 的抽象模型，开发者无需关心容错、数据分发、负载均衡等细节上的问题，这些问题已经封装在函数库里。Google 这一创造性的设计，为分布式计算行业带来了新的发展机遇，很快就出现了类似于 Google“三驾马车”的开源实现，Hadoop 就是其中之一。Apache Hadoop[8]的核心是：HDFS 分布式文件系统和 MapReduce 分布式计算框架。自2007年被推出以来，它一直凭借着其在大数据处理领域的广泛的实用性、良好的易用性、系统的稳定性以及 Hadoop 自身丰富的生态圈，得到了学术界广泛的关注和研究，也是众多企业和用户的首选[9]。在短短几年内，Hadoop 就成为了到目前为止最为成功、最被广泛接受的大数据处理主流技术和系统平台，并且成为一种大数据处理方面的工业标准，得到了业界大规模的进一步开发和改进。由于在系统性能和功能方面存在不足，Hadoop 在发展过程中进行了不断的改进，自2007年推出首个版本以来，目前已经先后推出了数十个版本。

分布式计算提供了很多方法，不过在进行很多大规模数据处理和大规模资源需求时，缺乏弹性，不易管理，对用户没有亲和性。标准化的业务模块建设，哪怕是系统的复制性建设，也并没有提高系统的灵活性。集中的大型基础设施遇到了系统利用率不足的难题，不同的系统运行在独占的硬件资源中，低效、高能耗、空间问题逐渐凸显。因此，为了降低成本、提高运营灵活性、提升资源利用率，虚拟化开始部署在数据中心，云计算技术[10]应运而生。虚拟化屏蔽了不同物理设备的异构性，将基于标准化接口的物理资源虚拟化为逻辑上标准一致的逻辑计算资源（虚拟机）和逻辑存储空间。云计算技术实现了计算能力的极大规模可扩展整合，为多个外部用户提供服务。1984年，Sun 公司的联合创始人 John Gage 提出了“网络就是计算机”的理念[11]，用于描述分布式计算技术带来的新世界，而今天的云计算正在将其变成现实。

21世纪10年代，云计算作为一种新兴的技术已经得到了快速的发展，凭借其高可靠性、高可扩展性、极其廉价等特点，深受企业和用户的欢迎。2005年，Amazon宣布推出 AmazonWebService 云计算平台[12]。IBM 于2008年宣布在中国无锡太湖建立了第一个中国云计算中心，阿里软件于次年在江苏南京建立了首个“电子商务云计算中心”。在过去的一年里，计算科技的爆发，已然昭示时代的奇点已经来临：基于云计算的人工智能 AlphaGO 击败了人类的智慧[13]；阿里云打破了亚马逊记录，再次降低了人类计算成本，将世界顶级的计算能力变成了普惠科技的云产品。正如一个英国预言学家的断言“未来，云计算是比汽车轮子还要根本的工具”。云计算从技术上解决了大规模海量数据分布式存储、海量数据实时备份、并行计算和应用高度集成等问题。用户无需了解数据的传输和计算过程，这些服务都隐藏在云端，用户只需根据自己的需求通过网络来获取云端的应用服务和网络信息。

1.2 研究问题与现状

转码是将高码流转换格式使之适应异构网络和多终端环境的视频处理方法[5]。海量视频的转码任务既是数据密集型工作又是计算密集型工作，需要消耗大量的计算资源[6]，且耗时长。为了满足流媒体，边传边播的媒体特性需求，需要对海量媒体数据进行转码。现有方法主要从三个方面对转码速度进行提升：（1）算法加速，（2）GPU硬件加速以及（3）集群并行加速。基于算法加速利用的是原始视频编码信息，如运动矢量等，优化算法以减少重新编码的计算量；或者通过简化转码框架，选择性地缺省部分非关键性转码步骤，但需要后期补偿由于省略步骤带来的图像质量以及精度的下降。基于GPU硬件加速方法是采用CPU加GPU的异构计算框架，利用GPU的超强处理能力，完成转码过程中计算最密集耗时最多的部分，但由于需要特殊的硬件支持，而且视频相互之间存在很强的依赖性，处理效果并不理想。集群并行转码由主节点将原始视频进行分割，将分片分发到不同的从节点上进行转码，转码后再收集进行合并，它涉及到节点的管理、资源的分配等问题。通过并行加速，无损于视频的图像质量，取用方便，是一种较优的转码加速方式，其中Hadoop MapReduce并行计算框架最为广泛使用。

分布式集群并行转码需要对视频进行分割处理，分片的方式和分片的大小从很大程度上影响着系统的转码效率。分割后文件需要分发到各个数据节点，各个分片任务的调度也是很多研究的重点。Hadoop最新版本支持的调度器有：FIFO调度器、容量调度器和公平调度器。但是官方提供的任务调度算法往往不具备针对性，从而未能将系统的资源利用率最大化。从优化资源利用率的角度出发，在MapReduce中动态调整任务槽slot、根据任务的类型配置任务槽slot都可以有针对性地优化系统的资源利用率。YARN的内存、CPU资源都封装在container资源容器

中，通过动态调整container配置的方式，可以提高资源利用率。但是任务运行状态下对container的调整可能会导致任务的运行异常。运行出错的转码任务如果不能及时恢复，会影响到转码后的合并，影响系统的转码时间。

本文从优化集群转码效率、提高集群资源利用率的角度出发，针对音视频转码任务，研究基于Hadoop转码集群的加速方法，旨在实现海量多格式流媒体音视频文件的存储与分析一体化。

1.3 研究内容与贡献

基于云计算的应用将会逐渐渗透到每个人的生活中，无论是对物联网的发展，还是对我们的服务、生活都会带来深远的影响。本文研究了网络流媒体传输中产生的海量视频监控数据的分布式存储和分布式集群并行转码的问题，将整个系统移植到云平台下实现计算能力极大规模的可扩展。本文以降低视频转码时间和提高转码系统吞吐量为目的，对转码加速问题进行了研究，主要侧重于研究 Hadoop 集群转码优化和 Hadoop 的容器资源分配。

本文将开源云计算平台 Hadoop 应用于网络流媒体的传输中，利用该平台的分布式文件系统 HDFS 实现海量音视频数据的存储，利用 Hadoop 的分布式编程模型 MapReduce 实现对流媒体数据的分布式转码。当任务并发量大时，节点的处理能力有限，同时处理多个任务，任务间资源的抢占，会导致个别转码任务等待时间过长。所以任务资源的分配调度，极大的影响了转码系统的效率，本文对其进行了深入的研究。由于 MapReduce 分布式编程框架适合于处理解耦合数据，在未进行预处理的情况下，直接存储到 HDFS 上，用 MapReduce 进行转码，会导致转码后音视频文件的不可用。本文通过对各流媒体视频格式结构特性的研究，针对流媒体文件格式的特殊性，利用位置信息在 Hadoop 上对视频文件进行了分割。

本文的贡献主要包括如下内容：

（1）设计了 Hadoop 平台下基于位置信息的视频分割方法。本文通过修改 MapReduce 的 getsplit 分片方式，通过在分割点位置前多读数据包的方式，避免了分割在关键帧处导致的转码后视频跳帧的情况。本方法成功地在 Hadoop 上实现了 MKV、MPG、TS、WMV 和 AVI 五种视频格式的分割，减少了额外的文件读写开销，避免了系统的性能瓶颈。

（2）设计了基于样本资源需求的容器资源配置方法。本文通过在样本集群中，对所有可能的内存和 CPU 核资源配置进行测试，找到转码时间最短的 container 配置，在集群变更后，只需处理单个样本 split 分片任务，分析其资源使用情况，利用基于样本资源需求的容器资源配置算法计算出 container 中内存和 CPU 核的系数值，即可得到新集群中最佳的 container 配置，从而提高集群的整体性能以及资源利用率。

1.4 本文组织结构

本文针对基于Hadoop平台的音视频转码技术及其资源优化方法进行了研究。首先介绍了音视频转码的在流媒体应用中研究的必要性。通过分析当前基于Hadoop平台的音视频转码技术的研究现状和Hadoop平台下资源调度分配的研究现状，提出了一种基于样本资源需求的容器资源配置方法和一种Hadoop平台下基于位置信息的视频分割方法。两种方法都分别进行了实证以及性能对比。本文的主要内容和结构安排如下：

第一章为绪论部分，主要介绍了课题的研究背景与意义，概述了研究内容与本文的主要工作。

第二章为研究基础Hadoop和FFmpeg的介绍，讨论了Hadoop的分布式存储框架HDFS和并行计算框架MapReduce，并对Hadoop的新型资源调度器YARN进行了详细的介绍，讨论了YARN的双层调度机制以其对两种资源（内存、CPU核）的分配和隔离，并陈述了其相应的研究进展。

第三章介绍了分布式媒体转码系统的设计。首先介绍了系统的整体架构，描述了系统的设计思路，系统要解决的问题，以及相应的解决方案，并对系统各个模块的功能以及系统的主要工作流程进行了详细的介绍。

第四章提出了一种 Hadoop 平台下基于位置信息的视频分割方法。首先简要介绍了 Hadoop 逻辑分片和逻辑记录的概念，然后详细介绍了基于位置信息的视频分割流程，分为两类：MKV；MPG、TS、WMV 和 AVI 等其他视频格式。最后结合 Super Video Splitter 对分割结果进行了实验评测和结果分析，并对转码集群进行了功能测试和性能测试。

第五章提出了一种基于样本资源需求的容器资源配置方法。首先介绍了节点并行度对转码速度的影响，以及 YARN 通过配置容器资源调整节点并发度的方法，然后介绍了一种基于样本资源需求的容器资源配置优化策略，详细地描述了容器资源配置的具体步骤。在物理机实验平台和虚拟机实验平台上进行了实证实验，并对实验结果进行了对比分析。

最后是结论部分，对本文的工作进行了总结并给出了下一步工作的展望。

第2章 重要概念及相关研究

本章先讨论了Hadoop的分布式存储框架HDFS和并行计算框架MapReduce，详细介绍了Hadoop YARN资源调度器和YARN的双层调度机制，以及YARN对两种资源（内存、CPU核）的分配和隔离，并陈述了其相应的研究进展，最后对本章进行了小结。

2.1 Hadoop基本架构

Hadoop是由Apache2.0许可协议发布的分布式系统基本架构，它可以部署在大量廉价硬件设备组成的集群上，并提供一组稳定性高可靠性好的接口以便开发相应的应用。用户可以在不了解分布式底层细节的情况下，开发分布式程序，利用集群的能力进行高速运算和存储。为了深入研究Hadoop的资源调度分配，需要掌握Hadoop的基本架构。因此，本章先对Hadoop的基本架构进行介绍。Hadoop框架的核心是：HDFS分布式文件系统和MapReduce分布式计算框架。其中，HDFS主要用于大规模海量数据的分布式存储，而MapReduce则构建在分布式文件系统之上，对存储在HDFS中的数据进行分布式计算。尽管Hadoop因MapReduce及其分布式文件系统HDFS而出名，但Hadoop这个名字也用于泛指一组相关的项目，如ZooKeeper、Avro、Pig、Hive、HBase等，这些项目都使用这个基础平台进行分布式计算和海量数据处理。

本章先介绍分布式存储系统HDFS的基础架构和MapReduce计算框架，然后对Hadoop YARN资源调度器进行详细介绍，最后介绍了相关的研究进展。

2.1.1 HDFS 架构

当数据集的大小超过一台独立物理机的存储能力时，就有必要对其进行分区并存储到若干台单独的计算机上。管理网络中跨多台计算机存储的文件系统被称为分布式文件系统（Distributed File System）。HDFS（Hadoop Distributed File System）作为Hadoop默认使用的分布式文件系统，是类似于Google GFS的开源实现。它提供了一个可扩展、高可靠、高可用的大规模数据分布式存储管理系统，基于物理上分布在各个数据存储节点的本地Linux系统的文件系统，为上层应用程序提供了一个逻辑上整体化的大规模数据存储文件系统。与GFS类似，HDFS采用多副本（默认为3个副本）数据冗余存储机制，并提供了有效的数据出错检测和数据恢复机制，大大提高了数据存储的可靠性。它以流式数据访问模式来存储超大文件，具有高容错性，适宜部署在廉价的机器上。HDFS可以提供高吞吐量的数据访问，非常适合大规模数据集上的应用。

HDFS的基本架构如图2-1所示，总体上采用了master/slave架构[14]，主要由以下几个组件组成：Client、NameNode、Secondary NameNode和DataNode。

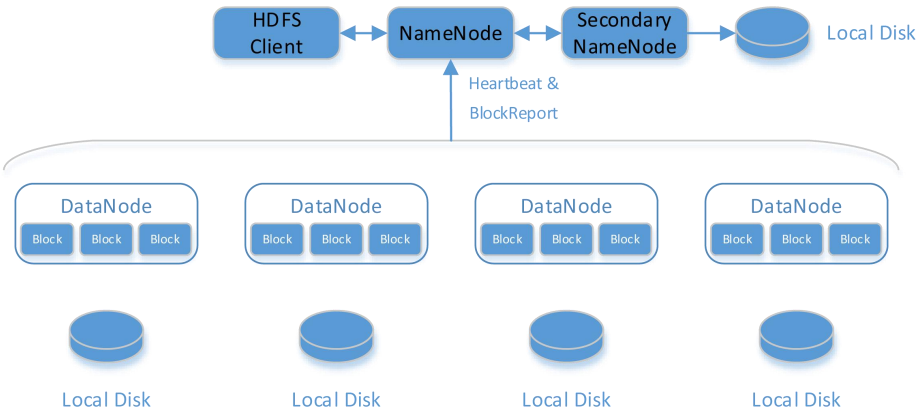

图 2.1 HDFS基本架构

（1）Client

Client（客户端）代表用户通过与NameNode和DataNode交互来访问HDFS中所有的文件。客户端提供了一个类似于POSIX（可移植操作系统界面）的文件系统接口，因此用户在编程时无需知道NameNode和DataNode也可以实现其功能。

（2）NameNode

整个Hadoop集群中仅有唯一一个NameNode。它用来管理HDFS的目录树和相关文件元数据信息，这些信息以“fsimage”（HDFS元数据镜像文件）和“editlog”（HDFS文件改动日志）的文件形式存放与本地磁盘上。NameNode也记录着每个文件中各个数据块所在的数据节点信息，但它并不永久保存块的位置信息，因为这些信息会在系统启动时由数据节点重建。此外，NameNode还负责监控各个DataNode的健康状况，一旦发现某个DataNode宕掉，则将该DataNode移出HDFS并重新备份其上的数据。

（3）Secondary NameNode

Secondary NameNode不仅需要为NameNode元数据进行热备份，同时需要定期合并fsimage和edits日志，并传输给NameNode。需要注意的是，为了减小NameNode压力，NameNode并不会自行合并fsimage和edits日志，将文件存储在磁盘上，而是交由Secondary NameNode完成。但是，Secondary NameNode保存的状态总是滞后于主节点，所以在主节点全部失效时，难免会丢失部分数据。在这种情况下，一般把存储在NFS上的NameNode元数据复制到辅助NameNode并作为新的主NameNode运行。

（4）DataNode

DataNode是文件系统的工作节点，每个Slave节点上都有一个DataNode。它用来存储实际的数据，并将数据信息定期汇报给NameNode。DataNode以固定大小的block为基本单位组织文件内容，默认值为64MB。用户上传一个大于block值的文件到HDFS上时，该文件会被切分成若干个block，存储在不同的DataNode上。Hadoop保证了数据的高可靠性，会将同一个block以流水线的方式写到若干个不同的DataNode上（默认是3，该参数可配置）。用户无需关心文件切割后再存储的过程。

2.1.2 MapReduce 架构

同HDFS一样，Hadoop MapReduce也采用了Master/Slave架构，具体如图2-2所示。它主要由以下几个组件组成：Client、JobTracker、TaskTracker和Task。下面就这几个组件分别进行介绍。

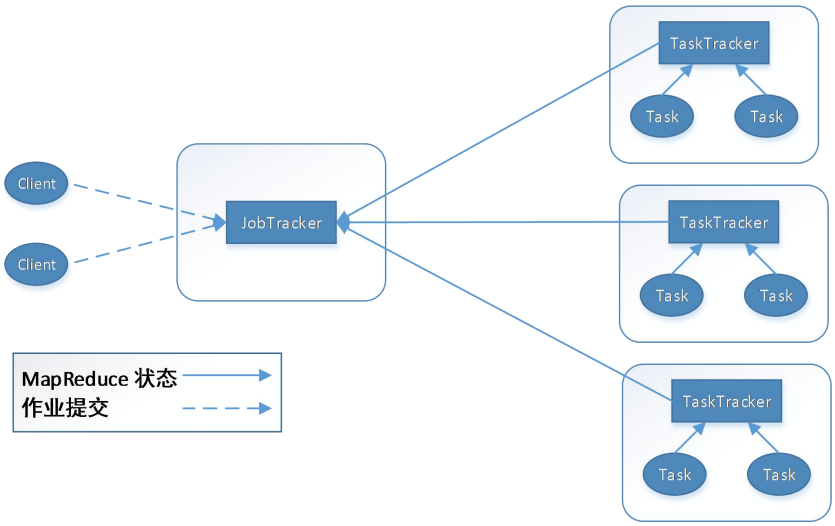

图 2.2 MapReduce 基本架构

（1）Client

Client负责提交用户编写的MapReduce程序到ApplicationMaster上；同时，用户可通过Client提供的一些接口查看作业运行状态。在Hadoop内部用“作业”（Job）表示MapReduce程序。一个MapReduce程序可对应若干个作业，而每个作业会被分解成若干个Map/Reduce任务（Task）。

（2）JobTracker

JobTracker主要是监控资源使用情况和对作业进行调度。JobTracker监控所有TaskTracker与作业的健康状况，一旦发现任务失败，会将其转移到其他任务节点；同时，JobTracker可以对任务的执行进度、资源使用情况等信息进行追踪，并将这些信息反馈给任务调度器，而调度器会在资源空闲时，选择合适的任务使用这些资源。

（3）TaskTracker

TaskTracker会周期性地通过Heartbeat汇报给JobTracker一些信息，其中包括

了本节点上资源的使用量以及任务的执行进度，同时接收JobTracker发送过来的命令并执行相应的操作（如启动新任务、杀死任务等）。TaskTracker使用“任务槽”（slot）等量划分本节点上的CPU、内存等计算资源。只有当task获取到slot后才有机会运行，而Hadoop调度器则是将各个TaskTracker上的空闲slot分配给Task使用。Slot分为Map slot和Reduce slot两种，分别供Map Task和Reduce Task使用。TaskTracker通过配置slot数目参数限定Task的并发度。

（4）Task

Task分为Map Task和Reduce Task两种。从2.1.1中可知，HDFS以固定大小的block为基本单位存储数据，而对于MapReduce而言，其处理单位是split。split和block的对应关系如图2-3所示。split是一个逻辑概念，它只包含一些元数据信息，比如数据起始位置、数据长度、数据所在节点等。用户可以自己决定split的划分方法。划分得到的split数目决定了map task的数量。

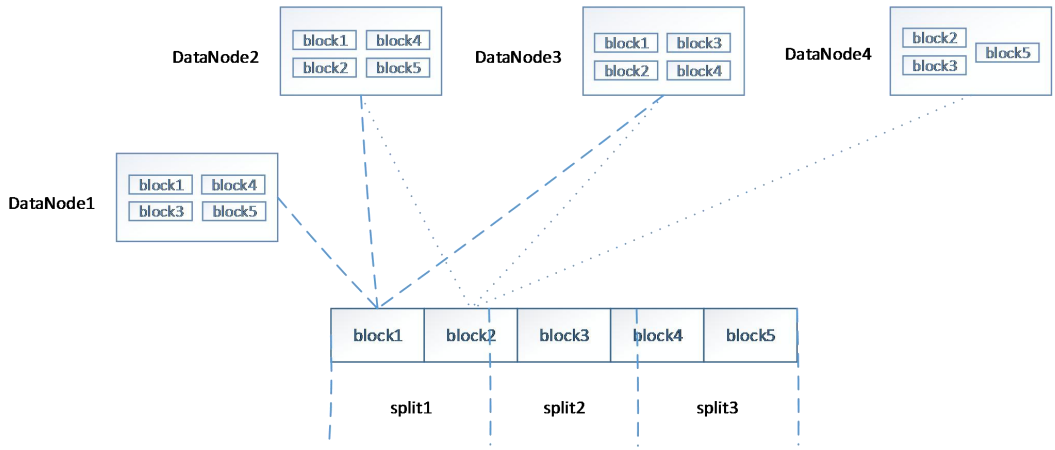

图 2.3 split与block的关系

Map Task 执行过程见图 2-4。Map Task 先将 split 迭代解析成一个个 key/value 键值对，依次调用用户自定义的 map()函数进行处理，临时结果被存放在本地磁盘上，其中临时数据又被分成若干个 partition，每个 partition 将被一个 Reduce Task 处理。

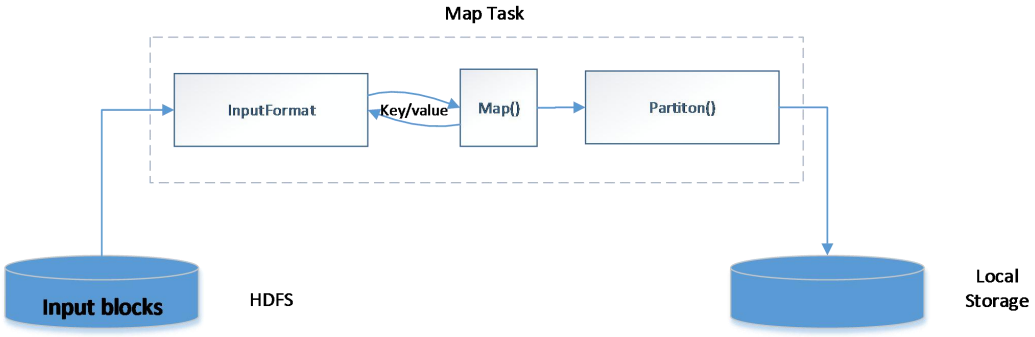

图 2.4 Map Task 执行过程

Reduce Task 执行过程见图 2.5。可划分为三个阶段：

1) 从远程节点上读取 Map Task 中间结果；
2) 根据 key 值对 key/value 键值对进行排序；
3) 依次读取<key，value list>，调用用户自定义的 reduce()函数进行处理，并将结果存储在 HDFS 上。

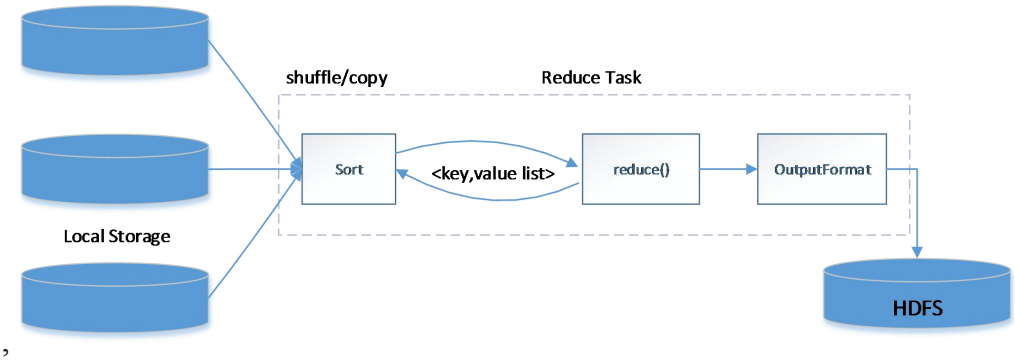

图 2.5 Reduce Task 执行过程

2.2 基于Hadoop YARN资源调度器

2.1.2 节介绍了第一代 MapReduce 框架（MapReduce Version 1.0，MRv 1）。随着数据量的高速增长和新型应用的出现，MRv1 在扩展性、可靠性、资源利用率和多框架支持等方面暴露出了明显不足，由此诞生了下一代 MapReduce 框架（MapReduce Version 2.0，MRv 2）。

YARN 是 Apache 的下一代 MapReduce 框架。它的基本设计思想是将 JobTracker 拆成两个独立的服务：一个全局的 ResourceManager（RM）负责资源的管理以及每个应用程序特有的 ApplicationMaster 负责单个应用程序的管理。

2.2.1 YARN 的架构

YARN 总体上仍然是 master/slave 结构。在整个资源管理框架中，ResourceManager 为 master，NodeManager 为 slave，ResourceManager 负责对各个 NodeManager 上的资源进行统一管理和调度。当用户提交一个应用程序时，需要提供一个用于跟踪和管理这个程序的 ApplicationMaster。它负责向 ResourceManager 申请资源，并要求 NodeManager 启动可以占用一定资源的任务。由于不同的 ApplicationMaster 分布在不同的节点上，因此它们之间不会相互影响。

图 2.6 描述了 YARN 的基本组成结构。YARN 主要由 ResourceManager、NodeManager、ApplicationMaster 和 container 等几个组件构成。

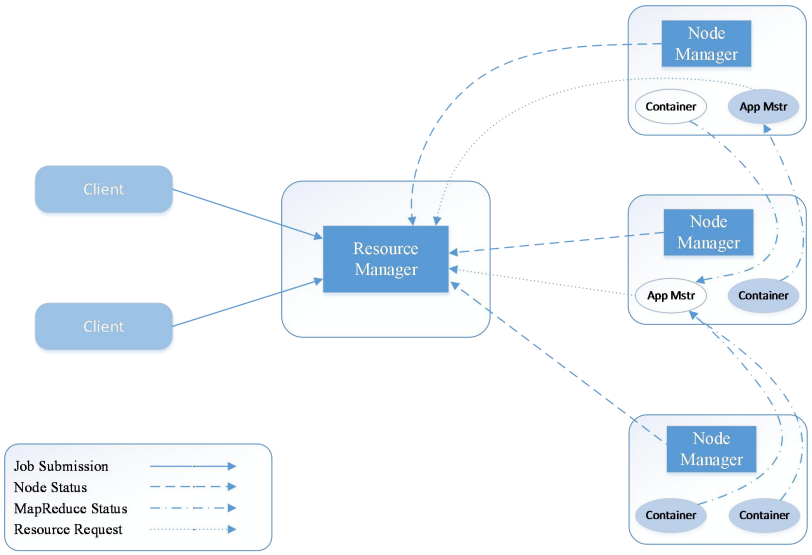

图 2.6 YARN 基本架构

（1）ResourceManager（RM）

RM是一个全局的资源管理器，是系统中将资源分配给各个应用的最终决策者。RM由两个组件组成：调度器（Scheduler）和应用管理器（ApplicationsManager，ASM）。调度器根据各个应用的资源需求进行调度，而资源分配单位用一个抽象概念“资源容器”（Resource Container，简称container）表示。Container是一个动态资源分配单位，它将内存、CPU、磁盘、网络等资源封装在一起，从而限定每个任务使用的资源量。此外，该调度器是一个可插拔的组件，用户可根据自己的需求设计新的调度器，YARN提供了多种直接可用的调度器，比如Fair Scheduler和Capacity Scheduler等。而应用程序管理器负责管理整个系统中所有应用程序，包括应用程序提交、与调度器协商资源以启动ApplicationMaster、监控ApplicationMaster运行状态并在失败时重新启动它等。

（2）ApplicationMaster（AM）

每个应用程序都会有一个AM，它实际上是一个简化版的JobTracker。AM主要负责同RM调度器协商以获取合适的容器，并跟踪这些容器的状态和监控其进度，以及与NM通信以启动/停止任务。AM出现故障后，ASM会重启它，而由AM自己从之前保存的应用程序执行状态中恢复应用程序。

当前 YARN 自带了两个 AM 实现：一个是用于演示 AM 编写方法的实例程序 distributedshell，它可以申请一定数目的 container 运行一个 shell 命令或者 shell 脚本；另一个是运行 MapReduce 应用程序的 AM——MRAppMaster。此外，还有一些其他的计算框架对应的 AM 正在开发中，比如 Open MPI、Spark 等。

（3）NodeManager（NM）

NM是每个节点上的框架代理，主要负责启动应用所需的容器，监控资源（内存、CPU、磁盘、网络等）的使用情况并将之汇报给调度器。一方面，它会定时

地向RM汇报本节点上的资源使用情况和各个container的运行状态；另一方面，它接收并处理来自AM的任务启动/停止等各种请求。

（4）Container

Container 是 YARN 中的资源分配单位，它封装了多维度的资源，如内存、CPU、磁盘、网络等。当 AM 向 RM 申请资源时，RM 为 AM 返回的资源便是用 container 表示的。YARN 中每个任务均会对应一个 container，且该任务只能在该 container 中执行，并仅能使用该容器代表的资源量。需要注意的是，container 不同于 MPv 1 中的 slot，他是一个动态资源划分单位，是根据应用程序的需求动态生成的。截至本文完成时，YARN 仅支持 CPU 和内存两种资源，且使用了 Linux Container 进行资源隔离。

2.2.2 YARN 的资源调度机制

YARN 资源调度器的核心是其资源调度机制，其中 YARN 采用双层资源调度机制对资源进行分配。

YARN 采用的双层资源调度机制如图 2-7 所示。第一层级是 ResourceManager 将资源分配给 ApplicationMaster，第二层级是 ApplicationMaster 将收到的资源分配给其内部具体的 NodeManager。

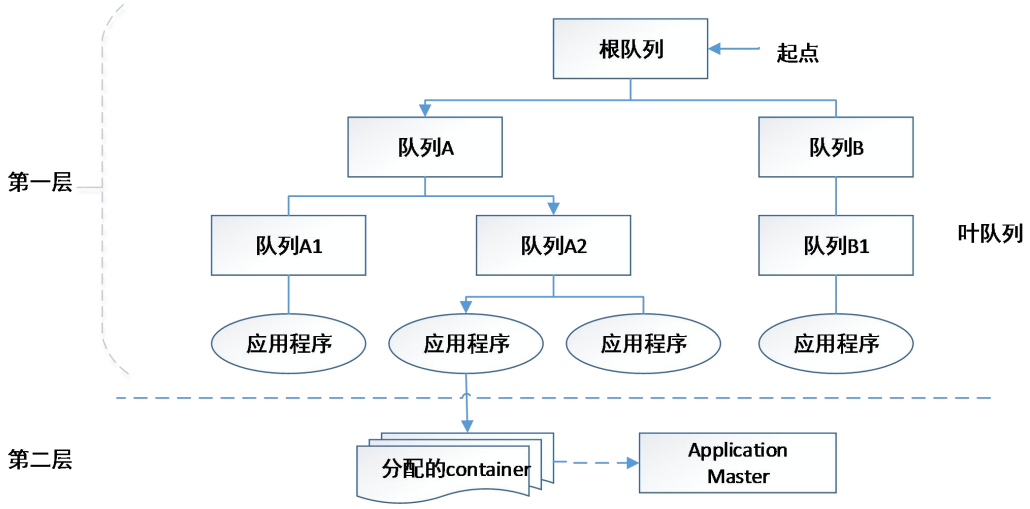

图 2.7 YARN 双层调度机制

YARN 的资源分配过程是异步的，具体过程如图 2-8 所示。

1) NodeManager 周期性通过 HeartBeat 向 ResourceManager 汇报节点信息；
2) ResourceManager 收到节点信息后，返回需释放的容器列表的心跳应答；
3) 同时触发一个 NODE_UPDATE 事件，如果此时有可释放的容器，则会触发资源的分配；
4) Scheduler 收到触发的事件后，会将该节点上的资源分配给内部的应用程

序，并将分配结果存放到一个内存数据结构中；

5) 应用程序的 ApplicationMaster 通过向 ResourceManager 发送周期性心跳，来获取最新分配的容器资源；

6) ResourceManager 收到信息后，将分配的容器通过心跳应答返回给 ApplicationMaster；

7) ApplicationMaster 收到新分配的容器列表后，再将这些新容器分配给内部任务。

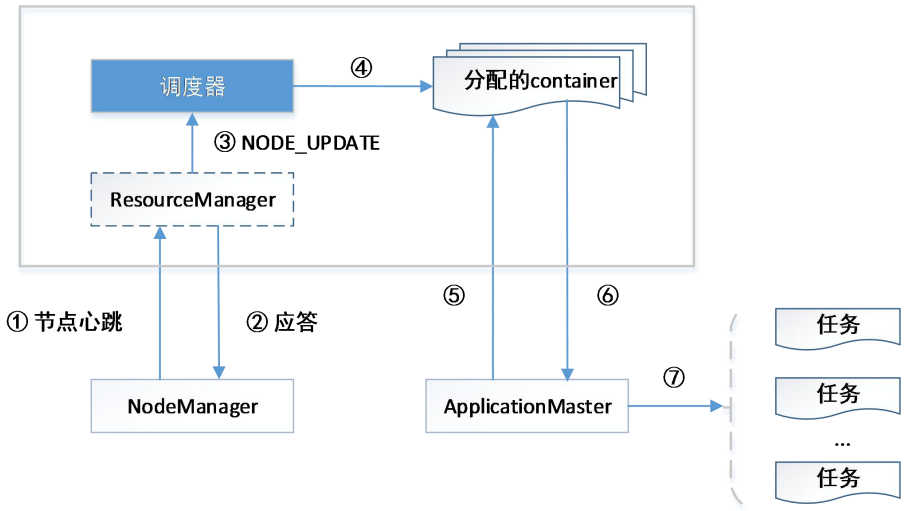

图 2.8 YARN 资源分配过程

心跳信息的请求伴随着每个 map 任务的数据本地化信息，特别是输入分片所在的主机和相应机架信息。调度器使用这些信息来做调度决策（像 jobtracker 的调度器一样）。理想情况下，它将任务分配到数据本地化的节点，但如果不可以这样做，调度器就会相对于非本地化的分配优先使用机架本地化的分配。请求也为任务指定了内存需求。在默认情况下，map 任务和 reduce 任务都分配到 1024MB 的内存，但这可以通过 mapreduce.map.memory.mb 和 mapreduce.reduce.memory.mb 来设置。

在 YARN 中，资源管理由 ResourceManager 和 NodeManager 共同完成，其中 ResourceManager 中的调度器负责资源的分配，而 NodeManager 则负责资源的供给和隔离。ResourceManager 将某个 NodeManager 上的资源分配给任务后，NodeManager 需按照要求为任务提供相应的资源，甚至保证这些资源应具有独占性，为任务运行提供基础的保证。

不同于 MRv 1 中基于 slot 的资源模型，YARN 采用了基于真是资源需求量的调度模型：集群中各个节点周期性向 ResourceManager 汇报各类资源使用情况，而 ResourceManager 则根据各个作业的真实资源量需求，结合一些资源约束，进行资源分配和任务调度。用户提交应用程序后，对应的 ApplicationMaster 负责将应用程序的资源需求转化成符合特定格式的资源请求，并发送给

ResourceManager。一旦某个节点出现空闲资源，ResourceManager 中的调度器将决定把这些空闲资源分配给哪个应用程序，并封装成 Container 返回给对应的 ApplicationMaster。

ApplicationMaster 发送的资源请求形式如下：

<Priority，Hostname，Resource，#Container>

- Priority：资源优先级。不同于 MRv 1 中的优先级概念，YARN 允许优先级是任意正整数，而具体怎样规划各种资源的紧急程度，完全由应用程序的 ApplicationMaster 将资源分为三种优先级，分别是 PRIORIYT_FAST_MAP、PRIORITY_REDUCE、PRIORITY_MAP，优先级分别是 5，10 和 20（Priority 对应的数值越小，优先级越高）。ResourceManager 将优先为优先级高的任务分配资源。
- Hostname：期望分配的资源所在节点。调度器会优先为应用程序分配马努数据本地化的资源，这样可避免移动数据从而提高效率。
- Resource：表示需要的资源量。截至 2.0.3-alpha 版本，YARN 仅支持 CPU 和内存两种资源。
- #Container：需要符合以上资源描述的 Container 的数量。对于 MapReduce 而言，同一个作业的所有同类型任务需要的资源量是相同的。

如上所述，Hadoop YARN 同时支持内存和 CPU 两种资源的调度[15]。默认只支持内存，如果想进一步调度 CPU 资源，需要进行一些配置。YARN 的内存分配方式不同于 MRv 1，后者中的 tasktracker 在集群配置的时候设置了固定数量的槽，每个任务在一个槽上运行。槽有最大内存分配限制，这对集群来说是固定的，导致当任务使用较少内存时无法充分使用内存资源（因为其他等待的任务不能使用这些未使用的内存）以及由于任务不能获取足够的内存而导致作业的失败。

在 YARN 中，资源被分为更细的粒度，所以可以避免上述问题。具体而言，应用程序可以请求最小到最大限制范围的任意最小值倍数的内存容量。默认的内存分配容量是调度器特定的，对于容量调度器，它的默认值最小值是 1024MB（由 yarn.scheduler.capacity.minimum-allocation-mb 设置），默认的最大值是 10240MB（由 yarn.scheduler.capacity.maximum-allocation-mb 设置）。因此，任务可以通过适当设置 mapreduce.map.memory.mb 和 mapreduce.reduce.memory.mb 来请求 1GB 到 10GB 间的任意 1GB 倍数的内存容量（调度器在需要的时候使用最接近的倍数）。一些重要的资源参数见表 2.1。

表 2.1 YARN 资源参数

参数名	含义
yarn.nodemanager.resource.memory-mb	表示该节点上 YARN 可使用的物理内存总量，默认是 8192（MB）
mapreduce.[map|reduce].memory.mb	每个 Map 或 Reduce 任务需要的物理内存量，默认是 1024（MB）
yarn.nodemanager.resource.cpu-vcores	表示该节点上 YARN 可使用的虚拟 CPU 个数，默认是 8
mapreduce.[map|reduce].cpu.vcores	每个 Map 或 Reduce 任务需要的虚拟 CPU 个数，默认是 1

内存和 CPU 这两种资源，是两种性质截然不同的资源。内存资源的多少会决定任务的生死，如果内存不够，任务可能会运行失败；相比之下，CPU 资源则不同，它是一种“弹性”的资源，使用量大小不会直接影响到应用程序的死亡，它只会决定任务运行的快慢。所以，YARN 对这两种类型的资源使用了不同的资源隔离方案。CPU 的资源隔离方案采用了 Linux Kernel 提供的轻量级资源隔离技术 Cgroups；对于内存而言，它是一种“限制性”资源，使用量大小直接决定着应用程序的死亡，Cgroups 会严格限制应用程序的内存使用上限，一旦使用量超过预先定义的上限值，就会将该应用程序“杀死”，因此无法使用 Cgroups 进行内存资源隔离，而是选择了线程监控的方式。

2.3 FFmpeg

FFmpeg[16]是一组开源计算机程序，可用于记录、转换数字音频、视频，并将其转换为音视频流。它为音视频的录制、转换以及流化提供了完整的解决方案。它包含了音视频编解码库 libavcodec，和音视频容器复用和解复用库 libavformat，以及用于转码多媒体文件的 FFmpeg 命令行程序。

FFmpeg 在 Linux 平台下开发，但它同样也可以在其它操作系统环境中编译运行，包括 Windows、Mac OS X 等。这个项目最早正在 2000 年由 Fabrice Bellard 发起，现在由 Michael Niedermayer 维护。许多 FFmpeg 的开发人员都来自 MPlayer 项目，而且当前 FFmpeg 也是放在 MPlayer 项目组的服务器上，平均每三个月发布一次新版本。FFmpeg 的组件详见表 2.2。

表 2.2FFmpeg 组件

命令行工具		媒体库	
ffmpeg	转换音视频格式	libswresample	音频重采集、类型转换与合并库
ffserver	现场直播和录音广播的 HTTP（RTSP）多媒体流服务器	libavcodec	编/解码库
ffplay	使用 SDL 和 FFmpeg 库的媒体播放器	libavformat	I/O 和合并/分离库
ffprobe	显示媒体信息	libavdevice	特殊设备的合并/分离库
		libpostproc	后置处理库
		libswscale	颜色转换与缩放库（图像拉伸、像素格式转换）
		libavfilter	基于图的帧编辑库（加特效、滤镜）

2.4 国内外研究现状

2.4.1 集群加速转码国内外研究现状

早在 2009 年，国外视频编码器领导厂商 On2 Technologies 公司已与 Zencoder 公司联合开发的 On2 Flix Cloud 随需视频转码服务。Flix Cloud 主要在 Amazon EC2（Elastic Compute Cloud）平台上运行，大幅度节省了视频转码成本，降低了企业快速转换大量视频内容的入门门槛。视频发布者采用 Flix Cloud 服务，无需交付启用成本或签订长期合同，并且即日起可以开始视频转码工作。Flix Cloud 通过运用云计算服务，来执行视频转码所需的 CPU 密集计算任务。Flix Cloud 采用基于“按需付费”的模式，只在特定时段使用自己所需的带宽和服务器资源，实现较低的项目开展成本。此外，Flix Cloud 还为大多数常用转码案例提供了设计好的“编码配方”（encoding recipes），确保以最少的配置工序获得最佳的视频质量。Flix Cloud 满足了客户一方面要将整个内容库立即转换成高清和移动格式的需求，同时又要确保较低的运营成本的市场需求。Flix Cloud 是一项具有高度可扩展性的虚拟服务，拥有其独特的优势。

国外学术界的一些学者在云转码领域做了一些研究工作：

不少国外的学者提出利用 MapReduce 并行计算框架进行分布式转码[17][18]，

采用的方法是事先将分割好的视频文件上传至 HDFS，直接利用 Hadoop 并行框架来搭建集群转码系统，但是并未对系统进行任何优化，转码性能较单机而言有所提升，却未能充分发挥集群转码的优势。Wang F[19]等提出了基于云架构的满足全球不同地区用户需求的流媒体服务平台，该平台不仅提供可伸缩性资源供应，而且造价低廉，最后作者在上做了验证。

有学者通过研究视频的分片方式来对转码任务进行加速。Garcia A[20]等对不同分片大小下的转码时间进行了比较，得出了在合理分片大小范围内，视频分片越大转码时间越短的结论，因为视频分片越大，视频分片数越少，转码过程中的调度和任务启动等开销越少。Jokhio F[21]等通过先分割小的分片，然后增大分割的分片大小的方式，减少任务启动的延时。Tian C[22]等提出了对视频进行粗细两个粒度的视频分割，先将文件进行粗粒度分割，再分成粒度更小的分片，从而避免某个节点上任务运行过慢造成的性能瓶颈。

Joen M[23]等提出了一种内容感知的视频分割方法来优化分布式编码系统的整体性能，在第一部分中通过考虑视频内容的改变更高效地执行分割，并且在第二部分中使用改变分段的编码顺序来有效地进行调度。Sambe Y[24]等提出了一种新的动态调度算法来解决异构系统中节点计算能力上的差异，以缩短总的转码时间。

国内对云转码的研究起步得较晚，但也有不少知名企业拓展业务领域，提供了一系列完美的视频解决方法，其中就包含了云转码。

阿里云旗下的媒体转码产品，就是为多媒体数据提供的转码计算服务[25]。它以经济、弹性和高可扩展的音视频转换方法，将多媒体数据转码成适合在 PC、TV 以及移动终端上播放的格式。截止 2016 年 11 月，已开放北京、杭州、上海和深圳四个服务区域，海外数据中心也即将开服。阿里云媒体转码预设多种转码模板，根据内容分析推荐的智能模板，支持常见设备和应用场景的静态模板。

腾讯云提供一站式媒体转码分发平台——点播 VOD（Video on Demond），从灵活上传到快速转码，从便捷发布到自定义播放器开发，为客户提供专业可靠的完整视频服务。腾讯云点播拥有遍布全国的 BGP 网络，覆盖 17+运营商，400 多个 CDN 视频加速节点，拥有超过 10000 台分布式转码集群，实现 2000 并发转码，保证了转码的质量和效率。

国内的学者也对利用 Hadoop 进行集群转码进行了相关研究。张浩[26]利用 Hadoop、FFmpeg、mencoder 以及 avidemux 开源工具，设计出一套商业化的云计算产品解决方案。李晓波[27]基于 Hadoop 构建并行转码集群，基于 Struts2+Spring+Hibernate 架构部署 Web 管理平台，使 Web 管理平台和 Hadoop 构成一个完整的集存储、转码和管理为一体的系统。宋晨伟[28]等利用 Hadoop 和 FFmpeg 开源工具搭建了分布式并行转码集群。但上述学者的研究都只是简单地

应用 MapReduce 并行框架，并未对其进行性能的优化。华中科技大学与中国移动研究院合作设计的云转码系统[29][30]，实现了网络上视频的实时转码，给用户提供了即点即播的流畅视觉体验，设计了数据放置策略和传输策略，减少数据的迁移，并用流水线处理转码过程，转码当前任务的同时对下一个任务分片进行传输。李伟[31]等提出了一个基于云的在线视频转码系统，为在线大数据量的视频转码提供经济和 QoS 保证的解决方案。通过分析导出 QoS 值与转码所需 CPU 核数之间的关系，最小化在特定 QoS 约束下的资源预留。

2.4.2 YARN 资源调度分配国内外研究现状

综述[32]指出了当前 YARN 广泛应用的调度策略中存在优先约束和公平约束的问题，例如 Fair Scheduler 会导致低调的资源分配。为了解决公平约束的问题，国外的一些学者提出了 LATE（Longest Approximate Time to End）调度算法[33][34]，LATE 算法考虑了集群的异构性，分析节点速度的快慢并推测任务执行的快慢。但是 LATE 算法推测任务的方式是静态的，导致性能仍然较低。还有学者提出了动态比例共享调度（Dynamic Proportional Share Scheduling）[35]，它允许用户根据任务进度和剩余时间对作业进行调整来控制获得的资源量。而 K Kc [36]等提出的截止时间约束调度器确保了只有能在截止时间前完成的任务才被调度。

性能分析或其他运行时间预测技术可以粗略地估计出 map 任务的执行时间[37][38][39]，然而当集群中多个 reduce 任务并发执行时，预测 reduce 任务的执行时间就变得很难。文献综述[40]提出了数据本地化可以有效减少集群内网络传输的开销，而这往往是数据密集型计算的一个瓶颈。文献[41][42]通过研究调度算法增加数据本地性从而避免不必要的数据迁移来优化 Hadoop 的性能。

一些国外的研究关注于给集群或者任务分类：Kapil B S[43]等将任务负载分类并根据集群的当前负载将特定类别分配给特定集群；Divya M[44]等则是将集群标记为 IO 密集型或 CPU 密集型。也有一些学者从动态调度的方向去研究：Elkholy[45]等根据节点上的可用资源情况，动态地扩张或收缩每个节点的资源容量；Huang C C[46]等提出了一种动态适应槽和复杂度感知调度算法，将视频分段的复杂度作为其任务执行的优先级，通过监视分布式集群的处理状态，对其任务槽数动态地进行负载均衡的调整。Kc K [47]等基于一个回馈控制器，动态地调整 container 的并行度；Su H[48]等提出了冻结/解冻 container 的概念，当由于资源限制任务需要取消时，冻结 container 中的部分计算，对其进行零惩罚的保存处理。Lee S K[49]等认为通过控制 vcore 的数量给每个 container 分配同样的负载可以最小化任务处理时间上的差距，从而在异构环境下优化性能。

国内的学者对 Hadoop 资源优化也做了深入的研究。李媛祯[50]提出了一种基于蚁群算法和粒子群优化算法的自适应资源调度算法，获取负载、内存、CPU 等资源使用情况来初始化蚁群信息素矩阵，实现资源的合理分配。曾婉琳[51]等通过

对 max_mappers 和 max_reducers 参数进行优化，提高任务并行效率。丁晓安[52]等提出了一种弹性 container 的概念，基于对资源的实时统计值。赵颖[53]提出了根据容器内进程对资源使用情况的变化，动态地调整容器资源量的配置，从而达到减少资源浪费和提高集群并行度的目的。

所以对于分布式转码来说，资源的分配和容器的配置很大程度上影响了系统的性能，所以资源的自适应和动态调整 container 容器是一个很好的优化方向。

2.5 本章小结

本章介绍了 Hadoop 的整体框架，并对 Hadoop 的两个核心组件——HDFS 和 MapReduce 进行了详细的介绍。然后介绍了 Hadoop MRv 2 的资源调度器 YARN，详细介绍了 YARN 的双层资源调度机制，内存和 CPU 两种不同类型资源的分配和隔离，以及视频转码用到的开源工具 FFmpeg，并陈述了相关研究的进展。

第 3 章　分布式媒体转码系统的设计

本章将介绍基于 Hadoop 的分布式媒体转码系统的总体设计：首先简要介绍系统的主要功能及其在整个系统中所处的位置；然后介绍系统的整体架构；描述系统的设计思路，先介绍了系统要解决的问题，针对这些问题提出相应的解决方案；然后描述了系统各个模块的功能以及系统的主要工作流程，最后对本章进行了小结。

3.1 系统简介

本文实现的基于 Hadoop 的音视频转码系统是嵌入式与网络计算实验室与湖南双菱公司合作的面向数字教育的全媒体大数据与智能挖掘平台的子系统。云转码系统基于分布式计算技术，主要针对广播电台、广播电视台录制的原始媒体资源以及用户上传的音视频文件进行快速并行转码为适应于网络传输的流媒体格式，并对海量的网络流媒体进行集中式存储。云转码系统的在整个系统中的位置如图 3.1 所示。

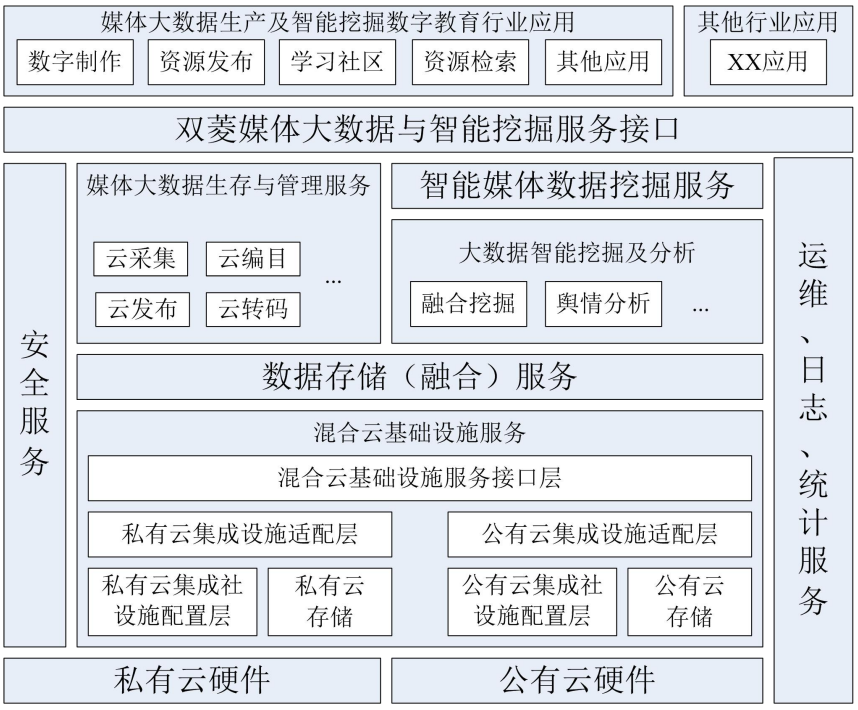

图 3.1 面向数字教育的全媒体大数据与智能挖掘平台

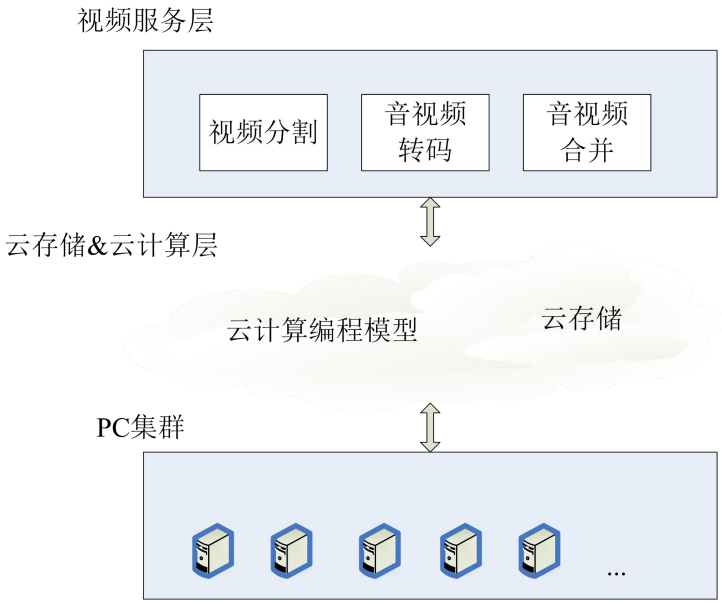

图 3.2 分布式转码系统架构

本系统的整体架构如图 3.2 所示，包括了视频服务层、云存储和云计算层以及 PC 集群。其中视频服务层包括了视频分割、音视频转码、音视频合并三个关键模块。

3.2 系统设计思路

3.2.1 系统拟解决的问题

基于 Hadoop 的音视频转码系统旨在设计并实现一种高吞吐量高速并行的海量音视频文件转码架构。该架构将原本适用于单机转码的 FFmpeg 迁移到分布式环境 Hadoop 下，优化了视频文件作为非结构化数据在 Hadoop 下的分割方式，并在保证高吞吐量的前提下，修改 container 的配置，提高集群资源利用率，大幅度提升媒体转码的性能。系统面临的主要问题为：

（1）音视频文件的分割。在 Hadoop 运行环境中，HDFS 是默认的文件系统，要处理的数据需要预先放置在 HDFS 中，并以数据块的形式存储。一个文件可以由一个或多个数据块组成，其中数据块的大小可调（默认值为 64M/块）。由于视频文件的非结构化特性，视频的压缩编码方式会导致视频中帧与帧之间产生较强的关联性。如果未经预处理直接由 HDFS 切分进行存储，会导致 MapReduce 处理后视频分片的跳帧，甚至导致转码后视频文件的不可播放。MapReduce 分布式架构适合处理非耦合数据，即数据段与数据段之间关联性比较弱的数据类型，视频分块间的强关联性与 MapReduce 编程模型的思想相冲突。目前多数研究都是借助开源工具进行视频的预分割，这些方法都需要在本地先分段再上传 HDFS，增加了额外的文件读写开销。而且视频分割处理一般是固定在某一个节点上完成的，当视频处理服务请求量过大时，该节点会成为系统的一个性能瓶颈。实验测试了

一些开源的视频分割器，现有的开源工具支持的视频格式有限，且拓展难度大，分割时间不理想，FFmpeg 自带的分割命令行每次都需要从头开始遍历，不能在截断点附近的时间标签开始遍历，不适用于海量视频的分割。

（2）Hadoop 平台下调用 FFmpeg。音视频转码需要调用的 FFmpeg 库是 C 语言版本的，而 Hadoop 本身是用 java 写的。要在 Hadoop 上运行 C/C++程序，需要借助系统提供的 API 接口。

（3）系统资源利用率没有保障。Hadoop 具有很高的灵活性，为用户提供了很大的自由去通过修改配置文件进行参数的设定。其中，container 的配置极大的影响了系统的效率。学者基于对资源的实时统计动态调整 container 配置的优化方法，在任务运行状态下调整 container 资源量可能会导致任务运行异常，冻结 container 的优化方法，不适用于 Hadoop 转码任务，因为视频文件分片的关联性，冻结未完成的子任务并对其结果进行零处罚的保存性处理，会导致转码后视频合并的出错。 实验表明转码任务的最优 container 配置与转码视频格式及参数无关，所以可以在任务启动前对 container 进行配置优化性能。由于 YARN 的资源规整化算法，container 的配置选择有限，可以通过对各种配置进行反复测试从而选取最优配比，但当集群变更时，原配置不一定仍为最优。如何在集群变更，资源总量变化的情况下，以最少的额外开销完成资源配置最优选项的探索，也成为本系统面临的主要问题。

3.2.2 解决方案

针对以上问题，本文采用了以下解决方案：

（1）Hadoop 平台下基于位置信息的视频分割方法：通过修改 MapReduce 的 getsplit 分片方式，实现在 Hadoop 上分割视频。通过在分割点位置前多读数据包的方式，避免了分割在关键帧处导致的转码后视频跳帧的情况。本方法成功地在 Hadoop 上实现了 MKV、MPG、TS、WMV 和 AVI 五种视频格式的分割，减少了额外的文件读写开销，避免了系统的性能瓶颈。

（2）Hadoop pipes 下调用 FFmpeg：Hadoop Pipes 采用的主要方法是将应用逻辑相关的 C++代码放到单独的进程中，然后通过套接字（socket）让 Java 代码与 C++代码通信。用户通过 bin/hadoop pipes 将作业提交到 Hadoop 集群中，Java 代码会使用 ServerSocket 创建服务器对象，然后通过 ProcessBuilder 执行 C++ Wapper library，C++ Wapper library 实际上是一个 Socker Client，它从 ServerSocker 中接受 key/value 数据，经过内部的 C++程序处理后，将结果返回给 ServerSocker，并由 ServerSocker 将数据写到 HDFS 或者磁盘上。

（3）基于样本资源需求的容器资源配置方法：容器的资源主要有内存和 CPU 核两种，通过实验发现改变每个容器所分配的内存量和 CPU 核数，从而调整每个 DataNode 上并发的进程数，可以提高集群的整体性能以及资源的利用率。当集群

变更时，只需处理单个样本 split 分片任务，分析其资源使用情况，即可得到新集群最佳的容器配置。

3.3 功能模块设计

音视频转码过程包括三个主要阶段：视频分割，音视频转码和音视频合并。下面分别对这三个过程的功能模块进行详细的介绍。

3.3.1 视频分割模块

视频是由一帧一帧的画面组成的，是一种非结构化的数据，其内部帧与帧之间存在着先后顺序关系。视频帧可分为 I（intra），P（predicted）和 B（bi-directional）三种。其中 I 帧被称为关键帧，是独立静态图像编码的帧；P 帧是由与其最邻近的前一个 I 帧或 P 帧预测得到的；而 B 帧则是由最邻近的前后两个 I 帧或 P 帧作为参考帧进行双向预测得到的。视频的分割必须严格按照关键帧的位置进行分割，不能随意分割视频。

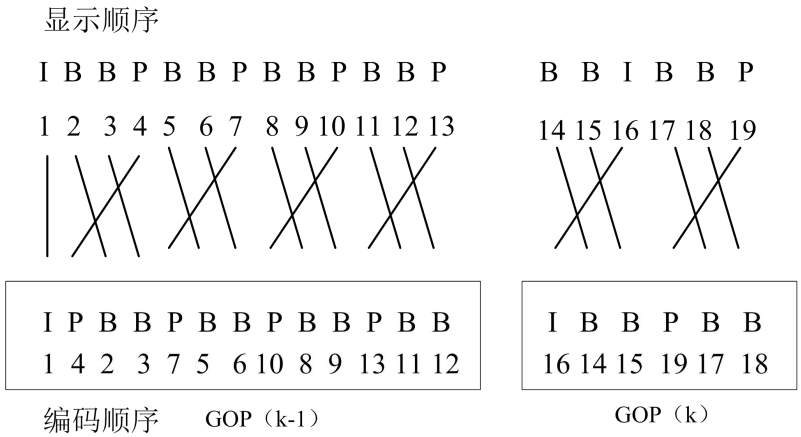

图 3.3 GOP 的分段

如图 3.3 所示，以 MPEG-2 为例，其中有两种 GOP（Group of Picture），分别是 Open-GOP 和 Closed-GOP。GOP（Group of picture）是一组连续的 IPB 画面，代表了两个 I 帧之间的距离。GOP(k-1)是 Closed-GOP，GOP(k)是 Open-GOP，在解码 GOP(k)的帧 B14 和帧 B15 时，需要 GOP(k-1)的帧 P13，如果视频分段后，某个段的第一个 GOP 是 Open-GOP，则在解码这个 Open-GOP 的时候需要前一个段的某一帧，从而造成段与段之间的关联性。另一种情况的 Closed-GOP 则不会发生这种问题，因为它是一个相对独立的完整的视频分段，帧与帧之间的预测都是在同一个 GOP 内完成。

本系统没有使用开源视频工具进行分割，因为这些方法都需要在本地先分段再上传 HDFS，增加了额外的文件读写开销。而且视频分割处理是固定在一个节点上完成的，当视频处理服务请求量过大时，该节点会成为系统的一个瓶颈。本

文基于 split 和 recordreader 是逻辑意义上的分割这一特点，通过修改 MapReduce 的 getsplit 分片方式实现在 Hadoop 上分割视频。根据位置信息，对于 MPG、TS、WMV 和 AVI 格式的视频，分片的时候往前多读取一秒的 AVPacket 数据包，对于 MKV 格式的视频，则多读取一个 cluster，以避免分割后转码结果出现跳帧的情况。

3.3.2 音视频转码模块

在转码过程中，使用到了 FFmpeg 的一些处理音视频的开源库。如 3.2.1 节所述，Hadoop 整个系统框架都是用 Java 语言编写完成的，在实际应用中，因为要用到第三方的 Java 库，需要采用 C/C++或者其他语言编写的作业，无法直接运行 C++程序，就需要使用 Hadoop 编程里的一些工具来辅助完成。而 Hadoop pipes 就是专门为 C/C++语言设计的，它采用 Socket 方式让 Java 与 C/C++通信。

Hadoop Pipes 采用的主要方法是将应用逻辑相关的 C++代码放到单独的进程中，然后通过 Socket 让 Java 代码与 C++代码通信，过程见图 3.4。用户通过 bin/hadoop pipes 将作业提交到 org.apache.hadoop.mapred.pipes 中的 submit 类，通过调用函数 setupPipesJob 对作业参数进行配置，用 JobClient(conf).submitJob(conf) 将作业提交给 Hadoop。进入 submitJob 函数后，Java 代码会使用 ServerSocket 创建服务器对象，即 socket 通信的服务器端，用于从 HDFS 中读取音视频数据。然后通过 ProcessBuilder 执行 C++ Wapper library，C++ Wapper library 从 ServerSocket 中接受 key/value 数据，经过内部的 C++程序处理后，将结果返回给 ServerSocket，并由 ServerSocket 将结果输出到 HDFS 上。

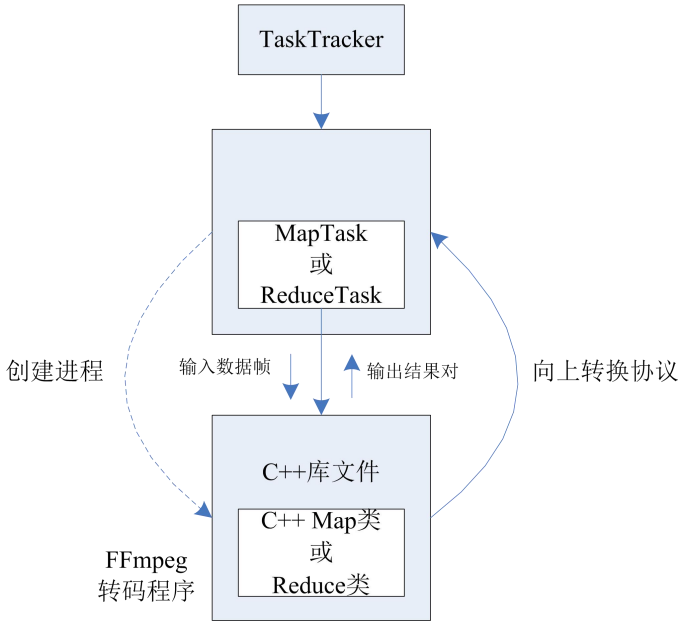

图 3.4Hadoop Pipes 通信方式

Hadoop pipes 任务提交命令的基本格式为：

Hadoop pipes

-D mapreduce.job.name=20160120_templatename //指定 Hadoop 的 job 名和模板名
-D hadoop.pipes.java.recordreader=true //设置 recordreader 为 java 端程序
-D hadoop.pipes.java.recordwriter=true //设置 recordwriter 为 Java 端程序
-D mapred.input.format.class=cn.itcast.hadoop.mr.wordcount.VideoInputFormat
//指定 mapreduce 端的输入数据格式，等号后面的部分为数据 class 类名
-D mapred.output.format.class=cn.itcast.hadoop.mr.wordcount.VideoOutputFormat
//指定 mapreduce 端的输入数据格式，等号后面为 java 编写的数据输出格式 class 类名
-libjars t2.jar //java 的 JAR 包，包括输入输出数据格式的 class 文件
-files config.ini //参数文件
-reduces * //指定 reduce 的个数
-input /***** //指定输入数据在 HDFS 上的路径
-output /***** //指定输出数据在 HDFS 上的路径
-program /test/transaudio1 //C++的可执行程序在 HDFS 上的路径

3.3.3 音视频合并模块

FFmpeg 提供了三种音视频合并的方法，官方推荐使用的是 FFmpeg concat 滤镜，它的成功率最高而且最为方便。FFmpeg concat 的作用是链接音视频流，将其一个接一个地连在一起。该过滤器作用于同步的音视频流的分段，同种类型的所有分段必须保证具有相同数量的数据流，并且在输出端数据流的数目也不会变。FFmpeg concat 的典型命令示范如下：

FFmpeg -f concat -i *.txt -c copy [outputfile]

需要注意的是，在执行命令进行合并处理之前，需要先创建一个文本文件，记录待合并的音视频文件名，concat 滤镜才会按照顺序对音视频文件分段进行合并。另外，concat 滤镜需要 FFmpeg1.1 及其以上版本支持，文件名的命名也应尽量避开奇怪的字符，否则会导致合并的出错。

3.4 分布式转码系统的工作流程

下面结合图 3.4 介绍整个系统的工作流程：

步骤①：将待转码的文件存储在 HDFS 上；

步骤②：运行 MapReduce 的 mapper()函数进行转码；

步骤③：通过 Hadoop pipes 提交作业进行 FFmpeg 转码，根据用户需求配置转码任务的配置文件，其中包括转码格式，以及一些可调参数（音频：采样率、声道数、比特率；视频：码率、分辨率、播放宽高比、音频声道、音频采样率、音频比特率）；

步骤④：利用 FFmpeg concat 进行转码后音视频文件的合并；

步骤⑤：获得最终的目标文件，存储到 HDFS 上。

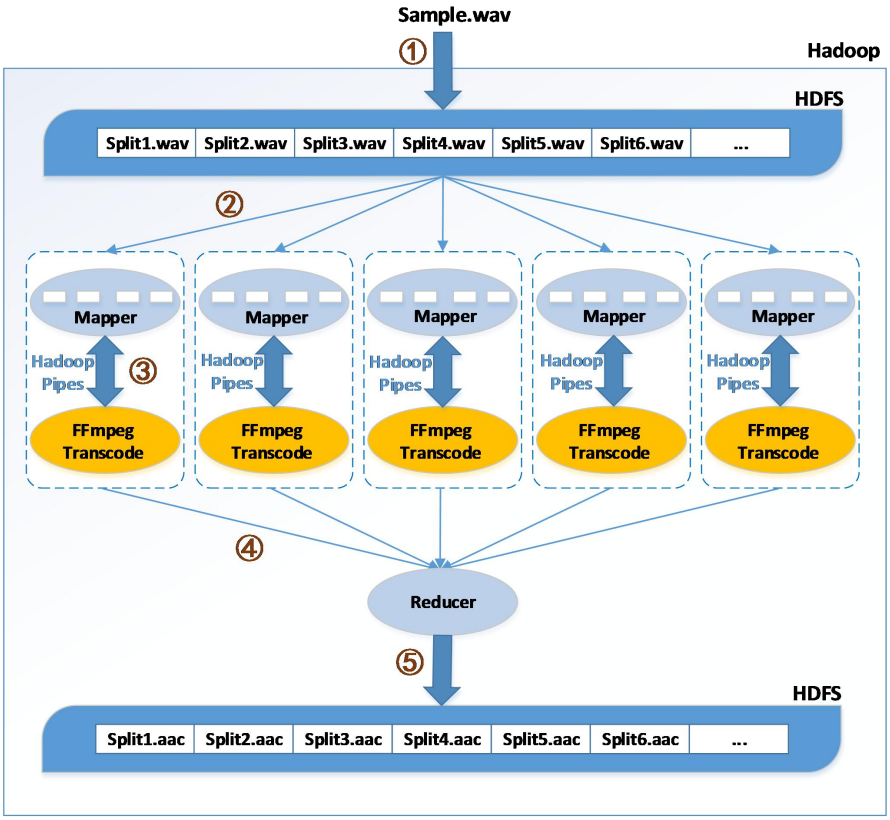

图 3.5 分布式转码系统流程图

3.5 本章小结

本章首先对 Hadoop 平台下音视频转码系统进行了简要的介绍，本系统是全媒体大数据与智能挖掘平台的一个子系统。从系统的架构出发，介绍了分布式媒体转码系统的设计思路，解决的问题；详细介绍了各个功能模块的设计以及整个系统的工作流程。

第 4 章 Hadoop 平台下基于信息位置的视频分割方法

由于视频的非结构化特性，帧与帧之间存在强关联性，Hadoop MapReduce 不能直接对视频进行转码。本章将介绍一种适用于 Hadoop 平台的基于位置的视频分割方法，介绍了对 MPG、TS、WMV、AVI 和 MKV 采取的不同的分割方法，详细描述了分割流程，并对分割方法的性能和系统的整体性能进行了测试，最后对本章进行了小结。

4.1 问题的提出

Hadoop MapReduce 并行计算框架适用于海量数据的处理，其结构简单，能有效支持数据密集型应用，非常适合并行处理音视频文件。但由于 Hadoop 内置数据类型有限，视频文件不能直接利用 MapReduce 框架进行处理。目前使用 MapReduce 处理音视频文件的方法主要分为两种：

1. 将音视频转换成二进制串行化数据处理。SequenceFileInputFormat 专用于 Hadoop 的高性能的二进制格式，如图片、视频、声音等媒体文件，其中具体键值对的格式由用户自定义。

2. 实现自定义的音视频文件接口。根据 MapReduce 模型数据流的特征，设计可以处理音视频文件的 Hadoop 数据类型，直接针对音视频文件进行处理，对视频帧进行分类，在关键帧处进行额外处理。

上述的两种方案，无论是重写 SequenceFileInputFormat 和 SequenceFileOutputFormat 接口还是直接自定义输入输出接口，所需的工作量都太大。所以很多学者选择直接使用开源工具在 Hadoop 某个固定节点上完成视频分割预处理，本文为了避免视频分割节点负载过大造成的系统性能瓶颈，寻求一种更经济的方式来实现音视频文件在 Hadoop 上的分割。

4.2 Hadoop下基于位置信息的视频分割方法

视频分割是为了将连续的海量视频流分割为独立的视频片段，根据需求的不同，可以分为按关键帧分割、按时间分割和按视频大小分割。最常见的方法是基于时间的分割，FFmpeg 提供了通过命令行基于时间分割视频的方法，这种方法需要遍历整个视频文件，对比时间标签，取出某段时间的视频帧。但是遍历整个视频文件十分耗时，FFmpeg 每次进行分割都需要从头开始遍历，而不能在截断点附近的时间标签开始遍历。出于对转码集群转码效率和负载均衡的考虑，本文采用基于视频大小的分割方法，尽可能将一个大视频元文件分割成相同大小的视

频文件，但是由于视频中帧与帧之间存在依赖关系，无法理想地实现相同视频大小的分割。

另外，FileInputFormat 定义的逻辑记录并不一定要匹配 HDFS 的文件块。例如，TextInputFormat 的逻辑记录是以行为单位的，那么很有可能某一行会跨文件块存放。虽然这对程序的功能没有什么影响，行不用担心会丢失或者出错，但这种现象应当引起注意，因为这意味着那些“本地的”map（即输入数据存储在进行运算的节点上的 map 任务）会执行一些远程的读操作。不过由此带来的额外开销一般都不是特别明显，可以忽略不做处理。

图 4.1 是一个被分成若干行的文件，行的边界与 HDFS 块的边界没有对齐的情况。分片边界与逻辑记录的边界对齐，这里是行边界，所以第一个分片包含了前 5 行，即第 5 行跨越了第 1 个 split 和第 2 个 split。第 2 个 split 从第 6 行开始。

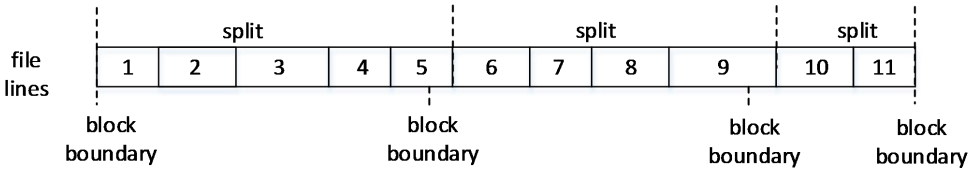

图 4.1 TextInputFormat 的逻辑记录和 HDFS 块

基于 Split 和 RecordReader 的这一特性，本文提出了通过改写 getSplits 函数中的 FileSplit 方法——makesplit 方法来实现在 Hadoop 上的视频分割。getSplits 函数需要修改的内容是：添加分割视频转码输出后的顺序以及转码开始判断参数的传递。由于 makesplit 函数中只提供了四个参数，其中 startpos、len 和 splitHosts 已经固定，因此将视频顺序参数和判断参数添加到文件路径 Path 中，然后再从 Path 中进行解析获取这两个参数。

利用 Hadoop MapReduce 进行视频分割转码的流程如图 4.2 所示。

在 MapReduce 读取分片执行转码任务之前，先读取 Pos 文件，确定分割点。Pos 是数据在媒体流中的字节偏移量，是视频文件 AVPacket 结构体内的附加信息。通过遍历视频文件读取出来，存在 Pos.txt 文件中备用。具体的分割流程如图 4.3 所示。

FFMPEG SDK 有两个地方可以用来控制写入的时间戳，一个是 AVPacket，一个是 AVFrame。用 av_read_frame( )函数可以读取一个 AVPacket 视频包，其中 AVPacket 是 FFmpeg 用来暂存解复用之后、解码之前的媒体数据（一个音/视频帧、一个字幕包等）及附加信息（pts、dts、pos 等），而 pts 和 dts 这两个参数是用来控制视频帧显示和解码顺序的时间戳，详情见表 4.1。但是，所需的 pts 应为解码得到的原始帧的 pts，而解码函数 avcodec_decode_video( )中得到的帧只是一个

AVFrame，其中所包含的 pts 是当前帧将来在还原播放时候的时间戳。所以，本文通过读取 AVPacket 获取 pts 和 dts 参数，对其进行转换。

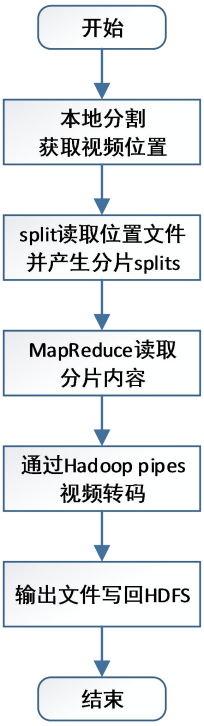

图 4.2 Hadoop MapReduce 视频分割流程

表 4.1 AVPacket 参数及其意义

参数名	意义
pts	显示时间戳
dts	解码时间戳
stream_index	所属媒体流的索引
data	数据缓冲区指针，size 为长度
duration	数据的时长
pos	数据在媒体流中的字节偏移量
destruct	用于释放数据缓冲区的函数指针
flags	标志域，最低位置“1”表示该数据为关键帧

判断当前 pts 时间戳与“下一个视频开始的时间点-1s”的关系，如果“pts 时间戳≤下一个视频开始的时间点-1s”，对分割时间段内的视频数据更新 pts 并保存，不在分割时间段内则更新结束位置继续读取下一个 AVPacket；如果“pts 时间戳＞下一个视频开始的时间点-1s”的话，需要在更新视频数据 pts 之前，对视频开始读取的位置进行更新，即认为该 AVPacket 是下一分割时间段内的第一个数据包。

“下一个视频开始的时间点-1s”表示的是视频开始时间的前一秒，在分割视频

的时候，通过将视频开始时间的前一秒作为视频读取的开始位置，即往前多读取了一秒钟的数据包，避免了分割恰好发生在关键帧处，转码后视频跳帧的情况出现。

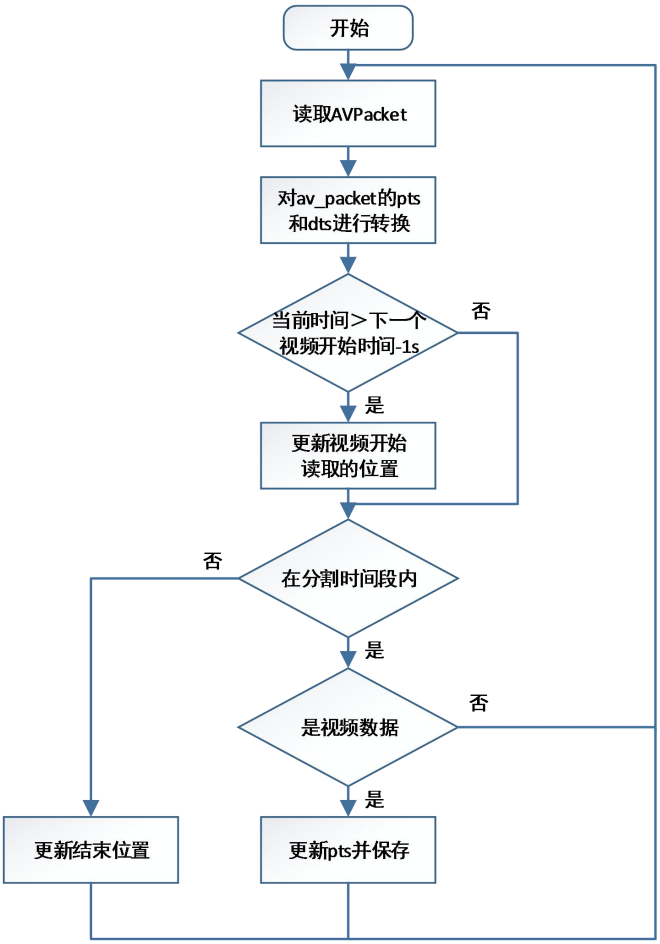

图 4.3 Hadoop 平台下基于位置信息的分割流程

判断当前 pts 时间戳与“下一个视频开始的时间点-1s”的关系，如果“pts 时间戳≤下一个视频开始的时间点-1s”，对分割时间段内的视频数据更新 pts 并保存，不在分割时间段内则更新结束位置继续读取下一个 AVPacket；如果“pts 时间戳＞下一个视频开始的时间点-1s”的话，需要在更新视频数据 pts 之前，对视频开始读取的位置进行更新，即认为该 AVPacket 是下一分割时间段内的第一个数据包。

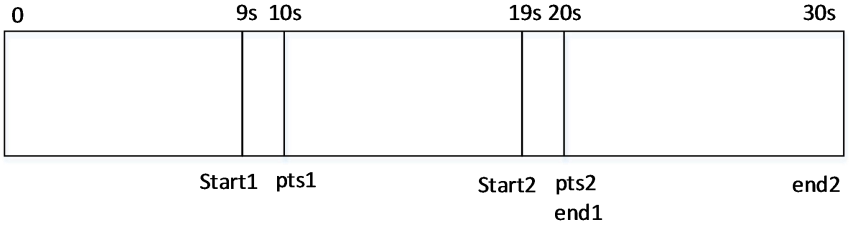

图 4.4 视频时长 30s 的文件分割点位置信息

“下一个视频开始的时间点-1s”表示的是视频开始时间的前一秒，在分割视频的时候，通过将视频开始时间的前一秒作为视频读取的开始位置，即往前多读取

了一秒钟的数据包，避免了分割恰好发生在关键帧处，转码后视频跳帧的情况出现。

以视频时长为 30s 的文件为例如图 4.4 所示，第一个 split 为 0-10s，第二个 split 为 9-20s，第三个 split 为 19-30s，视频读取的开始位置设为下一段视频开始时间的前一秒，分割视频段的结束时间为结束位置。第一个 pts，由于是直接读取数据而没有改变视频结构数据，因此在进行编码的时候，只需要判断 pts 的值是否相等，如果相等即为第一个 pts，可以开始进行编码。

<table>
<tr><td>EBML
Header</td><td colspan="4">EBML Version 文件版本</td></tr>
<tr><td></td><td colspan="4">DocType 文件类型</td></tr>
<tr><td rowspan="15">Segment
（音视频的实际数据）</td><td rowspan="2">Meta seek Information
（索引信息）</td><td rowspan="2">Seek Head</td><td rowspan="2">Seek</td><td>Seek ID</td></tr>
<tr><td>Seek Position</td></tr>
<tr><td rowspan="2">Segment Information
（识别文件信息）</td><td rowspan="2">Info</td><td colspan="2">Title</td></tr>
<tr><td colspan="2">Segment UID</td></tr>
<tr><td rowspan="3">Track
（音视频的基本信息）</td><td rowspan="3">Tracks</td><td rowspan="3">Track
Entry</td><td>Track Name</td></tr>
<tr><td>Track Number</td></tr>
<tr><td>Track Type</td></tr>
<tr><td>Chapters</td><td>Chapters</td><td colspan="2">Edition Entry</td></tr>
<tr><td rowspan="7">Clusters
（音视频帧数据）</td><td rowspan="7">Cluster</td><td colspan="2">timecode</td></tr>
<tr><td rowspan="5">Blockgroup</td><td>Block</td></tr>
<tr><td>Referenceblock</td></tr>
<tr><td>Block</td></tr>
<tr><td>Block</td></tr>
<tr><td>BlockDuration</td></tr>
<tr><td colspan="2">BlockDuration</td></tr>
<tr><td rowspan="2">Cueing data
（关键帧的 index）</td><td rowspan="2">Cues</td><td rowspan="2">Cue Point</td><td>Cue Time</td></tr>
<tr><td>Cue Position</td></tr>
<tr><td rowspan="2">Attachment
（关联文件）</td><td rowspan="2">Attachments</td><td rowspan="2">Attached
file</td><td>File Name</td></tr>
<tr><td>File Data</td></tr>
<tr><td rowspan="2">Tagging</td><td rowspan="2">Tags</td><td rowspan="2">Tag</td><td>Multitle</td></tr>
<tr><td>Language</td></tr>
</table>

图 4.5 MKV 文件结构信息

这种分割方式在 TS、WMV、MPG 和 AVI 等视频格式上都得到了完美的应用，但在 MKV 格式上，分割后的视频文件播放异常，转码报错。通过利用 MKV 文件解析工具查看文件，发现 MKV 的文件结构与其他视频格式不同，其音视频帧

数据存储在 cluster 结构中。MKV 文件结构信息详见图 4.5。

考虑到 MKV 视频格式与其他格式的不同，所以需要根据 cluster 的位置进行分割。但是由于在编码 MKV 的过程中没有进行帧数据的缓存，直接利用 cluster 的位置进行划分，会导致转码后视频图像的跳帧。本文采用向前多读取一个 cluster 的方法，由于 AVPacket 中含有变量 pos 值，因此可以使用 pos 来保存要转码的 cluster 的位置，并在进行编码的时候进行判断。需要注意的是，编码时选择向前多读取一个 cluster，而转码的时候依旧从原开始位置进行判断转码。

由于在分割 MKV 的时候，需要获取文件中 cluster 的位置。因此需要借助 EBML 和 Matroska 这两个库对 MKV 文件进行解析。EBML 是一种更为灵活的音视频框架，其扩展性强大，可以支持更多的音视频压缩格式拓展，此框架已应用于 Matroska。Matroska 是一种新的多媒体封装格式，也被称作多媒体容器，mkv 只是 Matroska 媒体系列中的一种文件格式。

EBML 和 MAtroska 两个库的安装过程如下：

1．超级用户下利用命令 tar -jxvf 解压文件 libmatroska-1.4.4.tar.bz2 和 libebml-1.3.3.tar.bz

2．解压进入后进入 libebml-1.3.3 下，利用依次执行./configure、make、make install 命令，这里一定要先安装 ebml 库，因为在 matroska 库的安装需要用到 ebml 库。安装成功后可以在/usr/local/lib/下看到库

3．进入 libmastroska-1.4.4 下，执行 ./configure --prefix=/usr PKG_CONFIG_PATH= “/usr/local/lib/pkgconfig/”，make 和 make install 命令，安装完成后在/usr/local/lib/下看到库

具体的 MKV 视频文件分割流程如图 4.6 所示。

打开 MKV 文件，将其与 EBML 流关联，跳过 MKV 文件中的 EBML 文件头部分，找到视音频实际数据的 segment 部分。在 segment 中寻找元素非空的 cluster 并输出其位置，接着在 cluster 中寻找 block 的位置，并将其输出。

至此，本文已经完成了在 Hadoop 上实现 MKV、MPG、TS、WMV 和 AVI 五种格式的视频分割。由于所有的文件分割都是基于位置信息进行的，因此分割后文件的文件头部分相同，所以在读取分割数据前应先读取文件头数据。本文将分割文件位置信息的第一个数据作为文件头的长度，在产生视频分片时将其添加到路径参数中。在读取文件时，先读取 header 长度的数据，再读取视频分片的数据。需要注意的是，在 Hadoop 下分割 MKV 的时候，Hadoop 读取媒体流中的偏移位置是相对于当前文件操作符的，因此每次读取分割文件需要将偏移减去头文件长度。

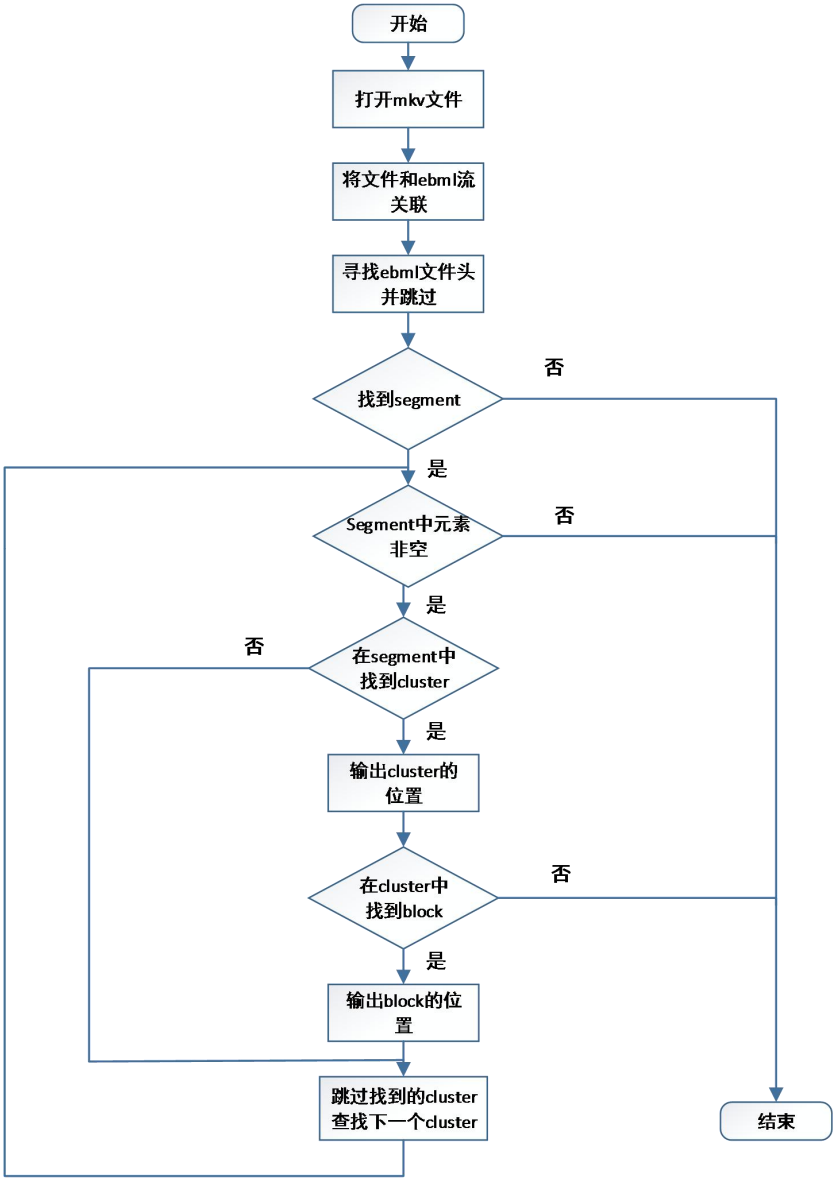

图 4.6 Hadoop 平台下基于位置信息的 MKV 视频文件分割流程

4.3 系统测试

4.3.1 实验测试平台

为了测试 Hadoop 下基于位置信息的分割方法以及整个音视频转码系统的性能，在 6 台物理机上搭建了分布式集群，其中 1 个主控节点，5 个转码节点。在此集群上部署的 Hadoop 音视频转码系统是同构集群，在此只介绍一台服务器的配置。具体的节点配置表 4.2 所示。

表 4.2 样本集群的节点配置

硬件配置			操作系统	软件配置		
CPU	内存	硬盘		Hadoop	FFmpeg	音频库
New Pentium Dual Core G4400(3.3G/ 3M/2 核)	4G(DDR4 2133)	500G(3.5"，SATA)	centOS6.5	Hadoop2.5.2	FFmpeg2.8	Fdkaac1.7

为了对系统进行有效的测试，选取以下测试数据（在这里以音频测试为例）。每组实验均进行 10 次，最后的结果取平均值。转码后的格式对转码时间并没有影响，采样率、码率决定了一个音频的音质和大小，FFmpeg 提供的 PCM 和 MPEG 系列的无损压缩方法在压缩效率上并没有明显的差别。具体如表 4.3 所示。

表 4.3 样本集群测试数据

文件名	文件大小	编码格式	声道	采样率	码率
Test1.wav	1.2GB	PCM	立体声	44.1khz	64kbps
Test2.wav	2.0G	PCM	立体声	48khz	2304kbps
Test3.mp2	225MB	MPEG	立体声	48khz	256kbps
Test4.wav	3.1GB	PCM	立体声	44.1khz	64kbps
Test5.wav	3.1GB	PCM	立体声	48khz	64kbps
Test6.wav	3.1GB	PCM	立体声	44.1khz	128kbps
Test7.wav	3.1GB	PCM	立体声	48khz	128kbps
Test8.wav	3.1GB	PCM	立体声	44.1khz	192kbps
Test9.wav	3.1GB	PCM	立体声	48khz	192kbps

为了对 Hadoop 分割方法进行有效的测试，选取以下测试数据，具体如表 4.4 所示。每组实验均进行 10 次，最后的结果取平均值。Hadoop 平台下基于位置信息的分割时间与 Super Video Splitter 的分割时间作对比，设计性能对比实验。

表 4.4 Hadoop 分割视频方法测试数据

文件名	文件大小	混合码率	帧率	采样率	声道
Test21.ts	10.4GB	PCM	29.97fps	44.1khz	2 声道
Test22.wmv	0.7GB	810kbps	29.97fps	44.1khz	2 声道
Test23.mpg	2.5GB	8597kbps	25.00fps	48.0khz	2 声道
Test24.avi	2.4GB	2054kbps	29.97fps	48.0khz	6 声道
Test25.mkv	1.2GB	1896kbps	29.97fps	48.0khz	2 声道

4.3.2 功能测试

Hadoop 音视频转码系统的主要过程是：上传音视频文件元数据到 HDFS（上传过程中完成视频的分割）、接受用户提交的转码请求、对音视频分片用 MapReduce 进行集群转码、记录视频分片的转码时间、合并音视频文件、转码后的最终结果存入 HDFS。本文系统支持同时提交多个转码任务，一个转码命令即为一个转码任务，如需提交多个任务需要在命令中声明任务的优先级。

通过 Hadoop 音视频转码系统转码后音视频文件播放正常，视频质量如图 4.7 所示，无跳帧卡顿现象出现，实现了预期功能。根据网络带宽的需求和终端的硬件处理能力，可以对视频转码过程中的视频参数进行指定，需要高质量的视频画质，可以通过配置较大的码率、分辨率来实现，对音频立体声的需求可以通过配置多音频声道实现。

图 4.7 Hadoop 集群转码后的一帧视频

4.3.3 性能测试

本文利用 Super Video Splitter 对视频文件进行分割，与 Hadoop 分割方法作对比。Super Video Splitter 是一种视频文件分割工具，支持 AVI/DivX、MPEGI/II、VOB、DAT、WMV、ASF 文件格式，可以把一个视频文件分割为多个视频文件，系统提供了不同的分割模式，可以设置分割后的文件数量或按照规定的大小进行分割。

实验结果如表 4.5 所示，Hadoop 下分割方法在分割时间上远远优于 Super Video Splitter，尤其是在数据量大的时候，性能的提升十分可观。由于 Super Video Splitter 工具分割后的视频段还需上传至 HDFS，额外的时间消耗也应计算在内。实验分析了 Hadoop 分割方法和 Super Video Splitter 同时分割 Super Video Splitter 不支持 TS、MKV 等视频格式的分割，Super Video Splitter 支持的视频格式有限，

且扩展难度较大，Hadoop 下基于位置信息的分割方法是应用于 Hadoop 集群转码上的较优选择。

表 4.5 Hadoop 下分割与 Super Video Splitter 分割的时间对比

文件名	Hadoop 分割时间（s）	Super Video Splitter 时间（s）	上传 HDFS 时间（s）
Test21.ts	100	*	1100
Test22.wmv	39	480	68
Test23.mpg	137	3014	162
Test24.avi	134	2460	154
Test25.mkv	29	*	128

*：Super Video Splitter 视频分割工具不支持该格式

通过本文提出的基于位置信息的视频分割，就可以成功地实现 Hadoop 平台下的音视频转码。实验测试了分布式转码系统的吞吐率，结果如图 4.8 所示。

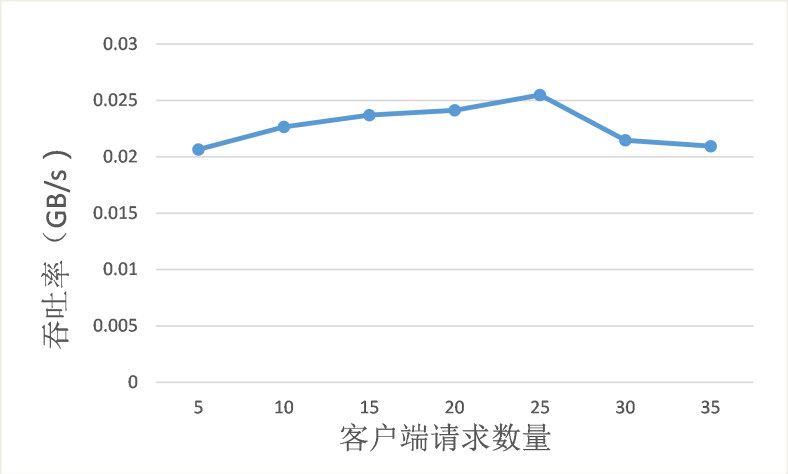

图 4.8 Hadoop 转码集群吞吐率

实验测试了相同转码格式、相同转码参数配置下，文件大小对转码速度的影响，结果如图 4.9 所示。横坐标是文件大小（单位是 GB），纵坐标是转码速率，即单位时间（S）Hadoop 集群转码的数据量。可以看出，随着数据规模的增大，Hadoop 转码的优势便体现出来，且优势更加明显。实验结果也从一定程度验证了 Hadoop 适应于处理海量大数据的特征，文件过小，Hadoop 系统启动的开销和作业的调度所需的时间开销，会使集群转码的优势没那么明显。

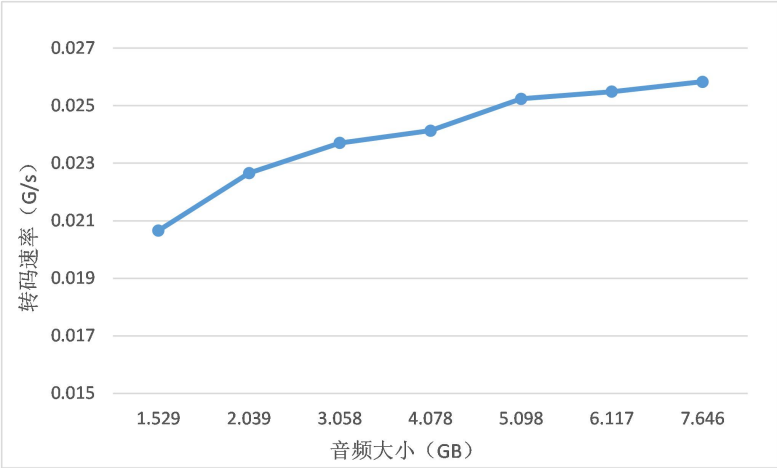

图 4.9 文件大小对转码速率的影响

实验选取 Test1.wav 文件，转码为 AAC 格式，来测试分片大小对转码性能的影响（其中 HDFS 的块大小为 64MB）。选取的分片大小和分片数量的关系如表 4.6 所示。

表 4.6 分片大小与分片数量的关系

分片大小（MB）	分片数量
7.7	160
15.4	80
30.8	40
61.6	20

分别运行转码程序，测得的系统转码时间与分片大小的关系如图 4.10 所示。

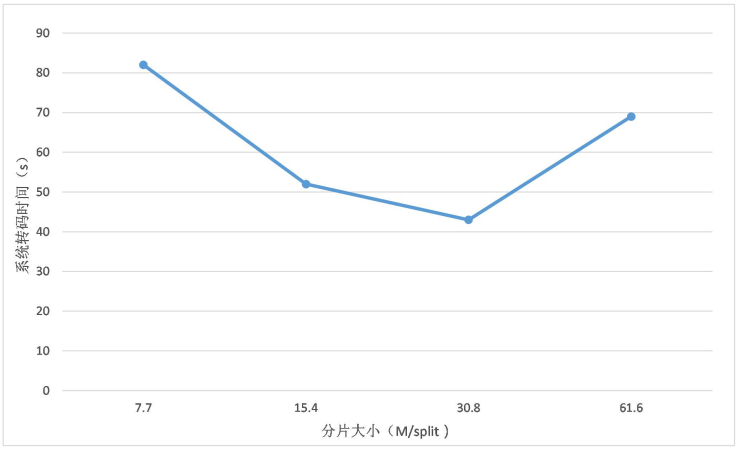

图 4.10 分片大小对转码时间的影响

图 4.10 反映出当分片大小过小时，尤其远小于 HDFS 数据块的大小（64M）时，系统的转码时间格外长。这是因为使用 5 个 DataNode 进行转码，每个节点分配到的转码任务数是不均衡的，任务量较大的节点拖慢了整个系统的转码进程。而当分片大小接近 HDFS 数据块大小时，系统的转码时间也会增长，这是因为

Hadoop 系统启动任务、任务调度过程传输分片都会带来额外的时间开销，如果单纯追求每个任务转码时间较短的话，初始化任务和任务的调度所占时间比例就会比较大。所以建议将视频文件的分片大小设为 HDFS 数据块一半的值。

为了验证上述结论，进行了文件分片粒度更细的测试。实验仍选取了大小为 1.2G 的 Test1.wav 文件，转码为 AAC 格式，来测试分片大小对转码性能的影响（其中 HDFS 的块大小为 64MB）。选取的细粒度分片大小和分片数量的关系如表 4.7 所示。

表 4.7 细粒度分片大小与分片数量的关系

分片大小（MB）	分片数量
20	60
30	40
40	30
50	24

测得的系统转码时间与细粒度分片大小的关系如图 4.11 所示。图 4.11 表明在分片大小粒度更细的情况下，在选取 30MB 左右的分片大小时，系统的转码时间最短。所以转码前分割，建议选取 HDFS 数据块一半的值作为分片大小。实验同时考察了在 HDFS 数据块大小为 128MB 时，分片大小对转码时间的影响，见图 4.12。结果表明，分片大小对转码时间的影响，与 HDFS 的数据块大小无关。

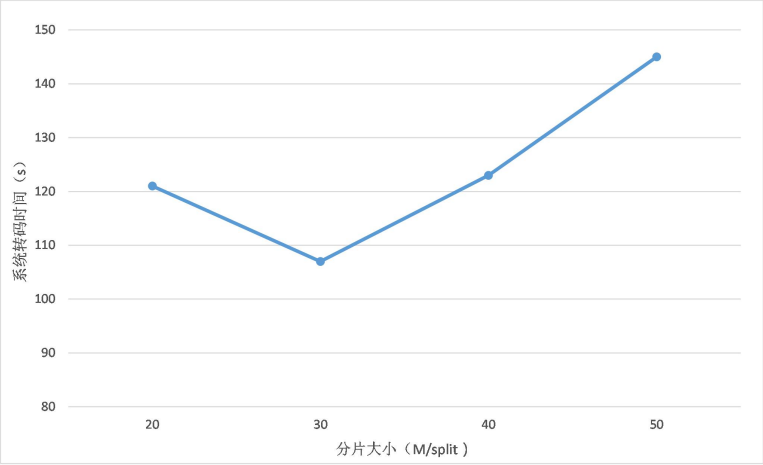

图 4.11 细粒度分片大小对转码时间的影响

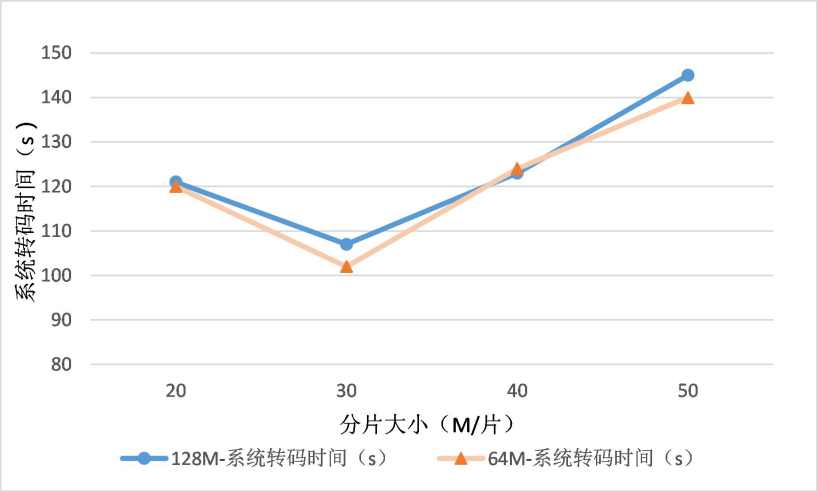

图 4.12 HDFS 块大小对分片大小选择的影响

实验比较了不同转码格式对转码时间的影响，与 Hadoop 集群转码作对比的是本地转码。第一组实验是 2GB 的 WAV 文件（Test2.wav）转 mp2（采样率为 48khz，码率为 256kbps，立体声）；第二组实验是 2GB 的 WAV（Test2.wav）转 aac（采样率为 48khz，码率为 64kbps，立体声）；第三组实验是 225MB 的 mp2（Test3.mp2）转 aac（采样率为 48khz，码率为 64kbps，立体声）。实验结果表明（见图 4.13），无论是哪种格式的转码，Hadoop 集群转码的性能都大幅度优于本地转码。

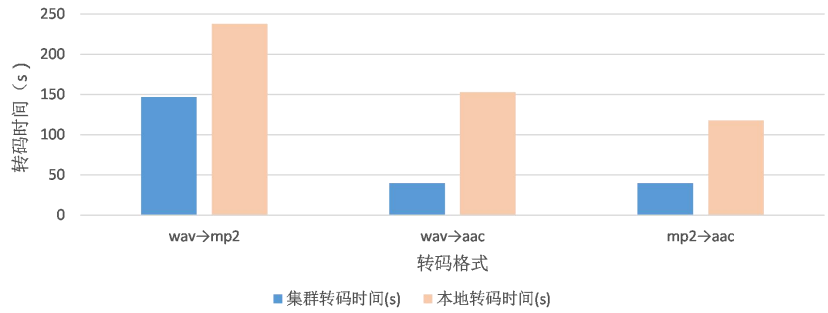

图 4.13 不同转码格式下集群转码与本地转码的时间对比

实验同样比较了不同转码参数对转码时间的影响，主要对码率（图 4.14 中的 br）和采样率（图 4.14 中的 sr）重点考察。分别对码率为 64Kbps、128Kbps、192kbps 和采样率为 44.1khz、48khz 的情况进行测试。实验选取了 3.1G 的 WAV 文件（Test4.wav、Test5.wav、Test6.wav、Test7.wav、Test8.wav 和 Test9.wav），转码为 AAC 格式，来测试不同转码参数对转码时间的影响（其中 HDFS 的块大小为 64MB，视频分片大小为 32MB）。音频采样率是指录音设备在一秒钟内对声音信号的采样次数，所以采样率越大，文件包含的信息量也就越多，所需的转码时间也就越长。实验结果表明，转码参数对转码时间影响不大，在各种参数下 Hadoop

集群转码的性能依旧大幅度优于本地转码。

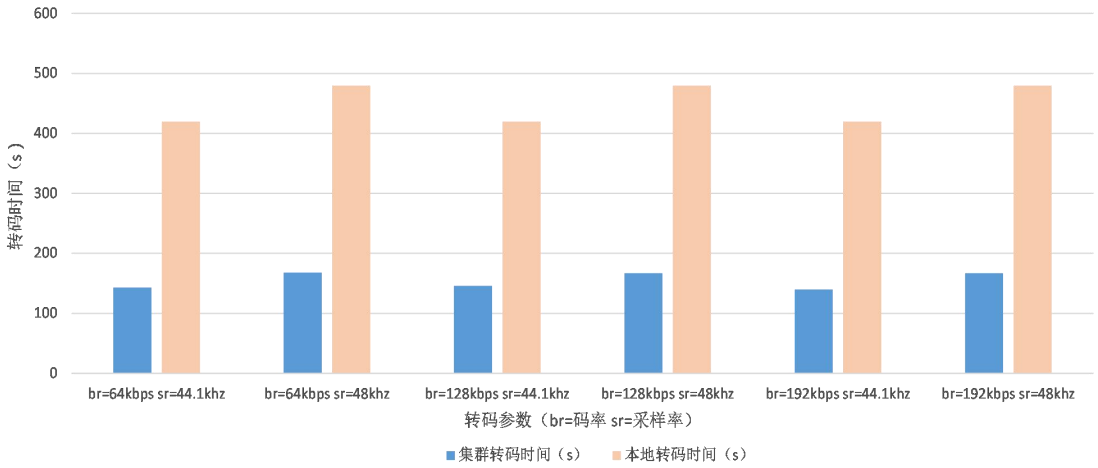

图 4.14 不同转码参数下集群转码与本地转码的时间对比

4.4 本章小结

本章提出了 Hadoop 平台下基于位置信息的视频分割方法，成功地在 Hadoop 上实现了 MKV、MPG、TS、WMV 和 AVI 五种视频格式的分割。并对该方法进行了性能测试，与 Super Video Splitter 分割器的结果作对比，验证了方法的优越性。最后对 Hadoop 音视频转码系统进行了功能测试和性能测试，比较了视频大小、视频分片大小、转码格式以及转码参数对转码时间的影响，并对结果进行了分析。

第 5 章 基于样本资源需求的容器资源配置方法

本章将介绍一种适用于基于样本资源需求的容器资源配置方法，介绍了方法的具体流程，详细描述了基于样本资源需求的容器资源配置算法，实验测试了优化后转码集群的性能，并对提出算法的正确性做了验证性实验，最后对本章进行了小结。

5.1 问题的提出

第 4 章实验表明，Hadoop 集群转码虽然在转码速度上大幅度优于本地转码，但是经实验发现，在虚拟化集群中，实验结果并未完美实现并行（即本地转码时间为 T，数据节点为 N 时，集群转码时间远大于 T/N）。多出的时间开销不仅是运行 Hadoop 系统所带的系统时间开销以及文件迁移开销，通过查看任务运行日志发现，由于节点上的资源有限，而转码又是极消耗内存资源和 CPU 计算资源的任务，当数据节点上的任务获取不到执行该任务所需的资源，就会一直等待。而 YARN 的资源调度机制分配给每个任务的资源量是固定的，容器配置的默认值是 <1G,1core>，当数据节点虚拟化分配得到的内存是 6G，CPU 核是 12 的时候，节点上会同时并发 6 个进程，而实际的内存并没有这么多，从而导致内存占用率过高的情况。同节点的子任务间抢占资源，未分配到资源的任务拖长了整个任务的运行时间。根据节点资源的实际情况，调节单个节点上任务的并发度，对转码的加速研究具有重要的意义。

进入 MRv 2 后，并发任务的数量对系统性能和资源利用率的影响变得更为明显。在 MRv 2 之前，每个节点上并发的 mapper 和 reducer 数量是由管理员指定的。如果想要修改这个数量，系统管理员需要通过设定 mapred.child.java.opts 和 mapred.child.ulimit 来修改每个 mapper 或 Reducer 的默认内存分配。Cloudera 的建议是 Mapper 与 Reducer 的比例为 3:2，其中总任务数应少于节点上的核数。这种建议旨在集群处于高负载情况下时，达到最佳的集群性能。但是，MRv 1 中严格的任务槽（slot）计算模型使得集群并行能力没有得到充分的发挥，自由度不够。

与此不同的是，在 MRv 2 中，mapper 和 reducer 并行的数量是系统管理员通过 YARN 基于资源配置计算得到的。每个节点分配一定数量的内存（通过设定参数 yarn.nodemanager.resource.memory-mb ）和 CPU 核（通过设定参数 yarn.nodemanager.resource.cpu-vcores）供 YARN 上的应用使用。一个 MapReduce 作业就是一个 YARN 应用，作业会向 YARN 索取资源容器来执行它的 map 和 reduce 任务，只需要设定参数 mapreduce.[map|reduce].memory.mb 和

mapreduce.[map|reduce].cpu.vcores。YARN 管理的内存和 CPU 可以在 map 和 reduce 间共享，当节点拥有任务执行所需的内存和 CPU 资源，节点就可以执行 MRv 2 任务。

这种方法是对 MRv 1 的改进，因为管理员无需将内存和 CPU 置入 slot 中。在 MRv 1 中，每个节点的并发任务数是由参数 mapred.map.tasks.maximum 和 mapred.reduce.tasks.maximum 定义的。而在 MRv 2 中，可以通过将每个节点分配到的资源数除以 YARN 分配给每个 MapReduce 任务的资源数得到单个节点的任务并行度（取内存和 CPU 两种类型资源计算出的最小值）。对于内存来说，用 yarn.nodemanager.resource.memory-mb 除以 mapreduce.[map|reduce].memory.mb 得到，对于 CPU 来说，则是用 yarn.nodemanager.resource.cpu-vcores 除以 mapreduce.[map|reduce].cpu.vcores 得到。所以 YARN 用户可以通过修改 container 的资源配置来指定节点的任务并发度。

container 是 YARN 中资源的抽象，它封装了某个节点上一定量的资源（CPU 和内存两类资源）。然而集群资源发生变更时，在众多资源配比下以较低的开销找到最佳的 container 配置是十分困难的，需要一种方法可以在集群发生更换的时候，动态自适应的改变 container 的配置，本文提出的基于样本资源需求的容器资源配置方法，很好地解决了这个问题。

5.2 基于样本资源需求的容器资源配置方法

基于样本资源需求的容器资源配置方法的流程如图 5.1 所示。

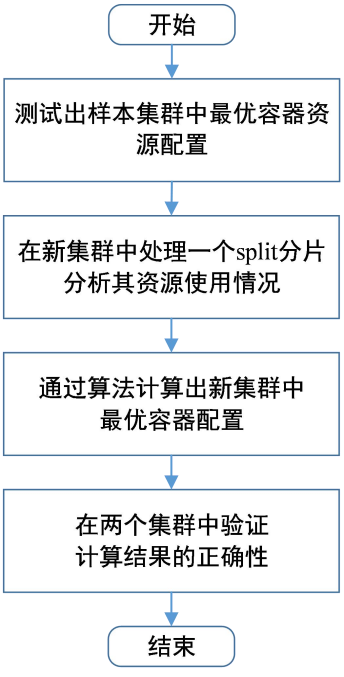

图 5.1 基于样本资源需求的容器资源配置方法流程图

在样本集群中，对所有可能的内存和 CPU 核资源配置进行测试，找到转码时

间最短的 container 配置。当集群更换，资源量变化时，在新集群上对一个音视频样本分片进行转码，分析转码单个 split 分片时内存和 CPU 使用情况，需要说明的是，这个 split 分片不具备特殊性，可以是任意的一个分片。通过容器资源配置算法可以计算出最优的 container 配比，此算法可以应用于虚拟化集群，在物理机集群中同样适用。算法的正确性在 5.3.2 节中进行验证，通过在两种性质不同的集群中测试得到验证结果。

算法 1：基于样本资源需求的容器资源配置算法

```
   Data:R, M
   Result:(x, y)
1 for i=1 to n do
2     SetContainer(j, R);
3     Update(R[i, j]);
4       tmp=AverageMem
5        x=ceil(M[0]*a\tmp);
6        if x>ceil(M[0]\1024) then
7             x=ceil(M[0]\1024);
8     tmp=AverageCpu
9        y=ceil(1*cs*b\tmp);
10       if y>M[1] then
11            y=M[1];
12 Procedure SetContainer(j, R)
13  if j=1 then
14       tmp=AverageMem;
15       a=4*tmp\R[0, 1];
16    if j=2 then
17       tmp=AverageCpu;
18       b=8*tmp\1*cs;
19 Return
```

容器资源配置算法的细节见算法 1。核心算法是一个简单的 for 循环，为每个节点配置资源参数（行 1-11）。核心算法部分在 SetContainer（j, R）中，基于样本集群中的实验结果计算内存配置因子 a 和 CPU 核配置因子 b，具体的计算见式（5.1）、（5.2）。参数 a（行 15 中）的计算公式中的“4”是实验得到的样本集群中最优内存的并行度。参数 b（行 18 中）的计算公式中的“8”是实验得到的样本集群中最优 CPU 核的并行度，值得注意的是，计算 CPU 核参数 b 的时候考虑了主频（CPU speed，cs）对计算结果的影响。二维数组 R[i, j]（见表 5.1）被用来存储集群中资源的容量和使用情况，其中 i∈[1, 3]，j∈[1, 2]。数组 M[k]（见表 5.2）被用来存储新集群中资源量，其中 k∈[0,1]。AverageMem 函数是用来计算 R[1, 1]，R[2, 1]和 R[3, 1]的平均值，同理 AverageCPU 是用来计算 R[1, 2]，R[2, 2]和 R[3, 2]的平均值。

$$\frac{R[0,1]}{(R[1,1]+R[2,1]+R[3,1])/3}\cdot a=4 \qquad (5.1)$$

$$\frac{1\cdot cs}{(R[1,2]+R[2,2]+R[3,2])/3}\cdot b=8 \qquad (5.2)$$

表 5.1 集群资源容量及使用情况配置数组 R[i, j]

参数名	含义
R[0, 1]	样本集群单个节点上的内存总量
R[1, 1]	转码单个样本分片需要的内存资源量的平均值
R[2, 1]	转码单个样本分片需要的内存资源量的最小值
R[3, 1]	转码单个样本分片需要的内存资源量的最大值
R[1, 2]	转码单个样本分片需要的 CPU 核资源数的平均值
R[2, 2]	转码单个样本分片需要的 CPU 核资源数的最小值
R[3, 2]	转码单个样本分片需要的 CPU 核资源数的最大值

表 5.2 新集群资源容量配置数组 M[k]

参数名	含义
M[0]	新集群单个节点上的内存总量
M[1]	新集群单个节点上的 CPU 核

当集群变更时，二维数组 R 中的数据需要被更新（行 3 中），更新的依据是分析转码一个样本分片的资源使用情况。每个节点的并行度取决于参数 x 和 y 的值（行 5-11），容器资源的具体配置参数可以通过用资源量除以并行度得到。需要特别说明的是，如果算法中第 5 行（或第 9 行）计算出的结果大于第 6 行（或第 10 行）节点的并行度，则将 x 或 y 设定为节点的最大并行度。

需要注意的是，算法中提到的 CPU 核不是 YARN 系统实际分配给每个任务的 CPU 核数，而是虚拟的 CPU 核。YARN 允许配置每个节点上可使用的物理 CPU 个数，以及物理 CPU 与虚拟 CPU 的比例，而用户申请资源时，申请的是虚拟 CPU。默认情况下，物理 CPU 和虚拟 CPU 是 1：1 的，如果集群是异构的，某些节点上的 CPU 拥有更强的计算能力，则可以通过调整物理 CPU 和虚拟 CPU 的比例，减小各节点之间的计算能力差异。虚拟 CPU 的概念借鉴了“物理内存”和“虚拟内存”，其主要目的是消除集群中 CPU 计算能力的异构性。

如 2.2.2 节所介绍的，Hadoop YARN 同时支持内存和 CPU 两种资源的调度。默认只支持内存，如果想进一步调度 CPU，需要进行一些额外的配置。Hadoop2.5.2 默认的调度器是容量调度器（CapacityScheduler），容量调度器使用资源计算器

（ResourceCalculator）来计算每个应用所需的资源。有以下两种类型的资源计算器：

1. 默认资源计算器（DefaultResourceCalculator）：资源计算（比如说计算 container 的数量时）的时候仅考虑内存的情况；

2. 主要资源计算器（DominantResourceCalculator）：资源计算时考虑内存和 CPU 两种情况。

默认情况下，容器调度器使用的是默认资源计算器，所以只支持内存的调度。如果想要使用主要资源计算器去调度 CPU，需要在“capacity-scheduler.xml”中修改以下下参数，即可实现对 CPU 资源的调度。

```
<property>
 <name>yarn.scheduler.capacity.resource-calculator</name>
 <value>org.apache.hadoop.yarn.util.resource.DominantResourceCalculator</value>
</property>
```

Hadoop YARN 为了易于管理资源和调度资源，内置了资源规整化算法，它规定了最小可申请资源量、最大可申请资源量和资源规整化因子。如果应用程序申请的资源量小于最小可申请资源量，则 YARN 会将其大小改为最小可申请量；如果应用程序申请的资源量大于最大可申请资源量，则会抛出异常，无法申请成功；规整化因子是用来规整化应用程序资源的，应用程序申请的资源如果不是该因子的整数倍，则将被修改为最小的整数倍对应的值，公式为 ceil(a/b)*b，其中 a 是应用程序申请的资源，b 为规整化因子。所以本文提出的算法中也使用了 ceil 函数对计算得到的结果进行规整化操作。

可申请资源量参数在 yarn-site.xml 中设置，具体参数说明见表 5.3。

表 5.3 Hadoop yarn-site 配置文件中的相关参数

参数名	含义
yarn.scheduler.minimum-allocation-mb	最小可申请内存量，默认是 1024
yarn.scheduler.minimum-allocation-vcores	最小可申请 CPU 数，默认是 1
yarn.scheduler.maximum-allocation-mb	最大可申请内存量，默认是 8096
yarn.scheduler.maximum-allocation-vcores	最大可申请 CPU 数，默认是 4

需要特别说明的是，对于规整化因子，不同调度器的情况不同，具体如下：

FIFO 和 Capacity Scheduler，规整化因子等于最小可申请资源量，不可单独配置。而 Fair Scheduler：规整化因子通过参数 yarn.scheduler.increment-allocation-mb、yarn.scheduler.increment-allocation-vcores

设置，默认是 1024 和 1。

5.3 实验分析

5.3.1 实验平台与配置

为了测试基于样本资源需求的容器配置方法的性能，并与本地转码和 Hadoop 集群转码进行比较，除了第 4 章中搭建的样本集群外，又搭建了两个特点差异较大的测试集群。集群 A 是用虚拟化工具虚拟出来的同构集群，共 3 个节点，其中 1 个是主控节点，其他 2 个是转码节点。集群 B 是物理机搭建的同构集群，共 4 个节点，其中 1 个是主控节点，其他 3 个是转码节点。具体的节点配置如表 5.4、5.5 所示。

表 5.4 测试集群 A 的节点配置

虚拟化工具	硬件配置			操作系统	软件配置		
	CPU-vcores	内存	硬盘		Hadoop	FFmpeg	音频库
XenSer-ver6.5	8 核	6GB	100GB	centOS 6.5	Hadoop 2.5.2	FFmpeg2.8	Fdkaac 1.7

表 5.5 测试集群 B 的节点配置

硬件配置				操作系统	软件配置		
CPU	CPU-vcores	内存	硬盘		Hadoop	FFmpeg	音频库
New Pentium Dual Core G4400 (3.3G/3M/2 核)	8 核	4G(DDR4 2133)	500G (3.5", SATA)	centOS 6.5	Hadoop 2.5.2	FFmpeg 2.8	Fdkaac 1.7

为了对系统进行有效的测试，选取以下测试数据（在这里以音频测试为例）。每组实验均进行 10 次，最后的结果取平均值。具体如表 5.6 所示。

表 5.6 容器配置方法测试数据

文件名	文件大小	编码格式	声道	采样率	码率
Test11.wav	3.1GB	PCM	立体声	44.1khz	64kbps
Test12.wav	3.1GB	PCM	立体声	48khz	64kbps
Test13.wav	3.1GB	PCM	立体声	44.1khz	128kbps
Test14.wav	3.1GB	PCM	立体声	48khz	128kbps
Test15.wav	3.1GB	PCM	立体声	44.1khz	192kbps
Test16.wav	3.1GB	PCM	立体声	48khz	192kbps
Test17.wav	1.5GB	PCM	立体声	44.1khz	64kbps
Test18.wav	3.1GB	PCM	立体声	44.1khz	64kbps
Test19.wav	6.1GB	PCM	立体声	44.1khz	64kbps
Test20.wav	9.1GB	PCM	立体声	44.1khz	64kbps

5.3.2 实验结果分析

为了体现系统优化后在转码时间上的降低，将本文系统的转码时间与原系统和本地转码两种情况进行对比，对比系统采用下述方式进行转码：首先将待转码视频上传 HDFS，在新系统中转码一个样本 split 任务，计算出该系统最优的 container 资源配置，比较转码时间，测试新系统中所有可能的 container 配比，验证基于样本资源需要的容器资源配置方法的正确性。下面是三个转码方式的对比实验以及实验结果。

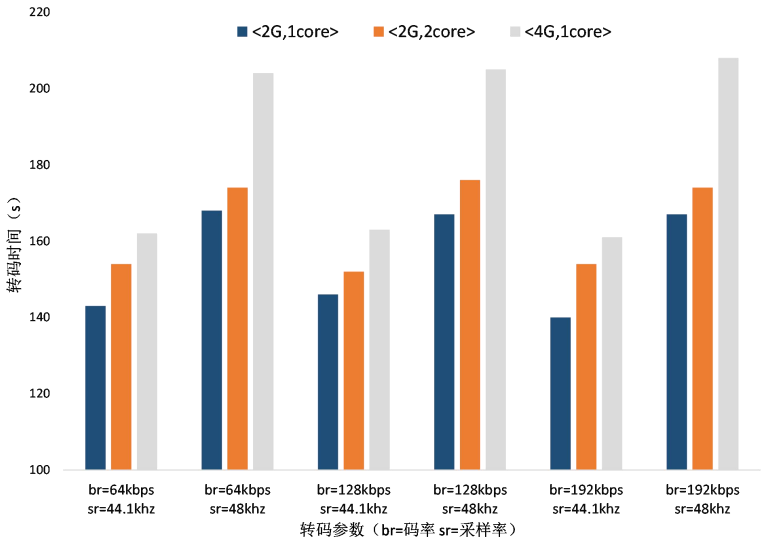

图 5.2 转码参数对 container 配置的影响

图 5.2 显示了在不同转码参数下，container 的资源配置对转码时间的影响，可以看出，无论码率和采样率怎么设置，container=<2G，1core>时，所有情况下的转码时间均最短。排除了视频参数对资源配置的影响，所以从 container 的资源

配置角度考虑系统的优化是可行的。

图 5.3 反映了本地转码、Hadoop 系统转码和 Hadoop 优化后系统转码的时间对比，可以看出优化后的系统转码时间大大缩短，而且随着转码任务量的增大，转码的优化程度逐渐提升，比未优化前提升了平均 24.5%的性能。

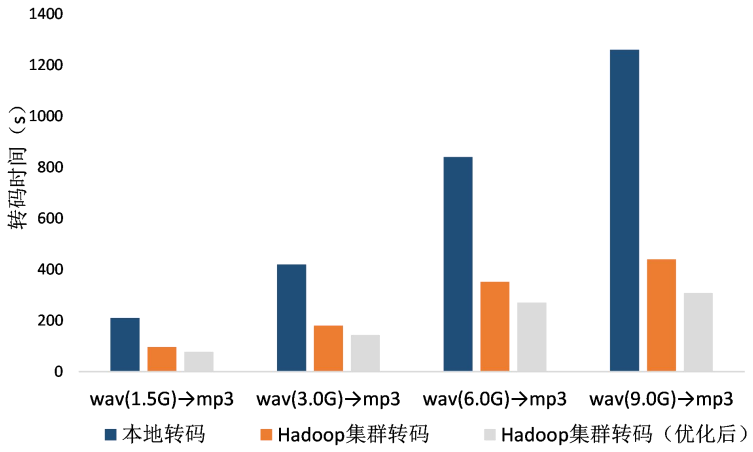

图 5.3 各转码方式下的转码时间对比

图 5.4 和图 5.5 显示了 Hadoop 转码集群优化前后的资源利用率情况的对比。优化后的内存负载比未优化前有所减轻，值得注意的是，优化后的 CPU 平均利用率约为 50%，对比未优化前的 39%，有所提升。基于样本资源需求的容器配置方法不仅优化了 Hadoop 转码集群的任务完成时间，还在一定程度上提升了系统资源的利用率。

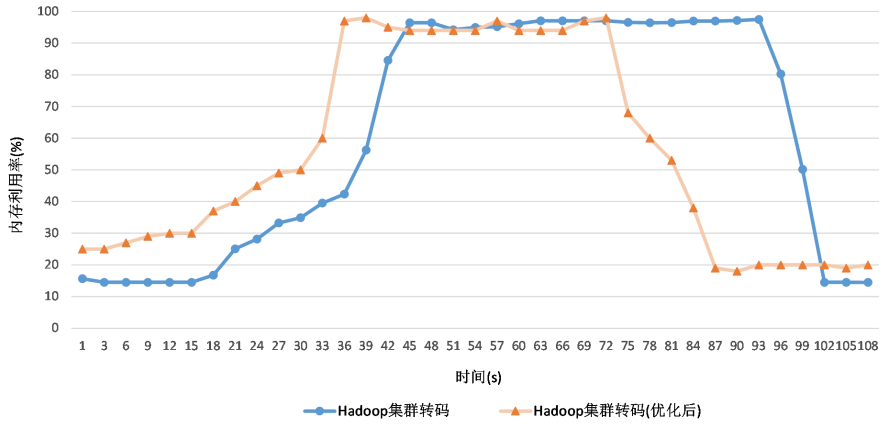

图 5.4 Hadoop 转码系统优化前后的内存利用率对比

本文在集群 A 和集群 B 上对基于样本资源需求的容器配置方法进行了验证实验，实验结果如图 5.6 所示，集群 A 中 container=<2G， 1core>时转码时间最短，而在集群 B 中 container=<1G， 1core>时转码时间最短，这与本文提出的基于样本资源需求的容器配置算法计算出来的容器资源配置的参数值一致，验证了方法

的正确性。

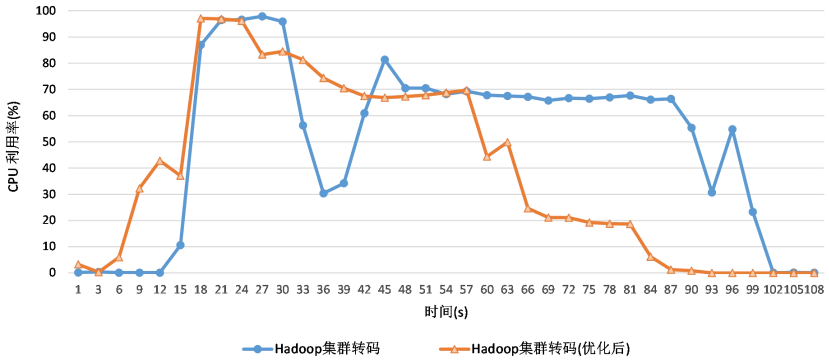

图 5.5 Hadoop 转码系统优化前后的 CPU 利用率对比

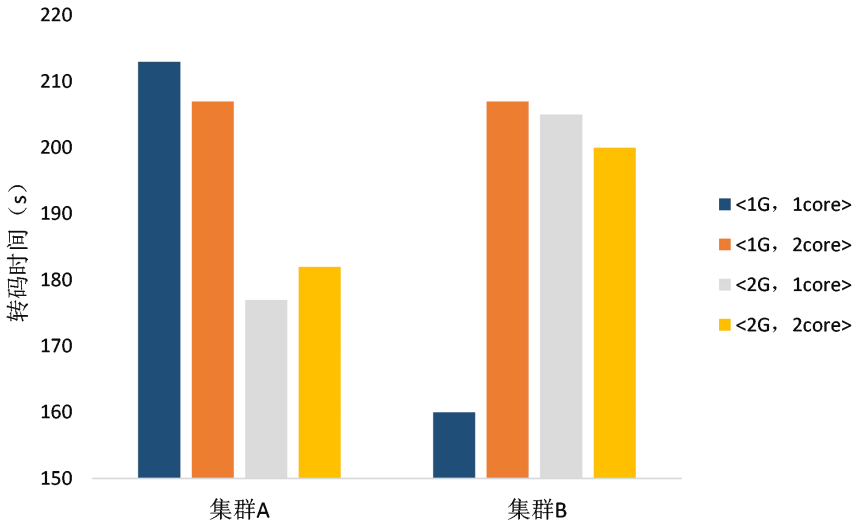

图 5.6 各容器配置下测试集群的转码时间对比

5.4 本章小结

本章详细介绍了基于样本资源需求的容器资源配置方法的过程，针对 Hadoop 转码系统的性能缺陷，提出了综合考虑内存和 CPU 两类资源的基于样本资源需求的容器配置方法，并对该方法进行了物理实验的测试和性能评价。最后验证了容器资源配置方法的正确性，以及在虚拟集群和物理集群的普适性。

结　论

随着物联网的不断发展，它产生的数据量也急剧增长，网络流媒体传输作为物联网行业的重要应用之一，其过程中产生的数据量是非常可观的。流媒体因为其特殊的非结构化特性以及其多样化的格式，使得传统的存储方式无法满足其存储需求。同时，为了满足多终端对媒体格式的多样化需求，以及网络异构性带来的媒体质量的多样化需求，需要对海量的视频进行转码。转码不仅是数据密集型作业，还是计算密集型作业，资源消耗量大，耗时长。在视频监控等实际应用中，需要将历史数据和当前数据进行对比分析，获取有用信息，这就需要寻求一种方法可以有效地存储和管理这些数据，实现存储和分析的一体化。分布式的存储与计算方式从技术上克服了这个难题。Hadoop作为开源的分布式大数据存储与分析的工具，凭借其广泛的实用性、良好的易用性、系统的稳定性以及自身丰富的生态圈，得到了学术界广泛的关注和研究。本文通过深入研究Hadoop框架和流媒体视频数据的特点，实现了Hadoop平台下海量音视频数据的转码，通过MapReduce对转码过程进行并行加速，结合container资源的分配和调度方式，提高资源的利用率，优化转码时间。

1. 本文工作

Hadoop平台下的音视频转码系统将海量音视频文件分块存储在HDFS分布式文件系统中，利用MapReduce并行计算框架对转码过程进行并行加速，修改MapReduce文件接口使其更好地适用于视频文件，结合container资源的分配和调度方式，提高资源的利用率，优化转码时间。主要工作及创新点如下：

提出了一种Hadoop平台下基于位置信息的视频分割方法，目前多数研究都是借助开源工具进行音视频的预分割，需要在本地先分段再上传HDFS中，增加了额外的文件读写开销。而且视频分割处理一般是固定在一个节点上完成的，当视频处理服务请求量过大时，该节点会成为系统的一个瓶颈。本文通过修改MapReduce的getsplit函数中的FileSplit方法——makesplit分片方式，添加分割视频转码输出后的顺序以及转码开始判断参数的传递，由于makesplit函数中startpos、len、splitHosts参数以及固定，所以将视频顺序参数和判断参数添加到文件路径Path中，然后再从Path中解析。本方法通过在分割点位置前多读数据包的方式，MPG、TS、WMV和AVI四种视频格式是在分割点位置前多读取一秒AVPacket数据包，MKV格式而是在分割点位置前多读取若干个cluster，避免了分割恰好发生在关键帧处导致转码后视频跳帧的情况。本方法实现了在Hadoop上MKV、MPG、TS、WMV和AVI格式视频的分割，并将其与Super Video Splitter分割器的结果作

对比，验证了方法的优越性，本方法避免了系统的性能瓶颈，节约了整体的转码时间。

提出了一种基于样本资源需求的容器资源配置方法，实验发现，当转码任务量大的时候，虚拟化进群会出现严重的内存过载现象，严重影响了系统的转码时间。通过改变容器分配的内存量和CPU核数，从而调整每个DataNode上并发的进程数，可以提高集群的整体性能以及资源利用率。本文提出的方法解决了在集群变更、集群资源量变化的情况下，以最少的消耗找到container最优配置的问题。本方法通过在样本集群中计算出内存配置因子和CPU核配置因子，当集群资源量变更后，只需在新集群上对一个split音视频分片进行转码，分析转码单个split分片任务所用的内存和CPU资源使用情况，利用基于样本资源需求的容器资源配置算法计算出最优的资源并发度，根据并发度即可得到container的最佳配置。本文设计了验证实验，在两个类型差异较大的集群上对本文提出的算法进行实证，验证了方法的正确性。此外，本方法不仅可以应用于虚拟化集群，在物理机集群中也同样适用。实验结果表明，相对于直接用Hadoop系统进行转码，通过优化container配置后的Hadoop转码系统在转码时间优化了25%左右，内存过载的时间缩短了25%，CPU的资源利用率从39%提高到50%。

2. 下一步工作展望

Hadoop集群并行转码作为一种较为理想的转码加速方法，目前已经取得了一些研究成果，但是相对于转码的性能和效果而言，依然显得不足，所提出的算法或多或少都存在一定的缺陷，若想实施与商业应用，还存在许多的挑战和问题。就此，笔者针对本文中的不足以及下一步的工作展望说明如下：

本文测试环境中的数据量未达到海量级别，只停留于GB级别，当执行TB、PB或者更高级别的数据操作时，系统能否依旧稳定运行，这还有待进一步测试。Hadoop提供了灰名单和黑名单的容错机制，保障了在节点失效或任务失败造成的异常得以及时处理，转码任务量过大时，系统能否保持其稳定性，也需进一步测试。下一步系统测试工作包括了TB级、PB级数据的上传和下载，以及转码的性能、异常处理的测试。

本文提出的基于样本资源需求的容器资源配置方法中，验证实验平台有限，只在两个集群中进行了实验。算法中考虑的参数包含了内存、CPU核以及主频，但是其他类型的资源，比如网络带宽和磁盘IO等，也会对系统的性能造成影响，可以在后续的研究中加入到基于样本资源需求的容器资源配置算法的资源矩阵内。

随着云计算技术的进一步发展，对于研究和应用不断深入和扩展，Hadoop转码技术的应用将不仅仅局限于视频播放、侦查安全等领域。将帮助人们解决更多的音视频在商业领域中的问题，云转码技术也将吸引更多学者的关注。

参考文献

[1] 曹洋，王建平．物联网架构及其产业链研究．技术经济与管理研究，2013，(2)：98-101

[2] Correcting the IoT History．http://www.chetansharma.com/，2017

[3] Xia F，Yang L T，Wang L，et al．Internet of things．International Journal of Communication Systems，2012，25(9)：1101

[4] 李立仁，李少军，刘忠领．智能视频监控技术综述．中国安防， 2009 (10)：90-95

[5] Conklin G J，Greenbaum G S，Lillevold K O，et al．Video coding for streaming media delivery on the Internet．IEEE Transactions on Circuits and Systems for Video Technology，2001，11(3)：269-281

[6] Ghemawat S，Gobioff H，Leung S T．The Google file system．ACM SIGOPS operating systems review，2003，37(5)：29-43

[7] The Apache Foundation．Apache Hadoop．http://hadoop.apache.org/，2017-03-28

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

[9] Huang L，Chen H，Hu T．Research on Hadoop Cloud Computing Model and its Applications．In：proceedings of 2012 Third International Conference on Networking and Distributed Computing (ICNDC’12)．Hangzhou，China: IEEE Computer Society，2012，59-63

[10] Armbrust M，Fox A，Griffith R，et al．A view of cloud computing．Communications of the ACM，2010，53(4)：50-58

[11] Schmidt E．网络就是计算机[J]．今日电子，1995，1：041

[12] Madduri R K，Dave P，Sulakhe D，et al．Experiences in building a next-generation sequencing analysis service using galaxy，globus online and Amazon web service．In：Proceedings of the Conference on Extreme Science and Engineering Discovery Environment：Gateway to Discovery．ACM，2013，34

[13] Churchland P S，Sejnowski T J．The computational brain．MIT press，2016，40-42

[14] 董西成．Hadoop 技术内幕：深入理解 MapReduce 架构设计与实现原理．第 2 版．机械工业出版社，2013，34-36

[15] A Ghodsi，M Zaharia，B Hindman，et al．Dominant resource fairness: fair

allocation of multiple resource types in USENIX NSDI，2011，11(2011)：24-25

[16] Tomar S．Converting video formats with FFmpeg．Linux Journal，2006，146(2006)：10

[17] Li Z，Huang Y，Liu G，et al．Cloud transcoder：Bridging the format and resolution gap between internet videos and mobile devices．In：Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video．New York，USA：ACM，2012，33-38

[18] Kim M，Cui Y，Han S，et al．Towards Efficient Design and Implementation of a Hadoop-based Distributed Video Transcoding System in Cloud Computing Environment．International Journal of Multimedia & Ubiquitous Engineering，2013，8(2)：214-224

[19] Wang F，Liu J，Chen M．CALMS: Cloud-assisted live media streaming for globalized demands with time/region diversities．In：INFOCOMIEEE．Orlando,USA：IEEE，2012，199-207

[20] Garcia A, Kalva H, Furht B. A Study of Transcoding on Cloud Environments for Video Content Delivery. In: proceedings of the 2010 ACM multimedia workshop on Mobile cloud media computing. New York, USA: ACM Press, 2010. 13~18

[21] Jokhio F，Deneke T，Lafond S，et al．Bit rate reduction video transcoding with distributed computing．In：Parallel，Distributed and Network-Based Processing (PDP)．Garching，Germany：IEEE，2012，206-212

[22] Tian C，Zhou H，He Y，et al．A Dynamic Map Reduce Scheduler for Heterogeneous Workloads．In：Proceedings of the 2009 Eighth International Conference on Grid and Cooperative Computing (GCC'09)．Washington，DC，USA：IEEE Computer Society，2009，218-224

[23] Jeon M，Kim N，Lee B D．MapReduce-Based Distributed Video Encoding Using Content-Aware Video Segmentation and Scheduling．IEEE Access，2016，4：6802-6815

[24] Sambe Y，Watanabe S，Yu D，et al．Distributed video transcoding and its application to grid delivery．In：Communications，2003．APCC 2003．The 9th Asia-Pacific Conference on．Penang，Malaysia：IEEE，2003，98-102

[25] 阿里巴巴．阿里云转码．https://www.aliyun.com/product/mts/．2017-02-25

[26] 张浩．MapReduce 编程模型在云海量视频转码中的研究：[成都理工大学硕士学位论文]．成都：成都理工大学，2012，55-60

[27] 李晓波．基于 Hadoop 的海量视频数据存储及转码系统的研究与设计：[浙江

工业大学硕士学位论文]. 杭州：浙江工业大学，2013

[28] Song C， Shen W， Sun L， et al． Distributed video transcoding based on MapReduce．In：Computer and Information Science (ICIS)．Taiyuan，China：IEEE，2014，309-314

[29] 郭奕希．基于 Hadoop 的视频转码系统设计与实现：[华中科技大学硕士学位论文]．武汉：华中科技大学，2011

[30] 陈珍．基于 MapReduce 的海量视频转码系统优化机制：[华中科技大学硕士学位论文]．武汉：华中科技大学，2013

[31] Wei L， Cai J， Foh C H， et al． QoS-Aware Resource Allocation for Video Transcoding in Clouds．IEEE Transactions on Circuits and Systems for Video Technology，2017，27(1)：49-61

[32] 李建江，崔健，王聃，等．MapReduce 并行编程模型研究综述．电子学报，2011，39(11)：2635-2642

[33] Yang S， Chen Y， Hsieh Y． Design Dynamic Data Allocation Scheduler to Improve MapReduce Performance in Heterogeneous Clouds． Cell biology international，2006，30(9)：681-687

[34] M Zaharia，A Konwinski，A D Joseph，R H Katz，and I Stoica．Improving MapReduce performance in heterogeneous environments．OSDI，2008(8):7-16

[35] T Sandholm，K Lai，Dynamic proportional share scheduling in hadoop．Job scheduling strategies for parallel processing，2010：110–131

[36] K Kc，K Anyanwu．Scheduling hadoop jobs to meet deadlines．In：Cloud Computing Technology and Science (CloudCom)．Indianapolis，USA：IEEE，2010，388–392

[37] A Verma， Ludmila Cherkasova， R H Campbel． Aria: Automatic resource inference and allocation for mapreduce environments．In：ICAC’11．Karlsruhe，Germany：2011：235–244

[38] J Polo， D Carrera，Y Becerra，et al．Performance-driven task co-scheduling for mapreduce environments．In：Network Operations and Management Symposium (NOMS)．Osaka，Japan：IEEE，2010，373–380

[39] Guo Z，Fox G，Zhou M．Investigation of Data Locality in MapReduce．In：proceedings of 2012 12th IEEE/ACM International Symposium on Cluster，Cloud and Grid Computing (CCGrid’12)． Ottawa， Canada: IEEE Computer Society，2012．419-426

[40] M Zaharia， S Shenker． Delay scheduling : A simple technique for achieving locality and fairness in cluster scheduling．In: Cluster Computing．Paris，France：

2010．265-278

[41] He，Y Lu，D Swanson．Matchmaking：A new mapreduce scheduling technique．In：Cloud Computing Technology and Science (CloudCom)，2011 IEEE Third Interational Conference on．Athens，Greece：2011．40-47

[42] J Tan，S Meng，X Meng et al．Improving reducetask data locality for sequential mapreduce jobs．In：INFOCOM．Turin，Italy：2013．1627-1635

[43] Kapil B S，Kamath S S．Resource aware scheduling in Hadoop for heterogeneous workloads based on load estimationComputing．In：Communications and Networking Technologies (ICCCNT)．Tiruchengode，India：IEEE，2013．1-5

[44] Divya M，Annappa B．Workload characteristics and resource aware Hadoop scheduler．In：Recent Trends in Information Systems (ReTIS)，2015 IEEE 2nd International Conference on．Kolkata，India：IEEE，2015．163-168

[45] Elkholy A M，Sallam E A H．Self adaptive Hadoop scheduler for heterogeneous resources．In：Computer Engineering & Systems (ICCES)．Cairo，Egypt：IEEE，2014．427-432

[46] Huang C C，Chen J J，Tsai Y H．A Dynamic and Complexity Aware cloud scheduling algorithm for video transcoding．In：Multimedia & Expo Workshops (ICMEW)，2016 IEEE International Conference on．Seattle，USA：IEEE，2016．1-6

[47] Kc K，Freeh V W．Dynamically controlling node-level parallelism in Hadoop．In：Cloud Computing (CLOUD)，2015 IEEE 8th International Conference on．New York，USA：IEEE，2015．09-316

[48] Su H，Zhu J，Mortazavi M，et al．A zero-penalty container-based execution infrastructure for hadoop framework．In：Computer and Information Technology；Ubiquitous Computing and Communications；Dependable，Autonomic and Secure Computing；Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM)，2015 IEEE International Conference on．Livepool，UK：IEEE，2015．640-647

[49] Lee S K，Bae M H，Eum J H，et al．Efficient vCore Based Container Deployment Algorithm for Improving Heterogeneous Hadoop YARN Performance．In：International Conference on Information Science and Applications．Singapore：Springer，2017．191-201

[50] 李媛祯．Hadoop YARN 资源分配与调度的研究：[南京航空航天大学硕士学位论文]．南京：南京航空航天大学，2015

[51] 曾婉琳，陈兴蜀，罗永刚．Hadoop 节点资源参数优化策略．计算机工程，

2016，42(1)：1-6

[52] Ding X，Liu Y，Qian D．Jellyfish: Online performance tuning with adaptive configuration and elastic container in hadoop yarn．In：Parallel and Distributed Systems (ICPADS)，2015 IEEE 21st International Conference on．Melbourne，Australia：IEEE，2015．831-836

[53] 赵颖．Hadoop 环境下的动态资源管理研究与实现：[上海交通大学硕士学位论文]．上海：上海交通大学，2015

附录 A 攻读硕士学位期间发表的学术论文

[1] Jing Yang, Renfa Li. A container resource configuration method in Hadoop Transcoding cluster based on requirements of a sample split. 2017 the 2nd IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA 2017)，2017.

附录 B 攻读硕士学位期间所参与的科研

[1] 新一代汽车嵌入式系统功能安全的建模与算法研究. 国家自然科学基金(61672217)

致　谢

不知不觉间，我已在湖南大学学习、生活了七个春秋。这期间的宝贵经历和收获，使我从懵懂少年蜕变成了一位有知识有理想的青年。值此论文完成之际，谨向在我多年求学生涯中所有帮助和关心我的老师、同学、朋友和家人致以诚挚的谢意。

首先衷心感谢我的导师李仁发教授。在研究生阶段的学习和课题研究中，李老师给予了我非常多的指导、建议和帮助。李老师的工作态度和钻研精神令人敬佩，无论寒暑假或周末，在基地实验室总能看到他的身影。李老师严谨的学术态度、渊博的学识、敏锐的专业洞察力、大气豁达的个人气质一直以来都深深感染和影响着我们，是我们永远的榜样。

衷心感谢李蕊老师为我们提供了在湖南双菱公司实习的机会，使我在开发领域得到了提高，将所学的理论知识应用于实际，感谢李老师在实习期间对我的启发和教诲，让我在研究上少走弯路，磨炼实践能力的同时也为我的论文工作提供了雏形。

衷心感谢实验室的白洋师姐、吴武飞师兄、周兰花师姐对我学习上的建议和不断的鼓励。

感谢实验室的朱立民、李万里、查里佳等同学，三年的学习时光里，大家团结互助、相互鼓励。感谢实习期间的项目组小伙伴，感谢秦凯强同学在项目走入僵局时的坚持，感谢刘四平同学孜孜不倦地为我解答疑惑，感谢大家的陪伴，让我在不断遇到困难的时候，仍然一直坚持下去。感谢何佩、任山和龙艳芳同学，本科到研究生七年时间，我们不仅是同学，还是一直朝夕相处的室友、好朋友，这份珍贵的友情我会永远珍惜。

感谢时刻关心我的父母和家人，他们是我永远的坚强后盾。感谢我的男友刘凯伦在我完成论文的过程中，给予了始终如一的支持、信任、安慰和鼓励。

最后，感谢审阅此论文及出席论文答辩会的老师们和专家们，感谢您们在百忙之中抽出时间对我的论文进行指导和帮助！

杨竞

2017 年 5 月 9 日