

分 类 号：TP393

单位代码：10183

研究生学号：2013532013

密 级：公 开



吉 林 大 学
硕 士 学 位 论 文
(学 术 学 位)

基于 Hadoop 的视频转码优化的研究

Study of Hadoop-based Video Transcoding Optimization

作者姓名：宋 扬

专 业：计算机系统结构

研究方向：云计算

指导教师：于秀峰 副教授

培养单位：计算机科学与技术学院

2016 年 4 月

未经本论文作者的书面授权，依法收存和保管本论文书面版本、电子版本的任何单位和个人，均不得对本论文的全部或部分内容进行任何形式的复制、修改、发行、出租、改编等有碍作者著作权的商业性使用（但纯学术性使用不在此限）。否则，应承担侵权的法律责任。

吉林大学博士(或硕士)学位论文原创性声明

本人郑重声明：所呈交学位论文，是本人在指导教师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。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结果由本人承担。

学位论文作者签名： 

日期： 2016年5月24日

基于 Hadoop 的视频转码优化的研究
Study of Hadoop-based Video Transcoding
Optimization

作者姓名：宋扬

专业名称：计算机系统结构

指导教师：于秀峰 副教授

学位类别：工学硕士

答辩日期： 年 月 日

摘要

基于 Hadoop 的视频转码优化的研究

在“互联网+”的大潮推动下，人们对视频的转码速度和质量等多方面需求日益增长。在 2015 年度的网络流量年报总结中可以看出，有关视频的流量成为了人们生活中消耗最大，占比例最大的一种。目前无论是工业界还是学术界，如何解决视频转码处理高效、高质量、高可用的“三高”问题，成为了研究的重点和核心。为此本文开展了视频转码的优化研究。

本文采用云计算技术对视频进行转码，使用云环境达到并行计算，完成高效和高可用，力求转码前后质量不会发生巨大偏差。本文设计了一个视频转码云平台。该平台采用典型的三层结构：IaaS 选用的是 Amazon 基础设施云；PaaS 选用的是 Hadoop；SaaS 中运行的是高性能的视频转码应用。本文提出了采用将 FFMPEG 和 MapReduce 技术融合实现视频转码优化策略。

文中创新性地提出了 S_MapReduce 结构并在 S_MapReduce 中引入虚拟 IP 机制来提高可扩展性和改变了 MapReduce 架构的不可跨域等多个问题；设计了心跳包流程并改进了负载均衡模块，使整个架构能高效、稳定的进行任务处理；对 MapReduce 的工作流和数据流进行修改和优化，提高计算架构的效率。

视频转码优化是基于云平台，将 FFMPEG 和 S_MapReduce 融合，实现分布式转码。主要工作包括：（1）根据固定场景和多个限定条件，将累赘的部分删除掉，为 FFMPEG 进行了合理化的瘦身。完成了针对于 S_MapReduce 的 NALU 设计，提出了防竞争机制，避免出现竞争和顶替错误（2）设计了 APU 格式来配合 MapReduce 进行分块处理，并优化了解码流程，提高了效率。（3）设计了二次校验算法并保证了数据的完整性和有序性。

实验环境使用了 19 个节点来建设转码云，在实验数据方面采用多层次、多维度，这样保证了实验的全面性和严谨性。本文中实验主要是对视频转码效果和视频质量进行测试。通过实验可以得知在效率方面比未修改的 FFMPEG 方法有显著提高，而随着数据量的增加本文方案效率的提升也逐渐加快。对比多点和单点架构可以得出结论，当单个文件大小是 10GB 时本文方案的效率提升可以达到对照方案的 16.4 倍。

关键词：

视频转码，虚拟 IP，FFMPEG，Hadoop，MapReduce

Abstract

Study of Hadoop-based Video Transcoding Optimization

Under circumstances of “Internet Plus”, people have the growing demand for video transcoding which with higher speed and quality than before. Based on the report of 2015 Annual Summary for Network Traffic, video created the largest proportion of the total network traffic consumption. For now, both industry and academic world are concentrating on how to transcode video efficiently with high quality and high-availability. This report focused on the research for optimizing video transcoding process.

To obtain an efficient process and an accurate result, the experiment used cloud computing method to transcode video. This method could enable the cloud to do the parallel computing. The report also designed a cloud platform of video transcoding. This platform used a three-level structure: IaaS, Paas and SaaS. We chose Amazon for IaaS level, Hadoop for PaaS level and a high performance video transcoding application for SaaS level. The report also introduced a video transcoding optimization strategy, which combined the FMPEG and the MapReduce technic.

This report mentioned an architecture called S_MapReduce. Introducing the virtual IP mechanism into S_MapReduce could improve many performances such as increasing scalability and making the MapReduce architecture able to cross-domain. A heart beat process was also designed in the S_MapReduce architecture, which ameliorated the load –balance module and made the whole architecture more stable than before. This process enhance the computing architecture performance by optimizing the MapReduce’s work-flow and data-flow process.

The video transcoding is based on the cloud platform, which combines the FFMPEG and S_MapReduce to implement the distributed transcoding. The main process include :(1) According to the fixed scene and several limited conditions,

dragged part would be deleted to weigh properly for FFMPEG. The design of NALU has been completed and anti-competition mechanism has been put forward to avoid competition and displacement error. (2) APU format has been designed to match MapReduce to deal with in block, optimizing the decoding process and improving the efficiency (3) Re-check algorithm has been designed to ensure the integrity and order of the data.

The whole experimental environment used 19 spot to build the transcoding cloud. The data in this experiment was chosen by multi-level and multi-dimension to insure that this test was universal and practical. Experiments were testing on the video transcoding performance and the after transcoding video quality. The result showed that under the cloud platform of video transcoding, the video transcoding became more efficient with the data size increasing. By using multi-spot method, when the single file size is 10GB, the efficiency promote to 16.4 times than the single-spot method.

Keywords:

Video Transcoding, virtual IP, FFMPEG, Hadoop, MapReduce

目 录

第一章 引言.....	1
1.1 项目背景及意义.....	1
1.2 国内外研究现状.....	1
1.2.1 国内现状.....	1
1.2.2 国外现状.....	2
1.3 本文主要工作.....	3
1.4 论文结构.....	3
第二章 相关技术基础.....	4
2.1 云计算.....	4
2.2 Hadoop 平台.....	4
2.3 MapReduce 计算架构.....	5
2.4 视频转码技术.....	7
第三章 转码云平台架构设计.....	9
3.1 视频转码云体系结构.....	9
3.2 技术选择.....	10
3.3 转码云 workflow 设计.....	11
第四章 S_MapReduce 架构.....	13
4.1 S_MapReduce 架构设计.....	13
4.2 S_MapReduce 心跳包设计.....	14
4.3 S_MapReduce workflow 和数据流设计.....	16
4.4 负载均衡模块设计.....	19
第五章 视频转码的优化设计.....	21
5.1 FFmpeg 解码.....	21
5.1.1 NAL 单元设计.....	21
5.1.2 解码过程.....	24
5.1.3 二次校验算法设计.....	25
5.2 S_MapReduce 和 FFmpeg 融合.....	26
5.2.1 S_MapReduce 和 FFmpeg 框架整合.....	26
5.2.2 S_MapReduce 分块策略设计.....	28
5.2.3 APU 原子处理单元设计.....	28
5.2.4 分割点设计.....	29
5.2.5 时间戳的单调性.....	31
5.2.6 转码 S_MapReduce 算法设计.....	31
第六章 实验与分析.....	34
6.1 实验环境.....	34
6.2 实验数据准备.....	35
6.3 实验与分析.....	36
第七章 总结与展望.....	42
7.1 总结.....	42
7.2 展望.....	42
参考文献.....	44
致谢.....	48

作者简介及在校期间所取得的科研成果.....49

第一章 引言

1.1 项目背景及意义

在数据量爆增、人们需求提高、业务逻辑不断复杂的背景下，人们对数据处理速度和数据处理结果的要求也逐步提升，云计算成为了人们生活中必备元素之一。云计算^[1]概念起始于 2008 年，它是将网格计算^[2,3]、分布式计算^[4]、并行计算^[5]和虚拟化^[6,7]等技术进行融合。云计算通过融合多种技术来提高计算能力，增加了可扩展性和安全性。云计算技术应用场景有很多，例如大型数据中心的数据处理，大规模的视频转码，因此本文将云计算技术应用于视频转码工作中。

在国内外视频转码研究方面的重点在于构建一套分布式部署的多媒体信息存储云和高性能转码云平台。其中构建一套完整、高效的转码云平台是整个项目的核心内容，高性能的视频转码云平台也是本文的核心重点。高性能视频转码云平台将云计算技术^[8,9]和视频转码^[10]技术融合从而提高整体效率。高性能的转码云平台可以将转码速度提高并且速度提高率和视频文件大小成正比。通过本文中所提出的二次校验可以保证视频转码传输过程中不会出现错误，保证数据块顺序不会打乱并可以做到原始数据的找回。本文的高性能转码云平台在转码质量上也可以做到较高水平。

1.2 国内外研究现状

目前，国内外有很多人在研究视频转码技术，这证明该领域是时下的研究热点并获得了极大关注。这些工作中每个人、每个团队都设计了各具特色的解决方案并实践了独特的见解和技术手段，它们都有值得本文工作学习和借鉴的地方。

1.2.1 国内现状

Mohohan^[11]是一款由台湾地区研究团队提出的一个分布式转码架构。采用了 Hadoop 技术和 FFMPEG 转码技术来提高转码效率。但它的主要工作方式没有发生革命性改变：还是通过将视频分块，在采用并行的方式进行系统优化，该项目应用的 Hadoop 是没有优化的版本。据本文作者通过文献^[11]查阅到的数据表明该

项目效率并没有提高很多并且在数据冗余和数据保护上没有做到完善、在平台二次开发过程中没有很好的预留接口,这些有待改进的方面导致后续升级和功能扩展时出现了些许问题。虽然有些许的不足,但是该项目为后续研究起到了良好的引导性作用。

Yang 的团队也在 2011 年也发表了一篇文章^[12], 在该文章中介绍了基于 Hadoop 的转码系统。他们的具体实现方式和 Kim 的工作类似, 只是在存储结构上有所差异, 并不是本质上的优化和改变。从效率角度来讲, 该文章所述原型实际实用效果不是很理想: 因为在真实环境中, 不可能所有的数据都在一个网段中并且省略不计网络带宽从某种程度上来讲要是不合理的。但是该研究对于本文工作还是有一定的启发性作用的。

1.2.2 国外现状

X264farm^[13]是一个完整的分布式转码系统。这个系统中实现了多个并行的转码器以提高转码效率。该系统存在诸多有待完善的问题: 虽然在整个架构中添加了负载均衡策略来提高整体性能而在负载选择上考虑不周全; 虽然进行了并行计算但是在负载均衡上没有做到实质性的突破。该工作借鉴了云计算^[4]平台的思路但其实是一个并行计算的系统。它将视频分块并进行并行处理, 通过基础调度算法来实现简单的调度优化, 本文认为这只能算是基础的负载均衡实现。虽然该系统有诸多问题, 但是对于 2006 年的技术背景来说此系统作为为后人的研究铺垫好道路。

Kim^[14]等人于 2013 年研发了一个转码系统, 该系统是融合了 Hadoop 技术和 FFMPEG 技术。该系统通过在 Hadoop 云系统中使用 FFMPEG 库来进行两种技术的深度融合。设计采用 FFMPEG 库下的 Xuggler 来对接口进行设计。它提供了一个云计算架构, 通过转码的代码进行数据流和工作流描述, 完善整个的业务流程, 提高转码效率。

Garcia^[15]等人在 2010 年提出了一个评价体系, 这个评价体系由于较为完备, 并且在评价方式清晰明了, 至今还有许多人的工作都参照该评价体系去对自己的系统设计做评估。这个评价体系可以多角度认证系统的效率等等。对于视频转码平台的评价也有了一定的技术保证。

1.3 本文主要工作

本文研究的核心内容主要是基于 Hadoop 的视频转码优化的研究，并且设计了转码云平台的体系结构。设计了一个 S_MapReduce 并行计算架构来提高视频转码效率，通过使用负载均衡策略来协调计算过程，提高整个集群的计算能力，这样就可以提高整体的运算能力，然后为用户提供更好的用户体验。高性能视频转码作为应用层的基本服务，将 FFmpeg^[16,17]和 Hadoop^[18,19]进行深度融合，应用 S_MapReduce，构建出一套完整的分布式转码的体系结构，完成高效的转码应用，并对数据进行检验比较和冗余保护，使其能更好的将数据完善化，提高转码速率，提高用户体验。

与上节同类工作不同的是在负载均衡设计和工作流设计上都体现了先进性和优越性，使整个计算环境可以达到较高的计算水平。在实现方式和设计思路方面借鉴了上节讲述的相关项目，对本文起到了深远的影响和深刻的启发。

1.4 论文结构

本文的结构安排如下：

第 1 章 简述本课题的项目背景和国内外工作对比，并简述本文的核心内容。

第 2 章 介绍了相关技术基础，并对相关技术做详细解释。

第 3 章 阐述转码云平台的体系结构、技术选择和工作流程设计。

第 4 章 主要介绍了 S_MapReduce 计算架构，详细说明优化和修改内容。

第 5 章 主要阐述视频转码的优化，给出了 FFmpeg 的 NAL 设计、转码与 S_MapReduce 融合及二次校验，并对相关内容做具体解释与分析。

第 6 章 介绍了视频转码实验与分析。搭建转码云平台，进行相关的性能测试，并对结果进行总结、归纳。

第 7 章，总结本架构的优势与劣势，并对下一步工作进行展望。

第二章 相关技术基础

2.1 云计算

云计算(cloud computing)中的云是对互联网的一种形象化描述,所谓的云计算^[20]就是一种按使用量计算费用的一种模式,是为用户提供高效计算的一种有效的途径。

云计算平台目前在业界有很多可供使用的产品,如新浪云计算平台、Google 云计算平台^[21]、Amazon 云计算平台^[22]、百度云计算平台等等。这些云计算平台种类众多并且各有特色,如何选择合理的平台成为了一个关键的问题。本文选择的是 Hadoop 云计算平台,它是 Apache 基金会的顶级项目而且各方面都很完备,从最下层的网络构架和数据存储到上层的数据处理和软件层处理, Hadoop 都做到了生态系统的完整实现和数据循环。

2.2 Hadoop 平台

当 Google 首次提出 MapReduce 后,大数据、云计算、分布式等概念变得越来越熟悉广为人知并深受学术界和产业界重视。在 2005 年 Apache 根据 Google 的 MapReduce 设计^[23]构建了 Hadoop 生态系统。第一个在真正实际应用场景中的 Hadoop 应用程序是 Yahoo! 公司^[24]于 2008 年构建的。Hadoop 诞生至今不断进行着升级改进工作,最新的 Hadoop 2.6.0 版本已经趋于完善,无论从调度上的转变还是从性能的提升,都远大于 Hadoop 1.x 的版本。在近期的版本中,添加的 Yarn 也对 Hadoop 的整体性能优化问题交出了一份满意的答卷。但是 Yarn 在稳定性上显示出明显的不足,这也是不能被完全应用于实际生产中的原因。

Hadoop 生态系统是一个健全的,稳态的生态系统。从底层架构到上层应用都展示出不同凡响的能力。在 2010 年, Hbase^[25]、Hive^[26]、Pig^[27]、Zookeeper^[28]等从属于 Hadoop 下的子项目都变成了主 Apache 基金会顶级项目,可以看出 Hadoop 在云计算方面和整个技术演进上都起到了举足轻重的作用。

2.3 MapReduce 计算架构

MapReduce 是一个分布式批处理计算架构,它的优越性在高效处理大规模数据。因此适用于离线处理数据。它可以提高系统效率并减少计算时长。MapReduce^[29]的执行流程如图 2.1 所示,首先将一个任务分成多个小片,其次进行 Map 操作将各个小片映射到各个计算节点。之后在计算后将任务结果 Reduce 回来并进行任务整合。最后输出给 HDFS^[30]端进行存储。

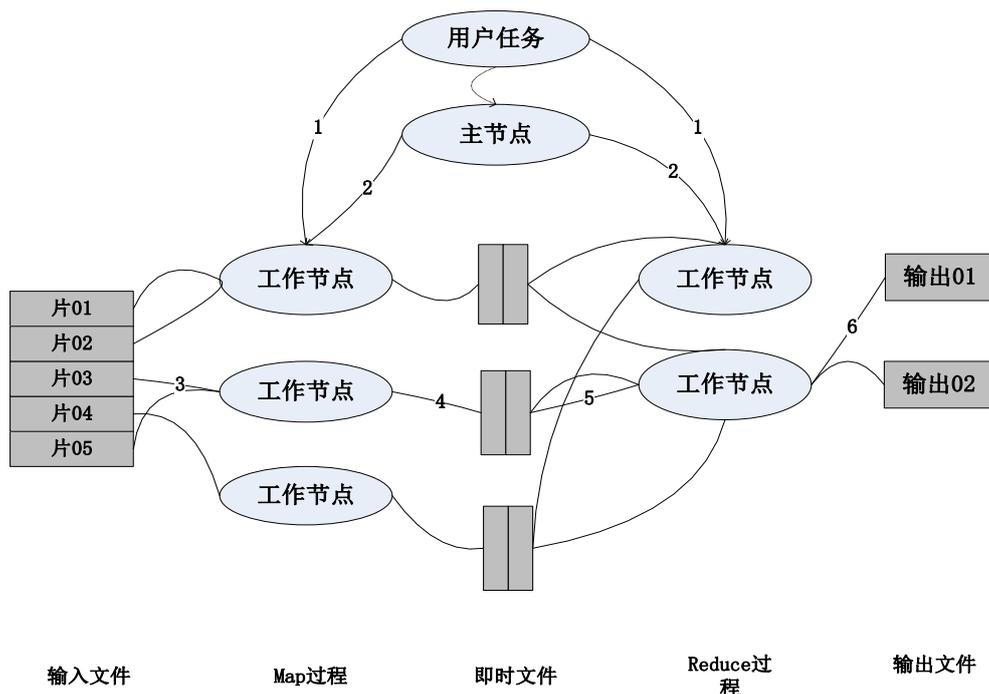


图 2.1 MapReduce 执行流程

具体流程如下：

1. 用户将任务发送给主节点并且通过主节点发送给工作节点
2. 主节点通过机器实时状态的反馈，借助调度算法来提高整体性能
3. 将任务进行分片，分解成 0~128M 每块。并且将不同块发送到不同节点上
4. 通过多个工作节点进行 Map 任务，然后将中间结果存储于即时文件中
5. 将即时文件发送到不同的工作节点上进行结果回收，也就是 Reduce 过程
6. 将结果输出并且存储于 HDFS 中

整个 Hadoop 给我们提供了可以使用的 MapReduce 的平台,可以对大规模数据进行快速的处理。MapReduce 在整个 Hadoop 发展进程中展现着举足轻重的位置。

MapReduce 在各个领域都展现出较强的数据处理能力。在深度学习^[31]领域,

MapReduce 可以提高深度学习的效率，帮助节点快速学习，优化大量迭代次数。在图像处理领域的结合也能帮助其快速完成图像^[32]处理，对图像的处理过程进行优化整合以便可以快速处理。在大规模的图算法^[33]中也给予了强有力的支撑。MapReduce 的例子还有很多，它的批处理和数据处理能力还是非常出色的，可以胜任很多应用场景。但是目前，MapReduce 主要应用领域在 log 日志的离线处理等文本类事务，还没有一套完整的体系来支撑转码云架构。这对于此类应用场景，MapReduce 还是大有可为的。本文也是基于 Hadoop 云平台搭建的转码云架构。

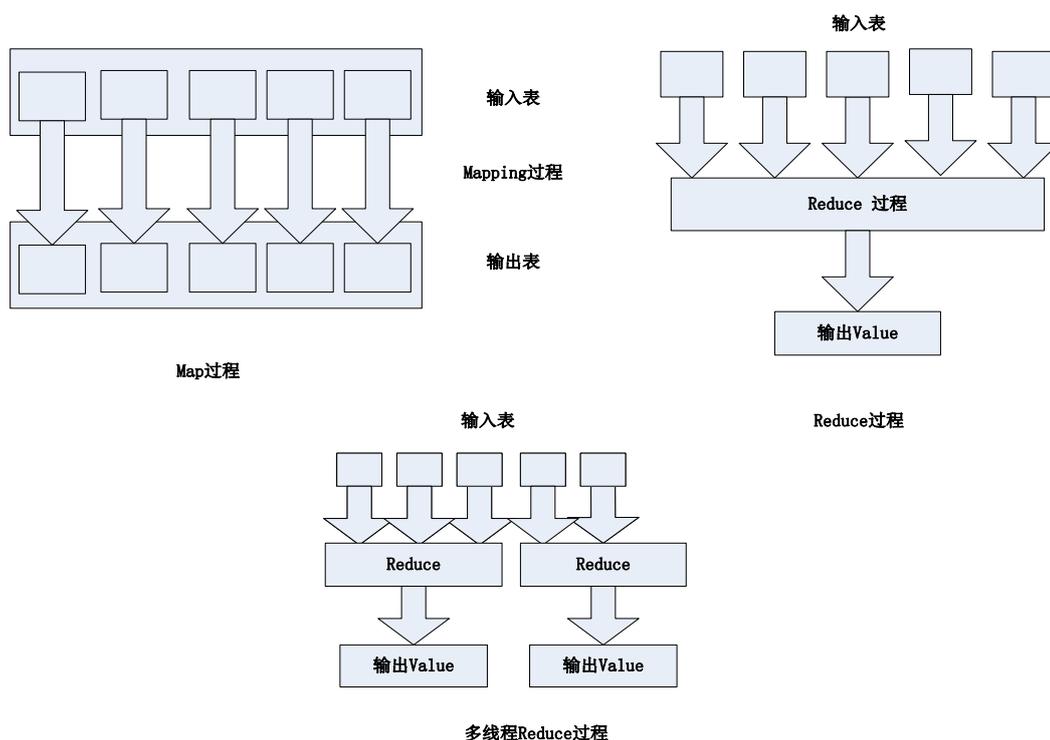


图 2.2 多线程处理过程

如图 2.2 所示，MapReduce 程序将列表形式的输入数据元素转换成输出数据，在功能性编程中使用两种不同的链表处理语句：Map 和 Reduce。MapReduce 的第一阶段称为映射，通过 Map 方式将数据以列表的形式发送出去，这阶段允许在许多机器上对数据进行并行处理。MapReduce 的第二阶段是 Reduce。将计算后的数据根据 key 值来匹配并回收数据（也就是 Reduce 过程）。

在 MapReduce 过程中，每个 MapReduce 数据元素是一个由键值和相应数据构成的键值对。在环境变量设定过程中，对编程方面没有明确并且严格的限制要求，允许对映射过程和统计过程进行重定义。Map 和 Reduce 之间的关系是 1:1。通过对数据进行 key 和 value 的拆分，可以更好的在各个节点进行计算并做到快

速回收。MapReduce 的设计是依赖于输入的值键。这个过程是十分关键的。

2.4 视频转码技术

1. 视频转码技术

视频是由图像流和音频流构成的，多个图片的连续播放配合相应的音频就形成了一个连续的视频影像。视频文件本身可以被称为容器（Container），信息被存放在文件中的位置是通过容器类型决定的（就像 AVI 和 QuickTime）。本文中所描述的流的概念是一个形象化表述，其实就是在固定的时间间隔内的一连串的数据元素。在数据流中，其数据元素被称为帧（Frame）。从数据流中读出来的数据称为包（Packets）。包就是一段数据，它包含了一段可以被解码成不同类型的原始帧的数据。每个包中包含了完整的帧。

视频转码就是将某种格式下的视频转换成另外一种格式的视频的一个过程。视频的格式有很多种，可以分为 FLV^[34]、AVI、MPEG^[35]等多种格式，不同的格式所对应的编码技术也各有不同。MPEG 又称为运动图像专家组。MPEG-1 广泛应用于 VCD 中，MPEG-2 广泛应用于 DVD 中，MPEG-4 是依托于第二代压缩编码技术的国际标准来进行编码的也是目前比较先进的一种编码格式。AVI 格式也叫做音频视频交错，是微软发布的一款视频格式。FLV 是 FLASH VIDEO 的简称，是一种新式的视频格式，在加载速度和文件大小占比上拥有很大的优势，但是随之而来的就是导入 FLASH 后，SWF 文件过大对在线播放也造成了巨大的传输压力。对于视频文件来说，分辨率、每秒帧数、比特率等众多指标是衡量和评判视频质量高低的重要基准。在存储方面，视频和音频总是使用某种有损压缩编码来存储的。视频转码也就是根据不同的编码格式改变了视频的帧速率或帧的尺寸等参数。针对于视频的主要属性也有相关指数和系数去分析视频的质量和压缩情况。

2. FFMPEG 技术

在本文了解的转码技术中：从转码和传输的过程上看 FFMPEG 的转码能力较为出众，并且是一个完全开源的软件。它可以架构在 Windows 和 Linux 中，这也省去了不同系统之间不兼容的问题。FFMPEG 是一个完整的开源解决方案，它集成了录制、转换、音/视频编码解码功能。FFMPEG 支持 MPEG、DivX、MPEG4 等 40 多种不同的格式进行编码，并且可以做到将 AVI、MPEG、OGG 等 90 多种解码方式。

FFMPEG 还有自己的播放器，使得用户在测试中更加方便快捷。FFMPEG 多媒体框架，采用 LGPL^[36] 或 GPL^[37] 许可证。FFMPEG 多媒体框架为录制、转换以及将音频与视频流化提供了一套完整的解决方案。目前最先进的音频/视频库 libavcodec 被应用在 FFMPEG 框架中。FFMPEG 的视频采集功能不容小觑，通过以下四种方式可以进行采集视频：1、视频采集卡，2、USB 摄像头，3、屏幕录制，4、支持将视频流传送给支持 RTSP 的流媒体服务器，支持直播应用通过 RTP^[38] 方式。

在 FFMPEG 整个架构中，编码 encode，解码 decode，错误检查 error_check 三个部分是最为重要的三个模块，FFMPEG 的编码和解码过程如下图 2.3 所示。

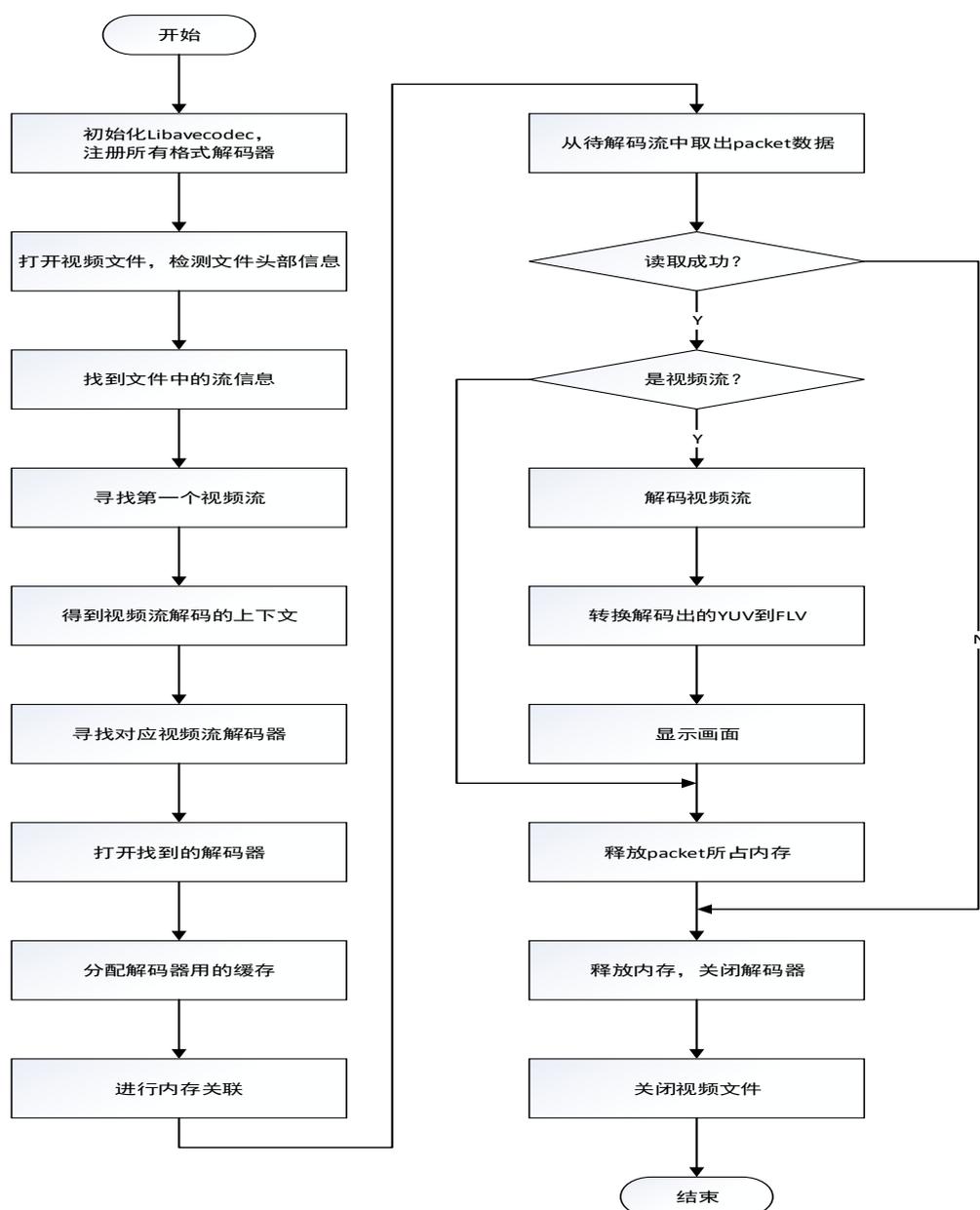


图 2.3 FFMPEG 解码过程

第三章 转码云平台架构设计

3.1 视频转码云体系结构

转码云平台体系结构主要是“3+1”层次，如图 3.1 所示，自底向上分别为 IaaS 基础设施层，PaaS 平台层和 SaaS 应用层 3 个层次。3+1 中的 1 就是存储中心层，该层次就是整个体系内的最底层并由磁盘阵列构成。本文中选取 Amazon 云平台作为基础设施层。选取 Hadoop 作为平台层设计基础，将 LVS 引入平台层，作为整个平台的负载均衡模块，使用 LVS 提高整个平台工作的效率。为了提高整个平台的弹性和容错能力在平台层中引入了虚拟 IP 机制，修改了心跳包流程。SaaS 层是应用层，在应用层中分为个人业务和管理业务两个方面。在个人业务方面，实现了以网页的形式为用户提供服务，用户可以在线进行视频点播、直播和在线的视频转码任务并提供播放功能。该业务是依托于修改后的高性能视频转码模块来完成的。管理业务是给后台工作人员使用的。后台工作人员可以通过网页做到对整个架构的流量管理、任务监测、系统状态掌控和相关数据展示。最下层的存储中心层主要磁盘阵列。

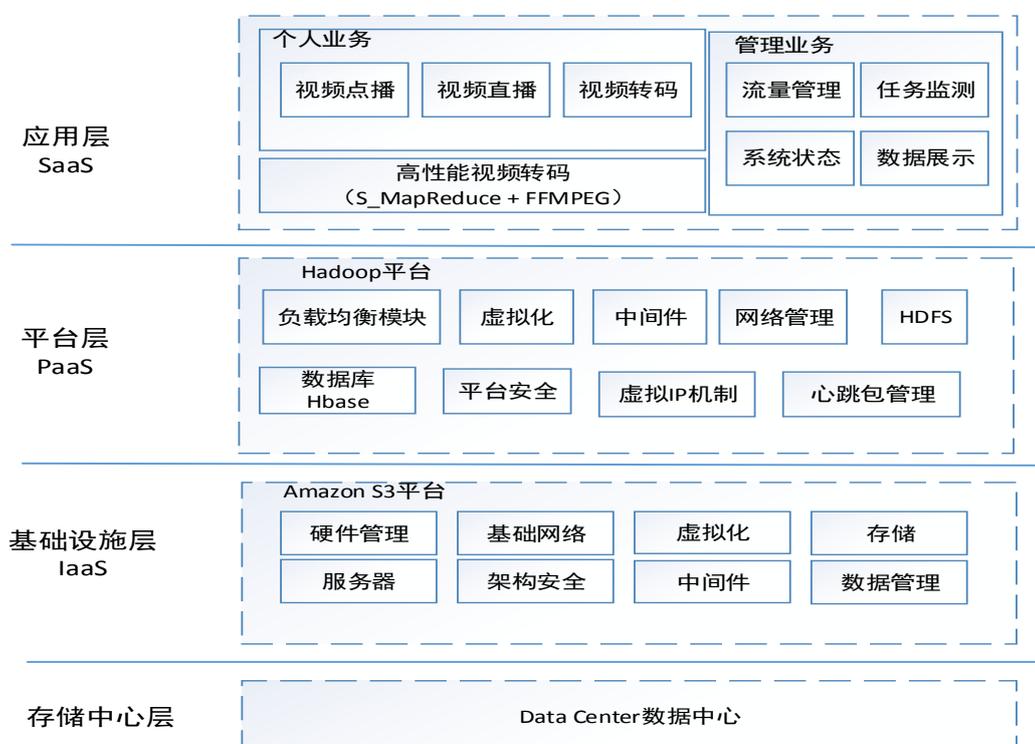


图 3.1 视频转码云平台体系结构

视频转码架构是基于云平台来搭建实现的，通过 Amazon 云平台为基础，上

层使用 Hadoop 生态系统构建视频转码的软件层，并且将视频转码使用 S_MapReduce 计算架构进行视频转码优化。运用分布式思想来构建转码云，使其更高效的完成转码任务。并且利用 HDFS 作为分布式类型的存储数据，存储过程是通过非结构化方式实现的。将图像和音频的数据进行拆分，通过矩阵的方式进行存储。具体操作方式是通过 JAI 库在 Map 中进行图像转化与调整。

3.2 技术选择

在 IaaS 中，可以选择的种类繁多，有开源和不开源，有国内的和国外的，如何选择一个适合的云服务是关键。根据实际项目背景和即将开发的应用类型来选择，锁定在 Amazon 和 OpenStack 两种，Amazon 是收费的云服务，OpenStack 是开源的。OpenStack 在成熟度上不如 Amazon，无法提供一个稳定的云服务。亚马逊 Amazon 虽然是一款收费的云服务，但是它为用户提供的计算节点能力和服务托管都是一流的，可以保证基本的项目需求。

在 PaaS 中，选择也是种类繁多，Hadoop、Spark 和 Storm 等等都是优秀的云平台。但是项目背景对用户的实时性并没有很高的要求，从分布式的角度上看，Hadoop 有着得天独厚的优势，其 MapReduce 计算框架是目前最为流行的一种分布式计算框架，所以本架构中选择 Hadoop 作为 PaaS。

在 SaaS 中，基于 PaaS 定好的提前条件下，MapReduce 也就成为了整个应用层的关键技术。视频转码技术种类繁多，有开源的，有不开源的。在技术层面上看，视频转码就是将采集到的视频通过固定的解码方式将数据打开，重新整理视频格式，最后再进行封装操作。视频转码技术的选择，从某种程度上来讲是核心要素，一个转码云的成功与否是和视频转码技术息息相关的。本文在众多视频转码技术中，选择了 FFmpeg，原因有如下三点。第一，FFmpeg 在项目构建和系统完备性上讲是处于领先地位的。第二，FFmpeg 技术在实现层面上来看是适配于不同操作系统下的，Windows、Linux、Mac 都是兼容的，并且完美承接的。第三，FFmpeg 在转码性能和可扩展性上来看，也是十分出色的。可以做到一个基础技术的作用。在 SaaS 中使用了 FFmpeg 和 S_MapReduce 来作为应用层的关键技术。

3.3 转码云 workflow 设计

转码云是将转码任务移交云平台上处理，workflow 就从原始的一台机器内进行调度转化为多点、多层次的调度。具体的 workflow 如图 3.2 所示。

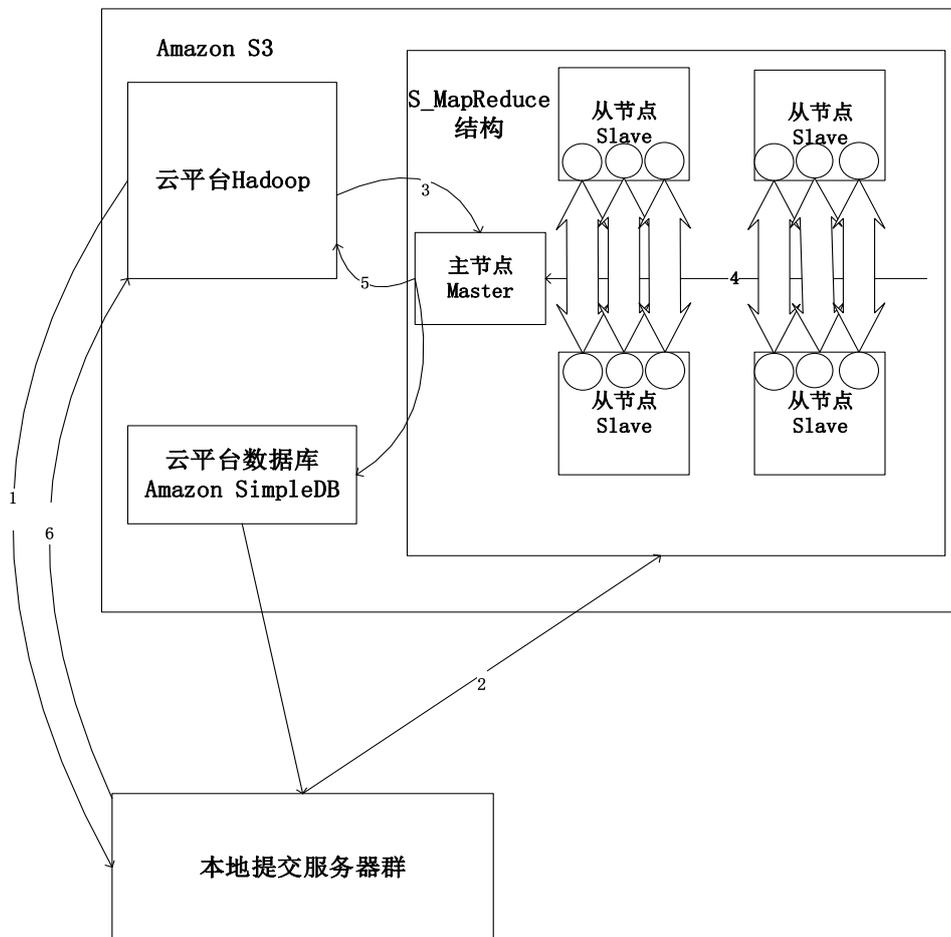


图 3.2 视频转码云 workflow

具体流程如下：

1. 在 Amazon S3 平台上,通过一个或者多个 HTTP 链接提交被转换的文件。
2. workflow 提交给 S_MapReduce, 然后对相关转码任务进行参数设定, 并进行存储和上传到 S3 平台上。

3. 每个码的操作方式都是来源于定义文件。主节点从 S3 中复制文件, 然后对分片进行复用操作, 并且注入到块文件中。最后随机分配到不同的从节点上。

4. Hadoop 集群工作是提交给 JobTracker, 这个过程就是分配任务。每个从节点开始处理 Map 任务分配。JobTracker 任务是将数据映射到本地存储数据的从节点上, 并且随机的选取 HDFS 上的节点。意味着几乎所有的从节点都是能够处理本地数据。一旦所有 Map 任务完成, 就开始进行 Reduce 阶段, 将集群中的节点和中间数据存储到 HDFS 上输出。

5. 在主节点进行合并操作，获取 **Reduce** 任务得到的结果，然后将其归纳成一个文件，将结果返回并复制一份到 **S3** 上保存为持久备份。
6. 将最终结果输出，然后复制到 **S3**，并回提到本地机器(或留在 **S3** 和服务给客户直接在 **web** 上)。然后重复步骤 3, 直到所有的文件定义文件处理工作。

第四章 S_MapReduce 架构

4.1 S_MapReduce 架构设计

在 Hadoop1.x 和 Hadoop2.x 下 MapReduce 版本是不同的。在 Hadoop1.x 版本中，MapReduce 架构是由一个 NameNode 节点来整体控制的，这样会出现 NameNode 拥堵，性能降低，安全性减弱的情况。Hadoop2.x 版本中出现了一个整体调度的模块，该模块减轻了 NameNode 的压力，但是在跨域和性能提高方面没有得到显著的提升。Hadoop2.x 版本虽然在某些方面相对于 1.x 版本来看提高了很多，但是在整体稳定性和可扩展性上不能与 1.x 进行媲美。目前，也没有 Hadoop2.x 应用于生产中的例子。以项目为背景，项目需要应用在真实的生产环境中，故选择 Hadoop1.x 版本进行操作。本文以 Hadoop1.x 版本为蓝本进行架构优化，使其能更好的与视频转码模块深度融合。

对于 Hadoop1.x 版本的 MapReduce，为了提高整个架构的完备性与灵活性，更好的实现多个数据中心之间的跨网段的任务输送，在原始架构基础之上增加了虚拟 IP 策略和负载均衡策略，对数据流和工作流进行整体调配优化。提出了一个 S_MapReduce(Smart_MapReduce)计算框架，如图 4.1 所示。

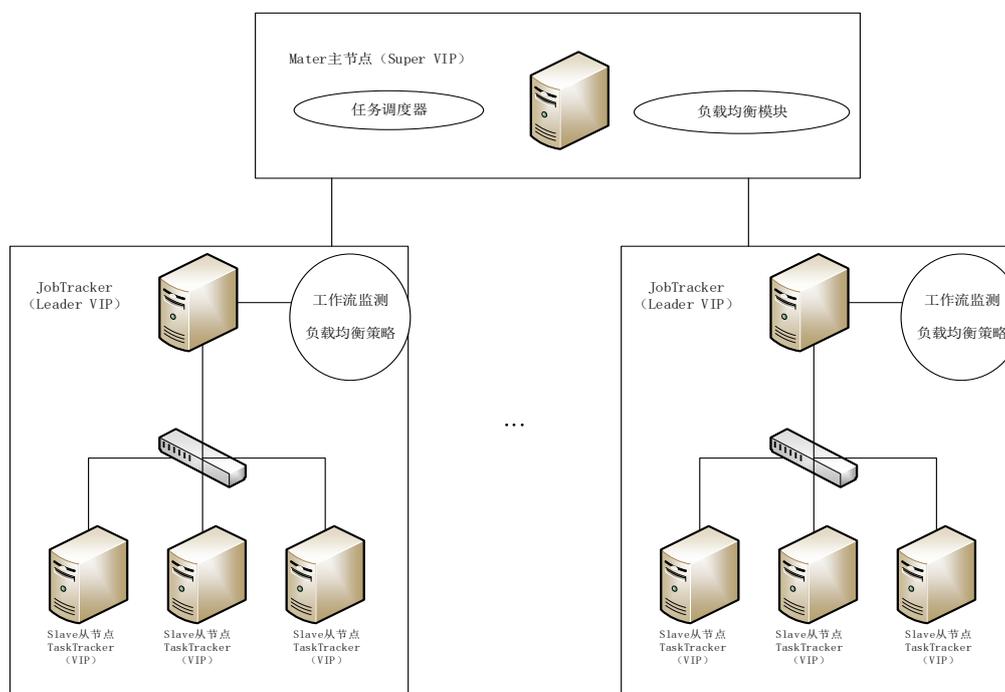


图 4.1 S_MapReduce 架构

S_MapReduce 架构由一个主节点 master 和多个从节点 slave 组成，主节点的

主要功能有分配任务、任务调度模块和负载均衡模块等等，从节点的主要功能是接收、执行任务和对数据进行调度。在整个架构中使用了虚拟 IP 机制，提高了整个架构的可扩展性、灵活性。主要核心部件功能如表 4-1 所示。

表 4-1 核心部件功能说明

组件名称	功能
主节点	管理元数据和目录树的 HDFS，它接收从属节点的心跳和执行结果。
从节点	运行任务，并将结果提交给主节点。
JobTracker	接收由用户设置的参数发布命令根据网络的动态。它把每个工作分为一个或多个块。
TaskTracker	执行派出由 JobTracker 任务
Leader VIP (Super VIP)	Super VIP 与 Leader VIP 工作模式类似，负责所有的任务。当一个任务是从客户端发送到集群中，它首先被节点的节点所在的节点集合。然后 Leader VIP 问题工作节点上的任务。

虚拟 IP 机制是框架的核心设计，虚拟地址被分配到多个域名服务器。这个设计可以提高整体架构的冗余性，服务器将失败的任务或者故障进行转移，不会影响正在进行的工作。

当 Super 虚拟 IP 从客户端接收任务时，它将虚拟地址转换为 Leader 虚拟 IP 的物理地址，并选择一个 Leader 虚拟 IP 来运行它。根据集群间的负载均衡策略进行选择。当 Leader 虚拟 IP 从一个 Super 虚拟 IP 接到一个任务，它派遣这些任务到对应节点，并开始实施负载平衡策略。因此，集群内的各个节点应保证实时传输消息和信息，以确保拟议好的框架下的资源可以得到充分的利用。也正是因为虚拟 IP 机制的特点，它可已被动态地映射到物理地址上，从而更容易做到平衡整个系统中各个节点的工作量，帮助系统达到一个相对高性能的水平。

4.2 S_MapReduce 心跳包设计

对 MapReduce 架构来讲，心跳机制决定了整个系统是否可以正常工作，是否做到了负载均衡。心跳包是一种 RPC(Remote Procedure Call Protocol)机制。RPC 又名远程过程调用协议，是通过网络方式来进行远程服务请求的一种协议。原始 MapReduce 和 S_MapReduce 的心跳包原理是相一致的，但是引入的虚拟 IP 机制在整个心跳包校验过程中起到了找寻地址的作用。也是更改心跳包流程的依据和凭证。在 S_MapReduce 框架，JobTracker 根据虚拟 IP (VIP) 机制来控制各个子节点接发心跳包的节奏和情况，根据获取到的信息去动态的部署子节点工作内容和分配任务，实现了动态可扩展能力的提升。所以通过精确定位和了解子节点工

作情况后，经过缜密的计算和动态调配，从而做到提高数据传输的速率，并提高了整个集群的容错能力。具体流程如图 4.2 所示。

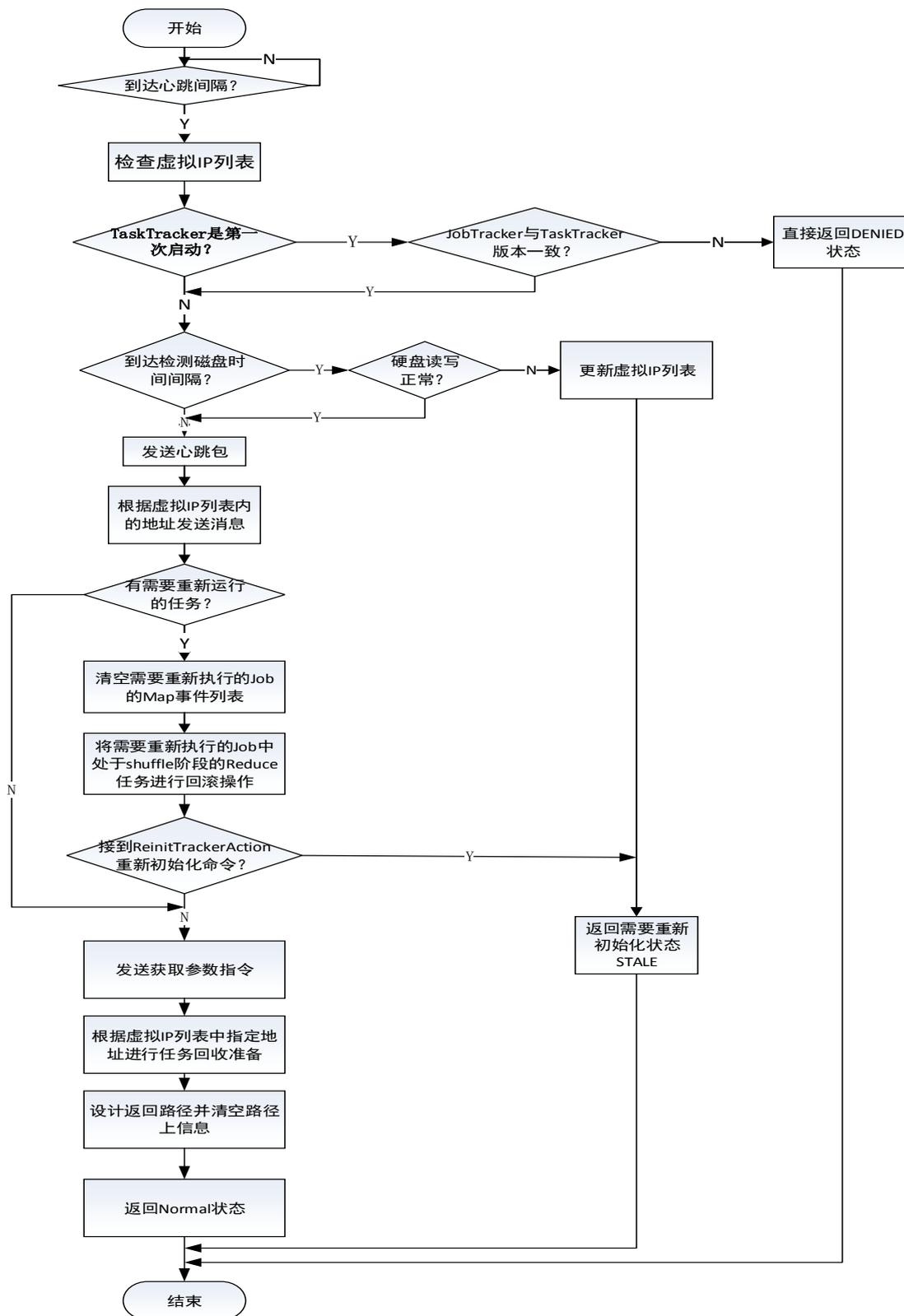


图 4.2 S_MapReduce 心跳包执行流程

重点步骤有如下 6 个步骤。

1. 根据虚拟 IP 的配置，设置集群
2. 判断是否发送心跳
3. 根据虚拟 IP (VIP) 配置文件找到相应的虚拟 IP (VIP) 地址，判断 TaskTracker 的位置和状态
4. 根据虚拟地址来锁定节点，并检查是否有损坏
5. 发送心跳包到指定的 TaskTracker 的虚拟 IP (VIP) 地址
6. 执行命令并返回结果给 Super 虚拟 IP (SVIP)

4.3 S_MapReduce 工作流和数据流设计

在 MapReduce 架构中，客户端将请求发送到 JobTracker 的虚拟 IP 所在的节点，这样 JobTracker 就可以监视每个 TaskTracker 的工作状态和是否健康，并根据实际情况使用负载均衡策略为任务分配资源。然后 JobTracker 可以根据 TaskTracker 的虚拟 IP 完成相关的管理工作。JobTracker 将发送心跳到 TaskTracker，从而确认 TaskTracker 是否还可以工作，并将结果反馈到 JobTracker。这一系列的功能都是依赖虚拟 IP 机制来进行完成的。这个机制保证了整个集群的稳定性和可扩展性。通过使用虚拟 IP 机制，可以实现动态的任务分配和节点从宕机状态恢复到正常状态后，可以将数据恢复并根据指令去进行工作。

在 S_MapReduce 框架中，我们使用虚拟 IP 机制来提高容错性能和计算性能。通过配置文件，JobTracker 找到自己的虚拟 IP，从而准备进行下一步任务分配的工作。负载均衡策略和网络流量等策略是否能做到提高效率，都是需要根据实时获取到的信息来动态调整的，关键也在于配置文件中阈值的选择。

一个任务执行的过程如图 4.3 所示，工作过程包括 8 个步骤，可以表述为：

1. 客户提出处理工作和提交工作的要求
2. 如果客户提交的 IP 与集群的虚拟 IP 相同，该节点在集群中的 Leader 虚拟 IP 开始发送数据包来检查 TaskTracker 工作节点的健康，并进行实时跟踪。
3. 如果节点是处于可以工作的状态时，它会反馈心跳包到 Leader 虚拟 IP 来证明它的存在。
4. 根据 Leader 虚拟 IP 获取的情况来分配任务，并根据获取到的网络情况做动态调配，并启用相对应的负载平衡算法来完成工作分配。
5. 负载平衡模块将分配策略返回给 Leader 虚拟 IP，并告知如何分配。将虚拟 IP 地址列表提交给 JobTracker。
6. JobTracker 将列表发送给 TaskTracker 并分配任务。

7. TaskTracker 开始根据收到的列表来分配计算节点去工作，在 Reduce 阶段之后，整个工作也就随之结束。

8. 任务返回结果到 Hadoop 的 HDFS 中。

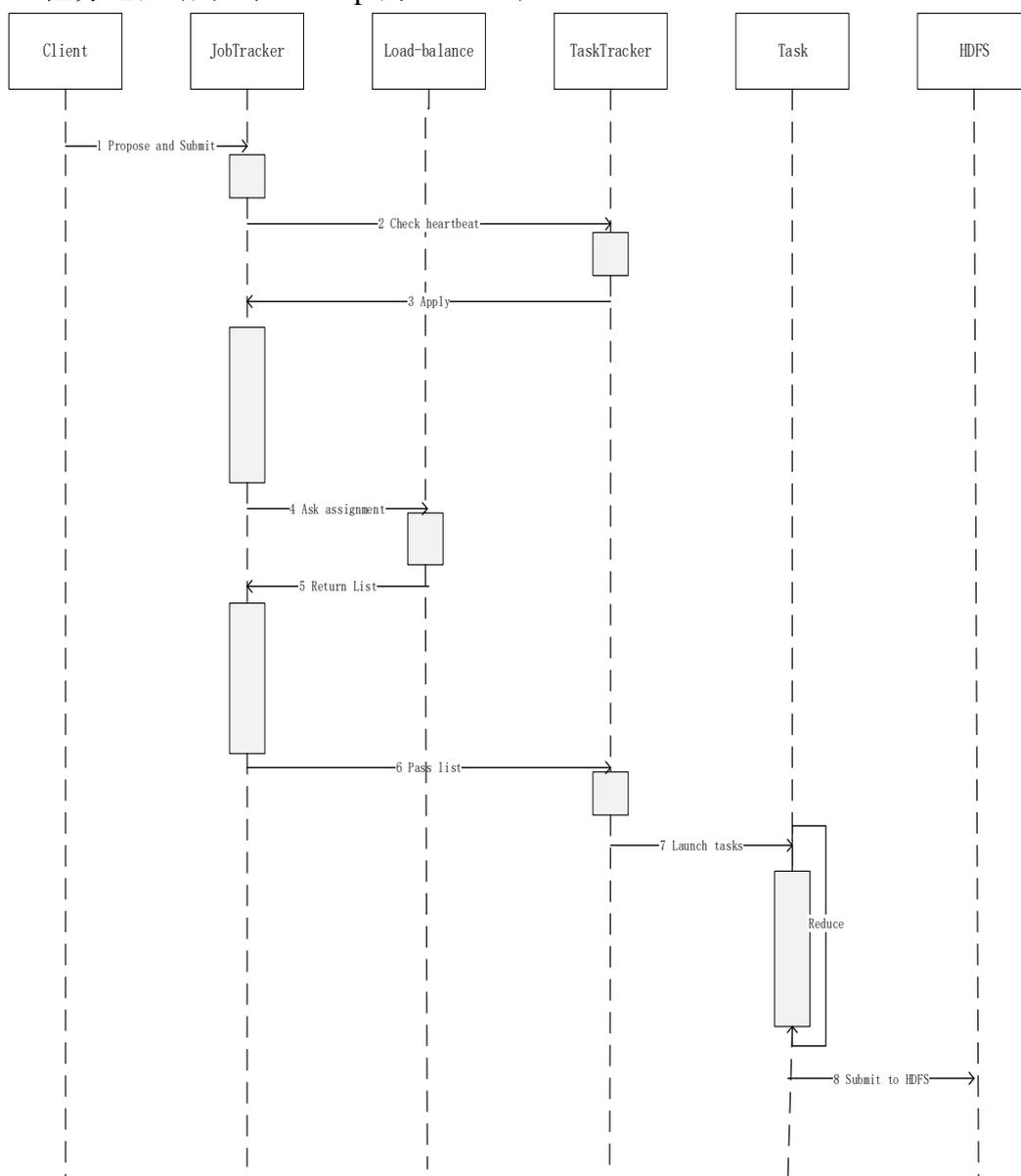


图 4.3 S_MapReduce 的工作流程图

数据流的优化也是 S_MapReduce 框架的一个重点优化的内容。在原有体系结构的基础上，通过构建虚拟 IP 机制来寻址并传输数据，这也就形成了一套全新的数据流设计，如图 4.4 所示。

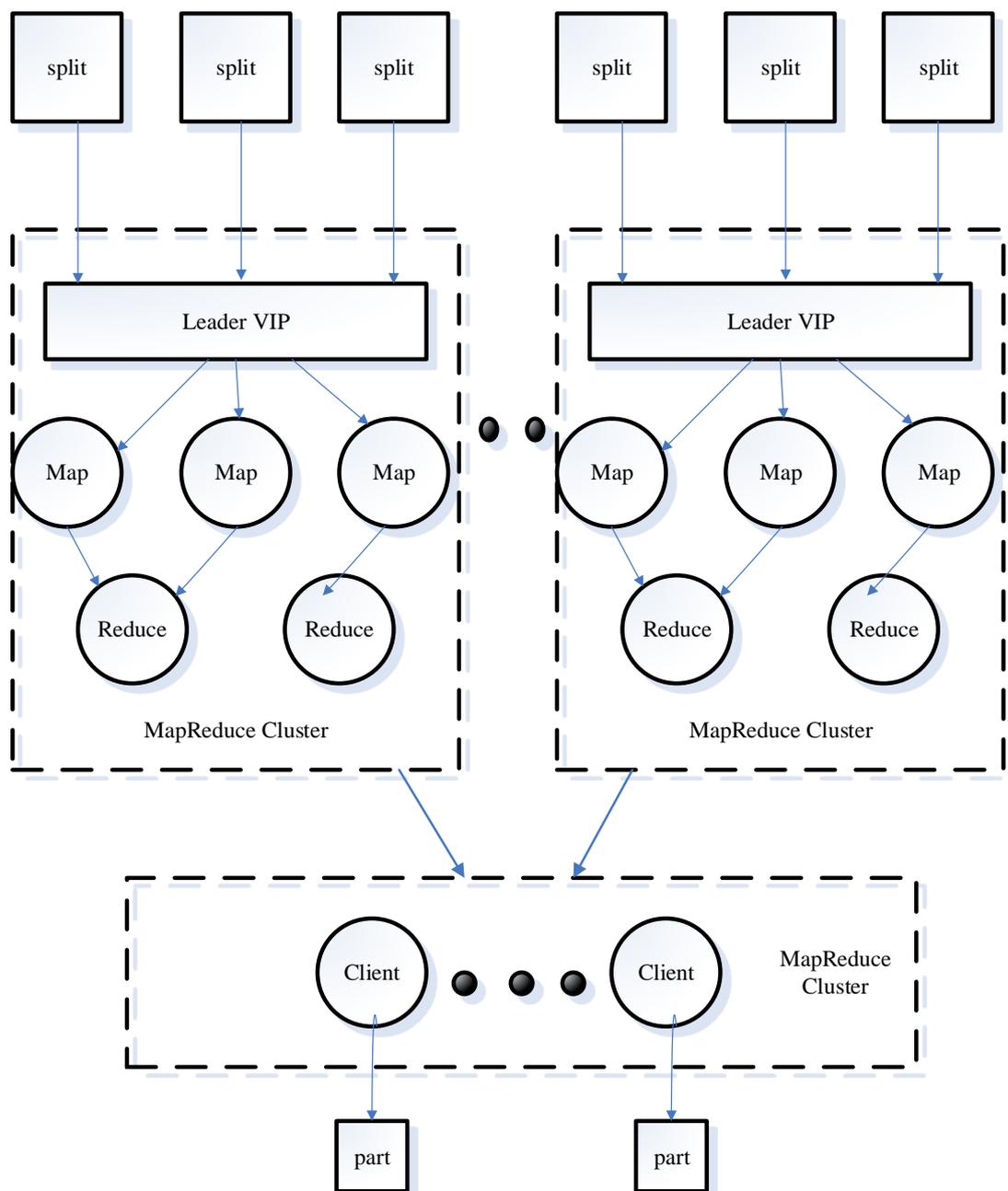


图 4.4 S_MapReduce 的数据流流程

数据流的处理就是控制数据走向和路径选择。在这个难点上，本文引入虚拟 IP 机制来完善整个过程。将数据根据规则进行分片操作，分好的数据片段也是 Map 映射后的一个过程，通过虚拟 IP 列表的指引，找到应该去往的节点，这个过程是经过数据分析的，是保证路径畅通的。在寻址过程中就是通过 Leader VIP 和 VIP 之间的配对来找到应该去往的节点。这个过程的实现就是数据流的走向。

4.4 负载均衡模块设计

根据虚拟 IP 机制，可以动态的分配资源，并且可以在同网段和不同网段内之间进行数据传输。根据这个特性，本文设计了全新的负载均衡模块，负载均衡模块通过分析心跳包获取的数据来对整个集群进行负载均衡策略的实施。模块中使用 LVS 来对整个集群进行负载均衡工作。

LVS，英文是 Linux Virtual Server，中文是 Linux 虚拟服务器，也是一个三层网络结构，正好适用于本架构的设计。LVS 是一个开源软件，该软件的主要功能是对集群进行 IP 负载均衡。通过虚拟 IP 机制来实现相应的算法，并且对整个集群进行负载均衡策略的实施。

负载均衡算法有很多，IPVS 实现了如下三种 IP 负载均衡的策略。第一个是 VS/NAT，全称是 Virtual Server via Network Address Translation。这种算法主要是通过网络地址进行转换的。第二个是 VS/DR，全称是 Virtual Server via Direct Routing。这种算法是通过真实网络，也就是 MAC 地址进行交互的。第三个是 VS/TUN，全称是 Virtual Server via IP Tunneling。这种算法是通过 IP 隧道的方式来进行优化的。但是针对于不同的网络状态和配置，IPVS 实现了十个负载均衡调度算法。轮询算法(RR)、加权轮转(WRR)和最少链接(LC)等十个算法，每个算法在不通过情况下可以发挥不同的作用。算法本身有自己的特性，可以根据特性设定相应权值，通过权值来选择算法。

在本设计中，将采集到的心跳包内的数据发送给负载均衡模块，负载均衡模块根据采集到的信息，选取其中的技术来对整个集群进行调度，以达到负载均衡。在采集信息过程中，需要心跳包返回相关数据，数据为 CPU 使用率、内存剩余空间、机器节点健康状态、往返心跳的时间和节点工作状态。根据这些采集到的数据来对过程进行建模，通过相关计算选择出使用哪种策略来进行任务分配。设定好负载后告知 JobTracker，JobTracker 根据设定好的策略分配任务给 TaskTracker。

负载均衡策略选择函数主要数据描述如下：

P_cpu: 一个心跳包获取的 CPU 的使用率；

P_mem: 一个心跳包获取的内存的使用率；

Ht: 一个心跳包获取的心跳包往返时间；

used: 表示为 CPU 的可用比率, $\{used=(1-P_CPU)|0 \leq used \leq 1\}$

mem: 表示为内存的可用比率, $\{mem=(1-P_mem)|0 \leq mem < 1\}$

t: 表示为经过处理后的心跳包发送和回收的时间和, $t \{t= ht/k, k=10|0 < t < 1\}$ 。设定一个阈值 $k=10(s)$;如果往返时间大于等于 10s, 则放弃该节点, 不作为计算节点使用。

V: 表示一台机器的性能描述, $V_i=(0.4 * used_i + 0.4 * mem_i - 0.2 * t_i)$

AV: 表示一个集群的性能描述, $AV = \sum_{i=1}^n V_i / n$, 根据 AV 的值选择负载均衡策略。

负载均衡策略伪代码:

```
load-balance (packet, n)
{ float V[], sum=0, AV;
  float used[], mem[], t[], P_cpu[], P_mem[], Ht[];
  for (i=0; i<n; i++){
    获取心跳包数据 (P_cpu[i], P_mem[i], Ht[i]); }
  for (i=0; i<n; i++){
    used[i]=1- P_cpu[i];
    mem[i]=1- P_mem[i];
    t[i]= Ht[i]/10;
  }
  for (i=0; i<n; i++){
    V[i]= (0.4 *used[i] + 0.4 * mem[i] - 0.2 * t[i]);
    sum+=V[i];
  }
  AV=sum/n;
  根据 AV 的值选择执行对应负载均衡算法。
}
```

第五章 视频转码的优化设计

FFMPEG 总共包含四个部分，编码、解码、传输和错误处理四大部分。本文主要工作集中在解码部分。根据需求将源文件转换成所需要的文件类型。本文对其进行如下几部分的优化处理。

1. 根据实验室项目具体情况，要求将任意类型的视频都转换成 FLV 格式，根据这种情况，对 FFMPEG 进行瘦身操作，将不适用于项目需求的内容删除，得到一个轻量级的 FFMPEG。并且将传输模块中的传输协议转换成适合于项目场景的传输协议。瘦身后的 FFMPEG 大小减少到原始 FFMPEG 的 55%左右，编译后的大小减少了 70%以上。

2. 修改了 FFMPEG 中解码部分的相关流程，并对划分 NALU 过程中可能出现的顶替和相互竞争的情况，设计出了一种解决方案来避免出现此类问题。

3. FFMPEG 中的容错部分虽然预留了相关接口，但是并没有具体实现。本文设计了二次校验机制，来保证视频转码和传输过程中的完全性和质量。

4. 设计了(key, value)对的格式，并设计了一个结合 S_MapReduce 的视频转码架构。

5. 引入时间戳、分割点和 APU 等机制来保证整个架构内部数据的高性能处理。

5.1 FFMPEG 解码

5.1.1 NAL 单元设计

FFMPEG 程序代码中主要分三个部分：

create();负责解码器句柄的创建；

decode();完成解码，因为修改了 NAL 单元，decode 内部都跟着发生了变化；

release();实现解码器句柄释放。

根据事先定义好的转码类型对数据流进行相对应的转码。在转码过程中首先将视频流分解成多个 NAL 单元，并以字节方式进行存储。随后将其存储在 Buffer 中，并且对其进行编码与解码。

NAL(Network Abstract Layer)，又称为网络抽象层。在视频编码中，整个体系框架由视频编码层（VCL）和网络抽象层（NAL）组成，VCL 负责数据内容，NAL 负责格式化数据和提供数据头部信息。

NAL 由一个字节大小的头信息和 VCL 带来的原始字节序列载荷 (RBSP) 的字节流组成。NAL 头格式如图 5.1 所示。

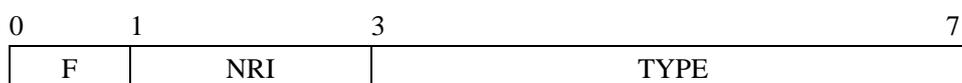


图 5.1 NAL 头信息格式

其中各个位含义为：

0: F, forbidden_bit, 冲突标记位, 1 表示有冲突, 0 表示无冲突;

1~2: NRI, nal_reference_bit, 处理的优先级, 值越大, NAL 越优先处理;

3~7: TYPE, nal_unit_type, 表示 NAL 类型。TYPE 值与 NAL 类型对应关系如表 5.1 所示。

表 5.1 TYPE 值与 NAL 类型对照表

type	NAL 类型
0	未规定
1	非 IDR 图像中数据不划分的片段
2	非 IDR 图像中 A 类片段
3	非 IDR 图像中 B 类片段
4	非 IDR 图像中 C 类片段
5	IDR 的图像片段
6	补充增强信息, 也就是 SEI 内容
7	序列参数的集合, 也就是 SPS 内容
8	图像参数的集合, 也就是 PPS 内容
9	分割符
10	一段序列的结束符
11	一段流的结束符
12	补充数据
13~23	保留空间, 可以后续其他功能添加使用
24~31	随意, 不做设定

NAL 的定义中对于长度没有给出明确限定, 所以在界定各个 NAL 单元时, 需要在头部添加界定信息。解码器将不同的 NAL 放在一个分组中, 每个分组有自己的头部信息, 解码器可以快速的解析出不同的 NAL, 找到 NAL 的分界点。通过这种方式将 NAL 进行逐个编码。

每个 NAL 都有一个起始码，起始码是采用行业标准，NALU 的起始码设定原则分为两种情况，第一个是 NAL 对应的 Slice 是一个帧的头，则用 4 字节表示，即 0x00000001；第二个是 NAL 对应的 Slice 不是一个帧的头，则用 3 字节表示，0x000001。后续的设计以 0x00000001 为例，当检测到起始码就进行读取 NAL，直到读取到下一个 0x00000001 为止，则表示当前 NAL 已经读取完毕。但是 0x00000001 出现在 NAL 数据中间部分怎么处理，本文引入了防竞争机制。NAL 中包含 0x00000001 或者是 0x00000000 的情况下，就会在最后一字节的前插入 0x03，形如：

0x00000000—>0x0000000300

0x00000001—>0x0000000301

解码器在解码时，首先逐个字节读取 NAL 的数据，统计 NAL 的长度，然后再开始解码。但是当在 NAL 中检测到 0x00000003 的时候，0x03 就应该被删除，并恢复原始数据。这样就可以做到预防出现竞争和顶替的错误出现。

如表 5.2 展示了一个 NAL 在解码器处理过程中数据的变化过程。NAL 数据是以十六进制方式表示的，包括 NAL 头信息和 RBSP 字节流如表中第 1 行所示，解码器接收 NAL 时在其前面加入起始码，以区分不同的 NAL，都是以 00000001 开始如表第 2 行展示。解码器读入时，一旦起始码 00000001 出现在 RBSP 字节流中，并不是下一个 NAL 的开始时，就会出现错误和竞争，针对于这种情况，采用防竞争机制，将 00000001 的最后一个字节前插入 0x03，结果在表中第 3 行给出，开始解码时遇到 0x00000003 的时候，0x03 就应该被删除，并恢复原始数据如表中第 4 行所示，保证解码的准确性。

表 5.2 NAL 数据处理变化

行号	类别	起始码	NAL 数据 (NAL 头信息, RBSP 字节流)
1	原始 NAL		67 42 00 1F E9 89 C8 00 00 00 01 68 CE 06 F2
2	带起始码	<u>00 00 00 01</u>	67 42 00 1F E9 89 C8 <u>00 00 00 01</u> 68 CE 06 F2
3	读入防竞争		67 42 00 1F E9 89 C8 <u>00 00 00</u> 03 01 68 CE 06 F2
4	解码防竞争		67 42 00 1F E9 89 C8 <u>00 00 00 01</u> 68 CE 06 F2

解码时 00000001 后面跟着是 67,42 等等数据。0x67 是起始码后的第一组数，也就是 NAL 的头数据。通过数位转换可以得出 0x67 的二进制码为：0110 0111，可以发现第 4 位-第 8 位是 00111，所以将其转化为十进制，得到 7，参考上表 5.1，可以得知，7 对应序列参数集 SPS，为解码提供依据。

5.1.2 解码过程

对于帧，主要分为 I 帧、B 帧和 P 帧这三种类型。I 帧是关键帧，又称为内部画面(intra picture)，通常是每个 GOP 的第一个帧。I 帧也是 B 帧和 P 帧的参考帧，也就是说 I 帧的质量直接影响整个视频的质量。P 帧被称为预测帧，B 帧被称为双向预测帧。B 帧和 P 帧都是服务于 I 帧的。宏块 (Macroblock) 是由亮度像素块和两个色度像素块组成的。整个编码过程都是以宏块为最小单位进行转码的，转码后，将多个宏块组成连续的视频流。

解码一帧的过程大体上说是三部分。

第一，对输入的字段进行了一系列的检查工作：例如宽和高，输入文件类型等。

第二，通过 `ret = avctx->codec->decode()`，调用了相应 AVCodec 的 `decode()`；函数，完成了解码操作。

第三，对得到的 AVFrame 的一些字段进行了赋值，例如宽高、像素格式等等。

其中第二步是最为关键的，它调用了 AVCodec 的 `decode()`；方法完成了解码。AVCodec 的 `decode()`；方法是一个指针，指针指向了具体解码器的解码函数。下面以 H264 类型的文件为例，进行具体阐述。

解码 NALU 函数 `Sg_decode_nal_units()`；过程：

第一步，调用 `ffs_h264_Sg_decode_nal()` 并且判断 NALU 的类型。

第二步，根据 NALU 类型的不同调用了不同的处理函数。其中处理函数可以分为两类。第一类是解析函数，第二类是解码函数。

(1) 解析函数：

`ffs_h264_Sg_decoder_seq_parameter_set()`；主要工作是解析 SPS 内容。

`ffs_h264_Sg_decoder_picture_parameter_set()`；主要工作是解析 PPS 内容。

`ffs_h264_Sg_decoder_sei()`；主要工作是解析 SEI 内容。

`ffs_h264_Sg_decoder_slice_header()`；主要工作是解析 Slice Header 内容。

(2) 解码函数：

`ffs_h264_Sg_execute_decoder_slices()`；主要工作是解码 Slice 内容。

对于 `decoder_slice()`；函数主要还是按照宏块 (8×8) 的类型解码流文件。对于宏块处理的过程基本上可以归结为 3 个步骤，最后得到结果。具体流程如下：

第一步，熵解码过程。当预先定义好的熵编码类型为 CABAC，则调用

`ffs_h264_Sg_decoder_mb_cabac()`；否则熵编码类型为 CAVLC，则调用 `ffs_h264_Sg_decoder_mb_cavlc()`；

第二步，解码宏块的过程。`ffs_h264_Sg_decoder_mb()`；

第三步，环路滤波过程。`Sg_loop_filter()`；

简单来将，解码一帧的数据，就是输入一个压缩编码的结构体 `AVPacket`，输出一个解码后的结构体 `AVFrame` 的过程。

从整体上来看，编码和解码的效率是整个评价体系的关键，视频转码过程中，编码和解码两个模块是核心部件。需要先将视频流提取出，将提取出的视频流进行拆分，通过需求将视频流转换成某种类型的视频流。转码后将视频流封装好，就可以提交给用户。

5.1.3 二次校验算法设计

所谓二次校验，也是一种数据保护的策略，本系统整体是应用 Hadoop 来进行构建的，HDFS 就作为底层存储，本身就有一套完整的数据保护机制和冗余机制。但是对于视频转码这个背景来说，是远远不够的。本文中所提到的二次校验，是在数据接收过程和发送过程中做的处理。因为计算架构选用的是 S_MapReduce 计算架构，就涉及到数据被分块，但是对于视频数据来说，分块的模式可能会造成帧与帧之间关联丢失和丢帧的困扰，本文通过对每个视频块进行帧标记，并对数据块进行块标记，每次在接收和处理过程中，都对标记位进行更新处理和匹配处理，通过跟正确顺序匹配表相对比，找出出现问题的位置和提供一个合理方法去解决问题。这也是二次校验的精髓，通过占用少量空间来换取整体性能和转码后的视频质量是值得的。

在本文中设计的二次校验中，其实就是对 APU 进行数据保护和匹配。APU 是原子处理单元，是以 key 和 value 键值对的形式出现的。APU 数据的 key 值是时间戳，本文设计的是对相同时间戳的 APU 数据段进行处理。在 value 中是多个图片组 GOP 为了保证数据的安全和顺序性，本设计中是对每个 APU 中 value 部分进行附加编码，在 value 中的第一个 GOP 的头和最后一个 GOP 的尾进行 flag 标记位设定。在转码后，通过比对各个 APU 之间的关系，并分析是否是正确顺序，如果是准确顺序，就进行下一步操作，如果不是，就将数据退回 buffer 中，继续循环，等待匹配成功。Buffer 的大小控制在不超过 5 个 APU 数据。

二次校验算法伪代码表述如下：

```

re_check(int TS, int count)// 参数定义, TS:时间戳, count: 缓存数量
{
    for (i = 0; i < count; i++) { //初始化
        AK[i] = 0; // 保存时间戳内APU[i] 中key值
        AVH[i] = 0; // APU[i].value 中第一个GOP的第一位,首位标记位
        AVT[i] = 0; // APU[i].value 中最后一个GOP的最后一位, 末位标记位 }
    Int *ptr = (int*)malloc(sizeof(APU)*count); //申请内存空间, 命名为buffer
    open_stream(); //打开数据流
    for (i = 0; i < count; i++){
        接收APU[i] 数据;
        更新(AK[i], AVH[i], AVT[i]); }
    for (i=0, j=i+1; i < count; i++, j=i+1){ //二次校验
        { While (1) //寻找相邻视频块
            { 获取APU[j] 数据; //从视频流或缓存中
                if (AVT[i] = AVH[j]) //相邻
                    进行复用和combine操作; break;
                Else 将APU[j] 存到buffer; }
            }
    }
}

```

5.2 S_MapReduce 和 FFmpeg 融合

5.2.1 S_MapReduce 和 FFmpeg 框架整合

对于一个视频转码架构来讲, 整体系统的流畅性和转码速度是极为重要的。对于视频转码的优化, 本文选择了整体的优化方案来进行优化。本文通过整合 Hadoop 下新版本 S_MapReduce 计算架构, 将 FFmpeg 的编解码和 S_MapReduce 进行深度融合, 构建出一套完整的视频转码云, 使其无论在计算性能还是实用性上都能做到趋近于完善, 如图 5.1 所示。

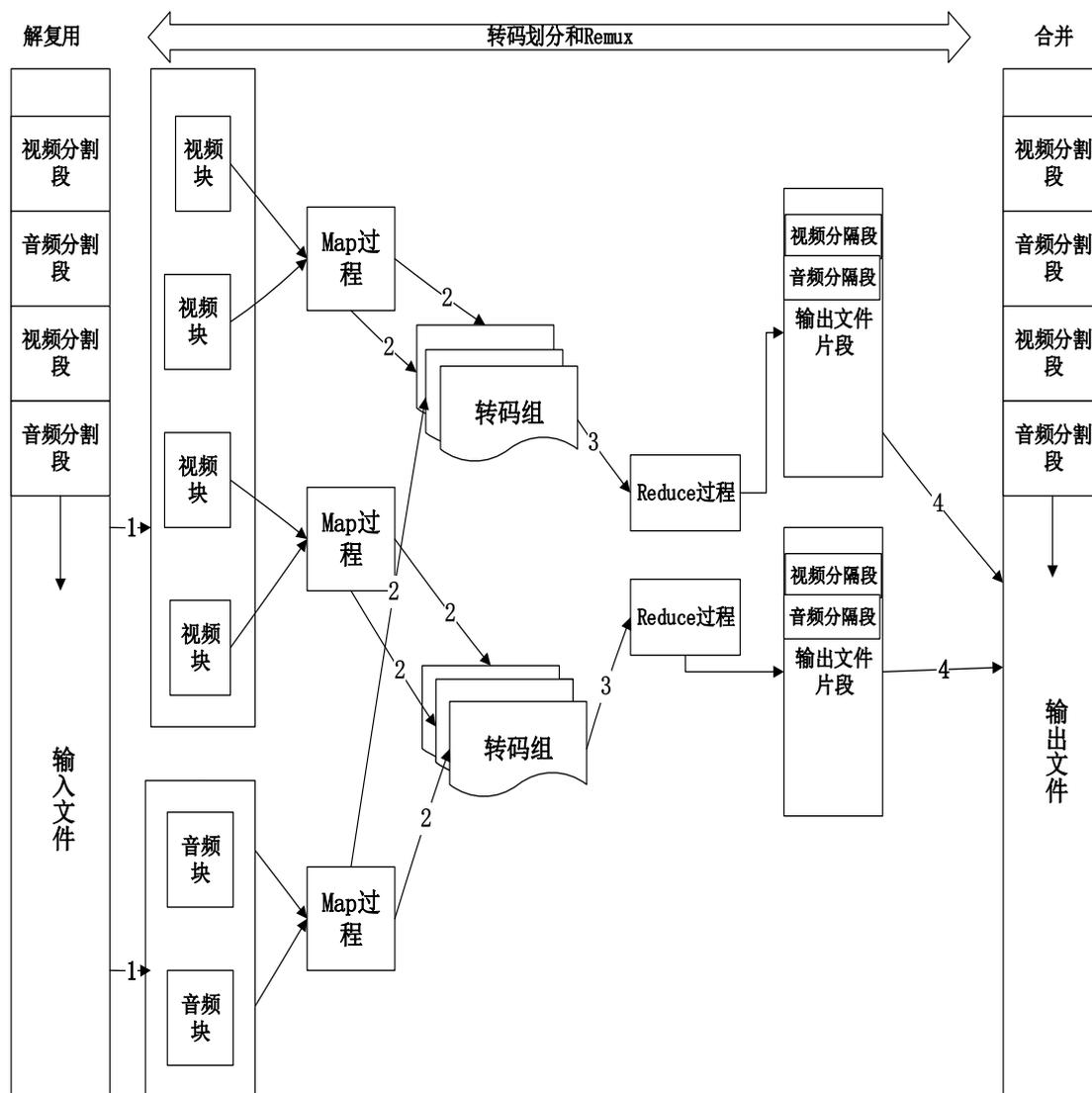


图 5.1 S_MapReduce 和 FFmpeg 框架整合

如图 5.1 所示，S_MapReduce 计算架构，是一种分散映射的模式，将整个视频文件分散成多个小文件进行分别操作。对于视频编码 FFmpeg 技术，可以通过修改编码模式，优化过程，使其能结合 S_MapReduce，形成一套整体的、高效的编码体系。关键步骤解释如下：

- 1、解复用：将输入文件分解成视频块和音频块。
- 2、转码划分：通过 Map 映射，将数据块分配到不同的节点进行转码任务。
- 3、Remux：Remux 主要任务是完成无损的提取出 HD-DVD 等高清影音文件里的主音频和主视频数据，这些输出块是根据实际情况 Reducer 数量进行分组分配的，并且相对应的 key 值对不能多出或者少于。所以是双向控制的一个过程。保证 1:1 对应。

- 4、合并：将最后阶段处理好的视频段拷贝到最终定好的地点，并对输出的

数据进行相关操作。

5.2.2 S_MapReduce 分块策略设计

S_MapReduce 计算架构是高效并且易于二次开发的。S_MapReduce 类型的转码设计根据自身特点,在整个架构设计上也存在着特点和特性。分块策略是一个重点研究内容。找到一个适合系统计算能力的块大小是非常困难的。本文尽可能的将块的大小变成尽可能大,以减少视觉质量的下降,但是小文件的可扩展能力是强于大块文件的,这也是一个难以平衡的事情。此外,通过调整集群的设置来做到以最大限度地提高 Hadoop 和 S_MapReduce 的性能。在对群集进行初始设置时,将 Map 任务槽和线程数设定为固定值。使用块数据作为并行计算的输入集,根据实际情况定义块的大小的方式会提高复用过程的效率。Hadoop 目前的版本还没有充分利用其集群动态分配任务的能力,本架构却较好的完成了这一工作,并且提高了动态分配能力。

通过相应的分块策略可以让系统的性能能得到充分的利用,将分块方式和机器性能进行分析,设计出一个适合于实验环境下的分块策略和任务数量。这样就能在单位时间内做到集群能力范围内最多的任务。

根据本文中的具体的实验环境和数据,设计相应的块来对数据进行分割,分割策略遵循不拆分帧,不使用超小块和超大块来进行转码任务分配。就像 10MB 的影音文件,不能使用 128MB 大小的块来对数据进行切分,这样会产生巨大的浪费,应该选用 8MB 或者 16MB,根据不同的节点数量选取相适应的块大小。

5.2.3 APU 原子处理单元设计

S_MapReduce 是分布式计算框架,通过将数据分割成 key 和 value 来进行操作,结构为(key, value)。将输入数据分散到多个原子处理单元中,这个过程称为解复用过程。对视频数据进行分块,将每个帧的视频头和尾进行标记和记录,这样可以进行头尾匹配。并对不同数据块的头和尾进行标记。由于新的数据包从输入文件读取,这些文件就被存储在每个数据流的临时缓冲区中。当缓冲区中的数据大小大于定义的“块”的目标大小时,就要创建一个新的块。使用视频我们清空缓冲区直到最近的一条关键帧,确保 GOP 结构放在一起。当使用音频时,我们完全清空缓冲区,因为它可以在任何时候被打破。这样就可以做到数据的保护,完整。具体形式如图 5.2 所示。

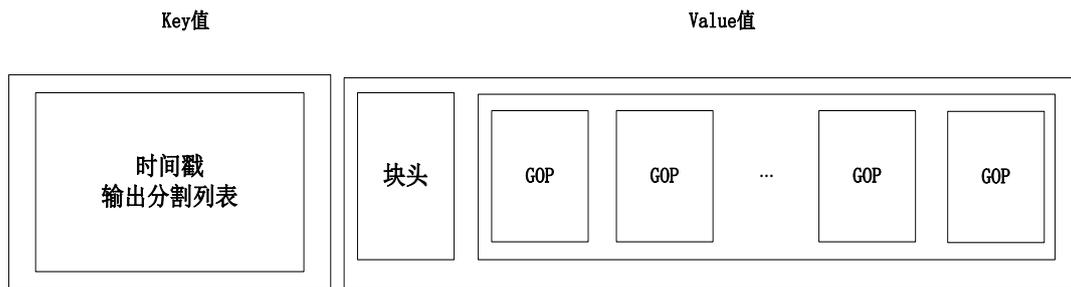


图 5.2 APU 格式设计

通过 APU 进行处理，并且对处理后的文件进行 Map 操作。映射到不同的节点上。一个数据块的大小主要是根据想要分成多少线程去处理，需要平衡输出端的数量和对整个过程进行负载均衡策略的应用，保证良好的性能。这个大小被指定以字节为单位，无论是音频还是视频。然后将切割后的文件进行封装和标记处理，方便后期输出时找到所对应的顺序。但是当处理音频的时候就可以考虑撤销标记，可以更好的提高效率。处理完的数据存放在 HDFS 中，此时的数据存储格式称为序列文件（Sequence-file）。适用于 HDFS 的处理。

5.2.4 分割点设计

当不同的块文件进行 Map 操作过程中，代表了特定时间下的特定视频与音频。而且每个块的大小不一定一致，也就导致了处理的时间也不一定是统一的。为了可以成功完成复用操作，需要时间一致才能同时返回 Reduce 函数。要做到这一点，本文设计并定义一个分割点，又名 split point，分割点本质的思想就是使在单位时间内完成视频和音频转码的数量一致。通过图 5.3 可以清楚的看到，在相同时间内，一个影音文件中视频的占比远远大于音频的占比。因此为了实现同时完成对视频和音频的转码，视频转码过程中，视频所需要的 Map 数量是远多于音频的。

分割点主要用于 Map 功能，表示在文件的哪个位置需要进行拆分，从而可以达到同时完成。当从 Map 过程划分各个输出块时，通过 key 值来进行配对，并且保证每个块的时间长度是相同的。不同类型的数据可以被分成大小一样的块，但是在单位时间内完成的效率差距很大，并不相同。

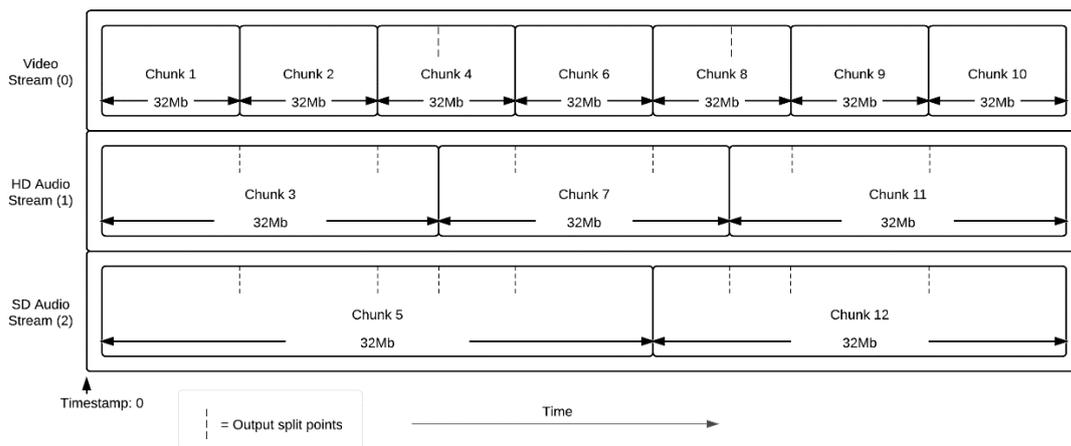


图 5.3 相同时间周期内的块分配

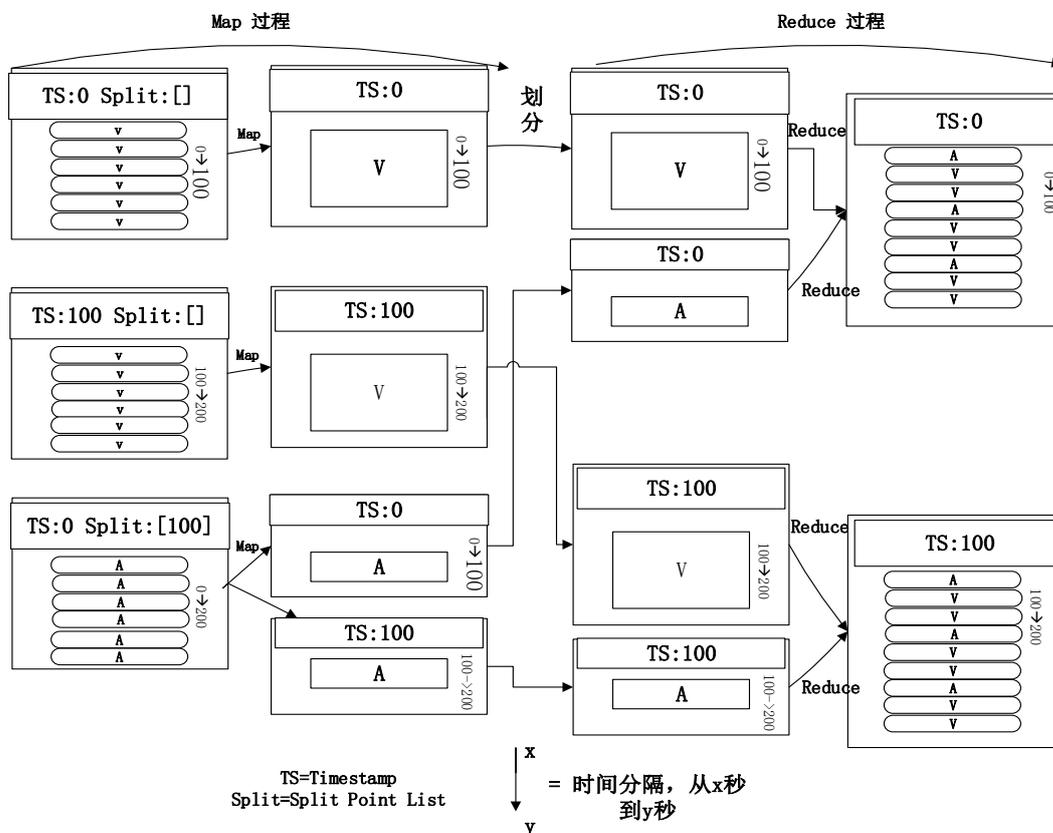


图 5.4 数据分块处理的流程

如图 5.4，视频和音频已经被区分出来，存到不同的分段中。图中 TS 代表时间戳，Split 代表分割点。从图中可以看出，在 Map 任务前将视频按时间戳分为两组数据，分别为 TS=0 和 TS=100。将音频进行分段来配合视频的速度，分割点为 100。通过 Map 后将数据分块。可以看出时间戳相同的视频和音频块回收到一起，故分成了 TS=0 和 TS=100 的两组分类。通过 Reduce 将两组不同的分类提交给 Merge 进行数据整合。通过图 5.4，可以看出视频使用两倍的时间来完成相

同量的音频处理，所以在上图中，视频的块数也是音频的两倍。

5.2.5 时间戳的单调性

虽然数据读取中存在交叉存取，但是运用高度准确的时间戳，在整个流中都会贯穿着非递减类型的时间戳，在回放过程中，每个分段的数据都是提前被缓存好的，并且是可以播放的。然后根据不同状态进行状态回溯，并且进行重新同步操作。这样就可以保证不同类型，不同段，不同点的视频流可以正常播放。

在分割点设定过程中，分割点的位置设定和个数对于整个系统的同步性是起到决定性作用的。如果设定不正确，那么在最后会发现丢失帧的风险。本文引入缓冲区的概念来进行解决这个问题。本文设计保持一个固定数量的块在缓冲区，并配备一个可变的输出队列。这样能将各个数据块的顺序排好，不会出现相关问题。根据实验测试，缓冲区大小定义 4 是较为适宜的。最多不超过 5。这样就可以做到既不会影响内存的数据占用，也会有充足的空间去缓冲数据块。

5.2.6 转码 S_MapReduce 算法设计

本文设计的转码是以 S_MapReduce 为计算框架的，从视频流的接收到转码后的外送，整个过程都是在计算框架上来实现完成的。具体内容如下：

1. Transcoder_Mapper 设计

该模块的主要工作是完成转码前的数据预备和 Map 映射。通过读取数据流的头文来对数据进行 key 值得设定。设定好后创建 APU 数据块，并将 key 值存储到 APU 头。设定好 APU 头后拉取数据，创建 Value 值，将数据块的 Value 值存储到 APU 的值域中，在 APU 中存储的是具体的数据。完成 key 和 value 的写入后，对得到的数据进行预处理，预处理后查看 Transcoder 函数，根据接口参数来对数据块进行格式重设定，并对视频和音频两种数据类型进行打包。在将数据包发送给不同节点前，应该做的是清空先前剩余或者遗留的数据块，对 buffer 做清空操作，清空操作后添加数据包。接下来就开始进行 Map 映射，映射方式根据负载均衡模块给出的路线来进行发送数据，发送的过程是依靠 key 值和时间戳来进行分配的，保证一个节点上的时间戳是一致相同的。Buffer 空间是固定的，达到数量就把 buffer 中的数据 Map 出去。如果当文件流已经停止，最后的 buffer 中还是有剩余，就单独处理这部分数据。在分配过程结束后，清空所有的 buffer，为下一轮数据做准备。具体流程见图 5.5。

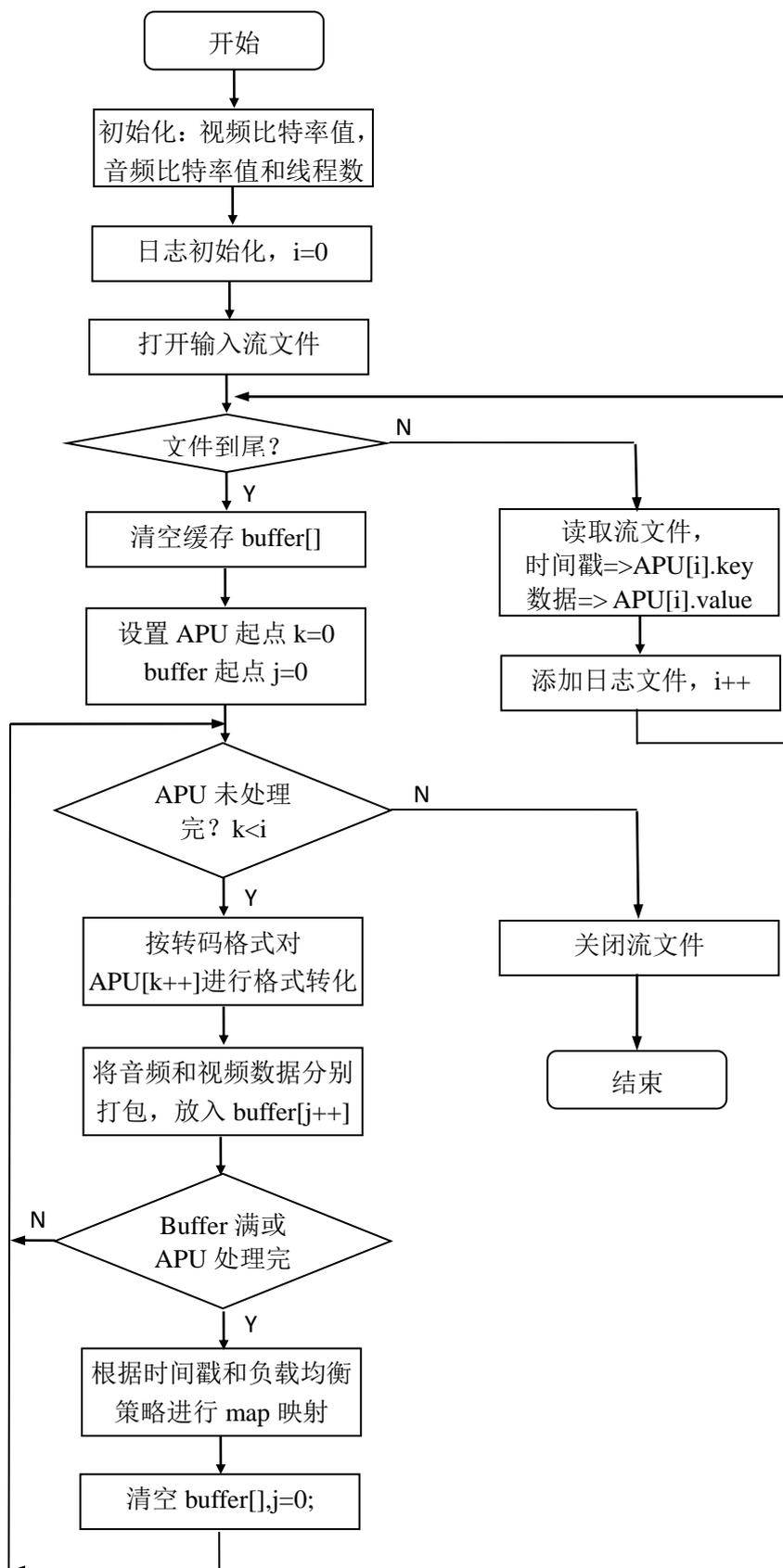


图 5.5 Transcoder Mapper 流程

2. Remux_Reduce 设计

Remux 的功能是将数据流中的视频和音频拆分出来, 在本模块 Remux_Reduce 中, 完成的是将数据从转码组中按照 key_list 来回收数据, 完成 Reduce 操作。之后将取到的数据流通过 Remux 来拆分出视频和音频两种不同的数据块。为最后的 merge 做准备。具体流程如图 5.6 所示。

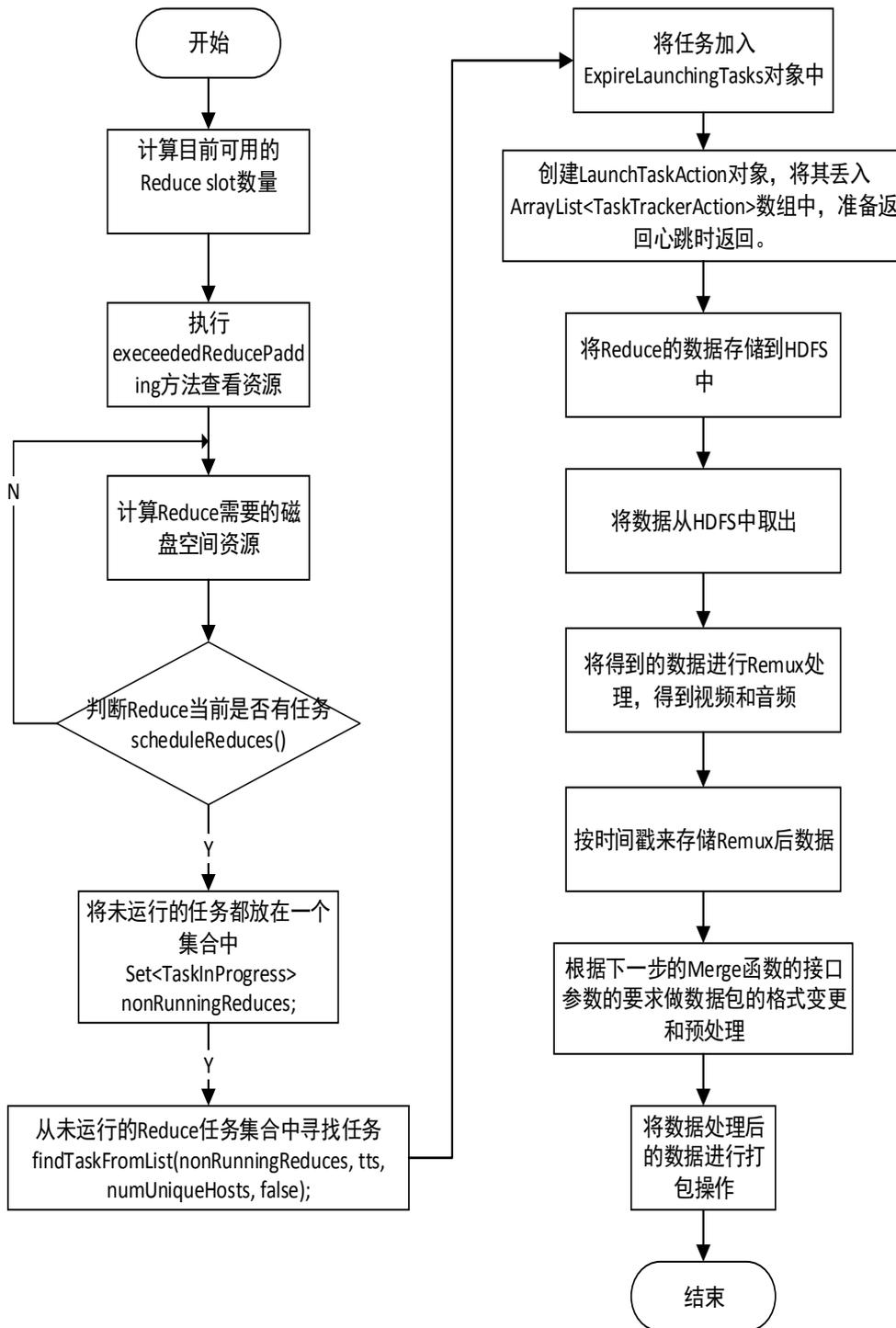


图 5.6 Remux_Reduce 流程

第六章 实验与分析

FFMPEG 的是一个开源项目，用 C 编程语言，具有良好的性能，可以支持众多的压缩格式的解码和编码。我们使用 Java 本地接口（JNI）来实现所需要的互操作，这样 FFMPEG 可用于 S_MapReduce 计算框架中，通过转码云来进行整合数据并输出。通过使用 C 的序列库来处理 Map 和 Reduce 块上的数据，这个库也称之为 TPL。

6.1 实验环境

实验机器环境是每个从节点的机器都是运行在 Amazon 云上的节点，每个节点的主要配置是 7GB 内存和 8 核的 2.33Ghz 主频的计算节点，能力趋近于 E5410Xeon 服务器，这种配置十分适合视频转码这个应用。计算能力约为 1.459 万亿次浮点运算，虽然在虚拟化过程中损失一些内存，但不会影响整个集群的运行。根据实验环境可以将每个节点最多分配 8 个 Map 任务和 8 个 Reduce 任务。该实验环境总共设定了 19 个节点，最多可以同时运行 152 个 Map 任务和 152 个 Reduce 任务，但是根据实验环境，每个节点设定的 Map 和 Reduce 个数是可以改变的。

该项目是利用 C++ 和 Java 混合来开发的，并使用 Xcode 和 Eclipse 进行实验代码构建模拟，主要是使用 Mac OS 来进行基础模拟实验。本地库文件需要被 Mac OS X 和 Linux 2.6 来编译，并动态链接 FFMPEG 的组件。使用一系列的脚本来调用库文件并且保证可以按需按需使用。Map 和 Reduce 函数使用 JUnit 来进行测试。需要对整个转码过程中的每个阶段进行测试，最后将数据汇总并展示出结果。结果的正确性验证，以及这些测试的性能，构成了我们的研究结果和评价的基础。

具体流程如下：

1. 使用 C++ 来编写对应于 FFMPEG 接口代码，并将其初始化
2. 使用 JNI 构建边界样本
3. 编写复用系统
4. Map 函数实现

5. Reduce 函数实现
6. 使用 Map/Reduce 方式来定义和编写视频转码部分
7. S_MapReduce 提交任务的 API 函数设计

6.2 实验数据准备

本文实验将进行可扩展性并行能力测试，输出视频的质量测试。对于这两个不同的测试重点，本文都使用相同的数据集进行相应的实验测试，并且对于数据的输入类型，质量和大小都有相应的标准。输入文件及其属性展示，如下表 6.1 所示。输入和输出文件都经过 Matroska 容器，保证了数据的可用性。输出文件中的每个视频和音频文件都是被处理过的，只有少许数据流被跳过（如章节标记和字幕）。

表 6.1 输入文件类型及其属性

FILE	文件大小	视频 pixels/sec	音频 samples/sec	长度 (sec)	比特率 (约) kbit/s	类型 (视频/音频)
1	4.6 MB	3.4M	64 Khz	35s	1097	WMA/WMV
2	183 MB	7.76M	96 Khz	1318s	1125	MPEG4/MP3
3	237 MB	5.84M	96 Khz	2755s	694	H.264/AAC
4	364 MB	6.92M	96 Khz	2617s	1114	MPEG4/MP3
5	798 MB	8.16M	96 Khz*6 streams	2643s	2408	H.264/AAC+AC3
6	1.54 GB	17.6M(HD)	88.2 Khz	2480s	4982	H.264/AAC
7	6.32 GB	16.8M(HD)	96Khz, 288 Khz	8294s	6106	H.264/AAC+AC3
8	10.52 GB	49.3M(HD)	288 Khz	9721s	8678	H.264/HEAAC

每个输入文件都是经过仔细筛选的视频文件，每个影音文件的视频大小、音频大小、时间长度、比特率和视频/音频类型都是不同的，都存在着自己的特点。特别说明 3 个文件，1 号文件本身包含多个视频流，在不增加多余音频的情况下增加了自己的长度。5 号文件的音频文件大小多余另外 7 个文件。7 号文件包含了一个 6 路视频（5.1 杜比数字流），每路的采样率是 96Khz。根据这些测试文件的特点，在分配视频块大小和构建 Map 任务时，都需要进行特别设定。

当要在并行环境下进行复用操作的时候，我们将输入文件分割成不同的数目的 Map 任务进行处理，正如表 6.2 展示的第一个文件的 Map 任务数量和目标块大小，通过这种方式进行处理可以大大提高计算可扩展性。所有的实验都根据表 6.2 给出的设定进行实验测试。块的大小是根据复用进程数目和 HDFS 中设定的

每块大小来进行调配的。每个块被分配到计算集群中的不同机器节点中。根据实验环境的计算能力，并且通过下表展示出的数据，我们将块的大小设定在 4MiB 到 32MiB。

表 6.2 输入文件的块大小与 map 设定

序号	文件大小	视频块大小(MB)	Map 任务数量
1	4.6 MB	4	1
2	183 MB	8	21
3	237 MB	8	28
4	364 MB	8	43
5	798 MB	8	95
6	1.54 GB	16	91
7	6.32 GB	16	376
8	10.52 GB	24	417

6.3 实验与分析

6.3.1 可扩展性测试

本文主要测试 MapReduce 在不同机器上的性能，这也是确保本文提出的解决方案能够正常的进行工作，并对整个集群进行测试，发现随着数目增加性能也在不断的提升。图 6.1 展示的是单点和多点的计算性能对比，不同文件大小展示出不同的效率提升走势，但是整体趋势是不会发生变化的，计算节点个数也由 1 个上升到 19 个，也就是说可执行 Map 任务的任务槽的个数从 4 个上升到 76 个。我们通过对处理每个文件三轮得到的时间取平均值来衡量各个集群大小的性能。图 6.2 和图 6.3 展示了不同大小数据在单点和多点时，效率的展示。纵坐标表示的基于单节点状态下，计算时间的平均值的激增程度，但是本图并不包括复用和合并的时间，只是考虑在分配过程中的时间效率。文件 1 到文件 5 的数据具体见表 6.3。（线程数：XC）

表 6.3 线程数量与速度比对照表

	4 XC	12 XC	20 XC	28 XC	36 XC	44 XC	52 XC	60 XC	68 XC	76 XC
File1	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
File2	1.0	2.1	2.5	2.6	3.0	3.0	3.1	3.2	3.5	3.5
File3	1.0	2.2	2.8	3.2	3.3	3.3	3.5	3.5	3.5	3.5
File4	1.0	2.4	3.4	3.5	4.6	5.2	5.5	5.8	5.8	5.8
File5	1.0	2.6	3.9	5.6	6.3	7.1	7.3	7.6	7.6	7.6

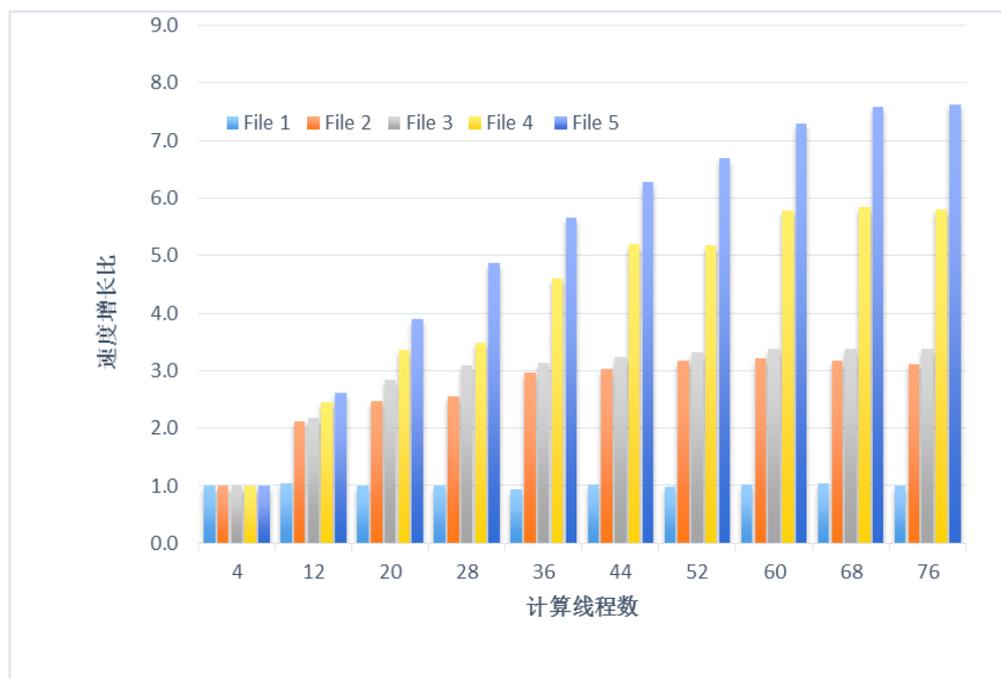


图 6.1 File 1 到 File 5 效率提升趋势

文件 6 到文件 8 的数据具体见表 6.4。

表 6.4 线程数量与速度比对照表

	4	12	20	28	36	44	52	60	68	76
	XC	XC	XC	XC						
File6	1.0	2.6	3.8	4.9	5.7	6.2	7.1	7.3	7.5	7.5
File7	1.0	2.9	3.9	5.9	7.5	8.3	9.4	10.9	12.1	12.7
File8	1.0	2.9	4.7	6.4	7.9	9.8	11.5	13.3	14.5	16.2

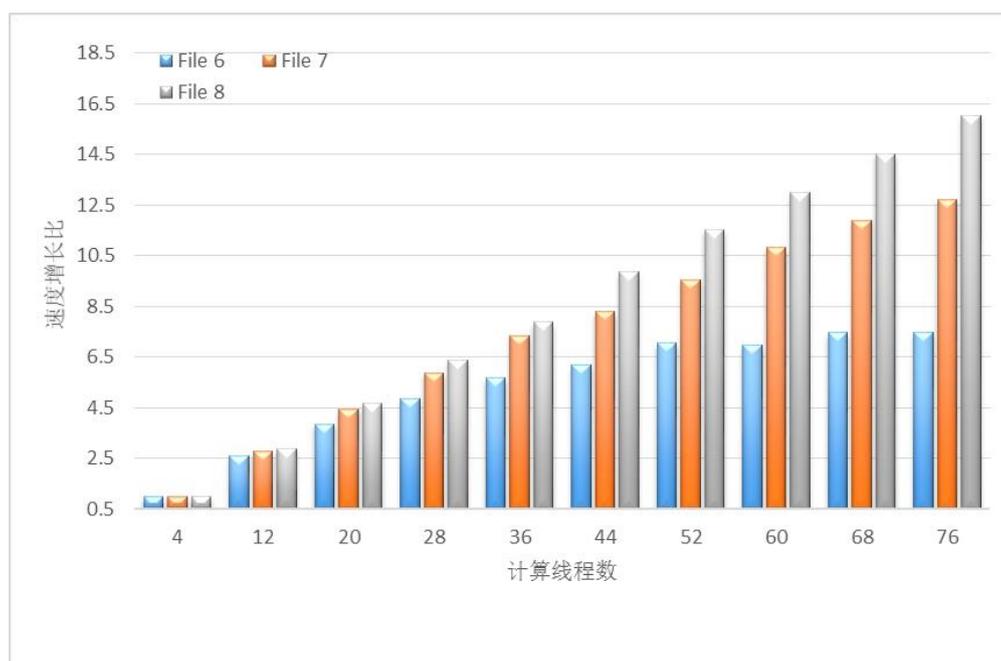


图 6.2 File 6 到 File 8 效率提升趋势

根据实验结果可以得知，利用集群的高效性是评价任何分布式系统好坏的重

要标准之一，尤其是使用按需付费的 IaaS 服务，就像本实验使用的 Amazon 云一样。确保添加计算节点到整体集群中时，无论数目多少，都能做到弹性高效可扩展。图 6.2 清楚地表明，文件大小小于块的大小的情况下，是不会获得性能提高。例如测试文件(File 1)主要原因是因为只有一个块来处理，这就是只有一个 Map 任务被映射出去，这样不能体现出集群的能力。测试文件 2、3 和 4 获取任务槽时，Map 任务槽数目超过了需要执行的 Map 数目（所对应的分别为 21，28 和 43），虽然得到了性能提升，但还不是最佳效果。当处理大文件时，就像文件 6/7/8 时，效率提升明显，跟单节点相比，效率至少可以提升 16.4 倍。

本实验中测试文件没有表现出性能随着计算节点数目增加而激增很多。这是因为在 Hadoop 平台下，MapReduce 任务数目没有达到一定数目，并没有提高过多效率。造成这种情况主要有两个因素：

任务开销：每一个 Map 任务运行会导致启动和关机时间的开销，以及过多数目的小文件造成了 Map 任务开销过大，减弱了效率。

任务过量：执行最后一波 Map 任务包含了许多不同的执行时间累加和，但是总消耗时间是依赖于集群整体性能和计算节点数目。加速的方式是通过增大或减小 APU 的大小，反过来就会造成本地机器上一些资源浪费。本实验可以将 Map 任务的大小缩小，变成更多的 APU 块来提高性能，但是就造成了过多的启动/关闭类型的开销。

对于本实验来看，展现可扩展性的最好用例是大文件 7 和 8。在每个 Map 任务进行操作时，设定适合的分块策略，使其能更好的适应实验集群，达到最佳效果。本实验环境最多可以进行 152 组 Map 和 Reduce，但是本实验最多只进行 76 组 Map 和 Reduce，原因是因为系统有自身开销，不能完全占用。但是也可以提升书目，只要不超过系统使用率的 90%，都是可以的。虽然启用任务量的增加，效果还是会增加更多。

6.3.2 视频质量评估

分析经过转码云平台转码后的视频的质量，本文选择使用 PSNR(Peak Signal to Noise Ratio) 方法进行测试，也就是均方误差的算法表示。PSNR 是评价图像和视频相关信息的客观标准。其数学表述方法为：

$$PSNR = 10 \times \log_{10} \left[\frac{(2^n - 1)^2}{MSE} \right] \quad (1)$$

其中 MSE 表示是原图与处理图之间的均方误差。也就是说 PSNR 越大，所代表的就是失真越少。

本文处理方法主要是比较原图和处理后的图进行帧比对和评估，并且将其分为低、中和高三种清晰度帧，然后通过并行的方式进行处理和比对。基本可以看到的是 PSNR 的标准值与平均值还是趋于相同的，可以保证一定的质量。

本实验是选取两个转码后的指标和信息，一个是固定码率系数模式 CRF (Constant Rate Factor)，一个是平均目标码率模式 AB (Average Bitrate)，通过测试这两个指标来评价输出质量。两个指标用处不同，CRF 的目标是维持整个视频的效果，而 AB 针对整个文件的特定的输出比特率。AB 经常受限于宽带的带宽等问题，如果换到一个带宽较大，可以通过更多数据量的时候，AB 可以保持其质量并防止视频回流，CRF 的作用也与其类似。但是 CRF 主要是用于存储那些在本地的二级存储的文件，类似于在线视频网站的应用。

CRF 模式可以使文件达到你期望它达到的质量，该方法可以为其提供最大的压缩率，转码效果可以达到较高水平。AB 方式的使最终文件达到你所要求的特定平均比特率。使用两边编码模式(2-pass)可以提高这种模式能效果。具体的结果见表 6.5 和表 6.6，表中正数代表提升，负数代表下降。通过下表中的文件 1 的结果可以发现，发现显示的不是结果，而是不是用。是因为文件 1 是一个多码流的文件，并且附带很多其他子文件，无法完成实验内容。

表 6.5 CRF 的 PSNR 差值

序号	文件大小	CRF 的 PSNR 差值计数			平均&标准差 CRF	
		LOWER	SAME	HIGHER	AVERAGE	STD. DEV
1	4.6 MB	不适用			不适用	
2	183 MB	17%	66%	17%	0.0017dB	0.0838 dB
3	237 MB	42%	20%	38%	0.8858 dB	0.2762 dB
4	364 MB	29%	48%	23%	-0.0872 dB	1.1720 dB
5	798 MB	25%	49%	26%	0.0018 dB	0.2501 dB
6	1.54 GB	40%	12%	48%	0.0307 dB	0.8126 dB
7	6.32 GB	33%	34%	33%	-0.0193 dB	1.3575 dB
8	10.52 GB	34%	36%	30%	-0.0185 dB	0.2972 dB

平均值： 0.1215dB 0.5855dB

表 6.6 AB 的 PSNR 差值

#	文件大小	AB 的 PSNR 差值计数			平均&标准差 AB	
		LOWER	SAME	HIGHER	AVERAGE	STD. DEV
1	4.6 MB	不适用			不适用	
2	183 MB	50%	1%	49%	-0.0631 dB	0.8877 dB

3	237 MB	45%	0	55%	-0.0026 dB	0.9663 dB
4	364 MB	44%	0	56%	0.0315 dB	1.5454 dB
5	798 MB	44%	0	56%	0.1986 dB	1.3757 dB
6	1.54 GB	44%	0	56%	1.1836 dB	4.5563 dB
7	6.32 GB	41%	0	59%	0.7456 dB	3.3247 dB
8	10.52 GB	44%	0	56%	0.0385 dB	1.9690 dB

平均值： 0.3406 dB 1.8904 dB

当编码器使用 CRF 时，很显然，在本文提出的输入块方式下，产生的平均质量并没有很大的提升。基本上大多数文件都是趋于平均 PSNR 差值，并且基本上都是小于 0.05dB。CRF 的 PSNR 差异的标准偏差为低，表明大部分的差异是接近平均和低水平帧的数目有提高或视频质量下降。当使用 AB 编码，非常明显的可以发现所产生的平均质量不会被我们的分块策略造成很大的影响。然而，进行 CRF 时，几乎每一帧针对于转码器转换出的帧都会造成或多或少的干扰，从而产生更大的标准偏差导致整体视觉质量降低。如图 6.3 所展示的，测试文件的内容是一个从文件 4 选取的 10,000 个帧，对其编码时使用 AB 和 CRF。AB 和 CRF 基本持平，这两种方式的基本趋于相似。

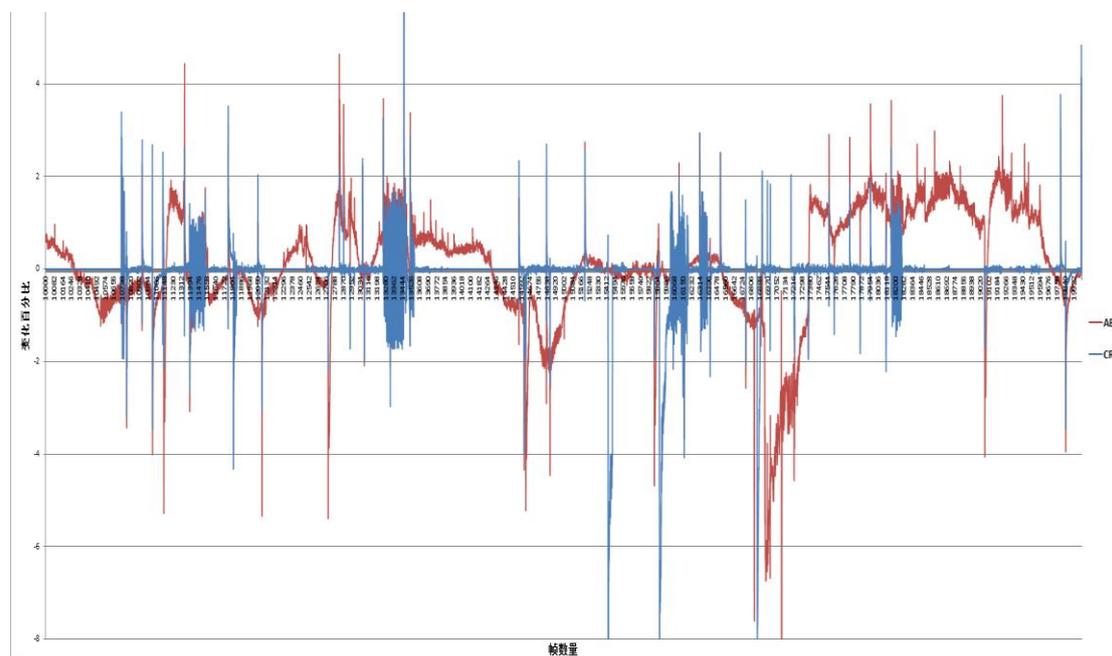


图 6.3 AB 和 CRF 编码对比

从本实验的输出文件是依靠 CRF 和 AB 编码两种类型进行分析的。我们比较了多个用例进行测试，通过实验发现，修改块的大小和块类型可以提高整个架构的速度和视频质量。块策略也成为了整个体系中尤为重要的一部分。

AB 编码依赖于前序编码的数据，并在不影响平均输出比特率信息的前提下

去优化效果。其主要原因是因为它只有一帧的数据存放在的之前的块中，而不是整个文件。正如图 6.3 一样，这也就造成了 AB 编码下输出的帧质量好于 CRF 编码。不同的人给视频的评价是不同的，没有办法做到统一协调。在本实验中，通过 PSNR 来进行客观的评定。在这个处理过程中，只能进行端到端的数据处理，不能随意更换转换位置。结果表明，各个视频之间并无质量过大的差异。

6.3.3 双尾 T 检验分析

T 检验 (T-Test) 是一种最为常见的假设检验方法，主要是验证总体得到的值是否存在较大差异。为了确保在视频质量分析期间，本实验得出的有关质量方面的差异没有错误，本文选择双尾 T 检验对数据进行概率分析。

1. 视频数据选择

总共比较 2 组数据，每组数据中含有 1 个视频，每个组进行转码后的数据和参考序列进行比对。第一组是使用的 CRF 编码的视频，而第二组是使用 AB 编码的视频。每个视频选取都是随机的，这样就确保了数据结果的客观性。

2. 视频属性与统计规则

每组实验随机的选取 8 个视频中的一个，并复制一份，两个一样的视频分别做 CRF 和 AB 编码，每个视频都带有自带属性，从色彩、视觉质量、音频质量和视频/音频并行度四个方面去评判。根据用户感受来加分，每个选项加 1 分，并对每组视频做三轮测试。总共是 8 组视频，做完后计算平均值，通过平均值来生成下表内的数据。

如表 6.7 所示，不同的切片的平均值和标准差，并且对 T 值进行双尾 T 检验。通过选取所有的总得分的平均值，可以获得足够的数据来执行双尾 T 检验。T 检验主要证明了无论是哪种视频，经过检验差距都并不是很大。不仅仅是视频类型决定了实验的结果，其实最主要的因素是转码前后尽可能的保持了影音文件的完整性以及相关参数的一致性。因此可以了解到经过转码后的视频和转码的参考序列没有很大的区别。

表 6.7 双尾 T 检验分析

	视频(CRF) MapReduce	视频(CRF) Reference	视频(AB) MapReduce	视频(AB) Reference
平均值	0.71	0.62	0.82	1.26
标准差	1.08	0.83	1.17	1.37
T 检验	0.34		-0.87	

第七章 总结与展望

7.1 总结

本文深入研究了 Hadoop 视频转码优化,具体工作包括:(1)分析 MapReduce 的计算架构并设计了全新的 S_Mapreduce 计算框架;(2)优化 FFMPEG 视频转码技术中的解码部分;(3)整合 S_MapReduce 和 FFMPEG 两个架构来提高整体的转码性能;(4)将视频流进行拆分以进行并行化处理,这提高整体性能并且成功验证了可以使用 S_MapReduce 的方式来提高视频转码的效率;(5)给出了一种有效的分块策略。

本文提出的架构在处理大文件时展示出的良好实验结果证明本文工作顺利达成了优化视频转码的目标。实验中使用 AB 编码和 CRF 编码分析视频并进行对比,可以发现转码后的文件与源文件相比,在转码质量没有很大落差的条件下提高了转码速度。

当使用计算集群来进行转码任务的时候,集群的多点计算能力可以帮助提高集群整体性能。本文实验表明随着数据量的增加集群特性的优势就逐渐增加。根据实验数据可以得到结论,转码速度可以提高至少 10 倍。但是不能过量的增加计算节点,这样不仅会造成浪费而且又会增加计算节点选择的时间。当使用本文输出的文件和参考序列的对比时可以明显的发现,当复用和合并过程被包括其中的时候 Reduce 过程的效率得到提高。如果用于处理的输入文件可以被分为几种不同的类型和不同类型的质量水平,那么复用过程可以进行一次,然后再进行后续使用。复用操作的开销基本是不变的,所对应的执行时间在整个转码过程总时间中的占比是在缩减的。当合理分块策略和计算机群个数设定合理时,本文的贡献在于可以使整个集群有良好的可扩展性和弹性。

7.2 展望

通过对本文的研究和撰写工作可以发现尚有许多待研究的问题,未来的工作可以分为两个方面。第一,未来需要努力实现双向编码方式,通过双向编码可以更好的提高转码的整体性能,并且可以做到提高压缩质量。通过本文的阐述,可

以证明 S_MapReduce 可以作为优秀的转码计算架构，虽然增加了相对应的复杂度，但是效率还是有显著的提升。如果将双向编码融入到系统中，就可以使整个平台的可扩展性得到提升，提升弹性负载能力；第二，需要调研 Hadoop2.0 版本，通过后期的努力可以将现今的 Hadoop 版本融入到整个系统中来解决资源分配的问题，通过使用双向编码可以做到提升编码质量和压缩程度。

参考文献

- [1] Buyya R, Yeo C S, Venugopal S, et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility[J]. Future Generation Computer Systems, 2009, 25(6):599–616.
- [2] Li Q T S S C, Taiyuan Iron amp Steel. Cloud Computing and Grid Computing[J]. Shanxi Electronic Technology, 2010.
- [3] Foster I, Zhao Y, Raicu I, et al. Cloud Computing and Grid Computing 360-Degree Compared[C]// Grid Computing Environments Workshop, 2008. GCE '08. IEEE, 2008:1–10.
- [4] Thain D, Tannenbaum T, Livny M. Distributed computing in practice: the Condor experience: Research Articles[J]. Concurrency & Computation Practice & Experience, 2005, 17(2–4):323–356.
- [5] Lewis T G, El-Rewini H. Introduction to parallel computing [M]. China Machine Press,, 2003.
- [6] Uhlig R, Neiger G, Rodgers D, et al. Intel virtualization technology[J]. Computer, 2005, 38(5):48–56.
- [7] Oshins J, Allsop B, Thornton A J. Configuration space virtualization: US, US8700816[P]. 2014.
- [8] Langmead B, Schatz M C, Lin J, et al. Searching for SNPs with cloud computing. [J]. Genome Biology, 2009, 10(11):: R134.
- [9] Zissis D, Lekkas D. Addressing cloud computing security issues[J]. Future Generation Computer Systems, 2012, 28(3):583–592.
- [10] Ahmad I, Wei X, Sun Y, et al. Video transcoding: an overview of various techniques and research issues[J]. IEEE Transactions on Multimedia, 2005, 7(5):793–804.
- [11] C.-H. Chen. Mohohan: An on-line video trans-coding service via apache

hadoop. [Online]. Available: <http://www.gwms.com.tw/TRENDAhadoopinTaiwan2012/1002download/C3.pdf>

[12] F. Yang and Q.-W. Shen, "Distributed video transcoding on hadoop," *Computer Systems & Applications*, vol. 11, p. 020, 2011

[13] R. Wilson. x264farm. A distributed video encoder. [Online]. Available: <http://omion.dyndns.org/x264farm/x264farm.html>

[14] M. Kim, Y. Cui, S. Han, and H. Lee, "Towards efficient design and implementation of a hadoop-based distributed video transcoding system in cloud computing environment," *International Journal of Multimedia and Ubiquitous Engineering*, vol. 8, no. 2, Mar. 2013.

[15] A. Garcia, H. Kalva, and B. Furht, "A study of transcoding on cloud environments for video content delivery," in *Proceedings of the 2010 ACM multimedia workshop on Mobile cloud media computing*, ser. MCMC '10. New York, NY, USA: ACM, 2010, pp. 13–18. [Online]. Available: <http://doi.acm.org/10.1145/1877953.1877959>

[16] A. Garcia, H. Kalva, and B. Furht, "A study of transcoding on cloud environments for video content delivery," in *Proceedings of the 2010 ACM multimedia workshop on Mobile cloud media computing*, ser. MCMC '10. New York, NY, USA: ACM, 2010, pp. 13–18. [Online]. Available: <http://doi.acm.org/10.1145/1877953.1877959>

[17] Baker M A, Dalale P, Chatha K S, et al. A scalable parallel H.264 decoder on the cell broadband engine architecture[C]// *IEEE/ACM International Conference on Hardware/software Codesign & System Synthesis*. ACM, 2009:353–362.

[18] Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters.[J]. In *Proceedings of Operating Systems Design and Implementation (OSDI, 2004, 51(1):107–113*.

[19] White T. Hadoop: The Definitive Guide[J]. O'reilly Media Inc Gravenstein Highway North, 2010, 215(11):1 – 4.

- [20] Mell, Peter, and Tim Grance. "The NIST definition of cloud computing." (2011): 20–23.
- [21] Moreno I S, Garraghan P, Townend P, et al. An Approach for Characterizing Workloads in Google Cloud to Derive Realistic Resource Utilization Models[C]// Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on. IEEE, 2013:49–60.
- [22] Palankar M R, Iamnitchi A, Ripeanu M, et al. Amazon S3 for science grids: a viable solution?" Pp[J]. Faculty Publications, 2008:55–64.
- [23] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107–113.
- [24] The Hadoop Distributed File System by Konstantin Shvachko , Hairong Kuang , Sanjay Radia , Robert Chansler
- [25] Taylor R C. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics[J]. BMC Bioinformatics, 2010, 11(6):3395–3407.
- [26] Thusoo B A. et al .Hive–apetabytescaledatawarehouse using Hadoop[C]// In Proceedings of the 26th IEEE ICDE. 2010.
- [27] Lassnig M, Garonne V, Dimitrov G, et al. ATLAS Data Management Accounting with Hadoop Pig and HBase[C]// Journal of Physics Conference Series. Journal of Physics Conference Series, 2012:956–958.
- [28] Okorafor E, Patrick M K. Availability of JobTracker machine in Hadoop/MapReduce ZooKeeper coordinated cluster[J]. Advanced Computing, 2012, 3(3).
- [29] Jeffrey D, Sanjay G. MapReduce: simplified data processing on large clusters[J]. Communications of the Acm, 2004, 51(1):107–113.
- [30] Wu J, Hong B. Multicast-based replication for Hadoop HDFS[C]// 2015 16th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). IEEE Computer Society, 2015:1–6.

-
- [31] Stonebraker M, Abadi D, DeWitt D J, et al. MapReduce and parallel DBMSs: friends or foes?[J]. Communications of the Acm, 2010, 53(1):64-71.
- [32] Cohen J, Cohen J. Graph Twiddling in a MapReduce World[J]. Computing in Science & Engineering, 2009, 11(4):29 - 41.
- [33] Plimpton S J, Devine K D. MapReduce in MPI for Large-scale graph algorithms[J]. Parallel Computing, 2011, 37(9):610-632.
- [34] 刘智国, 张雅明. 浅谈 FLV 视频格式[J]. 电脑知识与技术, 2008(20).
- [35] Gall D L. MPEG: A Video Compression Standard for Multimedia Applications. [J]. Communications of the Acm, 1991, 34(4):46-58.
- [36] Greenbaum E. Lisping Copyleft: A Close Reading of the Lisp LGPL[J]. International Free & Open Source Software Law Review, 2013.
- [37] Wacha J B. Taking the Case: Is the GPL Enforceable?[J]. Santa Clara Computer & High Technology Law Journal, 2005, 21(2):451-492.
- [38] Group A V T W, Schulzrinne H, Casner S, et al. RTP: A Transport Protocol for Real-Time Applications[J]. International Journal of Computer Applications, 2003, 2(2):459-482.

致谢

三年的硕士时光转瞬即逝。任凭时光飞逝，浓厚的师生情和友谊却长存。遥想三年前的我，接到了录取通知书是多么的欣喜，一切的一切都重现在眼前，不曾忘却。当来到了吉林大学的时候，心情是无法用言语来形容的，向往着学习生活、向往着校园活动、向往着科学研究。一切的一切都是那么的有吸引力，那么神秘。回首过去的三年，心中不免感慨，还有就是不可避免的伤感。感谢老师和同学对我的帮助，是你们给我指明了前进的方向，是你们教会了我做学术先学会做人的道理。

首先，我要对胡亮老师表示崇高的敬意与无尽的感谢，如果没有胡老师当年的指点迷津也不会有现在的我。也是胡老师给了我在网格计算与网络安全实验室学习的机会。胡老师无论在学术上还是做人上都努力做到尽善尽美，您是我一生的导师、一生的榜样。

其次，感谢我的导师于秀峰老师。感谢于老师当我在科研道路上遇到问题的时候为我指点迷津，帮助我在未来发展方向上选择的更加理性。

特别感谢车喜龙和赵阔老师，无论在学术上还是生活上都是尽职尽责的好老师，对我严格要求、细致指导、使我受益匪浅。感谢我读研期间陪伴我的同学们，感谢同学们营造了一个开放式、自学习式的实验室氛围，提高了我的表达能力和自学能力。尤其感谢孙承旭同学、付韬师兄和已经毕业的王海师兄，这三位挚友无论是生活上还是学习中都对我起到了巨大的正面作用，是我今后人生中的挚友，学习的榜样。

最后感谢我最亲爱的父母和家人，是你们的信任和支持给了我坚持研究下去的动力，是你们给了我信心和希望，你们辛苦了。

作者简介及在校期间所取得的科研成果

宋扬，男，汉族，1990年出生于吉林省长春市。2013-2016年在吉林大学计算机科学与技术学院攻读硕士学位，所学专业为计算机系统结构，研究方向为云计算，指导教师是于秀峰副教授。

本文作者在硕士期间，以第一作者身份发表1篇EI论文。

论文列表：

Yang S, Hao P, Hu J, et al. A New MapReduce Framework Based on Virtual IP Mechanism and Load Balancing Strategy[J]. Open Cybernetics & Systemics Journal, 2015, 9(1):253-261.

项目内容：

国家科技部项目，分布式部署的多媒体信息存储云研究与应用

吉林省重大科技攻关项目：物联网关键技术及应用(2011ZDG007)