学校代号 10532　　学　　号 S151000888

分 类 号 TP37　　密　　级 普通

湖南大学
HUNAN UNIVERSITY

硕士学位论文

异构分布式集群的视频转码与优化

学位申请人姓名 邓湘军

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 2018 年 5 月 8 日

学校代号：10532

学　　号：S151000888

密　　级：普通

湖南大学硕士学位论文

异构分布式集群的视频转码与优化

学位申请人姓名：　邓湘军

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：　2018 年 5 月 8 日

论 文 答 辩日 期：　2018 年 5 月 20 日

答辩委员会主席：　赵欢 教授

Video Transcoding and Optimization of Heterogeneous Distributed Clusters

by

DENG Xiangjun

B.E. (Changsha University of Science & Technology) 2015

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 Li Renfa

March, 2018

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

视频正不断以更多的表示格式，更多的设备类型和多种多样的网络环境进行制作和使用。视频转码是将一种视频编码格式转换为另一种视频编码格式的过程。然而，大多数时候，转码是一个计算密集型的过程。人们常常利用分布式计算技术来高效利用多机器，多核 CPU 和分布式计算资源在特定设施，家庭或专用分布式基础架构中可用的计算资源来处理复杂任务。在分布式集群中，视频被分割成多个分片，在多个机器上实现并行转码。Hadoop 是分布式计算技术中的一个非常流行的编程框架，为分布式系统提供了高可扩展性，满足了视频转码的高可扩展性需求。

在分布式集群中，机器之间的计算能力不一定会相同，计算能力的异构性是分布式系统中非常普遍的特性。本文从任务调度和系统架构的角度研究了异构分布式集群的视频转码加速问题。主要的工作和创新点如下：

（1）异构集群的负载不均衡使得集群的计算资源利用不合理，导致视频转码作业的执行时间远高于理想值。因此，一个负载均衡的任务调度算法能够实现分布式视频转码加速。Max-MCT 和 MLFT 任务调度算法的模型没有考虑视频分片的传输开销，使得任务调度模型不够精确，PLTS 算法考虑了分片传输开销但是没有平衡分片转码时间和传输时间，使得视频转码作业的预期完成时间仍然有优化的空间。为了有效利用集群的异构计算资源，本文从任务调度的角度出发，构建了 Hadoop 视频转码任务调度模型，为了优化这个模型，本文把它转化成了求解 NP 难问题，提出了一个本地感知的启发式算法 LA-MCT，这个算法的主要思想是平衡视频的分片转码时间和分片在集群内的传输时间，大量的仿真实验表明，本文的算法比现存的启发式任务调度算法如 Max-MCT、MLFT 和 PLTS 算法有更短的作业预期完成时间。

（2）为了加速整个转码系统的运行过程，从视频转码架构的角度出发，本文抽象了整个系统的工作流程，并使用 Alluxio 分布式内存文件系统代替现存的 HDFS 文件系统实现视频分片在系统内的缓存与共享，减少了视频分片在视频转码系统内的磁盘读写开销。本文搭建了一个小型 Hadoop 异构集群，设计并实现了一个视频转码系统，系统使用不同大小的视频数据，多次实验证明了基于 Alluxio 的 Hadoop 异构集群的性能优于现存的基于 HDFS 的 Hadoop 异构集群，视频转码速度提高了 5%以上。

关键词：视频转码；Hadoop；MapReduce；任务调度； HDFS； Alluxio

Abstract

Video is constantly being produced and used in more presentation formats, device types, and a variety of networks environment. Video transcoding is the process of converting a video encoding format to another video encoding format. However, most of the time, transcoding is a computationally intensive process. Therefore, people use distributed computing technology to efficiently use the available computing resources in multi-machine, multi-core CPU and distributed computing resources in a specific facility, home or private distributed infrastructure to handle complex task. In a distributed cluster, video is splitted into multiple segments and parallel transcoding is implemented on multiple machines. Hadoop is a very popular programming framework in distributed computing technology. It provides high scalability for distributed systems and meets the high scalability requirements of video transcoding.

In a distributed cluster, the computing power between machines is not necessarily the same, and the heterogeneity of computing power is a very common phenomenon in distributed systems. In this paper, we study the video transcoding acceleration of heterogeneous distributed clusters from the perspective of task scheduling and system architecture. The main works and innovations are as follows:

(1) The unbalanced load of heterogeneous clusters makes the cluster's use of computing resources unreasonable. As a result, the entire finish time of video transcoding job is much higher than the ideal value. Therefore, a load balancing task scheduling algorithm can achieve distributed video transcoding acceleration. The model of Max-MCT and MLFT task scheduling algorithms does not consider the transmission overhead of video segment, resulting in the task scheduling model inaccurate. The PLTS algorithm considers the segment transmission overhead but does not balance the segment transcoding time and transmission time. Therefore, the expected completion time of the job still has room for optimization. From the point of view of MapReduce task scheduling, in order to effectively use the heterogeneous computing resources of the cluster, in this paper, we first

abstract the Hadoop video transcoding task scheduling model. In order to optimize this model, we turn it into a solution to the NP-hard problem and propos a heuristic algorithm called LA-MCT, the main idea of this algorithm is to balance the video segment transcoding time and the segment transmission time in the cluster. A lot of simulation experiments show that our algorithm has shorter job finish time than the existing heuristic task scheduling algorithms such as Max-MCT, MLFT, and PLTS algorithms have shorter job finish time.

(2) From the video transcoding architecture point of view, in order to speed up the running process of the entire transcoding system, we abstract the entire system's operation flow. Alluxio distributed memory file system is used instead of the existing HDFS file system to implement video buffering and sharing within the system, which reduces the disk read and write overhead of video fragmentation in the video transcoding system. We have built a small Hadoop heterogeneous cluster, designed and implemented a video transcoding system, the system uses different sizes of video data, and many experiments have proved that the Hadoop heterogeneous cluster based on Alluxio outperforms the existing HDFS-based Hadoop. The speed of a video transcoding job increased by about 5%.

Key Words: Video transcoding; Hadoop; MapReduce; Task scheduling; HDFS; Alluxio

目 录

插图索引

附表索引

第1章 绪论

1.1 选题背景及意义

随着移动互联网的蓬勃发展，个人电脑、智能手机和平板电脑普及到人们的生活的方方面面。根据中国互联网络信息中心（CNNIC）在京发布第 41 次《中国互联网络发展状况统计报告》，截至 2017 年 12 月，我国网民规模达 7.72 亿，普及率达到 55.8%，超过全球平均水平（51.7%）4.1 个百分点，超过亚洲平均水平（46.7%）9.1 个百分点。网络视频用户规模达 5.79 亿，较 2016 年年底增加 3437 万，占网民总体的 75.0%。手机网络视频用户规模达到 5.49 亿，较 2016 年年底增加 4870 万，占手机网民的 72.9%[1]，如图 1.1 所示。因此，网络视频无时无刻不在人们的生活和娱乐中发挥着重要作用。

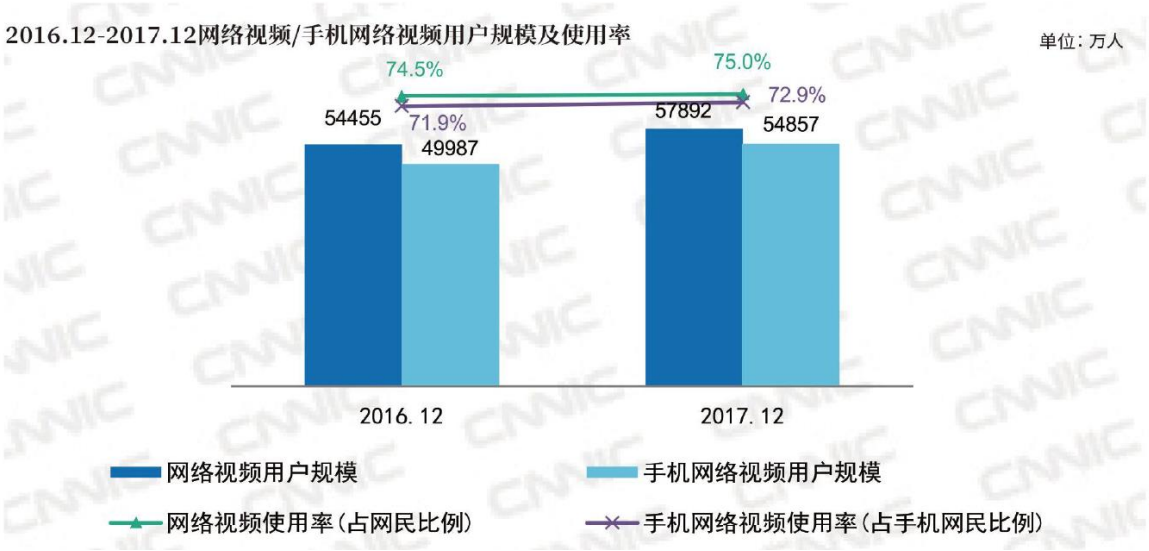


图 1. 1 2016.12-2017.12 网络视频/手机网络视频用户规模及使用率

移动终端的分辨率、屏幕尺寸是多种多样的。就拿手机屏幕的分辨率来说，根据友盟公司的 2017 年 1月手机分析报告，在 2017 年 1 月，Android 设备的分辨率主要包括以下 8 种：1280*720、1920*1080、854*480、960*540、1812*1080、1184*720、1776*1080 和 800*480[2]。一方面，不同的分辨率、屏幕尺寸、处理器和 GPU 的计算能力对于视频的需求是不一样的；另一方面，对于同一个移动终端，不同的环境下的不同网络带宽对视频的需求也不同。由于移动终端、和网络的异构性使得一个视频需要适配多种不同的环境。为了解决这个问题，视频提供方需要为用户提供不同的视频编码标准的视频。比如说，当用户的手机屏幕分辨率是昂贵的 1920*1080 的屏幕时，提供方给用户传输的是分

辨率较高的视频，而当用户的屏幕时廉价的 800*480 屏幕时，视频提供方只需要为终端提供低分辨率的视频即可，因为传输高分辨率的视频将会耗费更多流量，这对低分辨率的手机来说是没有必要的。

一个视频转化为另一个不同视频编码标准的视频的过程叫做视频转码（Video Transcoding），视频转码将已经压缩编码的视频码流转换成另一个视频码流，以适应不同的用户需求、不同的终端处理能力以及不同的网络带宽。本质上来说，转码是一个先解码，再编码的过程，因此转换前后的码流可能遵循相同的视频编码标准，也可能不遵循一样的视频编码标准。视频转码主要发生在以下几个场景：（1）不同视频格式间的转换，例如从 MPEG-2 或者 MPEG-4 转到 H.264；（2）内容传输，改变比特率满足不同网络带宽或者设备播放速度的需求；（3）清晰度，将高清视频转为标清甚至更低的清晰度，后者反向处理。

视频转码是一个计算集中型的任务，尤其是，小型设备（如 NAS 盒，路由器和移动终端）无法独自进行转码以满足按需的流媒体要求并提供可接受的用户体验。表 1.1 显示了几种常用的处理器的转码能力度量，它显示了给定示例视频的每秒帧数不同处理器实现的转码速度[3]，转码任务将采样视频分辨率从 4CIF 分别更改为 HD720 和 HD1080，该表清楚地表明，即使对于在单个转码进程上工作的相对高端处理器，转码也是一个计算密集型过程。而且处理转码任务的设备往往不止同时在处理一个任务，这将使得用户体验更差。

表 1. 1 不同处理器（设备）的视频分辨率转码能力测量

处理器		转码能力	
名称	规格	4cif - hd720	4cif - hd1080
Intel core 2 Duo E6550	2.33GHz 4GB RAM 4094Kb cache Desktop	26 fps	14 fps
Intel core 2 Duo E6570	2.10Ghz 4GB RAM 2048Kb cache Laptop	16 fps	10 fps
ARM cortex	250MHz 512MB Mobile	6 fps	2 fps

衡量视频转码系统的质量的最重要的指标是转码后的视频的图像质量与转码速度，本文的优化目标主要是视频转码加速。为了加快视频转码速度，改善用户体验，许多科研工作者做了大量的研究，通常来说，视频转码加速有以下三种改

善渠道：（1）转码算法加速；（2）GPU 硬件加速；（3）分布式并行加速。使用转码算法进行视频转码加速有可能带来图像质量失真的问题。使用 GPU 进行加速的视频转码系统的可扩展性不高，随着计算资源需求增大，很难有明显的视频转码加速效果。

随着云计算技术的兴起与流行，越来越多的应用运行在云集群上。在基于云视频转码系统[4-7]中，主节点将输入视频分割成小视频，然后将这些小视频分发到从节点上，各从节点独立地将这些小视频进行转码操作，然后再将转码成功的文件传输到主节点中，主节点将接收到的文件合并成一个完整的视频。分布式系统将会带来资源管理、作业调度和作业容错等问题。

MapReduce[8]是一种高可扩展性的分布式计算编程范式，可以同时使用数千台计算机并行地处理复杂的作业，因此是并行转码加速的有效解决方案。MapReduce 的开源框架 Hadoop[9]封装了分布式应用的实现细节，抽离出了应用的业务逻辑，为它们提供了简单的接口。使程序员只需要关注简单的业务逻辑，大大提高了生产力。由于视频可以细分成更小的原子单元，使得 Hadoop 平台的视频转码能够简单快速实现，并且视频转码系统能够有效利用 Hadoop 系统可扩展性高、资源管理方便、作业调度实现简单以及高容错性等框架本身的有点。

通常来说，现代云服务提供商通常由数千台性能不同的服务器组成，因此 Hadoop 集群往往是异构的。即使集群的异构性也可能不是一开始就设计好的，然而，随着时间的推移，集群中新加入的机器通常会与之前的机器硬件不同，导致计算能力的不同。而且，Hadoop 框架是为同构集群所设计的，如果在异构集群上不对视频转码作业进行优化，作业的执行往往会低于理想值，因此，异构集群非常普遍，对异构环境下的 Hadoop 集群的优化非常有意义。

在 Hadoop MapReduce 中，数据的共享使用的是 HDFS(Hadoop Distributed Filesystem)，HDFS 是一个通用的由廉价的本地磁盘组成的分布式文件系统。这意味着，在数据处理之前，用户需要将要处理的数据上传到分布式文件系统中，并为每个数据块提供三个副本（默认情况下）。

数据库天才，1998 年图灵奖获得者，詹姆士·格雷（James Gray）曾经说过：磁盘应该像磁带那样使用，而内存应该像磁盘那样使用。随着计算机硬件的不断发展，20 多年的断言现在已然成为现实。磁盘读写性能的发展以及远远不如内存的读写性能发展，同时，内存的制造方式的不断革新与推动，内存的制造成本越来越低，相对廉价的内存作为海量数据的存储介质成为一种可能。

谷歌公司发表大数据三篇经典论文[8-11]后，掀起了大数据狂潮，大数据生态圈的蓬勃发展，出现了分布式内存计算框架，如 Spark[12][13]、Ignite[14]和 Piccolo[15]。这些计算框架，不同于 MapReduce，将计算的中间结果存储在内存而不是本地磁盘，在很多作业中，确实减少了磁盘读写开销，加快了数据处理

速度，但是数据的存储与共享也是作业的执行瓶颈之一。基于内存的分布式文件系统的出现，如阿里巴巴公司研发的 Allxuio(原名 Tachyon)[16]，为分布式数据处理，提供了内存级别的数据共享。在 Hadoop 异构集群上进行视频转码时，一个通用的转码系统，把视频分片存储到内存分布式文件系统中，可以提高数据共享的速度。

1.2 研究问题

本论文的主要目标是研究异构分布式集群中的视频转码加速问题。视频转码的加速方法主要有三种，转码算法加速、GPU 硬件加速和分布式并行加速。转码算法的加速方法需要考虑到图像质量失真的问题，GPU 硬件加速可扩展性不高，而分布式并行加速能够使用 FFmpeg 避免图像质量的失真，Hadoop 分布式框架 Hadoop 提供了高可扩展性，使得分布式并行视频转码成为了一个有效的方法。

Hadoop 分布式并行视频转码需要把视频分割成分片，然后将分片调度到各个节点执行分片视频转码。Hadoop 框架的设计初衷是为同构分布式集群而准备的，一个负载均衡的任务调度算法能够有效利用异构计算资源，并行地加快作业处理速度。从视频转码任务调度的角度出发，在异构集群中，因为每个计算节点的性能不一样，作业中的任务的调度策略会给视频转码作业的最终完成时间带来极大的影响，如果异构集群的负载不均衡，那么不能充分利用异构集群的计算资源，使得作业的最终完成时间远远低于理想值。

分布式视频转码加速不仅仅只能考虑任务的调度，视频分片的存储与共享一样会消耗不少时间。从 Hadoop 视频转码系统架构的角度考虑，分片在分布式集群内的存储与共享需要用到 HDFS，这个 Hadoop 的基础组件将大量的廉价磁盘组织在一起，形成一个虚拟文件系统，在分片的读写过程中，需要读写磁盘，如果视频文件较大和分片数据较多，转码应用程序在执行时会有较长的 IO 读写开销。分布式内存文件系统 Alluxio 将内存组织成一个虚拟文件系统，提供了内存级的文件的共享与存储。使用基于内存的分布式文件系统，而不是基于磁盘的分布式文件系统，可以让被驱动节点分割后的视频分片的共享这项服务达到内存级别，提高视频转码的速度，从而改善系统的用户体验。本文研究 Alluxio 来替代 HDFS 作为视频分片存储和共享的介质，设计和实现一个基于 Alluxio 的 Hadoop 视频转码系统。

1.3 国内外研究现状

视频转码程序需要将输入的视频进行解码，还原出原始图像，然后根据再根据转码的参数要求重新将原始图像编码成新的图像，进而将这些新的图像组成用

户需要的视频文件。如果这样按部就班地进行转码，计算机需要很长的时间来处理。基于算法的视频转码加速方法利用输入视频的编码信息，比如残差、MPEG-4 宏块模式信息和运动矢量来减少重新编码视频的复杂度[17]。或者在转码过程中忽略部分转码步骤，再通过运动补偿技术来减少精度损失，比如基于变换域的转码，算法简单地解码来自比特流的变换系数和辅助信息，解释编码模式以便用较少的步骤适当地重新分离将在较低速率比特流中重新组合的离散余弦变换（DCT）系数[18]。或者截断比特流中的结构，减少分配给变换系数的比特数[19]。尽管这些方法加快了转码速度，但是也牺牲了图像质量。

基于 GPU 的硬件加速视频转码方法通常是 CPU 与 GPU 并存的异构环境下使用的。在这种方法中，CPU 主要负责逻辑运算，然后将 CPU 不擅长的复杂算术运算如视频转码过程中宏块的运动估计和模式选择分散在 GPU 中去并行执行[20]。虽然 GPU 相比于 CPU 的超强的计算能力确实能够大大加速转码过程，但是视频的宏块之间相互依赖，而且需要特殊的硬件去实现这种架构，使得可扩展性不高，并行加速效果并不显著。

目前存在有几种视频转码任务调度算法[21-23]，考虑到异构集群中的计算节点的异构性，F. Lao[21]将视频转码任务调度模型抽象成背包问题，并考虑了视频转码子任务的启动开销，为了解决这个任务调度问题，提出了 Max-MCT(Maximizing Minimal Complete)任务调度算法。S. Lin[22]提出了一种新的并行化视频转码框架，其中包括任务预分析，自适应阈值分割和最小完成时间（MFT）调度。基于此框架，他们提出了一种称为 MLFT（Minimum Longest queue Finish Time）的负载均衡调度算法，该算法通过将复杂任务划分为多个子任务并在分配队列中重新分配任务来缩短作业的最终完成时间。不幸的是，Max-MCT 和 MLFT 算法都没有考虑视频分片的传输时间，使得 Hadoop 视频转码的模型不够精确，从而导致集群负载不均衡。

当将视频分片的传输开销考虑到视频转码模型时，H. Zhao[23]提出了一个本地感知的任务调度算法 PLTS(Prediction-based and Locality-aware Task Scheduling)，该算法考虑了转码视频分片的本地性，并组合了 Max-Min 算法和 Min-min 算法的优点来平衡负载[24]，从而最小化作业的最终完成时间。但是该算法没有平衡视频分片的执行时间与传输时间，使得视频转码作业仍然有加速的空间。因此，在视频转码的任务调度的层面上，选择一个有效的调度策略，能够有效利用异构集群的计算资源，加速视频转码过程。

在 Hadoop 平台的角度上， Hadoop 生态系统中，最基础组件是 HDFS，它是 Hadoop 平台上的默认的文件系统，如果 Hadoop 需要处理数据，就需要将数据分块上传到 HDFS 中，当 MapReduce 作业运行时，如果从节点执行任务的数据不在本地，从节点需要将数据从存有数据分块的结点拉去数据到本地磁盘，然

后任务读入数据并处理。HDFS 作为一个文件系统，为集群中的计算结点提供数据的存储与共享。在使用 HDFS 作为文件系统的视频转码应用[24-[28]中，如果视频文件比较大，转码视频将在驱动节点被分割，然后驱动程序将分片上传到 HDFS 中，这中间设计大量的磁盘读写操作以及网络开销，这在整个视频转码作业过程中，占据了相当长的时间。

1.4 主要贡献

衡量一个视频转码系统的优化质量的一个重要指标是视频转码作业的最终完成时间，作业最终完成时间越短，系统的用户体验更高。本文以 Hadoop 异构集群环境为背景，考虑到视频流的基本特性，提出了一个本地感知的 Hadoop 视频转码任务调度算法 LA-MCT（Locality-aware Minimal Complete Time），并设计了一个视频转码系统，使用了 Alluxio 分布式文件系统减少视频数据共享时的磁盘读写开销，提高了视频转码系统的转码速度。本文的具体工作内容如下：

第一，介绍了视频流的基本特性以及转码的过程，并对比了三种不同的视频转码加速方法的相关研究，分析了他们的优点与不足。

第二，在 Hadoop 异构分布式集群环境下，使用并行转码的方法加速，由于 Hadoop 的高可扩展性，可以通过增加计算节点数量的方法扩展集群的计算能力，加快转码速度。为了有效利用集群的异构资源，本文抽象出了一个视频转码任务调度模型，它考虑了任务的启动开销、集群中计算节点的异构性和网络通信开销，这是一个车间作业调度问题（Job Shop Scheduling），这个问题被证明了是一个 NP 难问题，为了解决这个 NP 难问题，提出了一个本地感知的任务调度算法 LA-MCT（Locality-aware Minimal Complete Time），算法的核心思想主要是考虑视频分片的本地性，平衡视频的转码的执行时间与视频分片的传输时间，使得转码的最终完成时间接近于理想值。为了验证算法的效果，利用 Java 语言，使用随机生成的模拟数据，实现了提出的算法与其他几种现存的算法，包括 Max-MCT, MLFT 和 PLTS，推导出了这些算法生成的调度策略的最终完成时间，实验表明，本文提出的算法由于现存的算法，包括当前最好的算法 PLTS。

第三，设计了一个 Hadoop 异构集群上的视频转码系统，该系统主要创新点是，从视频分片共享的角度，使用 Alluxio 作为集群的分布式文件系统，减少了磁盘的 IO 开销，加速了视频转码过程。实验表明，随着被转码文件大小的增加，相比于使用 HDFS 作为分布式文件系统，视频转码速度提高了大约 5%。

1.5 章节组织

本文共由五个章节组成，全篇围绕 Hadoop 异构分布式集群上的视频转码与优化问题展开。

第一章，主要介绍 Hadoop 异构集群上的视频转码课题研究的目的和意义，并介绍了研究的主要工作是在保证视频质量的同时提高转码速度的问题，并总结了本文的主要贡献，简述了本文的主要研究工作以及文章的组织结构。

第二章，主要介绍了视频转码的底层原理、相关开源工具如 FFmpeg，并介绍了视频转码系统使用到的 Hadoop 相关的技术。另外，整理和总结了国内外对视频转码优化的研究现状，分析了他们的优点与不足。

第三章，将 Hadoop 异构集群上的视频转码的任务调度问题抽象成车间调度问题，这个模型考虑了视频分片的传输开销、任务启动开销，为了充分利用异构集群的计算资源，提出了 LA-MCT 算法，并且对 LA-MCT 算法效果的大量的推导与验证，与其他现存的算法的比较与评估。

第四章，主要介绍了本文提出的以 Alluxio 作为分布式文件系统的视频转码架构，并实现了该系统。同时用真实的测试视频进行转码实验，并与以 HDFS 作为分布式文件系统的视频转码架构作比较。

最后，对本文所做的工作进行了总结，以及对本文研究的不足的地方的改进方向。

第2章 相关研究和基础知识

2.1 引言

为了适应对视频的不同的用户需求、不同的网络带宽和不同的终端处理能力，视频转码技术随之而诞生。视频转码是一个运算集中的过程，这个过程需要对输入的视频流进行全解码、视频过滤/图像处理、并且对输出格式进行全编码。

本章详细阐述了视频转码的研究现状。由于本文的视频转码的优化重点是视频转码加速，因此，考虑到视频转码后图像质量和可扩展性，比较了各种视频转码加速渠道的优点与不足。最后，介绍了与本研究相关的实现和优化视频转码系统的主流技术与研究进展，包括 FFmpeg、Hadoop 和 Alluxio 等。

2.2 视频流结构和视频转码

目前存在不同的标准来规定压缩视频内容的比特流结构，以确保视频制作和播放设备的适当互操作性。视频比特流的具体结构对于不同的压缩技术是不同的，但是它们的一般结构都遵循着视频压缩的基本概念。图 2.1 以抽象的方式描述了 MPEG 比特流结构。从分布式转码的角度来看，这种结构有利于实现分布式并行加速，主要原因是我们能够找到一种合适的方式来在不同的处理单元之间分割比特流。为此选择合适的分片粒度是非常重要的，因为不同的粒度会导致系统延迟，影响视频转码的最终完成时间。并且影响系统实现的复杂性。利用较小粒度的分片（例如宏块）可能导致更高的通信和同步成本以及更高的实现复杂性。小粒度的视频分片中的数据依赖关系，使得系统处理一个分片时，要参考其他分片的数据，将增加系统处理一个分片的时间延迟。因此，处理视频序列需要比相同视频信息的 GOP 或帧消耗更多的时间。本文中的分布式转码器利用 GOP 作为分片的原子单元。总之，选择 GOP 作为分片的原子单元的主要原因是实现的复杂性以及在分片之间头部通信方面细粒度的副作用导致的。

视频转码器是一个软件，它解码给定的视频信号表示，并重新编码到另一个。图 2.2 显示了视频转码器的通用功能。视频转码器的任务范围可以从转换容器（即包含关于在视频文件中找到的流的元数据的头部）的视频格式这样的简单任务到用不同的编解码器（标准）。

一个视频转码器的最简单实现是“全解全编”转码器，如图 2.3 所示，这样的结构把解码器与编码器级联组合而成，它先将输入视频解码成原始的视频流数据

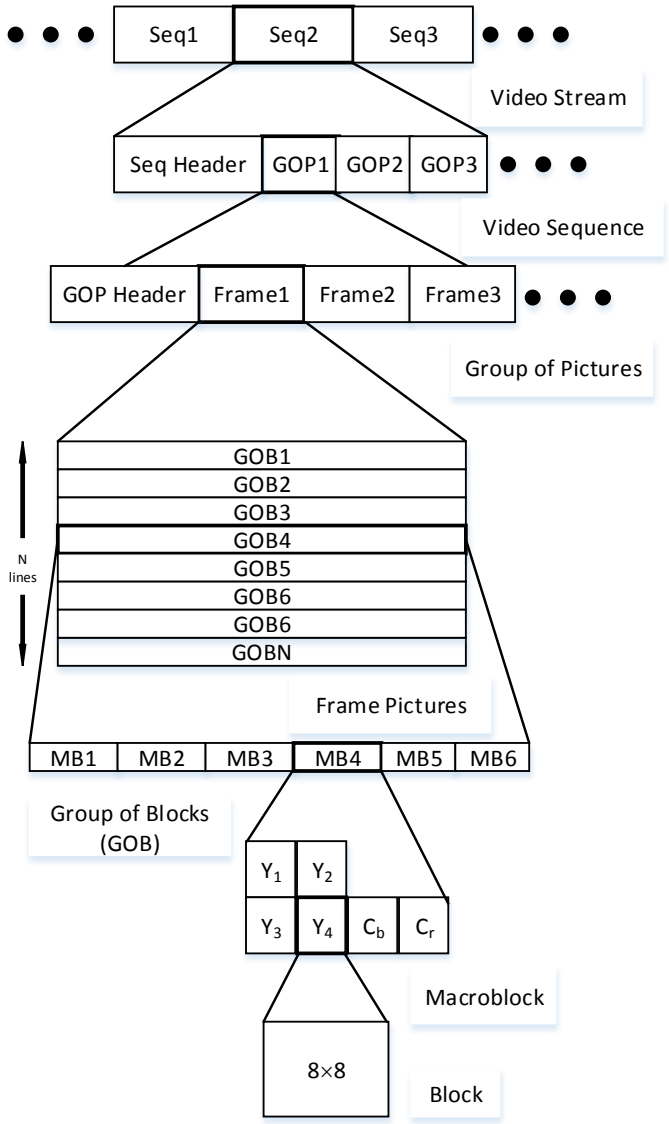


图 2. 1 MPEG 编解码器中的视频层次结构

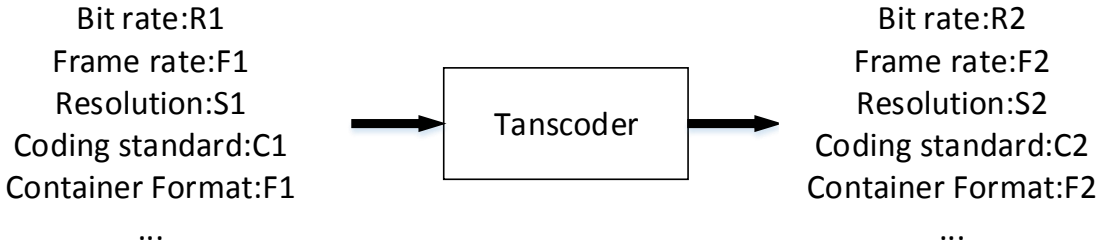


图 2. 2 一个通用的视频转码器

，再根据目标码率、帧率和空间分辨率要求进行编码。然而，如果按部就班地使用“全解全编”方法进行编解码，计算过程不仅复杂而且多余。事实上，由于视频编码需要进行大量的计算，完整的重编码过程非常耗时。另外连续编码往往导致更多的图像质量损失。因此，构成解码和重新编码的一些步骤可以通过重用在上次编码期间已经进行的编码而被省略。比如，视频转码器可以在解码过程中利用频域的压缩系数、运动矢量信息、输入视频的头信息以及宏块模式信息等，在编码后的视频质量损失不大的情况下，尽最大努力减小计算量。

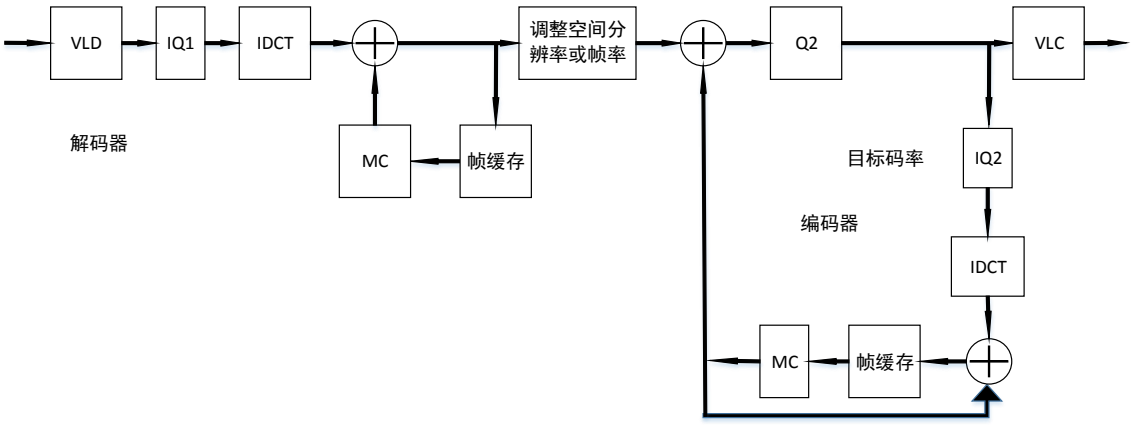


图 2. 3 全解全编转码器

2.3 FFmpeg 开源框架

FFmpeg[29]是一个完整的跨平台解决方案，用于录制，转换和流式传输音频和视频，它是处理多媒体内容（如音频，视频，字幕和相关元数据）的库和工具的集合。FFmpeg 的用户有 Google，Facebook，Youtube，优酷，爱奇艺，土豆等。它包含了一个用于多个项目中音频和视频的解码器库 libavcodec，以及一个音频与视频格式转换库 libavformat。FFmpeg 可以在 Windows、Linux 还有 Mac OS 等多种操作系统中进行安装和使用。基本上只要做视频音频开发都不能不使用 FFmpeg。FFmpeg 包含的组件如表 2.1 所示。

表 2. 1 FFmpeg 组件

组件	详情
libavformat	实现流协议，容器格式和基本的 I / O 访问。
libavcodec	提供了更广泛的编解码器的实现。
libavutil	包括哈希器，解压缩器和其他实用功能。
libswscale	实现颜色转换和比例缩放。
libpostproc	用于后期效果处理
ffmpeg	一个用于处理，转换和传输多媒体内容的命令行工具箱。
ffsever	基于 HTTP、RTSP 用于实时广播的多媒体服务器
ffplay	一个简约的多媒体播放器

2.4 Hadoop 框架

2.4.1 HDFS 分布式文件系统

HDFS 是 Google 公司的 GFS 的开源实现，它是一个易于扩展的分布式文件系统，运行在大量的普通廉价机器上，为大量用户提供稳定的文件存取服务。HDFS 架构如图 2.4 所示。它是一个 Master/Slave 架构，Master 是 Namenode，Slave 是 Datanode。

在图 2.4 中，Namenode 是一个中心服务器，它包含所有集群的文件系统元数据信息、监督健康状况的数据节点以及协调对数据的访问，是 HDFS 的中央控制器。Namenode 本身不拥有任何集群数据。这个 Namenode 只知道一个或多个数据块构成一个文件，并且这些块存储于集群中的 Datanode 中。

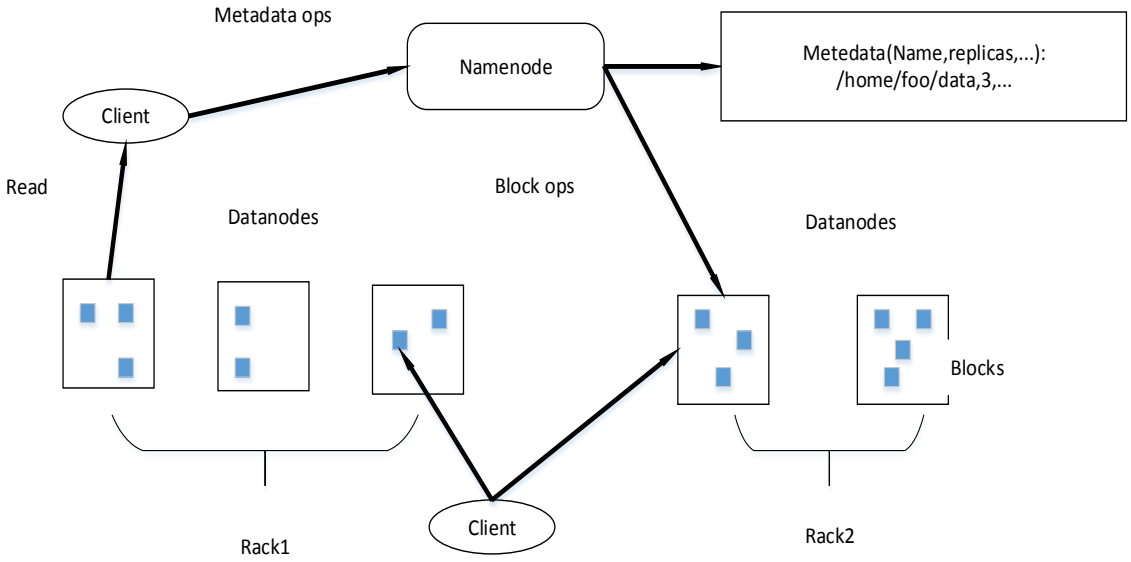


图 2. 4 HDFS 架构

Datanode 每 3 秒通过 TCP 连接向 Namenode 发送心跳信号，使用相同的端口号定义 Namenode 上的守护进程，端口号默认为 9000。每 10 个心跳信号构成一个块报告，Datanode 告知它的所有块给 Namenode。块报告允许 Namenode 构建它的元数据。

当系统进行文件操作时，Namenode 负责文件元数据的操作，Datanode 负责处理文件内容的读写请求，跟文件内容相关的数据流不经过 Namenode，Namenode 只知道文件的数据块在哪个 Datanode 中，如果经过 Namenode 会导致 Namenode 负载过高，成为文件系统的性能瓶颈。

客户端读取 HDFS 文件的流程如图 2.5 所示，详细过程如下：

（1） 客户端把要读取的文件路径发送 RPC 请求通知给 Namenode。

（2） Namenode 获取文件的元数据信息（主要是块的副本在 Datanode 上被存放的位置信息）返回给客户端。

（3） 客户端根据返回的信息找到相应 Datanode 逐个获取文件的数据块，使用 FSDataInputStream 输入流读取块数据，并在客户端本地进行数据追加与合并从而获得整个文件。

（4） 数据读取完成后，关闭输入流。

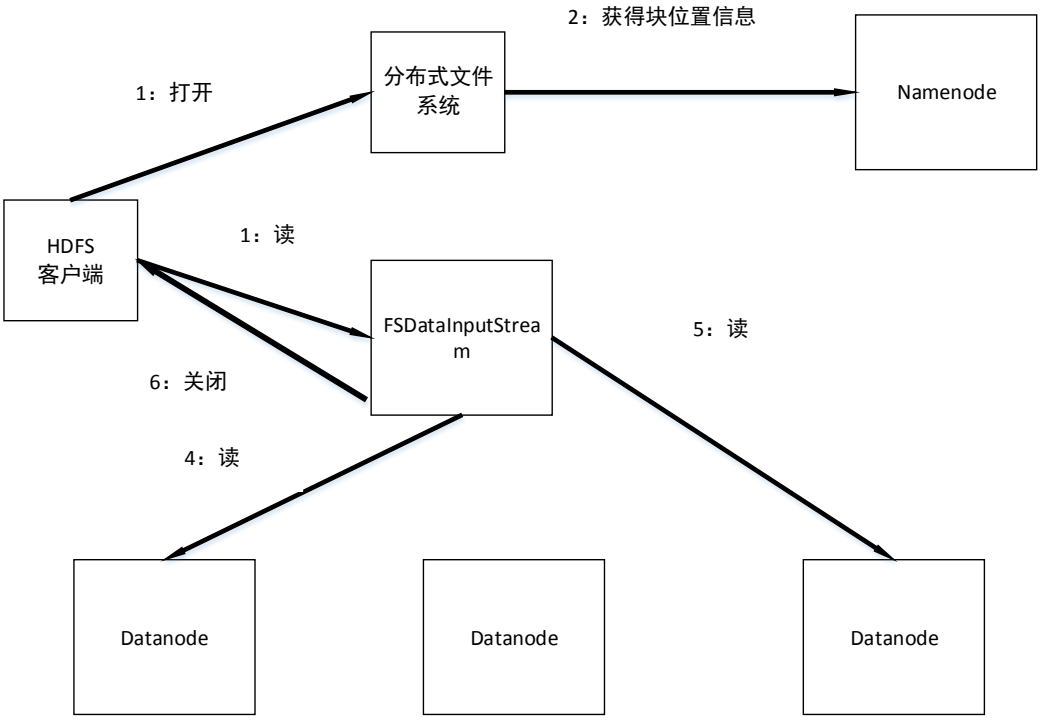


图 2. 5 读取 HDFS 文件流程

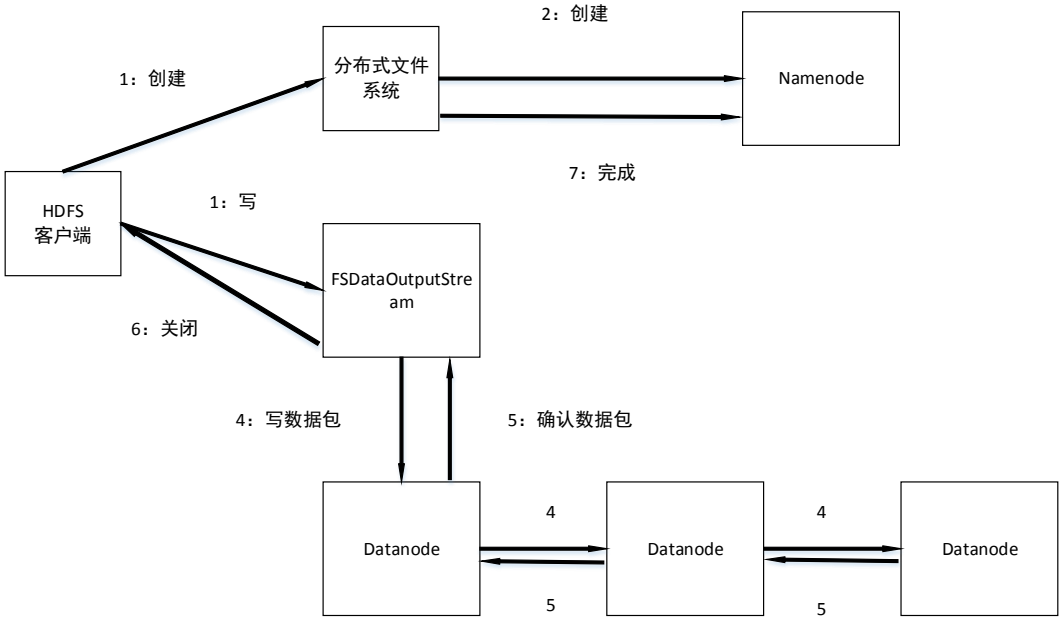


图 2. 6 写文件到 HDFS 流程

客户端写文件到 HDFS 的流程如图 2.6 所示，详细过程如下：

（1） 应用程序通过 HDFS 客户端向 Namenode 发起 RPC 请求，把要写入的文件元数据信息传输给 Namenode。

（2） Namenode 收到元数据信息后，检查要创建的文件是否存在以及是否有足够的权限。

（3） 若能够创建文件，Namenode 会返回给客户端一个该文件的记录，否则告知客户端创建异常。

（4） HDFS 客户端把文件切分为若干个 packets，然后向 Namenode 申请新的 blocks 存储新增数据。

（5） NameNode 返回用来存储副本的数据节点列表。

（6） HDFS 客户端把 packets 中的数据写入所有的副本中。

（7） 最后一个节点数据写入完成以后，客户端关闭。

通过 HDFS 的文件读写流程可知，每次从 HDFS 取数据或者存数据都要经过大量的磁盘 I/O 操作，如果一个 MapReduce 应用程序的输入数据量比较大，那么要从 HDFS 中读取的 HDFS 块就比较多，磁盘的读取与写入操作将会成为作业的性能瓶颈。

2.4.2 MapReduce 分布式计算框架

MapReduce 是大数据技术中一个经典的分布式计算框架，它能将单个计算作业分解成多个任务在多台计算机上并行执行，使得海量数据并行处理成为了可能。MapReduce 起源于搜索领域，主要为了解决 Google 搜索引擎面临的海量数据的处理的可扩展性不高的问题。总结它的设计目标，主要有一下三点。

（1） 易于编程：之前的分布式应用程序的设计（如 MPI）比较复杂，程序员需要把很多细节放在数据分片、节点间通信、容错、数据传输上，因而门槛较高。MapReduce 的设计目标之一是简化设计细节，将分布式应用程序的公共模块抽象成框架，程序员只需要关注业务逻辑，这样大大提高了开发效率。

（2） 高可扩展性：随着公司的用户量的增长和业务的拓展，积累的数据量（如用户日志）也逐渐增长，当数据量增长到一定程度后，现有的集群的计算能力和存储能力无法满足应用程序的数据处理需求，这时候只要往集群中添加机器，就能线性扩展集群的计算能力和数据存储能力。

（3） 高容错性：分布式系统中经常存在节点宕机、磁盘损坏、节点间通信失败、应用程序 bug 等问题，因此，任务执行失败和数据丢失的可能性较高。好在 MapReduce 的数据迁移或迁移计算算法能保持作业的

较高容错性。

Hadoop MapReduce 使用的是 Master/Slave 架构，如图 2.7 所示。主要包含四个组件：JobTracker、TaskTracker、Client 和 Task。下面将详细介绍一下这些组件。

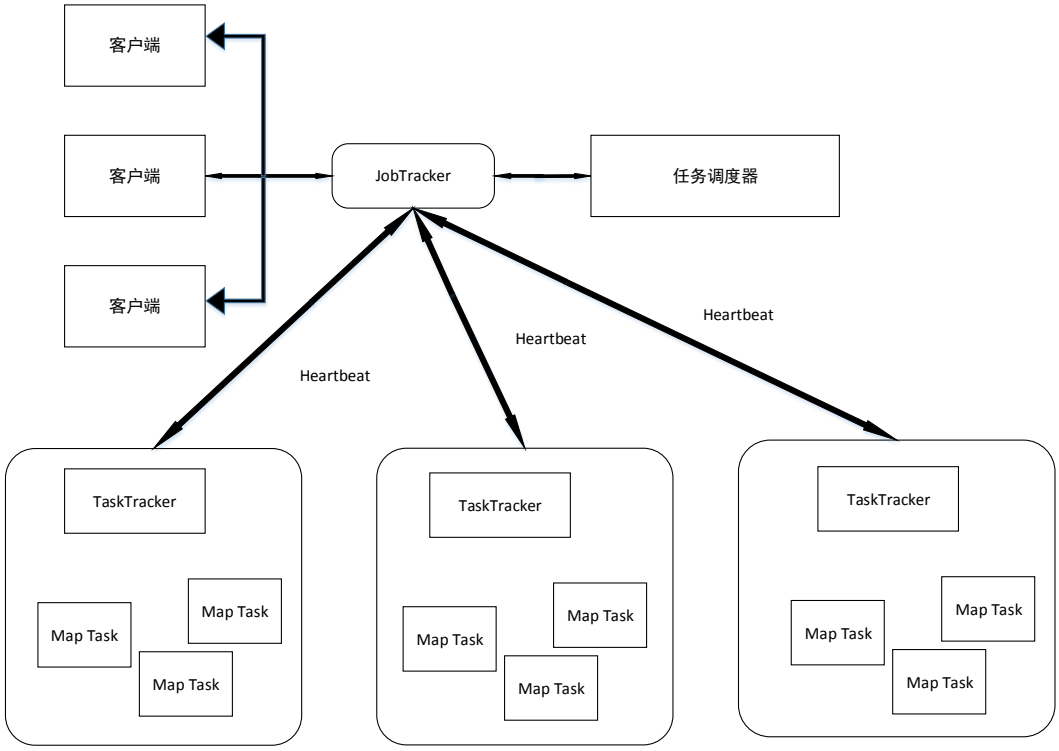


图 2. 7 MapReduce 架构

（1） 主节点 JobTracker

JobTracker 的主要作用是资源管理和作业调度。它监控 TaskTracker 和 Hadoop 作业的状态。如果任务执行失败，JobTracker 会将任务转移到其它计算节点上执行。JobTracker 保存着任务的执行进度、计算资源的使用情况等信息，并把这些信息通知给任务调度器，调度器根据调度算法分配任务到资源空闲的节点。在 Hadoop 框架中，程序员可以根据自己的需求设计自己的调度器，也就是说，任务调度器是可插拔的。

（2） 从节点 TaskTracker

TaskTracker 定期通过心跳机制 Heartbeat 将当前节点的任务的执行进度和资源使用状况发送给 JobTracker，监听 JobTracker 传送过来的指令并根据指令执行相应的操作（如启动任务、销毁任务等）。TaskTracker 使用计算槽 slot 作为资源（如 CPU、内存）划分的基本单元。任务在运行之前需要获取到 slot，Hadoop 调度器主要是针对计算槽 slot 进行分配。Slot 的类型有两种，Map slot 和 Reduce slot，Map slot 只能被 Map Task 使用，Reduce slot 只能被 Reduce Task 使用。TaskTracker 使用可配置的 slot 数目来控制任务的并发度。

（3） 客户端 Client

客户端 Client 为程序员提供了 MapReduce 应用程序的编程接口；同时，也提供了查询作业状态的接口。在 Hadoop 框架中，使用作业 Job 表示 MapReduce 应用程序。一个应用程序对应多个作业，一个作业能够被划分为几个独立的任务 Task。

（4） 任务 Task

Task 包含两种类型，Map Task 和 Reduce Task，他们的启动都由 TaskTracker 决定。虽然 HDFS 的数据存储的基本单位是 block，但是在 MapReduce 中，数据处理的基本单位是 split。Block 与 split 之间的关系如图 2.8 所示。split 是数据的逻辑划分，它不包含真实数据，只存储数据的元数据信息，如数据开始和结束偏移量、数据长度、存储数据的节点等。一个 split 对应一个 Map Task。split 的划分源于用户编写的划分函数。

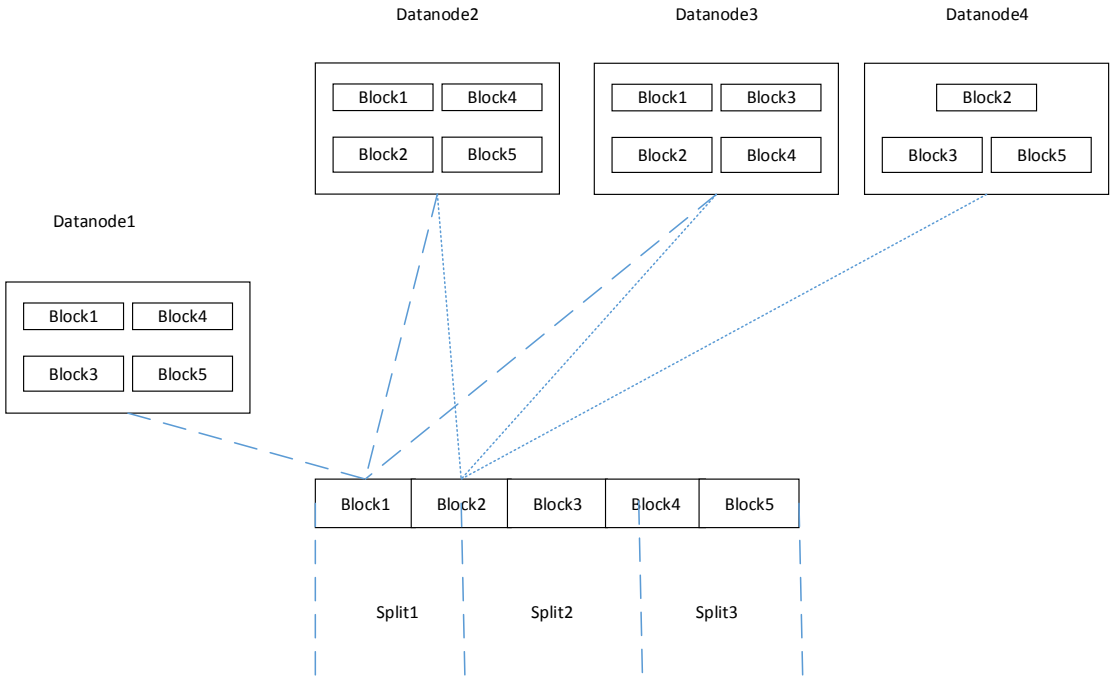


图 2. 8 block 和 split 的关系

MapReduce 为程序员提供了两个函数，map()和 reduce()，通过这两个函数，可以编写简单的分布式应用程序。map()函数的执行对应 MapReduce 的 Map 阶段，reduce()函数的的执行对应 Reduce 阶段。

Map 阶段如图 2.9 所示，map()函数的输入是 key/value 键值对，经过函数调用后，另外一系列键值对作为中间结果溢写到本地磁盘，MapReduce 会在底层将这些键值对根据键 key 进行分区，如果使用的是默认的分区算法，将进行哈希取模，使得键 key 相同的数据被传输到 reduce 函数上一并被处理。

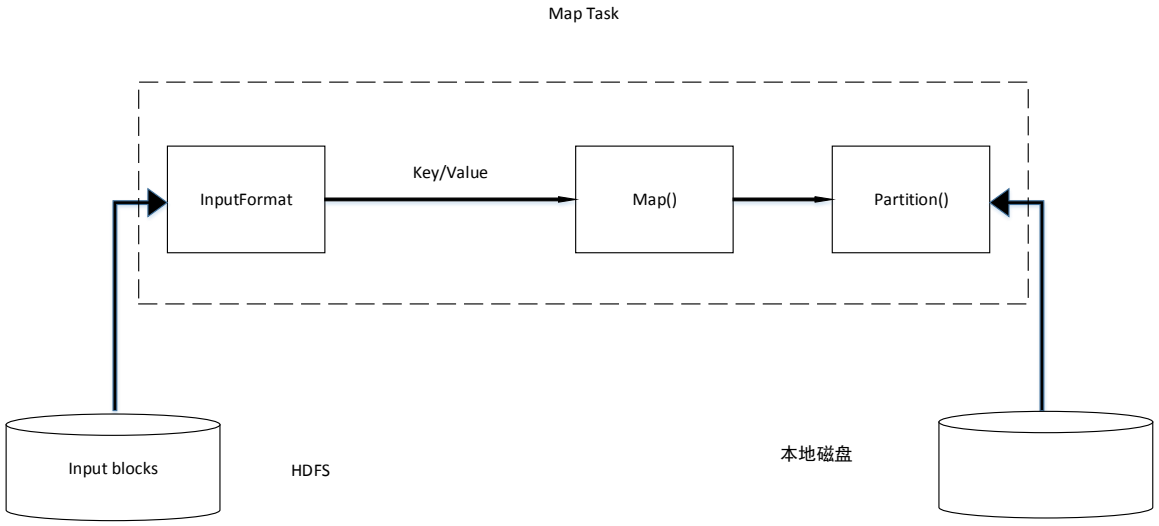


图 2. 9 Map 阶段执行过程

Reduce 阶段如图 2.10 所示，reduce()函数将 map 函数的输出的键 key 和 value 组成的序列作为输入，经过 reduce 函数调用后，生成最终的 key/value 键值对作为结果保存到 HDFS 中。

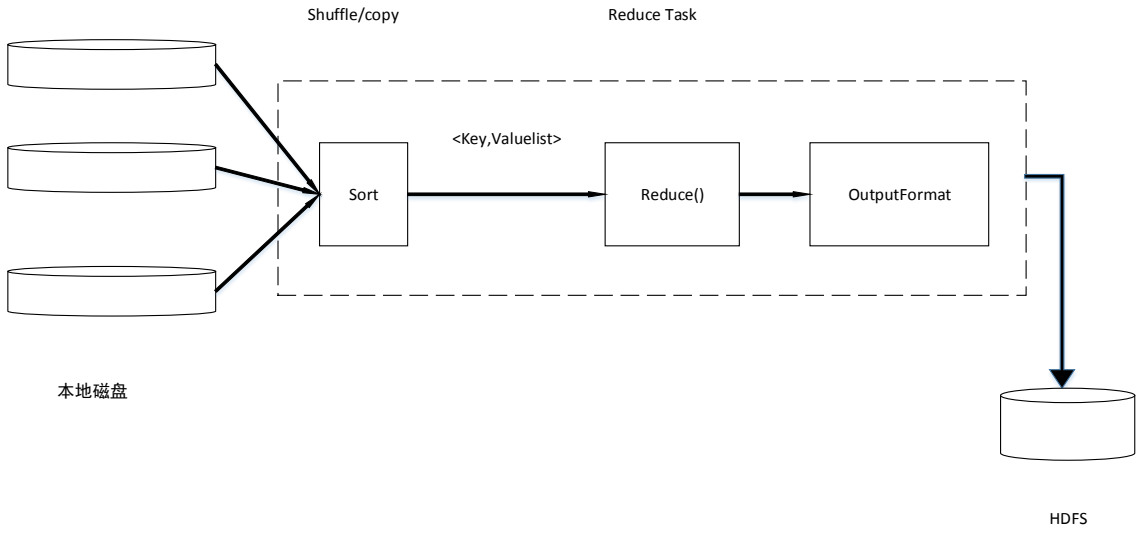


图 2. 10 Reduce 阶段执行过程

下面以大数据编程中的入门程序 WordCount 为例示范 MapReduce 应用程序的设计流程，WordCount 程序是统计文本中出现的单词的频率。

其中 map 函数如下:

```
map(String key, String value):
  words = SplitIntoTokens(value);
  for each word item in words:
        EmitIntermediate(item,”1”);
```

reduce 函数如下：

```
reduce(String key, Iterator values);
  int result = 0;
  for each v in values:
        result+=Integer.valueOf(v);
  Emit(key,String.valueOf(result));
```

MapReduce 程序编写完成后，指定好输入数据和输出数据的路径，将 WordCount 作业提交到 Hadoop 集群中。该作业的执行流程如图 2.11 所示，输入数据被 Hadoop 划分为多个输入分片 split，一个 split 将会对应一个 Map Task，Map Tas,一个个地读取 split 中的键值对 key/value，传入到 map()函数中处理，Map 阶段处理完后根据 Reduce Task 的个数进行哈希取模，分配到相应的 partition，然后使用归并排序将具有相同的 key 的键值对聚集在一起，传入 reduce 函数进行处理，处理完后将结果输出到文件中。

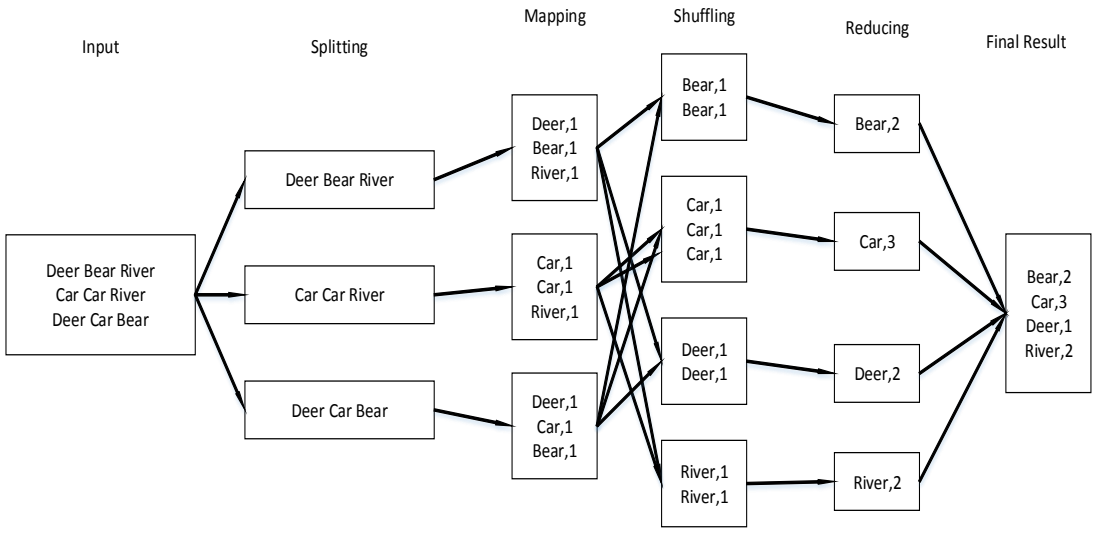


图 2. 11 WordCount 作业的执行流程

2.4.2 Hadoop YARN 框架

Apache Hadoop YARN （Yet Another Resource Negotiator，另一种资源协调者）是下一代 MapReduce 框架。在第一代 MapReduce 中，JobTracker 既要进行资源管理，又要完成作业控制，使得系统的可扩展性不足。另外，第一代 MapReduce 还有可靠性差、资源利用率低和无法支持多种大数据计算框架等缺点。这样的背景使得 Apache 和 Facebook 公司联合研发了 YARN。

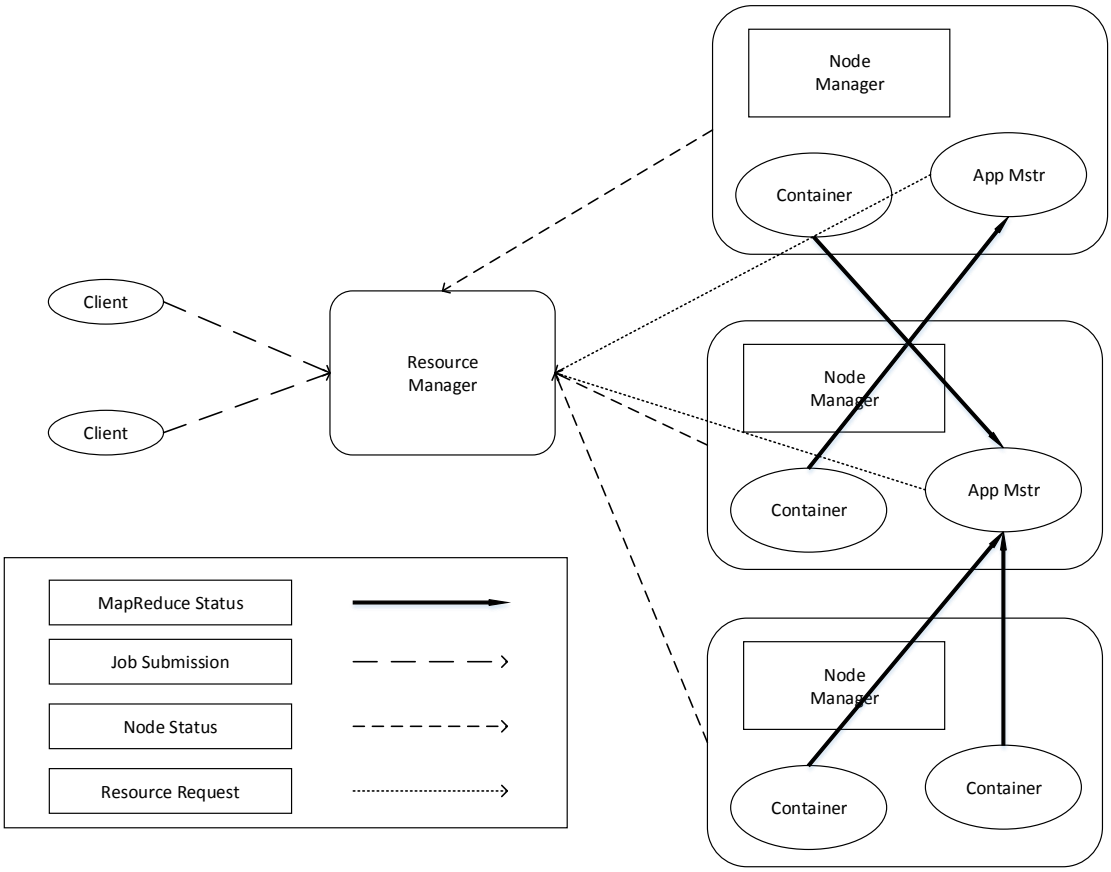


图 2. 12 Apache YARN 架构

同样地，作为分布式系统框架，它依然是一个 Master/Slave 架构，YARN 主要目标是将 JobTracker 的资源管理和作业调度进行解耦。YARN 架构如图 2.12 所示。主要有四个组件，包括 ResourceManager、ApplicationMaster、NodeManager 和 Container。资源管理的组件是 ResourceManager，作业调度的组件是 ApplicationMaster，ApplicationMaster 进程随着应用程序的提交成功而创建，它向 ResourceManager 申请计算资源。NodeManager 是 ResourceManager 在 slave 节点的代理，它向 ResourceManager 及时汇报当前节点上的资源使用状况和 Container 的执行状态，并根据来自 ResourceManager 的指令启动和销毁作业。Container 是 Hadoop 中资源分配的基本单元，它将内存、CPU、磁盘和网络等资源封装在一起。在 YARN 中，一个任务代表一个 Container。

2.5 Alluxio 分布式文件系统

Alluxio[30]是一个基于内存的虚拟分布式文件系统，它就像缓存一样，使用底层存储的数据，为上层的分布式计算框架提供存取服务。它能够加快大数据应用程序的数据读写速度。Alluxio 抽象了底层存储系统，为计算框架提供了统一接口，可以无缝迁移应用程序的存储平台到 Alluxio。图 2.13 描述了 Alluxio 在

大数据框架的层次。

图 2. 13 Alluxio 在大数据技术中的层次

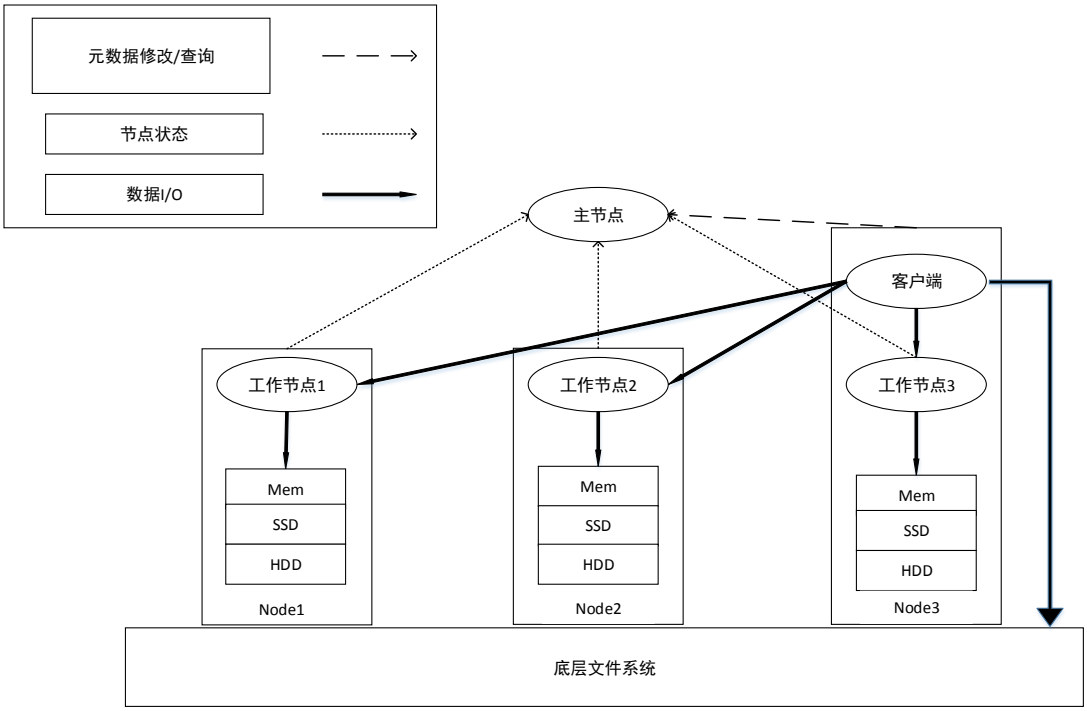


图 2. 14 Alluxio 的组成部分

与其它大数据框架如 HDFS、MapReduce、HBase、Spark 一样，Alluxio 也是一个 Master/Slave 架构，图 2.14 展示了它的四个组成部分，包括客户端 Client、底层文件系统 UFS、主节点 Master 和从节点 Worker。Master 管理存储数据的元数据信息，维护集群中从节点的状态信息。Worker 节点管理自己的 MEM、SDD 和 HDD，Client 作为客户端，为外界提供访问文件系统的 API 接口，UFS 为虚拟文件系统提供了数据备份的功能。Master 与 Worker 通过心跳机制保持连接，维护集群状态。Woker 使用缓存算法，在内存中存储热数据，数据的备份在 UFS 中。读取文件系统的文件时，Client 先向 Master 查询文件元数据信息，若元数据信息返回的信息表明 Client 数据在 Alluxio 中，则从 Worker

中获取文件，否则，从 UFS 中读取文件。

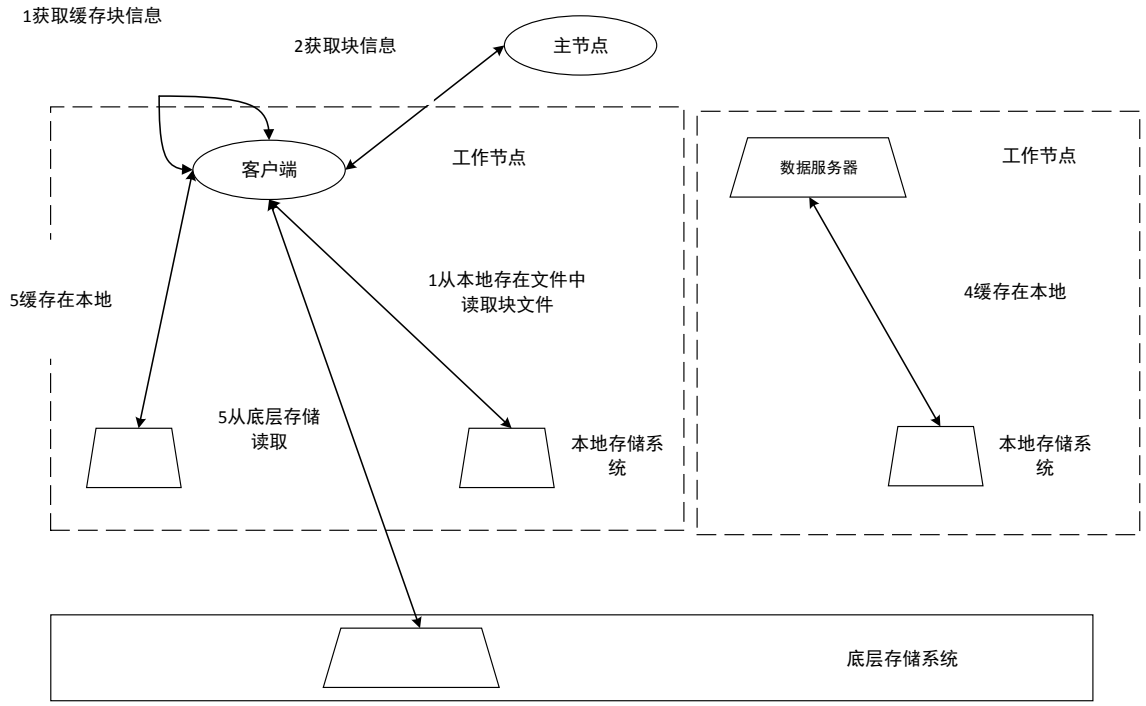


图 2. 15 Alluxio 读数据流程

Alluxio 读缓存是框架的重要设计思想，当上层计算框架访问数据时，或者客户端读取文件时，可以从 Worker 节点的本地内存读取文件，也可以从远端 Worker 节点的内存拉取，并且根据读操作所属类型参数来决定是否将文件缓存在本地。Alluxio 包含的读操作类型如表 2.2 所示。Alluxio 执行数据读取操作时，首先查看 Client 所在的 Worker 节点的内存中是否有要读取的数据块，若没有则请求其他 Worker 节点的数据，若仍然没有就向 UFS 读取。具体读取流程如图 2.15 所示。

由于 Alluxio 的写操作是直接往本地 Worker 的内存中写，也相当于为分布式应用程序提供了写缓存，从而提高了数据写的吞吐量。目前 Alluxio 框架写操

表 2. 2 Alluxio 的读操作类型 ReadType

读类型	作用
NO_CACHE	读取数据，不保存副本在 Alluxio 中。
CACHE	如果本地 Worker 节点无数据，那么在本地 Worker 中添加一个副本。
CACHE_PROMOTE	如果本地 Worker 节点有数据，将数据移动到 Worker 的最高层；如果本地 Worker 节点无数据，那么在本地 Worker 中添加一个副本。

表 2. 3 Alluxio 的写操作类型 WriteType

写类型	作用
CACHE_THROUGH	同步地把数据写入 UFS 和 Alluxio。
MUST_CACHE	同步地把数据写入 Alluxio，但不写入 UFS。
THROUGH	同步地把数据写入 UFS，但不写入 Alluxio。
ASYNC_THROUGH	同步地把数据写入 Alluxio，异步写入 UFS。

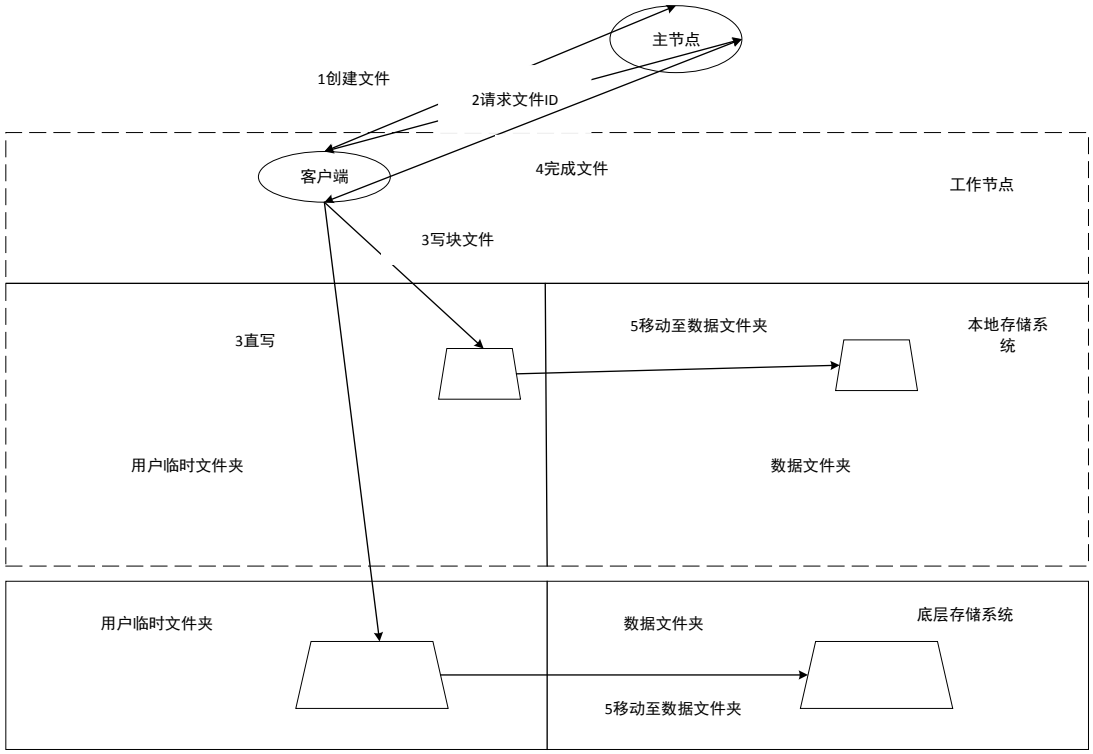


图 2. 16 Alluxio 写数据流程

作类型有表 2.3 中的四种类型，图 2.16 描述了写类型为 CACHE_THROUGH 的具体工作流程。Client 先与 Master 通信得到文件创建信息，如块 ID 号和输入流等，同时把数据写到本地 Worker 的内存和 UFS 中，写入完成后通知 Master 写操作结束。

Alluxio 作为一个以内存为中心的文件系统，加速了及时性要求高的应用程序。百度公司使用典型的百度查询来衡量其性能，如果使用原来的 Hive 系统，需要花 1000 多秒完成一个典型的查询；使用 Spark SQL—only 系统，要花 300 秒；但 Alluxio 和 Spark SQL 系统，只需要花 10 秒。实现了 100 倍的速度提升，并满足了交互式查询要求。去哪儿网（Qunar）的一个基于 Alluxio 的实时日志流的处理系统，重点解决了异地数据存储和访问慢的问题，从而将生产环境中整个流处理流水线的性能总体提高了近 10 倍，而峰值时甚至达到 300 倍左右。

2.6 视频转码相关研究与进展

2.7.1 转码算法加速

视频转码算法加速的主要是根据类型入手，不同的转码类型的转码加速思路是不一样的。视频转码的类型主要包括码率转换（Bitrate reduction）、空间分辨率转换（Spatial resolution reduction）、时间分辨率转换（Temporal resolution reduction）等[31]。

在视频转码码率转换的最初的研究成果是，H Sun [32]通过在保证各块的比特数满足要求的情况下丢弃高频 DCT（Discrete Cosine Transform）系数（如图 2.16）来减少计算量，同时在编码时使用运动矢量信息（motion vector）和编码模式（Coding decision modes）来避免重新进行运动估计和和编码模式选择。虽然这个方法精简了转码步骤，但是丢弃的这些系数会使得重新编码后的图像损失一部分精度。因此，文献[33]提出了一个有选择性地丢弃 DCT 系数的算法来减轻丢弃的系数带来的图像失真后果。文献[34],[35]提出运动矢量优化方法（Motion vector refinement schema,MVR）来改善运动估计精度。如果视频转码的目标是降低码率，那么主要是从变换域上的运动估计[36][37][38][39][40][41]入手，这样可以补偿漂移误差，但是，在不同的编码标准之间进行转换时，计算量非常大。

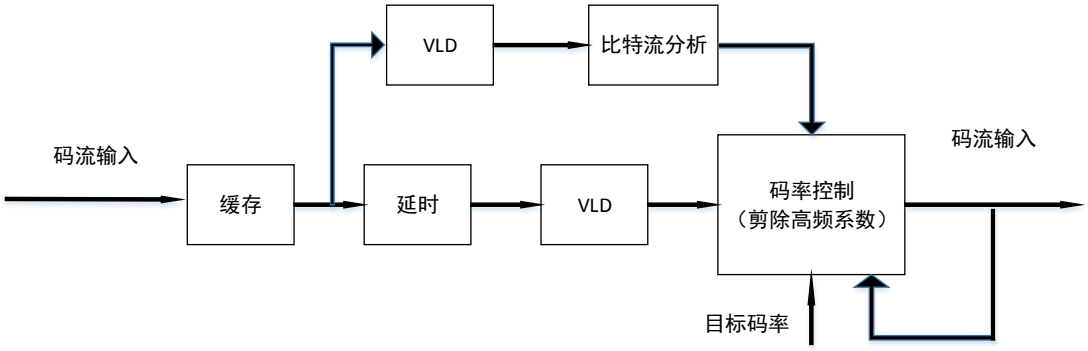


图 2. 17 截断高频系数加速视频转码

因为有些屏幕的分辨率较低，为了适应这些低分辨率的屏幕，往往需要进行空间分辨率转码。不同于降低码率转码的是，它往往需要在“全解全编”转码器中增加一个采样模块[31]。Shen G[42]和 Shanableh T[43]使用下采样的方法简化转码步骤。J Xin[44]和 N Bjork [45]使用运动矢量的映射算法和伸缩算法，这种方法在遇到脉冲噪声时变现不够理想。文献[45]提出了一个算法，他们的编码器会在运动估计操作执行之后从视频流的模式决策信息中决定宏块的新的编码模式。

为了适应网络带宽较低的场景，可以对视频进行时间分辨率转码，也叫作帧率转码。由于视频流中的 I 帧和 P 帧不依赖于 B 帧，文献[46]选择将 B 帧直接

丢弃，但是，大量地将 B 帧丢弃将会使得有些视频播放效果不连续。为了解决这个不连续的问题，Hwang J N[47]等人利用运动矢量的累积幅度来考虑是否将帧丢弃。文献[48]将输入视频中的 B 帧直接转换成 P 帧进行帧率转码。另外，他们还提出了运动矢量合成算法，因为丢弃帧会使得运动矢量给视频各帧带来的依赖关系中断，利用被丢掉参考帧的运动信息，将多个运动矢量合成为一个新的参考帧的运动矢量。

2.7.2 GPU 硬件加速

GPU 最初是专门用于计算机上图像运算的微处理器，随着 GPU 通用计算技术的发展，它在浮点计算、并行计算也有十倍甚至上百倍的计算速度。为了解决 GPU 上的复杂的计算问题，显卡厂商 NVIDIA 推出了 CUDA（Compute Unified Device Architecture）并行计算架构，该架构同时支持 C、C++和 FORTRAN 语言接口，方便程序员编写应用程序。

鉴于“全解全编”转码器的计算复杂性，许多人将视频转码的运动估计、帧内预测、DCT 变换、量化等关键环节转移到 GPU 中来计算。文献[49] 通过重新排列 4×4 块的编码顺序，提出了一种基于 GPU 的 H.264 / AVC 运动估计。在视频流的 I 帧中，帧内预测是一个耗时的环节，同时，这个环节需要用到相邻宏块的信息，换句话说，帧之间是相互依赖的，因此，帧内预测的并行原子单元不会是宏块级。这种依赖性对多核 GPU 上的计算带来了挑战，这在很大程度上依赖于并行数据处理来实现优异的加速。为了解决这个问题，文献[50]分析了帧内模式决策中的数据依赖性，并提出了新颖的基于贪婪的编码顺序来实现高度并行处理。文献[51][52]通过利用 CUDA 框架开辟负责 4×4 块的运动估计的线程在 GPU 上执行，然后计算出宏块的 SAD，合并成分块模式和运动矢量。上述并行算法的运动估计使用暴力搜索算法，遍历的点比较多，导致计算出的 SAD 过多了，影响了算法的运行效率。文献[53]提出了一种通过 NVIDIA CUDA 使用多核 CPU 和 GPU 的空间 SVC 可扩展计算模型。基于所提出的计算模型，然后提供解决该 CPU-GPU 协同工作架构的具有挑战性的数据转换问题的解决方案。文献[54]提出了一种新颖，快速，高度并行和可配置的结构，用于多视频编码标准（包括 H.261，MPEG-1,2,4，H.264 / AVC 和 AVS）中的锯齿扫描和可选扫描。 通过配置 ROM 数据可以支持任意扫描模式，并且所提出的架构可以大大减少处理周期。 这使得将扫描单元集成到变换流水线阶段成为可能，因此模式决策可能更有效。文献[55]介绍了 H.264 / AVC 基线轮廓熵编码器的高性能和低成本架构。在所提出的设计中，使用有效的方法来设计 CAVLC 以降低硬件成本。

尽管 GPU 能够优化“全解全编”转码器，成倍地提高视频转码速度，但是，但是，视频转码的复杂性使得完全在 GPU 并行计算难以实现，宏块之间相互依

赖导致难以扩展，GPU 价格不菲导致成本高昂，使得 GPU 硬件加速成为一个难点与痛点。

2.7.3 分布式并行加速

在开放 GOP 中，参考帧可以来自之前的 GOP。 在封闭 GOP 中，所有参考帧都属于同一个 GOP。 闭合 GOP 表示可以独立转码的原子单元。 通常，如果输入序列具有瞬时解码器刷新（IDR）帧，称为闭合 GOP 的编码结构，则该视频转码任务可以分成几个子任务。分布式集群中的机器将执行子任务完成转码。

为了将一个视频转码作业分解成多个任务，需要将视频进行分割，分割粒度的不同将大大影响视频转码速度，如果分片粒度过大，导致子任务数量较少，可能无法有效利用分布式集群的计算资源，如果分片粒度过小，导致子任务数量较多，将会增加任务的启动开销和网络通信开销。文献[56]描述了一个分布式视频转码系统，它可以同时将 MPEG-2 视频转码为不同速率的各种视频编码格式。视频转码器将 MPEG-2 文件沿时间轴分成小段，并对它们进行并行转码，该方法使处理器间通信开销最小化并消除来自重新编码的视频的时间不连续性。文献[57]计算了不同分片长度下的转码作业的最终完成时间，推导出分片数目如果在一个合理的数量范围内，任务启动开销和调度时间不会太多，那么视频分片的复杂度越高，转码时间越小的结论。文献[58]提出了多粒度分割算法，先粗粒度分割输入视频，再将部分视频进行细粒度分割，防止执行转码任务较慢的节点拖累整个作业的完成。文献[59]从任务启动开销的最小化角度入手，分析了分片中帧的数量相同的但大小不同和帧的大小相同但数量不同的情况，首先将分片切割成一个个 GOP（Group of Pictures）作为分片的原子单元，然后合成大的片段。

可扩展性是一个术语，它用于表示系统通过适当添加资源处理满足日益增加的工作量的能力。这个特殊的属性对于我们的分布式转码器很重要，以便维护用户暂停的一组必需的约束或系统的总负载要求。例如，某些视频转码请求可能需要比其他视频转码请求更多的计算能力，或者用户可能希望在相对较短的时间内开始播放视频。通过可扩展的系统，这种用户限制可以通过为各个转码器请求使用适量的资源来满足。可扩展的转码器还可以满足不同的负载条件。分布式计算是互联网上视频处理非常有前途的技术，它可以有效地使用分布式机器通过并行启动来处理繁重的工作。Dean 等人[8]提出了经典的分布式编程模型 MapReduce，它的高可扩展性为我们提供了一种有效的解决方案，可以对视频进行并行转码。目前已经有人对基于 MapReduce 的视频转码[60-62]采用先到先服务（FCFS）调度策略进行了研究，这些工作可以通过并行启动子任务来缩短处理时间，但仍有很大的改进空间，因为 FCFS 可能会导致负载均衡问题。因此，一些研究人员致力于构建在异构云环境上的视频转码服务[61-63]。 Huang 等人[63]提出了一种基

于云的代理，可以提供高质量的互联网流媒体，将视频转码过程定义为在线调度问题。 但是，他们没有考虑到子任务启动开销。 文献[61]提出了一个基于 MapReduce 的系统，可以将各种视频编解码格式转换为 MPEG-4 视频格式。Li 等人[62]实现了一个云转码器。 它只需要用户上传转码请求而不是视频内容，然后从互联网下载原始视频内容，根据用户需求转码视频，并通过加速的云内数据以高数据速率将转码后的视频传输给用户。杨竞通过在样本集群中，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。上述几项工作仅从并行化的角度缩短了转码时间，而不考虑异构集群中任务的负载均衡。

调度独立任务到异构分布式计算系统的目标是尽量减少作业的最终完成时间，这与车间调度问题（Job Shop Scheduling）类似，这是一个 NP 难题[65]。为了解决这个 NP 难问题，TD Braunt 等人比较了 11 种启发式算法，并证明了最有效的算法是 Min-min 算法[66]。 此外，由于分片的转码时间与分片复杂度成正比，Min-min 算法等效于视频转码模型中的最小完整时间（MCT，Minimum Complete Time）算法。但是这些算法是通用型算法，没有考虑到视频转码的特性。因此，F. Lao 等人[21]提出了一个考虑任务开销的模型，并使用 Max-MCT 算法解决了这个问题。该算法将这个调度问题抽象为虚拟背包问题，并将复杂的分片分配给计算能力较强的机器，以减少它们在子任务启动上的时间。Lin 等人[22]提出了一种新的并行化视频转码框架，其中包括任务预分析，自适应阈值分割和最小完成时间（MFT，Minimum Finish Time）调度。在此框架的基础上，他们提出了一种称为 MLFT 的负载均衡调度算法，该算法通过将复杂度较高任务分为多个子任务并在分配队列中重新分配任务来缩短最终完成时间。不幸的是，文献[21],[22]中的研究人员没有考虑机器之间段的传输开销。本地感知调度算法可以花费时间主要用于视频转码而不是视频数据传输。因此，在文献[23]中，提出了一种本地感知任务调度算法 PLTS，它结合了两种传统启发式算法 Max-min 和 Min-min 的优点来减少最终完成时间，并对异构集群进行负载均衡。但是，该算法不会平衡分片传输时间和任务执行时间。因此，最终完成时间的优化空间仍有很大的余地。

2.7.4 视频转码复杂度预测

在本文的第三章中，使用任务调度算法来实现异构集群上的视频转码作业的任务的负载均衡，要调度这些视频转码任务，需要进行对视频转码任务调度进行

建模，因此，需要预测视频的复杂度视频的复杂度。对于一个分片，它的复杂度是计算能力和转码操作时间的乘积。

Cheng 等人[71]设计了一个基于云的实时转码系统，用于将移动设备与常规桌面客户端连接成高清视频会议。他们引入了基于工作负载预测模型的面向转码的调度算法，以减少网络流量和计算开销。然而，他们的预测模型被应用于实时视频会议，因此不适合视频离线转码调度。Ma 等人[72]提出了一种针对云环境设计的 DASH（基于 HTTP 的动态自适应流媒体）视频转码的动态调度器。他们基于测量的统计和概率理论引入了视频转码时间（VTT）估计模型。然后他们设计了一个调度器，它选择了最快的处理器来运行高优先级的作业。Deneke 等人[73][74]研究了如何基于机器学习方法，支持向量回归和神经网络预测转码复杂度。然而，这两项工作首先需要对大量样本进行视频转码，然后利用机器学习理论或概率理论，根据某些参数（即持续时间，比特率，帧率，分辨率等）而不是视频特性来训练其分类器 （即运动估计/运动补偿），因此它们是粗粒度的。文献[23]将解码和编码过程划分为几个独立的模块，并确定每个模块的复杂性和各个参数之间的相关性。这个工作可以准确地分析视频转码复杂度，与转码时间[34]相比，运行时开销也非常小。

2.8 本章小结

本章首先介绍了视频流编码原理和视频转码的原理，并由一个最简单的视频转码器“全解全编”转码器开始，介绍了对视频转码加速的研究现状。同时，目前业界有一个视频转码开源框架 FFmpeg，使用这个框架可以快速地实现视频转码系统。为了实现分布式并行加速转码，在第四章将使用本章提到地 Hadoop 框架和 Alluxio 分布式文件系统。

第3章 Hadoop 异构集群上的视频转码任务调度算法

3.1 引言

在 Hadoop 异构集群中，每个节点的计算能力是不一样的，而 Hadoop 的最初的设计和优化是针对同构集群的，为了有效利用集群的计算资源，本章首先定义了 Hadoop 异构集群上的视频转码架构，以及对视频转码任务调度进行抽象和建模，并使用合适的调度算法完成任务的分配与调度。本章将重点介绍一个本地感知的视频转码任务调度算法 LA-MCT，并进行大量的模拟视频转码实验进行推导。

3.2 Hadoop 异构集群上的视频转码架构

图 3.1 显示了基于 Hadoop 异构集群的视频转码架构图，主要由分割器（Splitter），预测器（Predictor），任务调度器（Task Scheduler）和分片合并器（Merger）组成。因为分片的转码过程是相互独立的，并且在 Hadoop 中，一个 Split 对应一个 Map Task，所以集群上一个分片的转码过程称为任务，总之，任务是机器中的一个转码实例。分片和任务在本章中是相同的概念，但为了更好地描述，本文在某些场景中将任务称为分片。

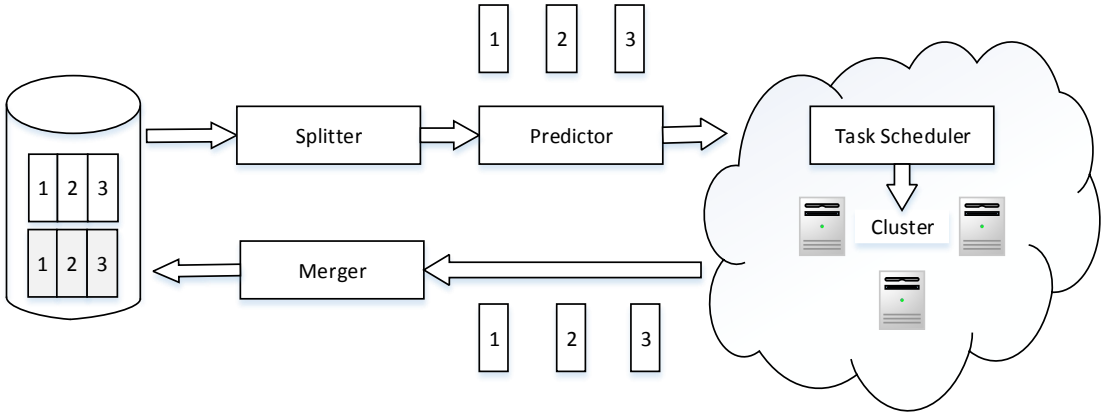


图 3. 1 视频转码基本架构图

在宏观上，视频由一系列视频序列组成，视频序列由 GOP 头和一些 GOP 组成。在视频编码中，GOP 结构规定了帧内和帧间排列的顺序。GOP 是编码视频流内的连续图片的集合。每个编码的视频流由连续的 GOP 组成，从中产生可见帧。 在压缩视频流中遇到新的 GOP 意味着解码器不需要任何先前的帧来解码下一个帧。 因此，GOP 是可作为原子单元被独立地进行视频转码。

当系统收到转码请求时，它使用分割器将视频分割成 GOP 级别的多个视频

分片。一般来说，与分片传输时间和转码时间相比，这个过程所耗的时间可以忽略不计。一方面，分片包含几个 GOP 而不是一个，因为如果一个任务只包含一个 GOP，整个转码过程将产生太多的任务，而在 Hadoop 中，MapReduce 任务主的生命周期包括三个阶段：启动，执行和销毁。任务启动和销毁时间是与任务无关的，如果分片的粒度越小，太多的任务会浪费太多时间来启动和销毁任务，从而增加整个转码作业的执行时间。另一方面，分片不能包含太多的 GOP。如果一个段中的 GOP 数量达到构成视频序列的值，因为处理视频序列需要比 GOP 或帧消耗更多的时间，导致更高的时间延迟。

在分割器将作业分成多个分片之后，系统使用预测器来预测分片复杂度。对于一个分片，它的复杂度是计算能力和转码操作时间的乘积。目前，存在很多视频转码复杂度预测算法[23],[67-74]，目前最好的复杂度预测算法来自文献[23]，因此，可以利用该算法来估计视频段的复杂度。在本章中，分片复杂度预测不是研究重点，本章主要关注任务调度，因此在进行模拟转码实验的时候每个分片的复杂度是预设的。

不同的 GOP 包含不同数量的帧，因此每个帧的复杂度往往不同，因此这些分片的复杂度是异构的。一般来说，集群提供多台具有不同计算能力的机器，所以 Hadoop 集群的转码能力也是异构的。每台机器的转码能力可以通过视频转码历史日志进行分析与估计[21-23]。任务调度器根据视频分片的复杂度、机器的转码速度、任务启动开销、视频分片在集群机器间的传输开销进行建模，并使用任务调度算法来实现负载均衡，从而有效利用异构集群的计算资源。集群中的集群根据调度算法的调度结果，启动 MapReduce 任务，执行视频转码，并把转码成功的分片保存到本地磁盘。

一旦任务执行完成后，转码后的分片将被传输到 Splitter 所在机器上，直到所有这些分片到达后，Splitter 将所有转码后的分片连接在一起。最后，一个视频序列将被生成并存储在磁盘中。

3.3 视频转码任务调度模型

在视频转码任务调度模型中，对于给定的视频，相比于任务的转码时间，分布式集群花费在在分割器 Splitter，预测器 Predictor 和合并上 Merger 的时间可以忽略不计[23]。因此，本章只针对任务调度进行建模。

Hadoop 集群由多个异构机器$\mathrm{J}=(1,2,\ldots,\mathrm{M})$组成，其中集合的大小是M。每台机器的转码能力表示为$\mathrm{P}=(p_1,p_2,\ldots,p_N)$。分割器将视频序列分成N个分片$\mathrm{V}=(v_1,v_2,\ldots,v_N)$，分片的大小表示为$\mathrm{S}=(s_1,s_2,\ldots,s_N)$。每个分片的复杂度可以通过预测器 Predictor 来预测，并表示为$\mathrm{C}=(c_1,c_2,\ldots,c_N)$。分片中的复杂度之和记为$C_{sum}=\sum_{i=1}^{N}c_i$。

模型忽略任务启动开销，一般情况下，分片c_i的在机器p_j的上的任务执行时间由分片传输时间，任务启动开销和分片转码时间组成。分片传输时间$d^i_{m',m}$与存储分片的机器和转码机器之间的带宽有关。任务启动开销是一个与任务无关的常量，记为o_m。 转码时间与分片复杂度c_i成正比，与机器的计算能力p_j成反比。那么机器上分片c_i的预期转码时间（ETT, expect transcoding time ）可以通过下式计算：

$$\mathrm{ETT}(\mathrm{v_i},\mathrm{m}) = \frac{\mathrm{c_i}}{\mathrm{p_m}} + \mathrm{d^i_{m',m}} + \mathrm{o_m} \tag{3-1}$$

其中$d^i_{m',m}$是从机器m'到m的分片v_i的传输时间，可以通过以下公式计算：

$$\mathrm{d^i_{m',m}} = C_{m',m} \times \frac{s_i}{B_{m',m}} \tag{3-2}$$

其中$c_{m',m} \in \{0,1,3\}$，当$c_{m',m} = 0$时表示任务分配给节点级本地机器，也就是任务的执行与存储是在同一节点上的，此时分片传输时间为 0；否则，分片传输时间与网络带宽$B_{m',m}$成反比。$c_{m',m} = 1$表明任务是执行在存储该分片的相同机架但不同机器上的任务，$c_{m',m} = 2$意味着任务在另一个机架的机器上执行。

在使用任务调度算法将所有任务映射到相应的机器后，每台机器将独立执行转码任务。机器j上被分配的任务可以表示为θ_m，θ_m上完成所有任务的期望完成时间 EFT（expect finish time）为：

$$EFT_{\theta_m} = \sum_{v_i \in \theta_m} ETT(v_i, m)$$

$$= \sum_{v_i \in \theta_m} (\frac{c_i}{p_j} + \mathrm{d^i_{m',m}}) + |\theta_m| \times \theta_m \tag{3-3}$$

任务调度结果记为$\theta = \{\theta_1, \theta_2, \dots, \theta_M\}$。 一旦θ中的所有任务执行完成，转码作业就完成了，整个作业的预期完成时间FT 可以表示为：

$$\mathrm{FT} = max_{\theta_m \in \theta}(EFT_{\theta_m})$$

$$s.t. \theta = \{\theta_1, \theta_2, \dots, \theta_M\}$$

$$\cup_{\theta_m \in \theta} \theta_m = J \; \forall \theta_{m1}, \theta_{m2} \in \theta, \theta_{m1} \cap \theta_{m2} = \emptyset \tag{3-4}$$

公式（3-4）与作业车间调度问题（JSS）相似，JSS 问题已被证明是 NP 难问题，所以这个任务调度问题也是 NP 难问题。获得 NP 难问题的全局最优解决方案是复杂而且耗时的。如果没有有效的算法，人们必须遍历所有可能的解决方案，导致时间复杂度高达$O(N^M)$。显然，如果分片数目比较多，找到最佳解决方案是不现实的。

3.4 LA-MCT 任务调度算法

在本章中，为了解决这个 NP 难问题，本章提出了一种新的启发式算法 LA-MCT。这个算法的核心思想是基于数据本地性和负载均衡策略。该算法迭代地

将所有分片划分为本地分片集合和远程分片集合。根据这两个任务集的特点，算法使用 MCT 负载平衡策略将这些分片映射到可选机器。最后，通过比较每次迭代中的预期完成时间FT来确定最佳划分模式。

对于每个片片，默认情况下在 HDFS 中有 3 个副本。根据视频是否在存储分片的副本的机器上进行转码，分片集和V被划分为两部分，它们是本地分片集L和远程分片集R。考虑一个分片，如果它只允许被映射到存储其副本的机器上，那么可以将它归类到本地分片集L中，所有这些分片组成集合$\mathrm{L} = (c_1, c_2, \dots, c_{|L|})$。相反，如果一个分片只被允许映射到没有存储其副本的机器，那么它将被分类到远程分片集R中，所有这些分片组成集合$\mathrm{R} = c_1, c_2, \dots, c_{|R|}$。 |L|和|R|是集合的大小。L和R所有分片中的复杂度和被表示为C_L和C_R。显然，它们满足以下等式

$$C_{sum} = \sum_{c_i \in L} c_i + \sum_{c_j \in R} c_j = c_L + c_R \tag{3-5}$$

算法 3.1 分片集划分算法

算法：分片集划分算法

输入：分片集合$\mathbf{V} = (v_1, v_2, \dots, v_N)$，分片复杂度集合

$\mathbf{C} = (c_1, c_2, \dots, c_N)$，本地分片集的分片复杂度之和$C_L$。

输出：本地分片集合L，远程分片集合R

```
1: 集合L = ∅, R = ∅;
2: temp = 0; isLocal = false;
3: for V中的每个分片v_i do
4:     if isLocal then:
5:         temp = temp + C_{v_i};
6:         if temp ≤ C_L then:
7:             L = L ∪ v_i;
8:         else
9:             把分片v_i分割为v_{i1}和v_{i2}使得v_i的复杂度小于并尽可能接
       近于C_L − (temp − C_{v_i});
10:            L = L ∪ v_{i_1}; R = R ∪ v_{i_2}; isLocal = true;
11:        end if
12:    else
13:        R = R ∪ v_i;
14:    end if
15: end for
16: return L , R
```

首先初始化$C_L = C_{sum}$，$C_R = 0$，算法不断减小C_L，直到它小于 0。考虑到尽可能大的C_L，需要为不断缩小C_L的值设置一个步长C_{step}步长。因为C_L的大小的不

同，C_{step}的大小的设置是一个复杂的问题。如果C_{step}太大，会降低调度结果的精确度; 如果C_{step}太小，导致算法的执行时间很长，使得模型无法忽略任务调度器中的任务调度时间。 因此，有必要在精度和计算时间之间达到合理的折衷。基于这个原因，令C_{step}的值等于V中的最小分片的复杂度。

算法 3.2 MCT 算法

算法：MCT 算法

输入：机器的计算能力集合P = ($p_1, p_2, \ldots, p_N$),分片v_i，分片复杂度C_i。

输出：最小完成时间 Min，有着最小完成时间的机器 m

```
1: 获得可以分配任务的集合machines;
2: min = ∞;
3: for machines中的每个分片m do
4:     根据公式（3-1）和（3-3）分别计算计算ETT(v_i, m)和EFT_θm;
5:     if ETT(v_i, m) + EFT_θm < min then:
6:         Min = min;m = j;
7:     end if
8: end for
9: return Min , m
```

对于每一轮迭代，都会生成一个新的C_L值，并根据公式（3-5），必然有一个C_R与之对应，然后使用任务集划分程序来确定本地分片集L和远程分片集R。由于C中的复杂度c_i是一个离散值，对于给定的一对C_L和C_R，算法从集合V中选择一个分片，并将其添加到本地分段集L中，直到 L中的所有分片的复杂度之和小于并且尽可能接近C_L。 显然，V中的剩余部分形成集合R。

算法 3.1 描述了将任务集V划分为L和R的过程。在第 6 行和第 7 行中，先将分片加入到本地分片集L中，在第 9 行和第 10 行中，如果再加入当前分片v_i到L中，那么L中的所有分片的复杂度之和将超过C_L。由于一个分片包含多个独立的 GOP 单位，因此可以进一步划分该分片，以满足L中所有分片的复杂度之和尽可能接近给定值C_L。当集合L“装满”后，isLocal 变量置为 true，此时，剩余的分片加入到远程分片集合R，代码第 13 行表示分片v_i被划分到集合R。

当由算法 3.1 生成集合L和R时，程序以降序复杂度的顺序遍历L和R中的分片，然后使用 MCT（Minimal Complete Time）程序映射这些独立的分片。分片按复杂度从大到小的顺序进行排序的原因是可以保证长任务优先执行，防止某些处理器在作业执行结束时处于空闲状态。为了避免重复使用 MCT 的代码，本文在算法 3.2 中定义它。

在算法 3.2 中，第一行中，对于每个分片，都有一个可以分配的机器列表

machines。正如之前提到的 LA-MCT 算法思想中，针对本地分片集L中的分片，返回的 machines 列表的元素个数是分片的副本个数，因为默认情况下每个分片有 3 个副本，因此默认情况下是 3，表示这个分片只能选择存储这三个副本的。针对远程分片集R中的分片，返回的 machines 列表的元素个数是集群的从节点个数减去当前分片的副本个数。在第 3-8 行中，算法遍历 machines 列表，计算出分片在每个机器上的转码任务的执行时间 ETT 和预期完成时间 EFT，选择拥有最小 EFT 的机器并把任务加入到该机器的任务集列表。

当L和R的所有分片都映射到相关机器后，可以根据公式（3-4）计算出FT的值，然后通过比较每一轮FT的值来找到最佳分配策略。

算法 3.3 LA-MCT 算法

算法：LA-MCT 算法

输入：机器的计算能力集合 P = $(\boldsymbol{p_1}, \boldsymbol{p_2}, \dots, \boldsymbol{p_N})$**，分片集合 V =** $(\boldsymbol{v_1}, \boldsymbol{v_2}, \dots, \boldsymbol{v_N})$**，分片复杂度集合C =** $(\boldsymbol{c_1}, \boldsymbol{c_2}, \dots, \boldsymbol{c_N})$**输出。**

输出：调度策略 $\boldsymbol{\theta_{best}} = \{\boldsymbol{\theta_1}, \boldsymbol{\theta_1}, \dots, \boldsymbol{\theta_M}\}$

1: ∀m ∈ J,$\theta_m = \emptyset$;$C_L = C_{sum}$, minFT = ∞;
2: 根据计算能力降序排序集合P ;
3: 从集合 C 中获得C_{step};
4: min = ∞;
5: **while** $C_L \geq 0$ **do**
6: 调用算法 3.1 生成集合 L 和 R;
7: 分别将 L 和 R 中的分片按复杂度降序排序;
8: **for** L 中的每个分片v_i **do**
9: 调用算法 3.2 获得应将分片v_i进行转码的机器 m;
10: 分配分片v_i到机器 m 上;$\theta_j = \theta_j \cup v_i$;
11: **end for**
12: **for** R 中的每个分片v_i **do**
13: 调用算法 3.2 获得应将分片v_i进行转码的机器 m;
14: 分配分片v_i到机器 m 上;$\theta_j = \theta_j \cup v_i$;
15: **end for**
16: 根据公式（3-4）计算 FT
17: **if** minFT < min **then:**
18: minFT = ft; $\theta_{best} = \theta$;
19: **end if**
20: θ = ∅;
21: **end while**
22: **return** $\boldsymbol{\theta_{best}}$

算法 3.3 描述了 LA-MCT 算法的主要步骤，在第 2 行中，先根据机器的转码速度降序排序集群的转码机器集合P。在第一行中，初始化$C_L = C_{sum}$在第 3 行中，算法查询复杂度递减步长C_{step}，正如之前提到的，取分片集合C的元素中的最小值。在第 5-21 行中，算法迭代每一轮递减的C_L值，在第 6 行，根据一个确定的C_L，返回一对集合L和R，在第 8-11 行中，使用 MCT 算法依次分配集合L中的分片，在第 12-15 行中，同样调用 MCT 算法分配R中的分片。在第 16 行中，使用 FT 计算公式计算这轮分配的完成时间。在第 17-18 行中，算法选取完成时间最小的一轮分配结果。

LA-MCT 算法主要花费在递减C_L值和使用 MCT 负载均衡策略分配分片上，因此 LA-MCT 算法的时间复杂度是$O(\frac{C_{sum}}{C_{step}} \times N \times M)$。

3.5 实验设计与评估

为了分析 LA-MCT 算法的效果，使用 Java 语言实现了这个算法，同时与近期的研究成果 Max-MCT、MLFT、PLTS 作比较。通过大量的模拟视频转码的数字仿真实验推导，本章的算法的预期完成时间FT接近最优完成时间。由于实验使用的时 Java 语言，需要安装 Java 运行环境，在本章中，使用 jdk1.8 作为 Java 虚拟机，使用 Intelij IDEA 14.1.4 作为开发工具，实现了前面提到的几种算法和本章提出的 LA-MCT 任务调度算法。

3.5.1 性能指标

考虑到视频分片的理想分布，即所有分片在本地进行转码，并且负载分布是最佳的，换句话说，转码作业的数据传输开销为 0.因此，理论上，当N个分片在M台机器上进行转码，最理想的作业完成时间是

$$FT_{opt} = \frac{\sum_{i=1}^{N} c_i}{\sum_{j=1}^{M} p_j} + o_m * \frac{N}{M} \tag{3-6}$$

显然，在实际环境中，有些分片需要传送到远程机器进行转码以平衡负载，换句话说，作业的分片传输时间通常大于零。本章的算法的目标是使FT尽可能接近FT_{opt}。 使用因子E_{ft}来评估不同算法的效率，表示为

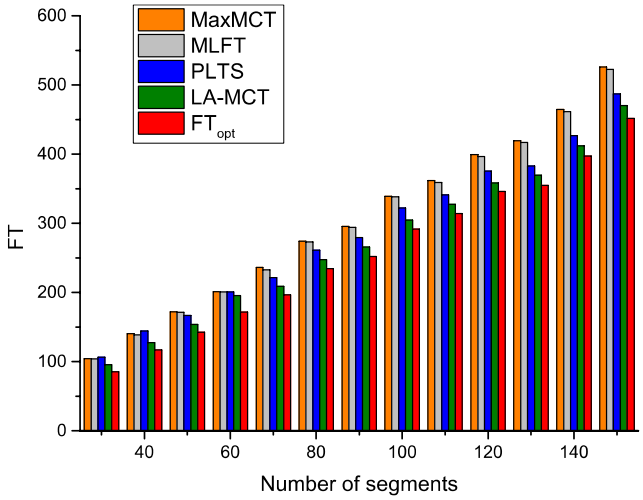
$$E_{ft} = \frac{FT - FT_{opt}}{FT} \times 100\% \tag{3-7}$$

为了验证 LA-MCT 算法的效率，本文使用 Java 程序进行模拟实验来评估不同的调度策略。 分片的复杂度在 40 到 100 之间，每个分片的 GOP 大小在 5 到 20 之间。为了排除偶然性，每个分片的复杂度和最大分段粒度是随机生成的。每台机器的计算能力随机分布在 1.0 到 5.0 之间，这意味着$1.0 \le p_m \le 5.0$。所有

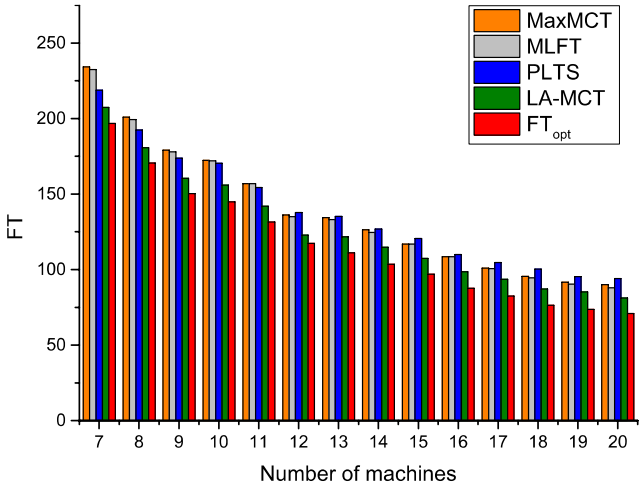
分片在 3 个随机选择的机器上存储 3 个副本。给定一个存储在机器m'中的分片，如果在机器m上执行转码，那么它就是本地任务，则$c_{m',m}=0$。如果m'和m在同一个机架上，则$c_{m',m}=1$，它们之间的网络带宽为 1000 Mbps。否则，$c_{m',m}=3$，带宽为 100 Mbps。任务的启动开销o_m是 5 秒钟。

在本章的实验中，对包括 Max-MCT，MLFT 和 PLTS 在内的视频转码任务调度算法也作为基准测试条件。在实验中，分别改变机器和分片的数量来评估视频转码作业的最终完成时间。对于每种情况，程序执行任务调度算法 10 次，然后计算他们的作业完成时间的平均值。

3.5.2 实验评估



(a) 不同的分片数量



(b) 不同的机器数量

图 3. 2 集群不同参数下的作业完成时间

图 3.2 显示了不同设置下的视频转码的FT，在图 3.2（a）中，横坐标是视频被分割后的分片数量，纵坐标是这批视频分片完成视频转码的最终完成时间。在

图 3.2（b）中，与（a）不同的一点是，横坐标修改为集群的机器数量。表 3.1 显示了该算法与其他三种算法之间的性能E_{ft}的值。在图 4（a）中，本章使用 10 台机器形成一个异构集群，将分片数目从 30 个变为 150 个。在图 4（b）中，分片数量固定为 50，通过不断调整集群中的机器数量，用来比较这些算法的FT。在表 1 中，列出了当机器数量为 10 并且分片数目从 30 到 150 时四种算法的因子E_{ft}。

由图 3.2 可知，LA-MCT 算法在所有情况下均优于其他算法。由于 MaxMCT 和 MLFT 算法都利用了将复杂分片分配给计算能力强大的机器的思路，但它们忽略了分片的数据通信开销，导致算法与理想完成时间FT_{opt}差距较大。当分片数量与机器数量之比过小时，也就是说，集群的计算资源较为充足时，PLTS 算法不如 MaxMCT 和 MLFT 算法，但随着这个比例的增大，换言之，随着输入视频的复杂度越来越大，它明显优于其他两种算法。原因在于 PLTS 算法使用 Min-Min 算法来映射任务，以便在执行数据量小的作业时将任务集中在更高计算能力的机器上。LA-MCT 算法可以确定需要在本地执行多少任务以及远程执行多少任务，从而平衡分片传输时间和任务转码时间，然后实现负载平衡。总之，本章的算法比其他三种算法更接近最优值FT_{opt}，并且优于当前最好的算法 PLTS。

表 3. 1 不同算法在不同分片数量下的E_{ft}值

分片数量	Max-MCT	MLFT	PLTS	LA-MCT
30	21.5	21.4	26.4	12.0
40	20.0	18.6	20.0	9.5
50	19.9	18.8	15.1	7.9
60	18.7	17.9	15.2	6.8
70	19.4	18.8	13.0	6.1
80	18.6	17.3	10.5	4.9
90	17.7	17.3	9.6	5.3
100	16.0	16.0	9.7	4.2
110	18.0	17.1	9.5	4.8
120	17.2	16.3	7.9	4.6
130	17.9	17.2	7.8	4.0
140	17.7	16.5	7.3	3.9
150	17.4	17.0	7.1	4.0

3.6 本章小结

本章详细描述了一个本地感知的异构 Hadoop 集群上的视频转码调度系统，该系统使用 Java 语言编写，核心算法是一个本地感知的任务调度算法 LA-MCT，用于在异构 MapReduce 集群上并行化视频转码，该算法可以平衡分片传输时间和任务转码时间。大量的仿真实验表明，通过改变集群从节点个数和输入视频的分片数量上，与现有算法（Max-MCT、MLFT、PLTS）相比，LA-MCT 使用最小的完成时间完成视频转码作业。

第4章 基于 Alluxio 的 Hadoop 集群视频转码系统

4.1 引言

尽管在第 3 章中，我们提出了一种性能比之前的研究都优越的视频转码任务调度算法，但是，对于一个 Hadoop 集群上的视频转码系统，视频文件还需要存储、分割、合并等操作。在视频进行分割成分片后，输入的视频分片和输出的视频分片在分布式系统内的存储和共享方式，同样也会影响视频转码系统转码一个视频的最终完成时间。因此，从架构方面对视频转码系统进行优化，也能有效提高视频转码速度。

4.2 视频转码系统模型

4.2.1 现存的架构

目前，在一个基于 Hadoop 集群的分布式视频转码系统中，一个一般的转码架构图可能抽象成图 4.1 所示。视频转码工作流程如下：

（1） 视频被 FFmpeg 分片，分片存储到本地磁盘。

（2） 使用 HDFS Client 的 API，将分片、转码参数（如比特率、分辨率、播放宽高比、 音频声道、 音频采样率、音频比特率）上传到 HDFS，供从节点从 HDFS 中下载文件。

（3） Hadoop 作业启动，在 Mapper 下载相应分片，使用 FFmpeg 命令对分片执行转码。分片被存储在本地磁盘，并将分片传输到应用唯一的一个 Reducer。

（4） Reducer 接收 Mapper 发送来的转码成功的分片，并启动 FFmpeg，使用 FFmpeg 命令合并视频文件。

（5） 将合并后的视频上传到 HDFS 以供用户使用。

在这个架构中，视频被 FFmpeg 分割后，为了让 Mapper 能够使用输入分片，需要将视频上传到 HDFS，因为 Mapper 读入的数据需要存储在分布式文件系统中。在这里，一个分片需要一次磁盘读写操作（读入内核缓冲区，写入到 HDFS）。Mapper 在使用 FFmpeg 执行分片转码前，需要从 HDFS 下载分片，一个分片需要一次磁盘读写操作。如果分片数目较多，或者分片文件较大，由于磁盘带宽是 0.2-2GB 每秒，假设一个 Datanode 需要读写 1GB 文件的话，磁盘带宽是 0.2GB 每秒，那么在不考虑网络带宽的情况下，从 Splitter 端的磁盘读入

内存缓冲区需要 5 秒,从 TCP 缓冲区写入到 Datanode 的磁盘需要 5 秒，在这里，磁盘读写将耗时 10 秒。同样地，Mapper 阶段中使用 FFmpeg 将分片转码完成后，需要将文件传输到 HDFS 上，供 Reducer 下载转码成功后的分片，如果转码后的视频文件大小与之前的几乎一样，读写磁盘耗时也是 10 秒。

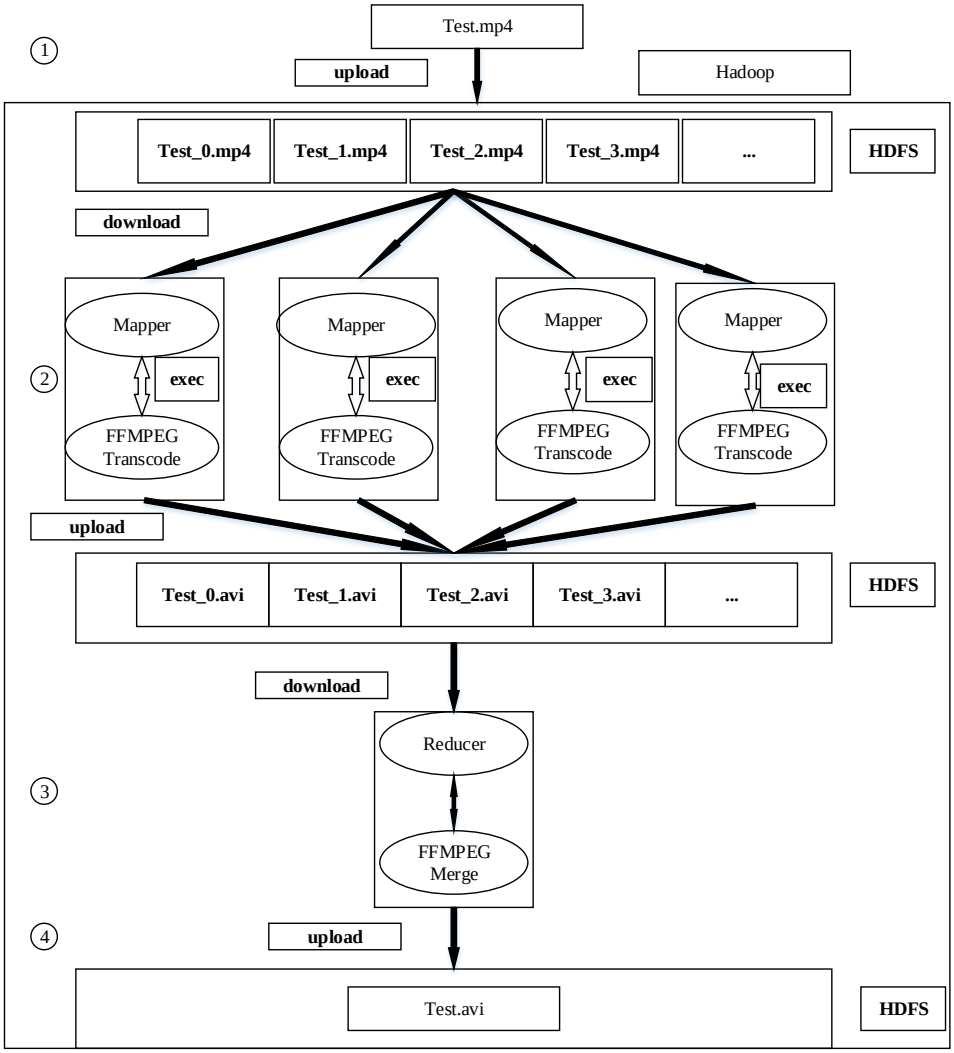


图 4. 1 基于 HDFS 的 Hadoop 视频转码架构

4.2.2 问题定义

基于以上分析，考虑到 4.1 中流程（1）中的上传到分布式文件系统的时间开销，结合第三章提出的公式（3-1）的分片c_i的预期转码时间 ETT，一个分片从被 FFmpeg 将输入视频分割而产生直到被 FFmpeg 转码成功然后被合并的预期总时间是：

$$\mathrm{ETT}_{total}(\mathrm{v_i},\mathrm{m}) = t1_{upload} + ETT(\mathrm{v_i},\mathrm{m}) + t2_{upload} + t_{merger} \qquad (4\text{-}1)$$

$\mathrm{ETT}_{total}(\mathrm{v_i},\mathrm{m})$可以完全描述视频转码系统内一个分片的生命周期，在这个生命周期里，$t1_{upload}$是待转码分片被传输到 HDFS 对应的 Datanode 所使用的时间，

$t2_{upload}$是 Mapper 阶段转码完成的分片上传到 HDFS 的 Datanode 所使用的时间，t_{merger}是视频文件在 Reducer 阶段被 FFmpeg 合并的时间，$t1_{upload}$和$t2_{upload}$的大小取决于分片的文件大小、文件系统的吞吐量（如磁盘带宽）和网络带宽，它们的计算公式相同，因此可以统称为t_{upload}，t_{upload}的计算方法是：

$$t_{upload} = \frac{s_i}{B_{fs}} + \frac{s_i}{B_{m',m}} \tag{4-2}$$

公式（4-2）中，B_{fs}表示文件系统吞吐量，如果文件系统是 HDFS，那么就是磁盘带宽，$B_{m',m}$表示网络带宽。因为视频分片需要被传输到 HDFS，因此分片在转码的生命周期内的所用时间也会受到网络带宽的限制。总的来说，考虑一个视频文件被分割成多个分片，那么一个视频转码 NodeManager 结点上的分片集合从最开始被 FFmpeg 分割到最终转码成功而被 FFmpeg 合并的总时间是：

$$EFT_{total_{\theta_m}} = \sum_{v_i \in \theta_m} t1_{upload} + \sum_{v_i \in \theta_m} \mathrm{ETT}_{total}(v_i, m) + \sum_{v_i \in \theta_m} t2_{upload}. \tag{4-3}$$

一个集群包含多个 NodeManager 结点，用来进行复杂的视频转码运算，一个作业被 MapReduce 使用分而治之的思想，将一个大的视频分割成多个任务，完成视频分片转码，最后在 Reducer 上的 NodeManager 使用 FFmpeg 合并，一个视频在集群内转码，所消耗的总时间是：

$$\mathrm{FT}_{total} = max_{\theta_m \in \theta}\left(EFT_{total_{\theta_m}}\right)$$

$$s.t. \theta = \{\theta_1, \theta_2, \ldots, \theta_M\}$$

$$\cup_{\theta_m \in \theta} \theta_m = J \; \forall \theta_{m1}, \theta_{m2} \in \theta, \theta_{m1} \cap \theta_{m2} = \emptyset \tag{4-4}$$

公式（4-4）表示视频在集群内所消耗的总时间是由集群内所有 NodeManager 节点的最大完成时间决定。

4.2.3 基于 Alluxio 的 Hadoop 视频转码架构

根据公式（4-4）中，如果本文需要优化它，需要根据（4-3）来考虑，在（4-3）中，$EFT_{total_{\theta_m}}$由三部分组成，包括$\sum_{v_i \in \theta_m} t1_{upload}$、$\sum_{v_i \in \theta_m} EFT_{total}(v_i, m)$和$\sum_{v_i \in \theta_m} t2_{upload}$。这三个部分是相互独立的，如果使用第三章提出的 LA-MCT 算法，那么$\sum_{v_i \in \theta_m} EFT_{total}(v_i, m)$可以与公式（3-6）中的$FT_{opt}$较为接近。对于$\sum_{v_i \in \theta_m} t1_{upload}$和$\sum_{v_i \in \theta_m} t2_{upload}$，显然还有优化的空间，它取决于每个分片的上传到文件系统的时间t_{upload}，对于t_{upload}，根据公式（4-2），它的大小由s_i、B_{fs}和$B_{m',m}$决定，由于输入视频的分片和转码成功的视频分片的大小s_i是无法改变的，如果要优化t_{upload}，只能从B_{fs}和$B_{m',m}$入手，针对$B_{m',m}$，只要提高集群的网络带宽即可，这不是本章的讨论重点，本章主要关注的是文件系统的吞吐量B_{fs}。

在现存的架构中，HDFS 作为大数据的基础组件，为 Hadoop 分布式集群提

供了数据的共享和存储服务，因为这个文件系统是将廉价磁盘组成在一起，所以它的吞吐量B_{fs}就是磁盘的带宽。表 4.1 列举了典型的数据中心的节点的介质的容量和读写带宽。由表可知，内存的读写速度是普通磁盘的 50 倍，是固态硬盘的 10-25 倍。另一方面，由于本文研究的 Hadoop 集群是异构的，组成集群的机器不仅在内存和 CPU 这两个计算资源维度上不尽相同，而且在磁盘的带宽上也不一定相同。在现实的生产环境上，集群的机器往往不一样，可能随着集群规模的不断拓展和磁盘性能的不断提高，集群中的机器的磁盘带宽也不相同。如果将一些文件大小较大的文件分配在一个磁盘带宽较小的机器上执行转码，t_{upload}的大小会受低带宽磁盘影响，使得作业的转码时间增加。

表 4. 1 典型的数据中心节点设置

介质	容量	带宽
硬盘驱动器(x12)	12-36TB	0.2-2GB/秒
固态硬盘(x4)	1-4TB	1-4GB/秒
网络	N/A	1.25GB/秒
内存	128-512GB	10-100GB/秒

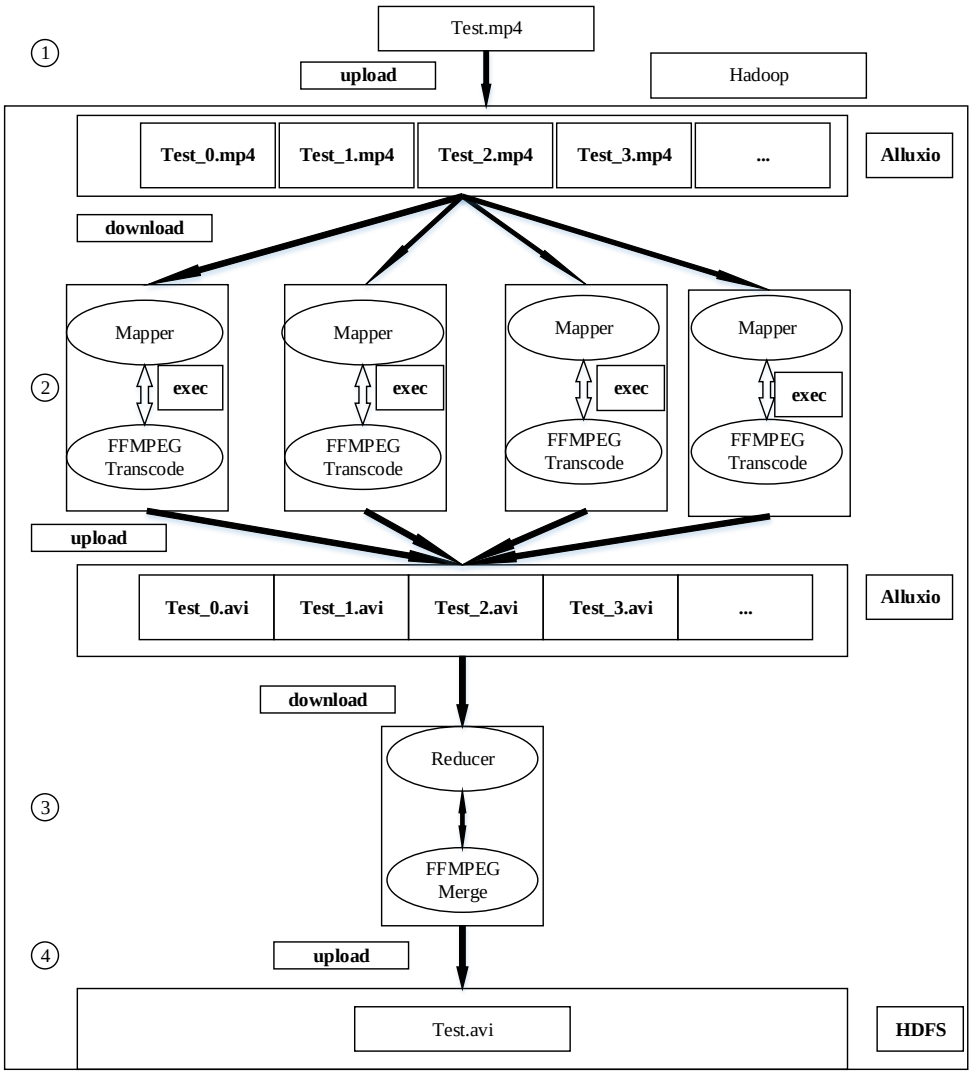


图 4. 2 基于 Alluxio 的 Hadoop 视频转码架构

鉴于此，如果使用阿里巴巴研发的基于内存的分布式文件系统 Alluxio 作为视频转码分片的共享与存储，将增大分布式文件系统的吞吐量B_{fs}的值，从而减小t_{upload}的值。对于一个机器，t_{upload}与s_i正比，与B_{fs}和$B_{m',m}$成反比，在s_i的值一定的情况下，随着B_{fs}的增大，t_{upload}一定会减小，那么，$EFT_{total_{\theta_m}}$肯定会减小，最终，FT_{total}也会随着B_{fs}的增大而减小。显然，如果本文使用内存作为视频分片的存储介质，能够达到视频转码加速的目的。

图 4.2 抽象展示了本研究提出的基于 Alluxio 的 Hadoop 视频转码架构，与基于 HDFS 的视频转码架构不同的是，视频被 FFmpeg 转码后，上传到 Alluxio 文件系统，以供 Mapper 算子的 map()函数下载视频分片，对于一个分片，将数据上传到分布式内存而不是分布式磁盘避免了一次磁盘 Read/Write。在分片被 FFmpeg 转码成功后，同样地，Mapper 会将转码成功后的分片上传到 Alluxio，来替代 HDFS，也避免了一次磁盘读写。通过减少两次分片的上传操作中的磁盘 IO 读写次数，加速了视频转码流程。

4.3 基于 Alluxio 的 Hadoop 视频转码系统设计

在 Hadoop 视频转码系统中，设计的主要思路是，系统环境搭建，定制 InputFormat， Mapper 阶段设计， Reducer 阶段设计。

4.3.1 系统环境搭建

本文的 Hadoop 视频转码系统的环境包括 FFmpeg 环境、Hadoop 环境和 Alluxio 文件系统搭建。FFmpeg 用于视频分割、视频分片转码和转码完成的视频分片合并。Hadoop 作为分布式集群的基础框架，提供了分布式存储系统 HDFS、分布式计算框架 MapReduce、资源管理和作业调度平台 YARN。Alluxio 作为 HDFS 的替代品，为视频转码系统提供内存级别视频分片共享与存储服务。

```
[root@master ffmpeg-2.8]# ll /usr/local/lib
total 257120
-rw-r--r--. 1 root root 155697112 Feb 24 22:55 libavcodec.a
-rw-r--r--. 1 root root   1854536 Feb 24 22:55 libavdevice.a
-rw-r--r--. 1 root root  29708494 Feb 24 22:55 libavfilter.a
-rw-r--r--. 1 root root  48834204 Feb 24 22:55 libavformat.a
-rw-r--r--. 1 root root   2195520 Feb 24 22:55 libavutil.a
-rw-r--r--. 1 root root  10325854 Feb 24 22:45 libfdk-aac.a
-rwxr-xr-x. 1 root root       959 Feb 24 22:45 libfdk-aac.la
lrwxrwxrwx. 1 root root        19 Feb 24 22:45 libfdk-aac.so -> libfdk-aac.so.1.0.0
lrwxrwxrwx. 1 root root        19 Feb 24 22:45 libfdk-aac.so.1 -> libfdk-aac.so.1.0.0
-rwxr-xr-x. 1 root root   5433992 Feb 24 22:45 libfdk-aac.so.1.0.0
-rw-r--r--. 1 root root    529994 Feb 24 22:47 libmp3lame.a
-rwxr-xr-x. 1 root root       943 Feb 24 22:47 libmp3lame.la
lrwxrwxrwx. 1 root root        19 Feb 24 22:47 libmp3lame.so -> libmp3lame.so.0.0.0
lrwxrwxrwx. 1 root root        19 Feb 24 22:47 libmp3lame.so.0 -> libmp3lame.so.0.0.0
-rwxr-xr-x. 1 root root    396672 Feb 24 22:47 libmp3lame.so.0.0.0
-rw-r--r--. 1 root root    671428 Feb 24 22:55 libpostproc.a
-rw-r--r--. 1 root root    449408 Feb 24 22:55 libswresample.a
-rw-r--r--. 1 root root   4806348 Feb 24 22:55 libswscale.a
-rw-r--r--. 1 root root   1269808 Feb 24 22:46 libx264.a
lrwxrwxrwx. 1 root root        14 Feb 24 22:46 libx264.so -> libx264.so.148
-rwxr-xr-x. 1 root root   1075256 Feb 24 22:46 libx264.so.148
drwxr-xr-x. 2 root root      4096 Feb 24 22:55 pkgconfig
```

图 4. 3 安装 FFmpeg 时必要的库文件

在 Linux 环境下，FFmpeg 的安装需要先安装 fdk-aac-0.1.4.tar.gz、ffmpeg-2.8.tar.bz2、lame-3.99.5.tar.gz、x264.tar.gz 等动态链接库。这样相关的动态链接库文件会随着 make 和 make install 出现在/usr/local/lib 文件夹下，如图 4.3 粗体部分所示所示，安装完成后，运行 FFmpeg 命令，出现 FFmpeg 版本信息即表示一切 FFmpeg 相关的库均安装完成。

```
<configuration>
  <property>
        <name> mapreduce.framework.name </name>
        <value> yarn </value>
  </property>
</configuration>
```

图 4. 4 mapreduce.framework.name 属性配置

```
<configuration>
    <!-- Site specific YARN configuration properties -->
    <property>
        <name> yarn.resourcemanager.hostname </name>
        <value> localhost </value>
    </property>
    <property>
        <name> yarn.nodemanager.aux-services </name>
        <value> mapreduce_shuffle </value>
    </property>
</configuration>
```

图 4. 5 resourcemanager 配置

slave 节点的 SSH 免密码登录，Hadoop 使用的版本是较新的 2.7.2。最后，需要在 mapred-site.xml 配置 mapreduce.framework.name 属性，如图 4.4 所示，以及配置 yarn-site.xml 中的 yarn.resourcemanager.hostname 和 yarn.nodemanager.aux-services 属性，如图 4.5 所示。

要使用 Alluxio 分布式文件系统，需要先配置 Alluxio。大部分使用默认设置即可。本章使用的 Alluxio 的版本是 alluxio-1.7.1。在 ${ALLUXIO_HOME}/conf 目录下，根据模板文件创建 alluxio-env.sh 配置文件。

在 alluxio-site.properties 文件中将 alluxio.master.hostname 更新为 Alluxio Master 的机器主机名。搭建完成后，使用./bin/alluxio format 命令格式化文件系统，启动并测试文件系统是否可用。

```
<property>
        <name>fs.alluxio.impl </name>
        <value>alluxio.hadoop.FileSystem </value>
        <description>The  Alluxio FileSystem (Hadoop 1.x and
2.x) </description>
</proerty>
<property>
        <name>fs.AbstractFileSystem.alluxio.impl</name>
       <value>alluxio.hadoop.AlluxioFileSystem</value>
       <description>The Alluxio AbstractFileSystem (Hadoop
2.x)</description>
</property>
```

图 4. 6 core-site.xml 中配置 Alluxio

为了使 MapReduce 应用可以与 Alluxio 进行通信，需要将 Alluxio Client 的 Jar 包包含在 MapReduce 的 classpaths 中。同时，需要在 Hadoop 的 core-site.xml 配置文件中配置 fs.alluxio.impl 和 fs.AbstractFileSystem.alluxio.impl 两个属性，如图 4.6 所示。配置成功后，为了让 MapRedude 应用在 Alluxio 上读写文件，Alluxio 客户端的 Jar 包必须被分发到不同节点的相应的 classpaths 中。可以在使用 hadoop jar...的时候加入-libjars 命令行选项，指定/<PATH_TO_ALLUXIO>/client/alluxio-1.8.0-SNAPSHOT-client.jar 为-libjars 的参数，这条命令会把该 Jar 包放到 Hadoop 的 DistributedCache 中，使所有节点都可以访问到。

4.3.2 视频分割

在视频被 MapReduce 并行实现视频转码前，需要将视频分割成视频分片，视频分片操作在 Hadoop 集群的作业提交 Client 中进行，Client 中安装了 FFmpeg，本章使用 mkvmerge 命令来分割视频，在命令中，指定参数是 split，并使用 size 参数指定每个分片的大小。

在本系统中，视频的分片大小是 HDFS 块的大小，默认为 128MB，如果视频的分片过大， MapReduce 中的 split 跨越多个数据块，使得任务增加网络开销；如果视频分片过小，MapReduce 将会启动过多的 Mapper 算子，但是因为

MapReduce 擅长处理的是数据文件较大的块，在处理数据量较小的块时，任务启动和销毁的开销在任务处理的过程中所占的比例较高，使得作业效率不高。因此，本文指定 size 参数是 64MB，也就是说，每个视频分片的大小是 64MB。

在生成的分块中，需要指定视频分片的文件名，系统使用递增序列命名被分割出来的视频分片，使用编号的文件名，可以在分片被转码完成后，使用 FFmpeg 合并，使得合并的视频流保持正确的顺序。

Client 调用 Alluxio 的 API 上传数据到分布式内存文件系统中，在这里，本章设置 Alluxio 的写数据类型是 MUST_CACHE，因为视频分片只需要被使用一次，在被转码完成后，视频分片就会被应用程序删除，因为视频不需要一直留在内存，更不需要备份到底层文件系统 UFS 中。

4.3.3 VideoInputFormat 设计

尽管在 Alluxio 中使用 MapReduce 来处理数据只需要简单地在配置文件中配置即可，但是对于一个 MapReduce 实现的视频转码系统来说，一个 MapReduce 的 Mapper 算子只处理一个 split。对于逻辑分区 split 中的每一个键值对 key/value，作为 Mapper 的子类的 map()函数的输入。在 Hadoop 的作业 main 函数中，需要使用 Job 的 setInputFormatClass 方法指定应用程序使用的 InputFormat 的子类。因此系统需要对视频分片划分成逻辑分区 split，并将框架提供的 InputFormat 的几个实现如 TextInputFormat、FileInputFormat、KeyValueInputFormat 等均不能满足视频转码的需求。

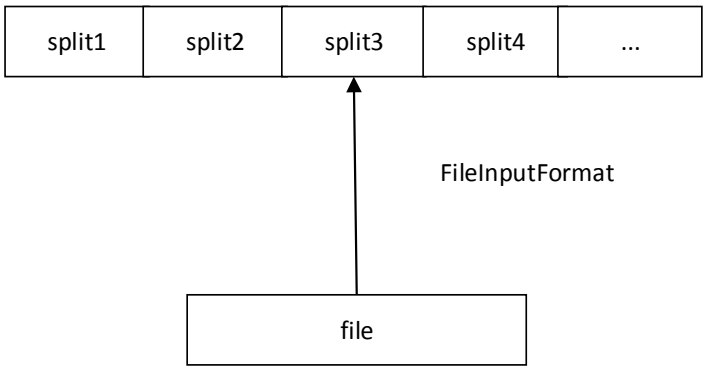


图 4. 7 FileInputFormat 原理

TextInputFormat 继承自 FileInputFormat，FileInputFormat 这个类继承自 InputFormat，FileInputFormat 这个类首先对输入文件进行逻辑上的划分,以 64M 为单位,将原始数据从逻辑上分割成若干个 split，每个 split 切片对应一个 Mapper 任务。对于 FileInputFormat 这个类，需要注意的是：FileInputFormat 这个类只划分比 HDFS 的 block 块大的文件，所以 FileInputFormat 划分的结果是这个文件或者是这个文件中的一部分。如果一个文件的大小比 block 块小,将不会被 FileInputFormat 这个类进行逻辑上的划分，此时每一个小文件都会当做

一个 split 块并分配一个 Mapper 任务,导致效率低下.这也是 Hadoop 处理大文件的效率要比处理很多小文件的效率高的原因。FileInputFormat 没有 RecordReader 的具体实现，因此，不能直接使用。

当 FileInputFormat 这个类将文件切分成 block 块之后，TextInputFormat 这个类随后将每个 split 块中的每行记录解析成一个个的键值对，即<k1,v1>。在 TextInputFormat 中，使用分隔符（默认是换行符）将文件分隔成多行，key 是当前行相对文件首行的偏移量，value 是文件当前行的数据。显然，使用当前分片的路径作为 value 值，就能让 Mapper 下载这个分片的数据到本地磁盘。

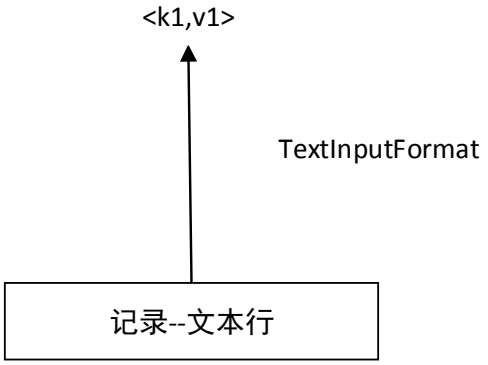


图 4. 8 TextInputFormat 原理

KeyValueInputFormat，这个格式也是把输入文件每一行作为单独的一个记录。然而不同的是 TextInputFormat 把整个文件行当做值数据，KeyValueInputFormat 则是通过 tab 字符来把行拆分为键值对。这在把一个 MapReduce 的作业输出作为下一个作业的输入时显得特别有用，因为默认输出格式正是按 KeyValueInputFormat 格式输出数据。

在本章的系统中，实现了一个适合于视频转码系统的 InputFormat，叫做 VideoInputFormat，它继承自 FileInputFormat，当 mapper 算子读取 split 并调用 createRecordReader 方法生成键值对时，JVM 会创建一个继承自 RecordReader 的实例，叫做 VideoRecordReader。在 VideoRecordReader 类中，应用程序重写了 RecordReader 类中的 initialize 方法。在 Initialize 方法中，需要读取输入分片所在的 Alluxio 文件系统的文件夹路径，然后将所有文件的文件名称保存到一个 ArrayList 中。

为了把视频分片映射成键值对 key/value，程序重写了 RecordReader 类中的 nextKeyValue()方法，这个方法中，把读入数据的顺序作为 key，value 是文件在 Alluxio 中的路径。

4.3.4 Mapper 设计

设计 Mapper 类就是实现 MapReduce 作业的 Mapper 阶段所完成的业务逻辑。在本章的视频转码系统中，由于输入的键值对 key/value 只包含了文件在

Alluxio 文件系统中的路径，因此，map()函数只要根据键值对中的文件路径信息，就可以从分布式内存文件系统中下载视频分片到 NodeManager 节点的本地磁盘。

要在 MapReduce 中使用 FFmpeg 执行视频转码，需要使用 Linux Shell 脚本执行 FFmpeg 命令。在 FFmpeg 命令中，指定输入文件、输出文件和转码目标参数即可进行视频转码。FFmpeg 输出的分片的文件名称，与输入视频一样，都是按顺序命名的，这样可以在使用 FFmpeg 命令合并的时候，保持正确的视频流的顺序。在 Java 中，需要使用 Runtime 类中的 API 执行 Shell 脚本，调用 Runtime 中的 getRuntime()方法获取 Runtime 类的一个实例，然后调用 exec()方法执行目标脚本。exe()方法返回 Process 类的一个实例，根据这个返回的实例的状态信息查看视频分片是否转码成功。

视频被 FFmpeg 完成转码后，新生成的分片将会存储在本地磁盘中，为了给 Reducer 合并，系统把新的分片上传到 Alluxio 文件系统中。同时，为了节约分布式系统的内存空间，需要把内存资源用到其他作业或者 MapReduce 的程序的执行而不是数据存储上，系统主动删除转码成功的输入视频，避免浪费内存资源。

Mapper 的输出依旧要包含文件的路径信息，只不过，这次保存的是输出视频存放在 Alluxio 文件系统的路径，以便于 Reducer 下载，并使用 FFmpeg 命令合并。

4.3.5 Reducer 设计

Reducer 的主要作用是合并 FFmpeg 程序输出的视频分片，在这里，本研究只有一个 Reducer，也就是说，所有的 mapper 输出的键值对都会被传输到这一个 Reducer 节点上。在 MapReduce 应用程序的 main 函数中。使用 JobConf.setNumReduceTasks(1)控制 Reducer 个数为 1。

Reducer 处理 mapper 端传送来的键值对，因为键值对里包含了文件的路径与顺序信息，FFmpeg 使用 concat 命令进行合并。

视频合并后，会保存合并后的视频到 Reducer 的本地磁盘，系统指定 Alluxio 的写类型为 THROUGH，上传视频到 Alluxio 的底层文件系统。在这里，本章使用 HDFS 作为 Alluxio 的底层文件系统，满足了转码后视频数据的高容错性，供用户下载使用。

4.4 系统测试与评估

为了测试基于 Alluxio 的 Hadoop 异构集群下视频转码效果，本研究搭建了一个异构 Hadoop 视频转码集群，该集群共包含 5 个节点，一个主节点，四个从节点。该集群的硬件配置信息如表 4.2 所示。

本章的视频转码系统，运行在这个集群上，Master 作为主节点，既是 YARN 的 ResourceManager，又是 Alluxio 的 Master，同时也是 HDFS 的 NameNode。对于 YARN 的 Container 的配置，系统使用默认的配置文件，也就是说，每个 Container，所占用的内存是 2GB，所使用的 CPU 核数是 1 核。

表 4. 2 异构集群的配置信息

节点编号	内存	CPU	硬盘	节点类型
Master	56GB(尔必达 DDR3 1333MHZ)	因特尔 Xeon（至强） X5670@2.93MHZ	希捷 ST31000528AS(1TB/7200 转/分)	主节点
Slave 1	8GB(三星 DDR4 2400MHZ)	因特尔 Core i7-6700@ 3.40GHZ 四核	西数 WDC WD10EZEX-08WN4A0(1TB/7200 转/分)	从节点
Slave 2	8GB(三星 DDR4 2400MHZ)	因特尔 Core i7-6700@ 3.40GHZ 四核	西数 WDC WD10EZEX-08WN4A0(1TB/7200 转/分)	从节点
Slave 3	8GB(金士顿 DDR3L 1600MHZ)	因特尔 Core i5-2410@ 2.30GHZ 四核	日立 HTS547550A9E384(500GB/5400 转/分)	从节点
Slave 4	4GB(记忆科技 DDR3L 1600MHZ)	因特尔 Core i5-4210H@ 2.90GHZ 双核	西数 WDC WD10SPCX-24HWST1 (1TB/5400 转/分)	从节点
Slave 5	8GB(威刚 DDR4 2400MHZ)	因特尔 Core i7-7700HQ@ 2.80GHZ 四核	Phison SM280128GPTC15T-S114-110(128GB/固态硬盘)	从节点

为了证明本章提出的优化方法的正确性，本章实现了一个参照系统，它就是基于 HDFS 的 Hadoop 视频转码系统，同样的，也运行在这个集群上。

为了比较这两个系统，对于同样一个视频，转码作业被执行 5 次，取 5 次视频转码的平均时间作为本视频的视频转码时间，并使用 10 个不同大小的视频对系统进行测试。视频数据的详细信息如表 4.3 所示。

表 4. 3 视频数据的详细信息

文件名	文件大小
Test1.rmvb	183MB
Test2.rmvb	296MB
Test3.rmvb	481MB
Test4.mp4	619MB
Test5.rmvb	924MB
Test6.mp4	1.4GB
Test7.mp4	2.1GB
Test8.mp4	2.5GB
Test9.mp4	2.9GB
Test10.mkv	4.2GB
Test11.mkv	4.7GB

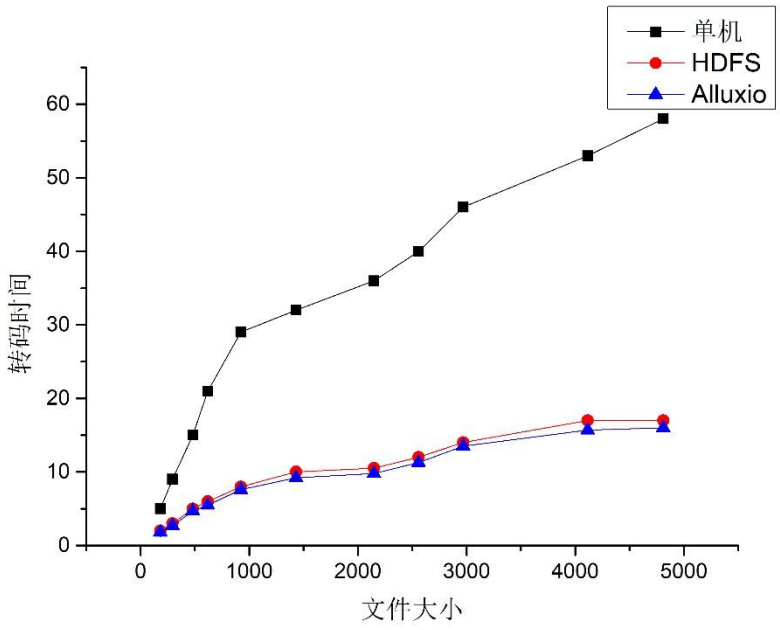


图 4. 9 不同视频文件大小的视频转码时间

图 4.9 使用折线图描述了表 4.3 中的视频在本研究搭建的异构集群上的视频转码时间，黑线代表在单机环境下直接用 FFmpeg 命令在 Slave1 下的视频转码时间，红线表示使用本研究搭建的集群，使用 HDFS 作为视频分片共享的文件

系统的转码时间，蓝线表示使用 Alluxio 作为分布式文件系统，来替代 HDFS 在搭建的异构集群下的视频转码时间。由图可知，使用 HDFS 作为文件系统，相比单机转码，一个视频被分散在多个机器下进行转码，转码时间缩短了约 70%。而使用本章提出的 Alluxio 作为文件系统的架构，随着文件大小的增加，相比于 HDFS，转码时间缩短了 5%以上。这个时间差距主要是因为作业执行时的 IO 读写开销导致的。

4.5 小结

本章详细介绍了 Hadoop 异构集群的系统架构，并对视频转码系统进行了抽象和建模，为了优化这个模型，本章使用 Alluxio 分布式内存文件系统来实现节点中的 Mapper 和 Reducer 视频分片的共享。为了验证本章的优化方法，使用了一个小型异构集群来运行本研究的应用程序，并与基于 HDFS 分布式文件系统作比较。大量的实验表明，本章设计的系统在不损失视频转码质量的情况下，为视频转码带来了 5%的速度提升。

结论与展望

1 工作总结

视频播放终端的异构性使得它们对视频的参数（分辨率、帧率等）需求不同，而且在动态的网络环境下，同一个视频终端对视频的质量要求也不一样。视频转码能够解决这个终端和网络的异构性问题，但是视频转码是一个计算相当复杂的过程，这个过程需要计算机的 CPU 拥有强大的计算能力。为了给视频转码提供足够多的计算资源，本文使用 Hadoop MapReduce 来利用集群的计算资源实现分布式并行转码，它让人们可以轻松地利用集群不断扩展的计算资源。在集群扩展的过程下，集群中的计算机的计算能力（CPU、内存）往往是异构的。为了加速异构集群上的并行转码，本文主要从任务调度和系统架构两方面来进行优化。本文的具体工作如下：

（1）在第二章，对视频转码加速的研究现状进行了分析与比较，共有 3 种加速方法，它们是视频转码算法加速、GPU 硬件加速和分布式并行加速，同时，得出分布式并行加速是一个既能避免图像失真又能提供高可扩展性的方法的结论。

（2）在第三章，为了对 Hadoop 异构集群的视频转码进行加速，在第 3 章，本研究在任务调度的层面进行了优化，并对视频转码任务调度进行了建模，这个模型是一个 NP 难问题。获得 NP 难问题的最优解需要非常高的时间复杂度，当数据量较大时，我们只能寻找 NP 难问题的近视最优解。因此，本文提出了一种时间复杂度较低的启发式算法 LA-MCT，这个算法平衡了视频分片的转码时间和分片的网络传输开销。大量的仿真实验表明，相比于现存的 Max-MCT、MLFT 和 PLTS 算法，LA-MCT 有着更短的转码完成时间。

（3）在第 4 章，从整个视频转码系统的方面考虑，在视频转码架构的层面上，对整个系统进行建模，考虑到视频分片数据的共享与存储，使用 Alluxio 内存分布式文件系统替代现存的以 HDFS 作为分布式系统视频分片的共享介质，减小了视频分片的磁盘读写开销。在设计系统时，系统使用 FFmpeg 命令分割、转码和合并视频，并设计了一个继承自 FileInputFormat 的子类 VideoInpFormat，它能将一个视频分片划分为一个逻辑分区，重写了 FileInputFormat 中的 RecordReader 方法，用于为 Mapper 算子提供 Key/Value 键值对。本研究搭建了一个小型 Hadoop 集群，使用真实的视频数据在本研究实现的系统上进行视频转码，相比于现存的以 HDFS 作为分布式文件系统的视频转码系统，转码速度提高了 5%。

2 研究展望

异构分布式 Hadoop 集群的视频转码的加速是一个极具挑战性的问题，它不仅要考虑减视频转码的特性，还要有效利用异构集群的计算资源。为了使异构集群的资源最大化利用，本文提出了 LA-MCT 启发式算法，本文使用仿真实验和模拟数据验证了算法的有效性，但是这毕竟不是真实的数据，在真实的 Hadoop MapReduce 环境下，其优化效果是未知的。因此，下一步，需要将本研究的 LA-MCT 算法应用到 MapReduce 计算框架中，去验证算法的效果。另一方面，LA-MCT 算法的任务调度进行建模的时候，虽然考虑了任务的启动开销，但是模型并没有把任务的销毁开销考虑在内，因为 Hadoop YARN 中的任务的销毁，需要 ApplicationMaster 与 NodeManager 进行通信，ApplicationMaster 检测到任务的执行进度是 100%之后，会通过 RPC 协议向 NodeManager 发送任务销毁的命令，这需要消耗一定的时间。另外，在 MapReduce 中，任务 Task 是会推测式执行的，对于一个任务，有可能在集群中存在多个 container 运行实例，NodeManager 会把这些实例的进度报告给 ApplicationMaster，当 ApplicationMaster 只要发现这些任务实例的执行进度有一个达到 100%，就会通知 NodeManager 销毁这个进度达到 100%的任务，并杀死其他未完成的任务，显然，一个任务存在多个运行实例会占用集群的计算资源，而本文并没有将任务的推测式执行考虑在内。最后，一个任务的执行是有可能出现错误的，比如机器宕机、网络环境异常等情况。如果任务执行失败， YARN 将会重新挑选一台机器去执行这个失败的任务，使得重新执行这个失败的任务的机器的所有任务的最终完成时间推迟，导致任务调度模型不够精确，最终导致转码作业的最终完成时间延长。因此，没有考虑任务的容错机制也是本算法的缺点之一。

在第四章中，本文提出的基于 Alluxio 的 Hadoop 视频转码系统，使用 Alluxio 来替代 HDFS 实现视频分片在分布式系统中的共享，虽然论文通过实验验证了提出的有效性，但是因为视频转码还有多种方式，如更改码率、帧率和空间分辨率等，第四章的实验只是对输入视频更改容器格式(更改为 AVI 格式)进行转码，这些实验不能证明对所有的视频转码方式都有效果。因此，下一步的工作是，对系统进行多种视频转码方式进行测试，以验证提出的架构的实用性。另外，对于一个小型集群，内存资源并不充足，如果使用本文提出的 Alluxio 架构，在应对文件较大的输入视频时或者同时运行多个 Hadoop 作业的情况时，如果把一部分内存资源用于数据的存储，那么相应的，YARN 的 Container 所能利用的内存资源就有可能不够，因此，本文提出的基于 Alluxio 的 Hadoop 异构集群转码架构，在内存资源足够多的情况下才会有更好的转码速度。因此，下一步可以设计相应的算法，应对内存资源相对不充足的场景。

参考文献

[1] 第 41 次《中国互联网络发展状况统计报告》. http://www.cnnic.net.cn/hlwfzyj/hlwxzbg/hlwtjbg/201803/P020180305409870339136.pdf, 2018

[2] 2017 年 1月 手机分析报告. http://tip.umeng.com/uploads/Mobile%20analytics%20report%20January%202017/mobile_analytics_report_january_2017.pdf, 2017

[3] Big Buck Bunny. A short computer animated film by the blender institute.www.bigbuckbunny.org, April 2008. Accessed: Jan 20, 2012

[4] D. M. Barbosa, J. P. Kitajima, and W. Weira. 1999. Parallelizing MPEG video encoding using multiprocessors. In Xii Brazilian Symposium on Computer Graphics and Image Processing. 215– 222

[5] Yen-Kuang Chen, Eric Q. Li, Xiaosong Zhou, and Steven Ge. 2006. Implementation of H.264 encoder and decoder on personal computers. Journal of Visual Communication and Image Representation 17, 2, 509–532. https://doi.org/10.1016/j.jvcir.2005.05.004

[6] Bongsoo Jung and Byeungwoo Jeon. 2008. Adaptive slice-level parallelism for H.264/AVC encoding using pre macroblock mode selection. Journal of Visual Communication and Image Representation 19, 2008: 558–572

[7] Tian Z, Xue J, Hu W, et al. High performance cluster-based transcoder. In: Proc of International Conference on Computer Application and System Modeling. IEEE, 2010:V2-48-V2-52

[8] Je rey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. ACM, 2008:107–113

[9] Hadoop 官网. http://hadoop.apache.org/, 2018

[10] Ghemawat S, Gobioff H, Leung S T. The Google file system. Acm Sigops Operating Systems Review, 2003, 37(5):29-43

[11] Chang F, Dean J, Ghemawat S, et al. Bigtable: a distributed storage system for structured data. Usenix Symposium on Operating Systems Design and Implementation. 2006:15-15

[12] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. Usenix Conference on Networked Systems Design and Implementation. USENIX Association, 2012:2-2

[13] Spark 官网. http://spark.apache.org/, 2018

[14] Ignite 官网. https://ignite.apache.org/, 2018

[15] Power R, Li J. Piccolo: building fast, distributed programs with partitioned tables. Usenix Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, Bc, Canada, Proceedings. DBLP, 2010:293-306

[16] Li, Haoyuan, Ghodsi, et al. Tachyon: Reliable, Memory Speed Storage for Cluster Computing Frameworks. 2014:1-15

[17] Metoevi I, Coulombe S. Efficient MPEG-4 to H.264 transcoding exploiting MPEG-4 block modes, motion vectors, and residuals. International Symposium on Communications and Information Technology. IEEE, 2009:224-229

[18] Bortot L, Chimienti A, Lucenteforte M, et al. An innovative low complexity MPEG-2 video transcoder operating in the DCT domain. Consumer Electronics, 2000. ICCE. 2000 Digest of Technical Papers. International Conference on. IEEE, 2000:312-313

[19] Domanski M, Marek J, aw. Fine grain scalability of bitrate using AVC/H.264 bitstream truncation. Picture Coding Symposium. IEEE, 2009:1-4

[20] Ho C, Au O C, Chan S H, et al. Motion Estimation for H.264/AVC using Programmable Graphics Hardware. In: Proc of international conference on multimedia and expo, 2006, 2049-2052

[21] Feng Lao, Xinggong Zhang, and Zongming Guo. Parallelizing video transcoding using MapReduce-based cloud computing. In IEEE International Symposium on Circuits and Systems, 2012: 2905–2908

[22] Song Lin, Xinfeng Zhang, Qin Yu, et al. Parallelizing video transcoding with load balancing on cloud computing. In IEEE International Symposium on Circuits and Systems, 2013: 2864–2867

[23] Hui Zhao, Qinghua Zheng, Weizhan Zhang, and Jing Wang. Prediction-based and Locality-aware Task Scheduling for Parallelizing Video

Transcoding over Heterogeneous MapReduce Cluster. IEEE Transactions on Circuits and Systems for Video Technology PP, 99, 2016:1–1

[24] Braunt T D, Siegel H J, Beck N, et al. A Comparison Study of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Ileterogeneous Distributed Computing Systems. 2000, 61(6):810–837.

[25] Kesavaraja D, Shenbagavalli A. Hadoop scalable Video Transcoding technique in cloud environment. In: Proc of IEEE International Conference on Intelligent Systems and Control. IEEE, 2015:1-6

[26] 张浩,孙淑霞 . Hadoop 云计算平台在视频转码上的应用.电脑与电信,2011(12):36-37+40

[27] 杨帆,沈奇威 . 分布式系统 Hadoop 平台的视频转码.计算机系统应用,2011,20(11):80-85

[28] 高东海. 基于 Hadoop 的离线视频数据处理技术研究与应用. 北京：北京邮电大学,2014,3-4

[29] FFmpeg. http://ffmpeg.org/, 2018

[30] Alluxio. https://www.alluxio.org/, 2018

[31] Lin J C, Huang X L, Yang Z X. Video Transcoding Techniques. Journal of Communication University of China, 2006

[32] Sun H, Kwok W, Zdepski J W. Architectures for MPEG compressed bitstream scaling . Circuits & Systems for Video Technology IEEE Transactions on, 1995, 6(2):191-199

[33] Eleftheriadis A, Anastassiou D. Constrained and general dynamic rate shaping of compressed digital video. In: Proc of International Conference on Image Processing, 1995. Proceedings. IEEE, 1995:396-399

[34] Youn J, Sun M T, Lin C W. Motion vector refinement for high-performance transcoding. IEEE Transactions on Multimedia, 1999, 1(1):30-40

[35] Youn J, Sun M T, Lin C W. Motion estimation for high performance transcoding. IEEE Transactions on Consumer Electronics, 1998, 44(3):649-658

[36] Assuncao P A A, Ghanbari M. Post-processing of MPEG2 coded video for transmission at lower bit rates. In: Proc of IEEE International Conference on Acoustics, Speech, and Signal Processing, 1996. Icassp-96. Conference Proceedings. IEEE, 1996:1998-2001 vol. 4

[37] Assuncao P A A, Ghanbari M. A frequency-domain video transcoder for dynamic bit-rate reduction of MPEG-2 bit streams . IEEE Transactions on Circuits & Systems for Video Technology, 1998, 8(8):953-967

[38] Wang J, Yu S. Dynamic rate scaling of coded digital video for IVOD applications. Consumer Electronics IEEE Transactions on, 1998, 44(3):743-749

[39] Chang S F, Messerschmitt D G. Manipulation and compositing of MC-DCT compressed video. Readings in multimedia computing and networking. Morgan Kaufmann Publishers Inc. 2001:188-198

[40] Sun H, Vetro A, Bao J, et al. A new approach for memory efficient ATV decoding . IEEE Transactions on Consumer Electronics, 1997, 43(3):517-525

[41] Safranek R J, Garg R. Methods for matching compressed video to ATM networks. In: Proc of International Conference on Image Processing. IEEE Computer Society, 1995:13

[42] Shen G, Zeng B, Zhang Y Q, et al. Transcoder with arbitrarily resizing capability. IEEE International Symposium on Circuits and Systems. IEEE, 2001:25-28 vol. 5

[43] Shanableh T, Ghanbari M. Heterogeneous video transcoding to lower spatio. Multimedia IEEE Transactions on, 2000, 2(2):101--110

[44] Xin J, Sun M T, Chun K. Motion Re-estimation for MPEG-2 to MPEG-4 Simple Profile Transcoding. Proc Int, 2002

[45] Bjork N, Christopoulos C. Transcoder architectures for video coding. In: Proc of IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE, 2002:2813-2816

[46] Wee S J, Apostolopoulos J G, Feamster N. Field-to-frame transcoding with spatial and temporal downsampling. In: Proc of International Conference on Image Processing, 1999. ICIP 99. Proceedings. IEEE, 2002:271-275

[47] Hwang J N, Wu T D, Lin C W. Dynamic frame-skipping in video transcoding. Multimedia Signal Processing, 1998 IEEE Second Workshop on. IEEE, 1998:616-621

[48] Xin J, Sun M T, Chun K. Motion Re-estimation for MPEG-2 to MPEG-4 Simple Profile Transcoding. Proc Int, 2002

[49] Kung M C, Au O C, Wong P H W, et al. Block based parallel motion estimation using programmable graphics hardware. In: Proc of International Conference on Audio, Language and Image Processing. IEEE, 2008:599-603

[50] Cheung N M, Au O C, Kung M C, et al. Highly Parallel Rate-Distortion Optimized Intra-Mode Decision on Multicore Graphics Processors. IEEE Transactions on Circuits & Systems for Video Technology, 2009, 19(11):1692-1703

[51] Lee C Y, Lin Y C, Wu C L, et al. Multi-Pass and Frame Parallel Algorithms of Motion Estimation in H.264/AVC for Generic GPU. In: Proc of IEEE International Conference on Multimedia and Expo. IEEE, 2007:1603-1606

[52] Rodriguez R, Martínez J L, Fernández-Escribano G, et al. Accelerating H.264 inter prediction in a GPU by using CUDA. In: Proc of Digest of Technical Papers International Conference on Consumer Electronics. IEEE, 2010:463-464

[53] Huang Y L, Shen Y C, Wu J L. Scalable computation for spatially scalable video coding using NVIDIA CUDA and multi-core CPU. In: Proc of International Conference on Multimedia 2009, Vancouver, British Columbia, Canada, October. DBLP, 2009:361-370

[54] An D, Tong X, Zhu B, et al. A novel fast DCT coefficient scan architecture. Picture Coding Symposium. IEEE, 2009:1-4

[55] Huang F M, Lei S F. High performance and low cost entropy encoder for H.264 AVC baseline entropy coding. In: Proc of International Conference on Communications, Circuits and Systems. IEEE, 2008:675-678

[56] Sambe Y, Watanabe S, Yu D, et al. High-Speed Distributed Video Transcoding for Multiple Rates and Formats. Ieice Transactions on Information & Systems, 2005, 88-D(8):1923-1931

[57] Garcia A, Kalva H, Furht B. A study of transcoding on cloud environments for video content delivery. ACM Multimedia Workshop on Mobile Cloud Media Computing. ACM, 2010:13-18

[58] Tian Z, Xue J, Hu W, et al. High performance cluster-based transcoder. In: Proc of International Conference on Computer Application and System Modeling. IEEE, 2010:V2-48-V2-52

[59] Jokhio F, Deneke T, Lafond S, et al. Bit Rate Reduction Video Transcoding with Distributed Computing. In: Proc of Euromicro International Conference on Parallel, Distributed and Network-Based Processing. IEEE Computer Society, 2012:206-212

[60] Huang Z, Mei C, Li L E, et al. CloudStream: Delivering high-quality streaming videos through a cloud-based SVC proxy. INFOCOM, 2011 Proceedings IEEE. IEEE, 2011:201-205

[61] 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

[62] Li Z, Huang Y, Liu G, et al. Cloud transcoder:bridging the format and resolution gap between internet videos and mobile devices. International Workshop on Network and Operating System Support for Digital Audio and Video. 2012:33-38

[63] Huang Z, Mei C, Li L E, et al. CloudStream: Delivering high-quality streaming videos through a cloud-based SVC proxy. INFOCOM, 2011 Proceedings IEEE. IEEE, 2011:201-205

[64] 杨竞. Hadoop 平台下的视频转码与优化. 长沙:湖南大学，2016,3-4

[65] Garey M R, Johnson D S, Sethi R. The Complexity of Flowshop and Jobshop Scheduling. INFORMS, 1976

[66] Braunt T D, Siegel H J, Beck N, et al. A Comparison Study of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Ileterogeneous Distributed Computing Systems. 2000, 61(6):810–837

[67] Jokhio F, Ashraf A, Lafond S, et al. Prediction-Based Dynamic Resource Allocation for Video Transcoding in Cloud Computing. In: Proc of Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. IEEE Computer Society, 2013:254-261

[68] Krishnappa D K, Zink M, Sitaraman R K. Optimizing the video transcoding workflow in content delivery networks . 2015, 34(4):37-48

[69] Shanableh T, Peixoto E, Izquierdo E. MPEG-2 to HEVC Video Transcoding with Content-Based Modeling. IEEE Transactions on Circuits & Systems for Video Technology, 2013, 23(7):1191-1196

[70] Ashraf A, Jokhio F, Deneke T, et al. Stream-Based Admission Control and Scheduling for Video Transcoding in Cloud Computing. Ieee/acm

International Symposium on Cluster, Cloud and Grid Computing. IEEE, 2013:482-489

[71] Cheng R, Wu W, Lou Y, et al. A Cloud-Based Transcoding Framework for Real-Time Mobile Video Conferencing System. In: Proc of IEEE International Conference on Mobile Cloud Computing, Services, and Engineering. IEEE Computer Society, 2014:236-245

[72] Ma H, Seo B, Zimmermann R. Dynamic scheduling on video transcoding for MPEG DASH in the cloud environment. In: Proc of ACM Multimedia Systems Conference. ACM, 2014:283-294

[73] Deneke T, Haile H, Lafond S, et al. Video transcoding time prediction for proactive load balancing. In: Proc of IEEE International Conference on Multimedia and Expo. IEEE Computer Society, 2014:1-6

[74] Deneke T, Lafond S, Lilius J. Analysis and Transcoding Time Prediction of Online Videos. IEEE International Symposium on Multimedia. IEEE, 2016:319-322

致　谢

时光飞逝，岁月如梭，在湖南大学读书一直是我从小的梦想，想不到今日却能以硕士研究生身份住在岳麓山脚下。每当走在岳麓书院的羊肠小道上，注视着岳麓书院楹联“惟楚有才，于斯为盛”的时候，我总能想到朱熹的理学。三年来，湖南大学校歌歌词“麓山巍巍，湘水泱泱。宏开学府，济济沧沧。承朱张之绪，取欧美之长，华与实兮并茂，兰与芷兮齐芳。楚材蔚起，奋志安攘。振我民族，扬我国光”不断萦绕在我的耳边，这是第一任校长胡庶华给我们湖大学子立的座右铭，我们要传承中华文化，学习欧美的长处，振我民族，扬我国光。伟大革命导师毛主席的题写的“实事求是，敢为人先”的校训警醒我，要立足现实，夯实基础，追求真理，脚踏实地，着眼于未来和长远，敢于竞争、敢于创新。岳麓山的革命先烈如再造共和的松坡将军蔡锷和中华民国的领袖黄公克诚等人的英魂鼓励我们湖湘学子要牢记我们的使命，为中华民族伟大复兴和社会主义现代化事业出一份力。

感谢我的研究生导师李仁发教授，他是一个非常有理想和学术追求的老师尽管已经年过六旬，但是心中却有很多年轻人身上都没有的拼劲，李老师是我的榜样，这三年来，对我悉心教导，对不同的学生，采用不一样的教育方法，并且处处为学生着想，真的非常感激他。

谢谢李蕊老师和刘彦老师在开题和中期检查对我的指导，提供了很多具有实际指导意义的建议，让我少走了很多弯路。特别感谢湖南双菱电子有限公司公司的周总和徐部长，是公司提供了我在研二一年的实习机会，同时课题的问题来源离不开公司的实际需求，公司也提供了研究初期的实验环境，让我更加熟练地掌握了 Spark 和 Hadoop 平台。

感谢黄晶师兄耐心地指导我的实验和小论文。谢谢课题组的袁娜、宋金林和屠晓寒等同学，在课题的研究过程中，我们朝夕相处、团结协作、互相勉励、共同进步，度过了美好快乐的研究生涯。

最后，谨向审阅本论文及答辩组的评委老师们致以深深的敬意和诚挚的谢意。由于本人研究水平有限，文中难免有不足之处，恳请各位老师批评、指正。

附录 A 攻读硕士学位期间发表的学术论文

论文发表：

[1] 邓湘军，黄晶，李仁发. A Locality-aware Task Scheduling Algorithm for Video Transcoding over Heterogenous MapReduce Cluster. ICMSSP, 2018.（已录用）

软件著作权：

[2] 邓湘军. 本地感知的异构 MapReduce 集群上的视频转码调度系统 v1.0: 中国, No. 02467622. 2018-04-10

附录 B 攻读硕士学位期间所参与的项目

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