**密级： 保密期限：**



**硕士学位论文**



**题目：基于混合存储的监控视频云计算平台中数据分布策略研究与应用**

**学 号： 2015110747**

**姓 名： 高阳阳**

**专 业： 计算机科学与技术**

**导 师： 马华东**

**学 院： 计算机研究院**

**2017年 11月 30日**

独创性（或创新性）声明

本人声明所呈交的论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢中所罗列的内容以外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京邮电大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

申请学位论文与资料若有不实之处，本人承担一切相关责任。

本人签名： 日期：

关于论文使用授权的说明

本人完全了解并同意北京邮电大学有关保留、使用学位论文的规定，即：北京邮电大学拥有以下关于学位论文的无偿使用权，具体包括：学校有权保留并向国家有关部门或机构送交学位论文，有权允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，有权允许采用影印、缩印或其它复制手段保存、汇编学位论文，将学位论文的全部或部分内容编入有关数据库进行检索。（保密的学位论文在解密后遵守此规定）

本人签名： 日期：

导师签名： 日期：

基于混合存储的监控视频云计算平台中数据分布策略研究与应用

摘 要

随着视频监控系统朝着大规模、海量数据和智能化的方向发展，基于云计算技术和智能视频分析技术构建监控视频离线分布式处理平台成为一种趋势。另一方面，基于SSD和HDD的混合存储架构以其大容量、低成本和高性能等优点被越来越多的云平台使用，用于实现云平台的高的数据读写性能。由于视频处理任务属于资源消耗型，视频任务处理过程存在大量的I/O操作，因此，构建基于混合存储的监控视频离线分布式处理平台很有必要。

然而，传统的监控视频离线分布式处理平台在分布式处理海量视频数据的过程中或者没有考虑集群节点负载和存储资源的动态性特点，或者没有考虑视频任务本身的资源需求以及集群节点间计算和存储能力异构性特点，容易造成集群高性能存储媒介SSD资源利用率低、集群负载不均衡等问题，进一步导致降低视频任务的处理效率。因此，如何提升基于混合存储的监控视频离线分布式处理平台中SSD资源的高效利用，保证集群的负载均衡，实现一个高性能的监控视频分布式处理平台是本文的研究重点。

首先，本文深入分析了当前主流智能视频处理算法，在实验基础上提出了一种面向监控视频数据块处理的时间预测模型 （PTPM），该模型综合考虑视频数据块特性如分辨率、帧率、时长以及集群节点存储和计算能力等，能够预测视频任务在不同节点上的处理时长。其次，本文提出了一种基于PTPM的视频数据初始放置策略（IPDS），该策略在进行视频数据初始放置过程中采用最小负载优先放置原则，将同属于一个视频处理任务的视频数据块作为基本的放置单位，对于每一个待放置的视频数据块，通过每次选取当前负载最小的计算节点进行放置来降低集群中各个节点的初始负载差异。同时，考虑视频任务分布式处理过程中节点存储资源和负载动态性特点，本文还提出一种视频数据动态迁移策略（LADM），该迁移策略包括一个节点内视频数据迁移方法（NLDM）和一个节点间视频数据迁移方法（CLDM），其中NDLM会周期性将本地HDD上的合适的视频数据迁移到本地SSD上进行放置等待后续处理以提升节点SSD存储资源利用率和节点任务处理效率，CLDM负责周期性将高负载节点上的合适的视频数据块迁移到低负载节点上放置以降低视频任务处理过程中的节点负载差异。最后，本文实现了基于Docker容器技术的视频浓缩算法功能镜像，使用交通数据作为数据源，对所提出的视频数据初始放置策略IDPS和视频数据迁移策略LADM进行了验证，实验表明IDPS和LADM能有效提升基于混合存的监控视频离线分布式处理平台中SSD的资源利用率，保证集群的负载均衡以及大大提高视频任务的处理效率。

关键词：云计算，混合存储架构，数据分布，数据迁移，离线视频处理

基于混合存储的监控视频云计算平台中数据分布策略研究与应用【翻译】

ABSTRACT

（空一行）

KEY WORDS:

目录

[第一章 绪论 1](#_Toc500768861)

[1.1 研究背景和意义 1](#_Toc500768862)

[1.2 国内外研究现状 3](#_Toc500768863)

[1.3 论文的主要研究内容 5](#_Toc500768864)

[1.4 论文组织结构 7](#_Toc500768865)

[第二章 相关技术介绍 9](#_Toc500768866)

[2.1 Docker相关技术 9](#_Toc500768867)

[2.1.1 Docker及其框架简介 9](#_Toc500768868)

[2.2.2 Docker容器集群技术 11](#_Toc500768869)

[2.2 混合存储体系结构相关技术 13](#_Toc500768870)

[2.2.1 HDD设备特性 13](#_Toc500768871)

[2.2.2 SSD 设备特性 13](#_Toc500768872)

[2.2.3 基于SSD和HDD的混合存储系统结构 14](#_Toc500768873)

[2.3 云平台数据分布相关技术介绍 17](#_Toc500768874)

[2.3.1基于元数据管理的数据分布 18](#_Toc500768875)

[2.3.2 基于算法管理的数据分布 19](#_Toc500768876)

[2.4 本章小结 21](#_Toc500768877)

[第三章 基于混合存储的监控视频离线分布式处理平台系统总体设计 22](#_Toc500768878)

[3.1 需求分析 22](#_Toc500768879)

[3.2 视频监控系统介绍 23](#_Toc500768880)

[3.3 基于混合存储的监控视频离线分布式处理系统架构设计 24](#_Toc500768881)

[3.3.1 总体架构设计 24](#_Toc500768882)

[3.3.2 工作流程设计 26](#_Toc500768883)

[3.4 核心模块设计 27](#_Toc500768884)

[3.4.1 本地视频数据迁移模块设计 27](#_Toc500768885)

[3.4.3 全局视频数据迁移模块设计 28](#_Toc500768886)

[3.4.4 核心数据结构设计 28](#_Toc500768887)

[3.4 本章小结 30](#_Toc500768888)

[第四章 基于混合存储架构的云平台中数据分布策略的实现 31](#_Toc500768889)

[4.1 基于Apache Hadoop的监控视频云平台数据分布策略 31](#_Toc500768890)

[4.1.1 Apache Hadoop默认的数据分布策略 31](#_Toc500768891)

[4.1.2 解决方案 32](#_Toc500768892)

[4.2 视频数据块处理时间预测模型 33](#_Toc500768893)

[4.3 监控视频数据初始放置算法 35](#_Toc500768894)

[4.3.1 数据放置算法 36](#_Toc500768895)

[4.3.2 数据放置算法实现 36](#_Toc500768896)

[4.4 监控视频数据块迁移策略 39](#_Toc500768897)

[4.4.1 相关变量定义 40](#_Toc500768898)

[4.4.2 NLDM本地视频数据迁移算法实现 41](#_Toc500768899)

[4.4.3 CLDM全局视频数据迁移算法实现 43](#_Toc500768900)

[4.5 本章小结 47](#_Toc500768901)

[第五章 系统实现及测试 48](#_Toc500768902)

[5.1 系统环境配置 48](#_Toc500768903)

[5.1.1 系统开发环境 48](#_Toc500768904)

[5.1.2 系统硬软件配置 48](#_Toc500768905)

[5.2 系统功能实现 49](#_Toc500768906)

[5.2.1 视频数据初始放置功能实现 49](#_Toc500768907)

[5.2.2 本地视频数据迁移模块实现 49](#_Toc500768908)

[5.2.3 全局视频数据迁移模块实现 53](#_Toc500768909)

[5.2.4 视频浓缩服务镜像实现 55](#_Toc500768910)

[5.3 系统功能与算法效果测试 55](#_Toc500768911)

[5.3.1 系统功能验证 56](#_Toc500768912)

[5.3.2 PTPM准确性验证试验 56](#_Toc500768913)

[5.3.4 IDPS初始数据放置算法效果验证试验 57](#_Toc500768914)

[5.3.5 LADM数据迁移算法效果验证试验 59](#_Toc500768915)

[5.4 本章小结 62](#_Toc500768916)

[第六章 总结与展望 63](#_Toc500768917)

[6.1总结 63](#_Toc500768918)

[6.2 展望 65](#_Toc500768919)

[参考文献 66](#_Toc500768920)

[致谢 71](#_Toc500768921)

[作者攻读学位期间发表的学术论文目录 73](#_Toc500768922)

# 第一章 绪论

## 1.1 研究背景和意义

近年来，随着智慧城市建设和人们公共安全意识的提升，视频监控系统已经成为城市安防建设的重要保障，随着视频监控系统规模越来越大以及智能视频处理技术如目标检测与跟踪、交通违章检测、车牌识别等在视频监控系统中的应用越来越广，视频监控系统开始朝着大规模、海量数据和智能化的方向发展。传统的基于单机的监控视频离线处理平台越来越难以应对当今海量视频数据的计算处理需求。基于云计算技术构建分布式的监控视频计算系统逐渐成为了监控视频离线处理平台的一种新型的解决方案[1,2]。

然而，以大数据为基础的分布式云计算平台对于数据的读写性能有一定的要求[3]，智能视频监控云计算平台更是如此。不同于传统的针对文本处理任务的云计算平台，智能视频监控云平台主要提供面向海量视频数据的分布式计算，而大多数的智能视频处理任务如视频浓缩、车牌识别等属于资源消耗型任务，其在执行过程中不仅会涉及到大量的逻辑运算，同时也伴随着大量的磁盘I/O操作，伴随着CPU/GPU技术的不断升级，集群的I/O能力成为了钳制提升视频计算任务处理性能的瓶颈。

当前，SSD（固态硬盘）由于其低能耗、存储密度大以及高的I/O性能等优点，成为了云计算平台主要的存储媒介之一[4]，基于SSD构建云计算平台可以有效提升整个平台的读写性能，进而提升整个平台的处理效率。然而相比于HDD（机械硬盘），SSD的存储容量小且单位存储空间的花费高，仅仅使用SSD作为整个云计算平台的存储媒介会导致云计算平台的构建和维护成过高。通过结合HDD和SSD，利用HDD的大容量、低成本以及SSD低能耗、高I/O性能等特性，构建基于HDD和SSD的混合存储架构的云计算平台，成为了目前主流的提升云计算平台处理性能的解决方案[5,6]。

然而，基于SSD + HDD的混合存储架构模式构建云计算平台，会进一步加剧集群资源的异构性。针对基于混合存储架构的云计算平台，如何制定高效的数据分布策略，实现平台中计算以及存储资源，尤其是SSD的高利用率，对于提升平台整体的计算性能至关重要，也是目前研究的热点问题[4-6],[10-13]。另一方面，由于视频数据的特殊性以及视频处理任务相对于文本处理任务的复杂性，使得制定高效的针对基于混合存储架构的监控视频离线处理平台的数据分布策略变得更加具有挑战性，主要面临以下两个难点:

（1）如何制定视频数据的初始放置策略，以减少各个节点间的初始负载差异。在分布式视频任务处理场景中，一个视频任务的最终完成时间由处理时间最长的节点确定，如果在视频数据初始放置时，不考虑各个节点的负载差异，一方面会造成处理能力弱的节点分配的视频数据量大，而处理能力强的节点分配到的视频数据量偏小，导致延长整个视频任务的处理时间，另一方面，节点初始负载差异过大也会增加视频任务处理过程中的数据迁移成本，由于传输视频数据会占用大量的网络传输资源，进一步导致降低整个视频处理任务的处理效率[14]，因此制定合理的视频处理的初始放置策略对于提升整个平台视频任务的处理效率至关重要。

（2）在视频任务处理过程中，如何根据各个节点的实时负载大小制定跨节点的视频数据的全局迁移策略，以实现整个集群的负载均衡。在视频任务处理过程中，各个节点可利用的SSD和HDD的存储空间大小随时间不停地发生变化，由于不同节点之间的计算能力和存储能力不同，并且各个节点的初始放置的视频数据量大小也可能不同，导致在视频任务处理的过程中，各个节点的负载差异会随着时间的变化越来越大，最终负载最高的节点会延长整个视频任务的完成时间。因此如何实现一个动态的全局视频数据迁移策略，使得能够根据各个节点的负载值以及可利用的存储空间大小将负载高的节点上的视频数据迁移到负载低的节点进行处理，以实现在视频任务处理过程中降低各个节点之间的负载不均衡程度和最小化整个视频任务的完成时间，对于提升整个计算平台的处理性能至关重要。

目前大多数的云计算平台采用Apache Hadoop构建[7]，其中Hadoop分布式文件系统（HDFS）作为底层的存储系统用于存储需要处理的海量应用数据，而分布式处理框架如Spark、MapReduce负责对HDFS上存储的海量数据进行分布式计算处理。然而，当前的Apache Hadoop平台默认采用循环轮询方案（round-robin）作为其数据初始分布策略，而不考虑节点存储介质本身的特性（SSD还是HDD）[4]，并且，Hadoop默认采用的负载均衡策略只考虑在任务处理过程中各个节点的负载率一致，而不考虑视频任务本身的特性以及节点存储介质异构性[8][9]，采用hadoop构建的视频云计算平台进行视频任务处理，可能会导致将计算敏感性的视频处理任务分配到计算能力弱的节点上，而将IO密集型的视频任务分配到IO能力弱的节点上，从而导致整个视频任务的处理效率低下。另一方面，HDFS在进行数据存储时，为了实现高容错以及方便上层计算框架如Spark/Mapdeduce的并行计算等，会将视频文件按照默认的数据块（Block）大小（64M）进行分割备份并保存在不同的计算节点，这导致在进行离线监控视频任务的处理过程中，即使是针对同一个视频文件，也不可避免地需要进行网络传输，而大量视频数据在网络中的传输会占用大量网络资源，进一步导致系统性能下降。

综上，使用Apache Hadoop构建基于混合存储架构的视频监控云计算平台不能保证对高性能存储介质的高效利用，并不是构建基于混合存储架构的离线视频分布式处理系统的理想方案。随着Docker应用容器引擎的开源，基于Docker的容器技术以其细粒度的资源分配、轻量级、一键式的部署分发流程（build one, run anywhere）以及易于扩展等优点成为了云计算领域的新贵，已逐渐成为主流的云计算技术之一[15,16]。同时，随着Docker官方的容器集群化管理技术Docker Swarm[17]以及Google开源的容器集群管理工具Kubernets[18]的日渐成熟，构建基于Docker容器技术的分布式计算集群不仅能够提供强大的计算能力，在资源管理、任务调度等方面也展现出了优异的表现[19]。另一方面，由于Docker Swarm和Kubernets在资源和任务调度相关源码实现上采用插件编写模式，并且Docker容器能够直接访问并获取宿主节点真实的物理资源状况，基于Docker容器技术构建分布式处理云平台可以更方便地扩展和集成自定义的数据分布和任务调度策略。因此，使用Docker容器技术构建基于混合存储架构的分布式监控视频离线处理平台很有必要。

## 1.2 国内外研究现状

云计算平台中的数据分布问题，随着大数据，尤其是视频大数据时代的到来，面临巨大挑战。如何制定合理的数据分布策略（包括数据放置策略以及数据迁移策略），优化云平台中的数据分布，对于提升云平台整体计算性能至关重要，国内外也有很多针对云计算平台中的数据分布策略的研究。

Dadi[20]等人主要考虑云平台中数据分布策略对于数据交付效率（将数据从集群存储设备分发给用户）的影响，提出一种基于P2P网络文件分发范式的数据分布方案，不同于现有大多数分布式存储系统中基于固定数目副本（比如HDFS中每一个数据块的副本数默认为3个）的数据分布策略，该策略可以在线性时间复杂度内求出每个数据块的最优副本个数以及其对应的具体的存储节点位置，从而实现数据交付过程效率的最优化。Poonthottam 等人[21]提出一种使用实时访问模式的Hadoop动态数据分布策略，这种新数据分布策略能够让任务更快的获取所需要的数据，从而实现访问时间和带宽的优化。最后，仿真实验表明这种数据策略的效果比HDFS默认数据放置策略更好。Kayyoor等人[22]提出了一种负载感知的云平台数据副本选择和放置算法，该算法将一组待查询数据项（可以是关系分区或者文件数据块等）上的预期查询工作建模为一个超图（hypergraph），并通过分析图与图之间的边连关系决定哪些查询数据项需要进行复制以及在哪些存储节点放置对应副本，通过优化数据副本分布，减少给定查询工作负载下获取相关数据项过程中涉及的存储节点个数，从而优化平均数据查询跨度，降低云平台环境中数据查询过程的总能耗。Zhang等人[23]提出了一种针对分布式离线视频处理平台中的视频数据放置算法，该算法采用批量分步放置策略，首先将待处理的所有的视频数据块按照关联度划分为不同的DBs，每一个DB是一系列关联度高的视频数据块集合，同时也是每一步进行数据放置的基本单元，该策略每一次选择一个DB放置在当前负载最小的节点上进行处理，最终保证各个节点的初始负载差异尽可能的小，从而有效降低了整个视频任务的完成时间。然而，以上这些研究提出的都是基于单一存储介质下的视频放置策略，并没有考虑云平台中存储介质的异构性，对于基于混合存储架构下的视频监控云平台，使用以上这些数据放置策略很容易导致高性能存储介质如SSD的存储资源利用率低，进而导致降低整个视频任务的处理性能。

近年来，混合存储架构下的云计算平台中数据分布策略的研究也得到了大家的关注，取得了一些研究成果。Kgil等人[24]提出了一种将SSD作为动态随机存储器（DRAM）和HDD之间的二级文件缓存的一种存储架构，当客户端发起一个数据读取请求时，会首先在DRAM中进行查找，如果命中数据就直接返回，否则根据闪存哈希表来判断请求的数据是否在二级缓存SSD中，如果在二级缓存SSD中，就直接返回客户端请求的数据，并同时将数据调入一级缓存DRAM中，否则将会从SSD缓存中选择一个数据块替换所缺失的数据块，并同时更新缓存哈希表，相比于传统的只有DRAM一级的缓存结构，该方法有效减少了磁盘的访问次数，提高了数据的命中率。Wan等人[25]提出了一种面向SSD+HDD混合存储架构的数据放置与迁移方法，该方法在进行数据的初始放置时综合考虑用户自定义的数据放置规则（比如在某个特定的时间段内某一数据块及其三个副本必须都放置在HDD上）和系统存储介质结构性特点，提出了一种基于用户规则（User polices）约束的数据初始放置策略并将其转化为数学线性优化问题，通过求解方程获得最优的数据初始放置策略，同时为了进一步提升数据的访问效率，该方法通过马尔科夫链的数据分类模型对数据的历史访问信息进行实时分析，将预测的未来一段时间访问频率高的数据对象迁移到SSD上进行放置，有效提升了混合存储架构下的系统数据访问性能。Chen等人[26]提出了一种基于SSD + HDD的高性能混合存储系统Hystor，该系统会周期性分析HDD上数据的用户访问日志，并将读写延时过长或者语义至关重要的数据块迁移到SSD进行存储，便于后续用户的读取。Yang等人[27]利用数据访问的空间局部性特点，将更新频率低（seldom changed） 并且读取频繁（mostly read）的参考数据块（reference data blocks）存储在SSD上，而将与参考数据块相对应的涉及频繁I/O操作的增量数据块以及相关的增量日志存储在HDD上，通过其提出的I-CASH智能检测和聚合算法，在用户进行数据写入操作时执行相应的数据相似性检测以及增量信息推导，将写入操作涉及的变更（或者增量）数据块存储在HDD上，并建立参考数据块和变更（或者增量）数据块的映射关系，在进行数据读取操作时将直接将增量数据块和对应的参考数据块相结合返回给用户。该方法即利用了SSD的高的数据读取性能的优点又避免了对SSD的频繁的随机写入操作，延长了SSD的使用寿命，同时利用数据块的强正则特性以及内容局部性特点，一个HDD磁盘块可以包含大量针对某些参考数据块的小规模的增量日志信息，使得一次HDD的读写操作可以同时完成多个I/O 请求，有效提升了平台的读取效率。Shi等人[28]提出了一种结合SSD、混合存储器（Combo Drive）以及HDD的分层混合存储系统，并同时提出了一种基于动态规划的多阶段数据分配算法，该算法在满足各个存储层空间资源约束条件下，通过周期性将低性能存储介质上的热数据（访问频率高）迁移到高性能存储介质上存储来优化数据的分布，提升整个存储系统的读取性能。

然而，以上这些研究在进行数据的优化分布过程中主要考虑通过提升高性能存储介质如SSD的利用率来实现高的读写性能，而没有考虑视频任务本身的资源需求特性以及大规模视频任务处理场景中各个节点的处理负载和可利用存储资源的动态性。不同于传统的文本处理任务，不同的智能视频分析任务在资源需求上呈现异构性，比如视频浓缩任务在处理过程中涉及大量的IO操作，属于IO密集型任务，而车牌识别任务在处理过程中涉及大量的CPU/GPU计算，属于计算密集型任务。在视频数据块放置过程中，如果不考虑视频任务本身的特性，很容易将计算密集型任务对应的视频数据块集合放置在计算能力弱的处理节点上，而将IO密集型的视频数据块集合放置在IO能力弱的处理节点上，从而导致任务资源需求和节点资源能力的不匹配，进一步可能导致增加视频任务处理过程中的网络数据传输量，降低视频任务的处理效率。另一方面，在视频任务处理过程中，各个节点的计算负载以及SSD和HDD的存储资源利用率随时间不停地发生变化，在进行视频数据放置时，如果仅仅以提升视频文件的读写性能为目标，而不考虑各个节点的处理负载以及各个节点的存储资源利用率，很容易导致存储性能高的节点上放置的视频数据过多，而可利用的存储资源大的计算节点上的视频数据却反而过少，导致即增加了节点之间的负载不均衡程度，也降低了节点的存储资源利用率，进一步地也延长了整个视频任务的处理时间。

## 1.3 论文的主要研究内容

基于SSD+HDD混合存储架构的监控视频离线分布式处理平台通过综合利用HDD的高容量、低成本以及SSD的高性能、低延迟特性来优化平台整体的读写性能，进一步提升平台视频任务处理效率。如何制定数据分布策略，保证集群各个节点SSD存储资源的高效利用和实现节点间的负载均衡对于提升平台视频任务处理效率至关主要。然而，目前云平台常用的数据分布策略即没有考虑海量视频任务处理场景中可利用存储资源以及节点负载的动态性特点，也没有考虑视频任务的资源需求特性，因此使用传统的方法构建基于混合存储架构的监控视频离线分布式处理平台，在进行海量视频数据的分布式处理过程中，无法保证高性能存储媒介SSD的存储资源的高效利用，也很容易出现节点负载不均衡情况，进一步地增加视频任务的处理时间。本文针对以上问题，通过分析当前主流的分布式计算框架以及智能视频处理算法，提出了一种基于视频任务处理时间预测模型（PTPM）的数据初始放置策略和数据迁移策略，并实现了基于容器技术的离线视频浓缩算法验证系统来验证算法的有效性。

主要研究内容有以下几点：

（1）监控视频离线处理任务时间预测模型

通过分析当前主流的智能视频处理算法，在实验的基础上提出了一种监控视频离线处理任务时间预测模型（PTPM），该预测模型通过结合视频数据块本身的特征，例如分辨率、帧率、时间长度等，视频处理任务的类型如视频浓缩、视频摘要提取、行为检测等，以及集群中各个节点的计算能力和存储能力等，在基于大量历史任务数据分析基础上，通过自适应的调整参数，建立时间估计函数，进而估算出视频数据块在不同计算节点上处理的花费时间，从而计算得出监控视频离线处理任务时间。

（2）一种视频数据初始化放置策略IDPS

基于PTPM时间预测模型，研究并实现了一种最小负载优先的视频数据初始放置算法。该数据初始放置算法将同属于一个视频处理任务的视频数据块作为基本的放置单位，对于每一个待放置的视频数据块，通过每次选取当前负载最小的计算节点进行放置来降低集群中各个节点的负载差异。

（3）一种视频数据动态迁移策略LADM

该策略包括一种基于线性规划的节点内视频数据动态迁移策略NLDM和一种负载感知的全局视频数据迁移策略CLDM。首先，对于每个计算节点，在其视频任务处理过程中，随着待处理视频数据块的减少，其节点的高性能存储媒介SSD的可利用的存储空间就越多。为了有效利用各个节点的SSD的存储资源，本文研究并实现了一种以最大化节点SSD率用率和提升节点视频任务处理效率为目标的基于0/1线性规划的本地视频数据迁移策略。在节点进行视频数据处理的过程中，通过周期性分析节点内各个存储媒介（SSD或者HDD）上剩余待处理视频文件的数量，节点内部各个存储媒介的读写速率和剩余存储空间大小、以及节点本身的计算能力等，建立以最小化节点处理负载为目标的0/1线性规划方程，通过求解方程来选取HDD上需要迁移到SSD上的待处理视频数据块，节点的本地数据迁移模块（Local Data Migration Module）根据所获取的待迁移视频数据块信息执行实际的迁移操作。该策略通过优化节点上视频数据块的分布，进一步提升了节点SSD利用率以及降低了节点视频处理任务的处理时间。另一方面，由于不同节点之间的计算能力和存储能力不同，并且各个节点的初始放置的视频数据量大小也可能不同，在视频任务进行过程中，节点之间的负载差异也就随时间的变化越来越大。为了保证视频任务处理过程中集群的负载均衡，进一步提升视频任务的处理效率，本文研究并实现了一种以实现集群负载均衡为目标的负载感知的全局视频数据迁移策略CLDM，该策略会首先对各个节点的负载大小分析，将集群节点划分为高负载节点集合 和 低负载节点集合，并通过综合考虑各个节点可利用的存储资源情况以及各个节点之间的网络传输带宽等因素，将负载高的节点上的合适的视频数据块迁移到负载低的节点进行后续处理，以使得在视频任务处理过程中降低整个集群的负载不均衡程度，同时提升了整个视频任务的处理效率。

（4）基于Docker容器技术的离线视频浓缩算法验证系统

基于以上提出的数据放置和迁移策略，实现了一种基于Docker容器技术的视频浓缩算法功能镜像，使用交通车辆监控视频作为数据源，对所述的视频数据初始放置算法和视频数据迁移算法进行了验证。该系统使用SSD + HDD的混合存储架构作为整个计算集群的存储后端，上层采用Docker容器技术实现视频计算任务的分布式处理。当用户提交视频摘要任务后，各个节点会通过视频浓缩算法功能镜像生成容器计算实例读取本地放置视频数据进行处理，同时系统会按照基于PTPM时间预测模型的视频数据初始放置策略和视频数据迁移策略进行数据的放置和移动，以优化集群中的数据分布，保证集群任务处理过程中的负载均衡和高性能存储媒介SSD的高效利用，大大减少视频浓缩离线任务处理的总时间。

## 1.4 论文组织结构

本论文将按照以下六个章节展开：

第一章：绪论。首先介绍了本文的研究背景，然后介绍了目前云计算计算平台中的数据放置策略的研究现状，最后介绍了本文的主要研究内容。

第二章：相关技术介绍。本章主要介绍构建基于混合存储架构的视频监控云处理平台的相关的一些背景知识，包括Docker及其集群构建相关技术，混合存储体系结构介绍等，最后对当前云平台常用的数据分布方案进行了详细介绍。本章为后续研究和平台开发奠定基础。

第三章：监控视频离线分布式处理系统架构。本章我们首先对智能视频监控云平台中的视频离线分布式处理系统进行需求分析，然后结合ITU-T标准介绍通用的视频监控系统，然后是监控视频离线分布式处理系统的架构设计、工作流设计以及核心组件设计。

第四章：云平台中视频数据放置策略研究，这部分是本文的核心。首先建立监控视频处理任务时间预测模型（PTPM），并详细介绍了建立模型的过程。然后介绍了基于PTPM时间预测模型的初始数据放置策略，并详细介绍该策略的实现过程，最后介绍基于CTPM时间预测模型的数据迁移策略，详细介绍该策略的实现过程。

第五章：系统实现与测试分析。首先，描述了对基于容器技术的离线视频浓缩算法验证系统中主要组成部分的详细实现；接着论述了视频浓缩算法功能镜像的实现；之后介绍了测试环境配置；最后基于视频浓缩算法功能镜像对本文提出的PTPM模型的准确性、视频初始放置策略（IDPS）、视频数据迁移策略（LADM）的性能进行实验测试，并对实验结果进行了分析。

第六章：结束语。对本文的所有研究工作做出总结，结合目前行业热点,展望监控视频云计算平台中数据分布策略未来的研究趋势和分析了本文提出的数据分布策略进一步的优化方向。

# 第二章 相关技术介绍

本章主要介绍了后续章节涉及的一些技术知识，主要包括与本文所优化的视频监控云平台相关的云环境相关技术，混合存储体系结构相关技术以及当前云平台中数据分布相关技术。

## 2.1 Docker相关技术

### 2.1.1 Docker及其框架简介

Docker是2013年由PaaS服务提供商dotCloud公司发起的一个开源项目，是一整套跨平台、可移植并且简单易用的容器解决方案，从发展至今已经成为主流的云计算技术之一[18]。Docker底层基于Linux提供的LXC轻量级虚拟化技术（如Namespace、CGroup等）来为容器提供资源隔离和安全保障。作为资源分割和调度的基本单位，Docker容器实例负责为每一个应用程序（服务）封装完全独立的运行时环境。通过对Docker容器实例生命周期的管理，Docker能够让应用的分发、部署和管理变得前所未有的高效和轻松。

Dcoker框架基本的架构如图2-1所示。



图2-1 Docker框架图

可以看到，Docker的实现采用松耦合的设计思想，不同的功能被封装成不同的组件，各个功能组件（模块）通过一套完整的机制对外提供服务，处理用户的请求。主要包括Docker Client，Docker Daemon， Dirver，graph，libcontainer等6大模块。

（1）Docker Client

Docker Client是Docker中负责接收用户服务请求的客户端模块。当前端用户发起容器服务请求后，由Docker Client负责与Docker Daemon之间建立通信，并将用户服务请求转发给Docker Daemon。当Docker Daemon处理完成当前的请求后，再由Docker Client负责将请求结果返回给用户。Docker Client主要通过tcp://host:port的方式和Docker Daemon之间建立通信,其中host和port分别为Docker Daemon进程所在宿主的IP以及Docker Daemon的监听端口，默认为2735。

（2）Docker Daemon

Docker Daemon是整个Docker框架中最核心的一个功能模块，它是一个常驻在宿主后台的系统进程，主要功能就是接受Docker Client发送过来的用户请求，将这些用户请求解析并转化为对应的系统调用完成相关的容器操作。Docker Daemon本身主要由Docker Server和Docker Daemon两个子模块组成，其中Docker Server主要负责接收和解析Docker Client请求，将解析后的内容发送给Docker Engine。Docker Engine根据解析后的元信息拉取合适的功能镜像创建容器实例处理请求。

（3）Driver

Driver是Docker架构中的驱动模块。Docker Daemon接收到用户请求后，需要先将用户请求转化为对应的系统调用来实现容器实例的创建，同时在管理和销毁容器实例的过程中，也不可避免地需要进行大量的系统调用如获取Docker容器运行时信息，获取Graph的存储和记录等。为了将对系统操作的调用从Docker Daemon内部相关业务逻辑中抽离出来，Docker设计了一套针对系统调用的操作接口，这些接口按照功能主要划分为三类：用于容器管理相关的驱动execdriver,用于网络管理相关驱动networkdriver以及用于文件存储相关操作的驱动graphdriver。

（4）Graph

Graph主要用于在本地宿主保管已经下载的镜像文件以及记录容器实例和镜像之间的对应关系，主要由GraphDB和Repository两部分构成。其中GraphDB是一个运行在本地的小型图数据库（基于SQLite构建），主要用于记录节点的命名以及节点之间关联关系。Repository则是一个本地目录，用于实际存储所有已经下载的镜像文件的相关信息，包括容器镜像的元数据，容器镜像文件大小，以及该容器镜像所代表的具体的rootfs。

（5）libcontainer

libcontainer是Go语言实现的一个库文件，主要目的是通过该库来直接操作系统内核中与容器相关的API而不需任何其他依赖。

基于Docker技术构建服务（应用）具有以下优点优势。

（1）自动化测试和持续集成。Docker利用其Docker镜像可编程的特性帮助应用发开人员消除了线上线下的环境差异，保证了应用生命周期的环境一致性和标准化。开发人员使用Docker镜像实现标准开发环境的构建，开发完成后通过封装完整环境和应用的镜像进行迁移。因此，测试和运维人员可以直接部署软件镜像来进行测试和发布，大大简化了持续集成、测试和发布的过程。

（2）跨平台的可移植性。Docker具有良好的适配性，几乎可以运行在各种云平台中，使得应用开发者再也无需担心受到云平台的捆绑，同时也让应用多平台混合部署成为可能。目前支持Docker的IaaS云平台包括但不限于亚马逊云平台（AWS）、Google云平台（GCP）、微软云平台（Azure）、OpenStack等，还包括如Chef、Puppet、Ansible等配置管理工具。

（3）高效的资源利用及隔离。Docker是一种操作系统级别的虚拟化，与底层共享操作系统，没有管理程序（Hypervisor）的额外开销，性能更加优良，系统负载更低，在同等条件下可以运行更多的应用实例，可以更充分地利用系统资源。同时，Docker拥有不错的资源隔离与限制能力，可以精确地对应用分配CPU、内存等资源，保证了应用间不会相互影响

（4）组件可复用。Docker采用AUFS联合文件系统使得镜像变得可分层、可编程，镜像之间不再是相互隔离的，不同的镜像可以组合成一个新的镜像。每个镜像都带有一个唯一的标签，通过标签可以实现镜像的回溯，加载指定的镜像。镜像的制作和修改过程也变得很简单，只需编辑Dockerfile文件重新创建即可。

### 2.1.2 Docker容器集群技术

Docker技术的出现使得应用（服务）的分发、部署、打包等变得异常简单和高效，然而，采用单机模式的Docker容器技术来对外提供服务已经无法满足当下日益增长的业务需求，如何基于Docker技术构建和管理分布式的容器集群，以应对海量的业务处理需求，同时保证平台服务的稳定性和扩展性等至关重要，也是目前的研究热点[29]。例如Docker公司自己研发的关注于容器编排以及集群化部署和管理的组件（技术）就有Fig/Compose 、Machine、Docker Swarm[30]等，同时Apache的Mesos以及Google开源的Kubernets[19,31,32]也在容器集群化管理方面有很好的表现。本小节主要介绍Docker官方的容器集群管理技术Docker Swarm以及google开源kubernets容器集群管理技术，为后续章节实现容器化监控视频分布式处理平台提供理论基础。

Docker Swarm是Docker官方提供的一种Docker集群管理工具，由于使用标准的Docker API作为其前端用户的访问入口，因此用户只要有单机Docker的运维和开发经验就可以很快上手，学习和二次开发的成本低。同时，由于仅仅关注于容器集群的资管理和调度，Swarm本身非常轻量，占用的系统资源也非常少，并且Swarm引入了服务的概念，而不再将容器实例作为其集群基本的管理和调度单元，同时也不再需要额外的KV存储支持服务模型，使得集群的扩容缩容、服务发现、负载均衡以及滚动更新等功能实现更加容易。最后，由于Swarm各个功能模块基于插件化机制（Batteries included but swappable）实现,使得用户可以很容易进行功能扩展。

Kubernets是除Docker Swarm之外当下最为火热的容器集群管理技术，是谷歌内部大规模容器集群管理技术Borg的一个开源实现版本。它构建于Docker之上，允许用户创建和管理Docker容器集群，并为容器化集群提供包括资源调度、服务发现、运行监控、扩容缩容、负载均衡以及失败冗余等一系列完善功能。 同Docker Swarm一样，Kubernets也提出了服务的概念，并将服务（Service）作为其分布式集群架构的核心，由于Kubernets本身不限定任何编程接口，所以不论是使用Java、Go还是C++、Python进行服务的编写，都可以毫无困难地映射为Kubernetes的Service，并可以通过标准的TCP通信协议进行交互。如表2-1所示，是Docker Swarm和Kubernets技术的对比，可以看出，相比于Docker Swarm，Kubernets在设计模式和工程实践上具有好的优势。本文在构建容器处理集群时也采用Kubernets作为平台底层支撑的管理工具。

表2-1 Docker Swarm和Kubernetes对比

|  |  |  |
| --- | --- | --- |
|  | Docker Swarm | Kubernetes |
| 设计初衷 | 跨宿主集群的容器管理 | 支持分布式、服务化的应用框架 |
| 核心功能 | 管理节点、调度任务、服务发现、滚动更新、容器HA、通讯安全 | 资源调度、服务发现、服务编排、资源逻辑隔离、安全配置管理、自动回滚、job任务支持、内部域名服务、健康检查等 |
| 服务发现机制 | 通过Consul实现 | 内置 |
| 支持的容器类型 | Docker | Docker、Rkt |
| 经典案列 | SA Home Loans、宜信大数据、新浪微博 | Pearson、Box、PingCAP、美团、网易、华为、阿里巴巴等 |

## 2.2 混合存储体系结构相关技术

不同类型的存储设备之间在性能、容量以及价格等方面的差异很大，混合存储架构通过综合利用各个存储设备之间的差异性，通过将判定为语义重要或者访问频率高的数据放置在性能较好、存储容量小的存储设备，而将访问频率少的数据放置在存储容量大、性能低下的存储设备，实现在低成本下高的读写性能。目前在混合存储架构实现中常用的存储介质主要是机械硬盘和固态硬盘两种。

### 2.2.1 HDD设备特性

HDD全称Hard Disk Drive(机械硬盘)，简称硬盘，是目前被广泛使用的一种持久化存储媒介之一[3]，主要由磁道、扇区、柱面以及磁头数4个基本结构组成。磁盘的工作原理主要是通过利用特定的磁粒子的极性来记录数据。磁头在写入数据到磁盘的过程中，首先利用数据转换器将接收到的数据（比如电脑可识别的机器码）转换为电脉冲信号，随着磁盘的转动，磁头将不同的电脉冲信号转换为磁粒子的不同存储极性实现数据的写入。从磁盘读取数据的过程与此相反。磁盘读写具有对称性，并且在进行数据重写时使用的是原位更新的方式。

磁盘的读写性能主要和本身磁盘片的个数以及磁盘的旋转速率有关。通过增加磁盘片个数或者提升磁盘的旋转速率一定程度上可以提升磁盘的存储效率。然而，由于磁盘主要通过磁头的机械移动实现数据的读写，而磁头移动速度和数据的读写速度存在不只一个数据量的差距[34] ,使得基于磁盘设备进行数据的随机访问需要耗费较多的时间才能够完成指定操作。虽然通过利用磁性硬盘并发技术和磁头短距离移动技术可以提升磁性硬盘的随机访问性能，然而这两种技术都会使得磁性硬盘本身的存储空间利用率低，硬件成本增加，存储系统能耗变大，因此并不是工业界理想的提升磁盘随机读写性能的方案。到目前为止，虽然磁盘的顺序读写性能是可以被接收的，但是其随机读写性能并没有得到大幅度的提升，随着计算机CPU/GPU运算速度的不断提升，磁性硬盘的读写速率和计算机的运算速率之间的差距越来越大，磁盘读写性能成为了钳制计算机计算能力的重要瓶颈之一。

### 2.2.2 SSD 设备特性

不同于磁性硬盘，SSD全称Solid State Drive(固态硬盘)，是一种完全建立在半导体芯片如FLASH上的一种存储媒介。由于是一种纯电设备，数据读取和写过过程不涉及机械移动（操作），使得固态盘相比于磁性硬盘具有更高的读写性能，是目前比较理想的高性能存储媒介之一。固态盘的基本特性如下：

（1）I/O接口特性

虽然和磁性硬盘实现数据持久化存储和读取的原理完全不同，但是SSD的接口规范、功能以及使用方法却磁性硬盘保持一致，这隐藏了固态盘的独有特性，也使得计算机设备或者其他存储系统可以无缝地实现固态盘和磁性硬盘之间的数据迁移操作。

（2）读写特性

不同于磁性硬盘，固态硬盘在进行数据写入时，有一个写前擦除限制。固态硬盘被分为多个存储单元块，每个存储单元块则有许多页组成。固态盘按照页面单元执行数据的读取和写入操作。然而，在进行数据写入操作之前，固态盘会首先基于块单元进行数据的擦除操作，也就是说，如果需要对某个存储单元块中的页面进行写操作，需要提前将该块上的数据擦除，而不能通过原地覆盖更新的方式写入数据。由于写前擦除操作限制，导致固态盘的读写性能呈现非对称特性，其读取数据的速度快，而写入数据的速度慢。另一方面，固态盘也具有擦写次数限制。固态盘存储单元的物理结构中主要通过氧化物进行电场的创建。写入数据时，电子穿过氧化物存储电荷，记录一个电位值，即写入一个数据，擦除数据则会向相反方向进行。但是电子穿过氧化物的次数越多，氧化膜就好越弱，最终电场就不能阻止电子的自由活动了，这个存储单元也就失效了，因此，固态盘具有擦写次数限制，频繁的写操作会降低固态盘的使用寿命，而磁性硬盘则不存在这样的限制。

（3）物理特性

由于固态硬盘是一种纯电设备，不像磁性硬盘一样需要安装额外的机械设备（如磁头、磁臂）来实现数据的读写功能，因此，在相同的条件下，固态盘的能耗要比磁性硬盘的能耗低15倍左右，具有低能耗的优点。同时，由于是电子设备，没有额外的机械移动器件，因此不管是人为的物理摔震，还是计算机数据高速处理的场景，固态盘都能从容应对，具有防震抗摔的特性。最后，由于固态盘的制作器件一般都比较先进，使得同样存储容量大小的固态盘相对于磁性硬盘的来说具有更轻的重量，并且固态盘在工作状态也几乎不产生噪音。

### 2.2.3 基于SSD和HDD的混合存储系统结构

在前面两个小节，我们分别对两种主流的存储媒介SSD和HDD进行了简单的介绍，本小节我们主要介绍基于SSD和HDD的混合存储系统结构以及其优势，为后续章节实现基于混合存储架构的监控视频离线分布式处理平台的实现提供理论支撑。

一个理想的云存储子系统应该满足（1）能够在低成本前提下满足海量数据的存储需求。（2）能够具有海量数据规模下与计算性能相匹配的高性能的数据读写能力。然而，目前任何一种基于单一存储介质构成的存储系统都不能满足上述大容量、低成本和高性能等存储要求，这主要是由目前的存储媒介固有的物理特性决定的[35]。随着SSD技术的日趋成熟，结合SSD的高性能、低延迟和HDD的低成本、大容量的特点构建基于SSD和HDD的混合存储架构成为一种新型的满足云平台大容量、低成本和高性能读写性能要求的解决方案[10-13]。

目前主流的基于SSD和HDD的混合存储系统主要分为两种：（1）将SSD作为HDD的缓存（2）将HDD和SSD作为同一层级的存储设备。下面我们主要对这两种不同类型的混合架构进行介绍。

首先，我们介绍SSD作为HDD的缓存的混合存储架构，其架构图如图2-2所示。



图2-2 SSD作为HDD缓存的混合存储架构示意图

在该架构中，SSD的主要作用是用于缓存HDD上主数据的子集。当用户需要进行数据访问时，会优先在SSD中进行查找，如果SSD中已经缓存了当前需要的数据则直接返回给用户，否则就需要从HDD上查找并将查找结果缓存在SSD中以提升后续数据访问效率。根据写操作的不同，该架构又可分为只读缓存架构和读写缓存架构两种类型。其中只读缓存结构中，SSD只用于保存清洁的数据，一旦用户对HDD上的主数据进行了更新操作，其对应的在SSD上的缓存副本就会失效。读缓存结构则不同，用户对主数据更新的同时会相应的更新SSD上缓存的副本数据（如果SSD上当前已经有更新数据的副本的话）。进一步地，读写缓存结构又可划分为写返回缓存和写直达缓存。其中写返回缓存会将HDD上的写入缓存在SSD中，之后就始终保持其SSD上副本和HDD上原始副本的同步，消除HDD的写入。由于在写返回结构中，已经缓存在SSD上的副本数据和HDD上的最初的原始副本数据不一致，就需要增加额外的开销来保持数据的一致性，防止某些突发情况下如系统重启和断电等的数据丢失。写直达缓存结构直接转发写入操作到SSD和HDD中，由于同时在SSD和HDD上保存数据副本，所以写直达缓存结构以写入性能为代价避免了额外的元数据更新。

将SSD作为HDD的缓存的混合存储架构通过按需将用户访问的数据缓存在SSD中，基于SSD的低延迟和高读取性能的优点大大增加了后续用户访问相同数据的效率，然而该架构无法提升整个系统的存储容量，并且该架构比较依赖缓存策略的制定， 如果缓存策略制定不合理，缓存命中率过低，会导致无法有效利用SSD的存储资源，降低系统的数据访问效率。

将HDD和SSD作为同级存储设备的混合存储架构是目前更为普遍的一种实现高性能数据访问性能的方案[35]，其架构图如图2-3所示，主要原理是将SSD和HDD按照同级别的存储设备，进行统一编制。存储容量是SSD和HDD的存储容量之和。在该混合存储架构中，数据或者放置在SSD上或者放置在HDD上，而不能同时放置在两个存储媒介中，即每个数据仅被放置在一个存储媒介中。为了最大程度提升系统数据访问性能，该架构会基于系统提供的数据历史访问记录如数据读操作次数、数据写操作次数等将数据划分为“冷”数据和“热”数据两种类型，并通过周期性从HDD迁移“热”数据到SSD中进行放置，保证SSD上存储的都是频繁访问或者语义重要的“热”数据。



图2-3 SSD作为HDD同级存储设备的混合存储架构示意图

基于初始的分配策略的不同，该混合存储架构又可以分为“热”类型和“冷”类型以及“热冷互换”类型三种。其中在“热”类型架构中，所有的待处理数据都被初始放置在HDD上，SSD上的初始放置的数据为空。系统每隔一段时间就会将HDD上的判定为“热”类型的数据集迁移到SSD上进行放置。相反的，在“冷”类型架构中，待处理数据初始都被放置在节点的SSD上，HDD上初始放置的数据为空，系统每隔一段时间会将判定为“冷”类型的数据集迁移到HDD上进行放置。不同于“冷”类型和“热”类型，“冷热交换”类型的架构中，所有待处理的数据初始会被随机放置在系统的HDD或者SSD上，也就是初始情况下，HDD和SSD都放置有数据，然后系统每隔一段时间就会将HDD上判定为“热”的数据迁移到SSD上进行放置，同时将SSD上判定为“冷”的数据迁移到HDD上进行放置。

## 2.3 云平台数据分布相关技术介绍

在云平台离线分布式处理的场景中，海量数据在集群各个存储节点（或安装有存储设备的计算节点）中放置，集群计算节点并行从存储节点中读取与之关联的数据进行处理。如何进行数据的放置，优化集群中数据分布，对于提升集群的处理效率，保证数据处理过程中集群的负载均衡以及降低数据在不同节点之间的网络传输等至关重要。云平台在设计海量数据的分布（放置）方案时至少应该考虑以下三个因素：

（1）故障域隔离。云平台一般都运行在（跨多个机架）的物理节点组成的集群上，海量数据被分布在集群各个节点的物理存储设备如磁盘中。为了降低由于节点故障导致的数据丢失的概率，提升系统的容错性，在进行数据放置时，需要考虑将同一份数据的不同副本合理放置在不同的故障域中，比如将同一数据的不同副本放置在两个不同机架的存储节点中。

（2）负载均衡。云平台中各个存储节点的磁盘容量以及IO性能呈现差异化、异构性特点。在进行数据分布时，需要综合考虑各个节点的存储资源特性，保证数据均匀分布在各个不同的存储节点，避免出现数据分布不合理造成的某些节点过载而某些节点空闲的情况。

（3）最小化数据迁移量。一方面，随着集群中新的节点的加入或者由于某些原因导致的已有的节点从集群中移除，需要将一些数据从当前存储节点迁移到新的节点进行放置；另一方面，在云平台并行处理数据的过程中，由于各个节点的计算以及存储能力的不同，随着任务的进行，其对应的不同存储节点上剩余的待处理数据量也不同，此时，也需要进行数据的迁移操作，将剩余数据量过多的节点上的数据迁移到剩余数据量少的存储节点上放置。为了避免过多的数据迁移操作导致的集群性能下降，在进行数据分布时，需要综合考虑集群节点的缩容扩容能力以及集群负载动态性，保证数据迁移过程中尽量迁移较少的数据量。

目前在云平台中常用的数据分布方法主要分为两大类：基于元数据管理（Metadata management）的数据分布方案和基于算法管理（Algorithm management）的数据分布方案[36-38]。

### 2.3.1 基于元数据管理的数据分布

基于元数据管理的数据分布方法通过引入元数据机制，在进行数据放置时，首先需要通过集群元数据服务器或者用户来指定待处理数据的具体的存储位置，然后将待处理数据和其具体的存储位置之间的映射信息存储在元数据中，由元数据管理服务器统一管理。该方法可以根据具体的用户需求精确地进行数据的放置，比如用户可以指定将确定的某个数据的副本放置在某个具体的存储设备上。目前GFS[39], HDFS[40]以及Openstack Swift[41]等云平台中比较常见的分布式存储系统都采用基于元数据管理的数据分布方法。

以HDFS分布式文件系统为例。HDFS集群如图2-4所示，其存储节点主要有NameNode和DataNode构成，其中NameNode主要负责存储如名字空间、访问控制信息、数据块（Block）的具体存储信息等元数据，而DataNode负责存储具体的用户数据。



图2-4 HDFS数据分布示意图

在进行数据放置时，HDFS会首先将用户数据分割为固定大小的数据块（Block），默认为64MB，然后将这些Blocks按照默认的副本放置策略放置到不同的DataNode中，同时并将对应的元数据信息存储在NameNode中进行管理。其中HDFS在进行数据块的副本放置时为了提高数据容错率，降低机架故障导致的数据丢失，默认采用的是简单的机架敏感（rack awareness）的副本放置策略。其具体的放置策略如下：

（1）将数据块的第一个副本放置在提交数据块的节点上，并将对应的数据块ID、机架ID等信息报告给NameNode节点进行存储。

（2）将该数据块的第二个副本随机放置在本机架上除去放置第一个副本数据的节点的其他任意一个DataNode节点上，并将对应的数据块以及存储设备元信息汇报给NameNode节点。

（3）将数据块的第三个副本随机放置在与前两个副本放置的节点所在的机架不同的机架中的任意节点上，并同时向NameNode汇报对应的具体的存储相关元信息。

数据块副本放置完成后，集群的NameNode就存储了整个集群的数据映射信息，各个DataNode定期通过心跳通知NameNode自己的健康状态，一旦发现有节点或者机架出现故障，NameNode通过遍历查询元数据表（Metadata Table）获取故障节点上相关存储信息，并将其存储的数据块迁移到网络传输代价最小或者空闲的健康节点进行存储。

基于元数据管理的数据分布方案通过元数据服务器统一进行数据存储位置信息的管理，用户在进行数据访问时只需通过查询元数据表即可快速获取具体的存储信息，同时该方案也支持用户精确制定具体的数据存储位置，对于存储大量需要进行批量处理的数据是一种不错的数据分布解决方案。然而，在针对文件大小不固定、并且小文件规模比重很大的海量数据存储时，该方案会导致系统维护的元数据表过大，一方面使得难以合理利用系统存储空间，同时由于元数据信息过于庞大，查找元数据信息这一过程的效率下降，导致系统整体的数据访问性能受限于元数据服务器的带宽和计算资源，进一步地对系统的扩展性造成影响。

### 2.3.2 基于算法管理的数据分布

基于算法管理的数据分布方案摒弃了集中式的元数据管理机制，而是通过使用确定性的数据分布算法（策略）来计算出用户数据（以及副本）的具体的存储位置。系统在进行数据放置时，通过将如表示数据的唯一的标识符、集群存储节点的拓扑结构、确定的数据存放规则等作为算法输入参数，通过运行算法来得到具体的存储位置信息。只要算法输入不改变，则系统的任何节点都可以根据该算法获取特定数据块的具体存储信息，而不需要进行元数据的管理和查询操作，并且该方法可以实现将计算分散到各个节点。目前在Dynamo[42]、GlusterFS[43]、ceph[44-45]等分布式文件存储系统中被采用。

基于算法管理的数据分布方案中采用的数据分布算法主要包括基于一致性Hash的数据分布算法，基于CRUSH算法的数据分布策略以及改进的基于弹性Hash算法的数据分布策略等[36,37,42]，其中基于一致性Hash的数据分布算法是目前基于算法管理的分布式文件存储系统中最常用的数据分布算法，其算法基本思想如下：

首先将哈希值空间看做一个收尾相连的圆环，然后通过特定的哈希函数对存储节点的唯一特征值（通常是存储节点IP）做哈希，将其映射到哈希圆环的特定的位置上，在进行数据写入时，假设每个待存储数据具有唯一的键值，同样地，首先通过相同的哈希函数基于数据的键值进行对其哈希，并将结果映射到相同的圆环上，然后从该数据的哈希值映射的位置开始按照顺时针方向出发，遇到的第一个存储节点即为该数据的存储位置。数据的读取采取同样的方式。另一方面，当有存储节点加入或者移除时，该算法通过分担沿圆环顺时针方向距离最近节点上的数据来实现新节点的存储资源的利用和降低已有节点的存储负担。

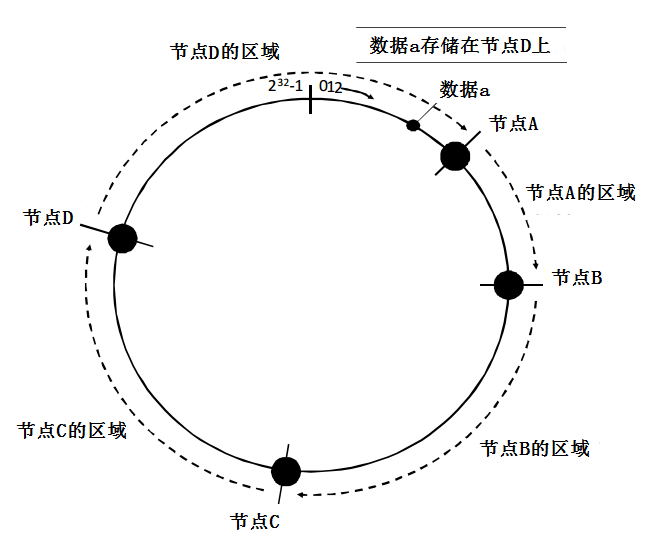


图2-5 基于一致性Hash算法的数据分布示意图

最后，为了实现集群存储节点之间的负载均衡，避免突发情况下某些节点负载过高的情况，比如某个存储节点失效，该节点数据被移动到其顺时针方向邻近的第一个健康节点上存储，导致该节点上数据量翻倍。该数据分布算法在一致性Hash思想的基础上提出了虚拟节点（Virtual Node）的概念[42]。算法会维护一个虚拟节点和物理节点映射列表，在进行节点映射时，通过哈希函数将虚拟节点而不是真实的物理存储节点均匀映射到哈希环上，每一个虚拟节点可以对应多个物理节点。并通过调整物理节点和虚拟节点之间的映射关系来实现数据的调整。虽然该算法也需要维护一份虚拟节点和物理存储节点的映射关系表，但是该映射表只维护简单的虚拟节点和存储节点的映射关系，并不需要太多的存储和计算资源，不需要使用单独的元数据服务器进行维护。而GlusterFS则通过改进后的弹性Hash算法和文件目录在存储池中定位数据，不需要额外的虚拟节点和物理存储节点映射关系表。

基于算法管理的数据分布方案相对于基于元数据集中管理机制的数据分布方案来说，由于减少了管理以及检索元数据的资源需求，并且可以将计算分散到各个节点，提高了系统资源利用率，也提升了系统的扩展性。

## 2.4 本章小结

本章首先介绍了Docker云计算相关技术，包括Docker技术框架以及Docker集群相关技术，然后介绍了混合存储结构相关技术，主要是介绍了主流的基于SSD和HDD的混合存储架构类型，最后介绍了分布式云平台中常用的数据分布方法。本章节主要为后续章节打下理论基础。

# 第三章 基于混合存储的监控视频离线分布式处理平台系统总体设计

本章我们主要介绍基于混合存储的监控视频离线分布式处理平台的设计与实现过程。首先给出当前智能视频监控云平台中视频离线分布式处理系统的需求分析，然后简单介绍我们将要使用的视频监控系统的组成以及各个组成部分的作用，接着给出了本论文提出的基于混合存储架构的监控视频离线分布式处理系统的架构设计和工作流设计，并详细说明了所述监控视频离线分布式处理系统架构核心组件设计，最后是本章小结。

## 3.1 需求分析

如今，随着Web 2.0的发展，基于IP网络的视频监控系统被广泛应用于各个行业（如交通、安全、环境、城市管理、卫生）。这些视频监控系统通过其部署的前端摄像机无时无刻不在进行视频数据的采集, 以满足同一时刻数以百万计的移动或者终端用户的监控视频服务需求，以中国电信全球眼系统为例，其单个系统部署的高清摄像头的数量就多达2万个，每一个高清摄像头每一秒采集的视频数据大约6M左右，其2万个高清摄像头一天产生的视频数据量就高达1280TB。视频监控系统开始朝着大规模、海量数据方向发展。另一方面，为了避免全人工的从海量视频数据中提取有价值的语义（关键）信息，智能视频分析算法如视频浓缩、车牌识别、越界检测得到了快速发展并迅速在视频监控领域得到了普及，这些视频分析算法能够自动化地针对监控视频数据进行分析并从中获取有价值的语义信息，大大降低了人工地对海量监控视频数据进行分析获取信息的成本和有效提升了从海量监控视频数据中获取信息的效率。视频监控系统呈现智能化的发展趋势。

随着视频监控系统向着大规模、海量数据以及智能化的方向发展，基于单机的视频离线处理平台已经无法应对其海量视频数据的存储和计算需求，结合云计算技术和智能视频分析技术构建分布式的智能监控视频离线处理云平台成为目前主流的应对视频大数据处理问题的解决方案。并且，由于大多数的智能视频处理任务如视频浓缩、车牌识别等属于资源消耗型任务，其在执行过程中不仅会涉及到大量逻辑运算，同时也会进行大量的磁盘I/O操作，为了进一步提升云平台的I/O性能，实现海量数据规模下于云平台计算能力性匹配的数据读写性能，提升云平台的视频任务处理效率，构建基于SSD和HDD的混合存储架构的监控视频离线处理云平台至关重要。

## 3.2 视频监控系统介绍

视频监控系统是一种专注于视频应用技术的通信系统，其通过所部署的前端摄像机远程捕获多媒体（例如视频，图像和各种报警信号）数据，并基于管理的宽带网络以友好的方式将其呈现给终端用户，并保证多媒体数据的质量以及多媒体数据传输过程中的安全性和可靠性。根据国际电信联盟标准化部门的相关标准（ITU-T）[46]，一个标准化的智能视频监控系统如图3-1所示，主要由六大模块组成，它们分别是中央管理单元（CMU），业务控制单元（SCU），媒体内容分发单元（MDU），媒体内容存储单元（MSU），房屋单元（PU）和客户单元（CU）。



图3-1 视频监控系统架构

其中，中央管理单元CMU是整个视频监控系统的核心单元，负责视频监控系统各个功能组件（单元）之间的协同调度和服务管理。SCU用于访问PU和CU之间的服务控制和信号呼叫控制。 MDU用于将多媒体数据从PU传输到CU，MDU的主要功能包括多媒体信息接收，多媒体数据处理，媒体路由，媒体传输，媒体转发，媒体复制等等。MSU用于检索，存储媒体，并提供流媒体服务能力。MSU主要有网络存储设备NVR组成，NVR通常用作本地存储设备，用于连续记录来自网络摄像机的监控视频。由于预定义的资源限制，NVR具有最大的存储容量或吞吐量限制。例如NVR可以同时从16个摄像机接收视频流，并且可以为这些摄像机记录一个月的视频数据。此外当NVR出现故障时，数据可靠性难以保证。PU是视频监控系统中的前提子系统，它实现多媒体信息捕获，多媒体编码，报警信号输出和多媒体网络传输的功能。PU通常是IP摄像机等各种视频监控设备，其可以通过网络将实况视频流传输到其他系统组件。CU是视频监控系统内的客户单元，用于向最终用户呈现多媒体信息（例如视频，图像和报警信号）。

## 3.3 基于混合存储的监控视频离线分布式处理系统架构设计

### 3.3.1 总体架构设计

我们的监控视频离线分布式处理系统基于Docker容器虚拟化技术构建，不同于我们之前研究工作中提出的基于单一存储架构下的容器化视频监控云平台，由于采用混合存储模式，本文构建的监控视频离线分布式处理系统在架构设计和工作流程都更为复杂。如图3-2所示是整个系统的总体架构图：



图3-2 基于混合存储的监控视频离线分布式处理平台架构

该系统根据其各自的功能的不同主要划分为五大功能组件（模块）。

（1）视频监控系统VS

VS部分的就是我们在3.2提到的视频监控系统。在VS中，PU负责不断地采集视频数据，MSU由众多的网络视频记录器（NVR）组成，NVR从PU接收视频数据流并将视频存储在其本地的物理存储设备上。同时，NVR可以通过网络访问接口支持视频数据管理的各种操作，如上传，下载和检索等。VS可以向Controller组件发送视频处理任务的请求来分布式的处理监控视频处理任务。

（2）Cluster Manager

集群管理器是整个系统的核心部分，是一个集中式的容器集群管理组件，底层基于google的Kubernets容器集群管理器，主要负责从VS接收视频任务处理请求，控制从VS本地存储设备拉取视频数据到容器分布式计算集群中放置，转发容器集群产生的处理结果到VS。同时，在容器集群进行视频数据块处理的过程中，集群管理器组件还负责控制全局视频数据迁移模块GDMM根据全局视频数据迁移策略CLDM（将在第四章节详细介绍）执行数据视频数据迁移任务。

（3）Container Cluster

容器集群由许多异构低廉的物理计算节点构成，是整个系统实现视频任务分布式处理的核心部分。每一个物理计算节点（简称为）SH-Server都配置有高性能低容量的存储媒介SSD和高容量低性能的磁盘存储介质HDD用于放置需要处理的视频数据块，同时每一个物理节点都配置有一个Docker计算引擎用来创建处理视频数据块的容器计算实例。另一方面，每一个物理节点都运行有一个节点管理器进程（组件）Node Manager，节点管理器组件主要负责从Cluster Manager接收分发的视频任务处理请求元信息并根据视频任务处理请求元信息控制节点Docker计算引擎拉取远程镜像生成容器计算实例进行视频数据块的处理，同时，节点管理器组件还负责控制本地视频数据迁移模块LGMM执行本地视频数据块迁移算法NLDM（将在第四章节详细介绍）执行单节点异构储存媒介间视频数据迁移任务。

（4）GDMM

全局视频数据块迁移模块（Global Data Migraion Module）的主要功能是在视频任务处理过程中周期性地执行全局视频数据迁移策略CLDM将高负载节点上的视频数据块迁移到低负载的节点上放置用于后续处理，优化视频数据块在容器集群中的分布。详细的设计细节将在本章节第4小结介绍。

（5）LDMM

本地视频数据迁移模块(Local Data Migration Module)主要功能是在视频任务处理工程中周期性执行本地视频数据迁移策略将本地节点HDD上存储的合适的视频数据块在满足存储空间约束的前提下迁移到本地高性能存储媒介SSD上放置等待后续处理，从而提升SSD的存储空间利用率和节点的处理性能。详细的模块设计细节将在本章节第4小结介绍。

（6）Image registry

镜像仓库是一个远程的存储中心，用于存储监控视频离线分布式处理系统需要的所有镜像文件，每一个镜像文件都封装有一个特定的视频处理算法（视频摘要处理、车牌识别、越界检测等），在开始视频任务分布式处理时，每个节点的Docker引擎会查看本地宿主是否有对应的生产容器计算实例的镜像文件，如果没有，则会从远程镜像仓库中拉取并保留在本地宿主，并通过镜像文件生成容器计算实例负责视频数据块的处理。

### 3.3.2 工作流程设计

在本部分中，我们专注于分布式系统中监视视频处理的工作流程。PU采集的监控视频数据通过网络传输存储到本地的MSU中，并且可以根据智能应用的请求进一步分析。监控视频离线分布式处理系统工作流程如图3-3所示：

首先，用户通过CU向VS发起监控视频离线处理任务请求，并通过任务请求页面设置该离线任务处理请求所涉及的一些重要参数，例如视频处理任务处理类型（视频浓缩、越界检测）、视频数据在MSU中的存储位置等，由VS系统分装这些信息后发动给Cluster Manager组件。比如用户在向VS发起监控视频浓缩离线任务请求时，需要在相关界面设置视频浓缩的一些参数，例如视频的浓缩比、感兴趣区域、每个画面同时出现的最大目标个数等。同时选择处理哪些视频数据。当用户提交处理请求后，VS会首先根据请求信息获取待处理视频数据在MSU中的具体存储位置等信息，然后将这些信息和用户提交的参数信息一起封装成约定的消息格式发送给Cluster Manager组件。

Cluster Manager组件接收到来自VS的处理请求消息后，一方面会对接收到的消息格式进行分析，从中获取待处理的视频数据在MSU中的具体存储地址、视频任务处理类型等元信息，同时还会获取Container Cluster中可用节点的相关元信息（如节点编号、节点CPU核数、节点SSD可利用空间大小、节点HDD可利用空间大小等）为后续进行实际的视频数据放置和分布式处理做准备，待相关视频数据元信息和节点元信息获取成功以后，Cluster Manager会启动监控视频任务处理时间模型（PTPM），利用视频数据相关元信息和集群节点相关元信息初始化PTPM模型。PTPM初始化以后，Cluster Manager会从VS中的MSU中依次拉取待处理视频数据块SVDB并调用视频数据块初始放置算法（IDPA）计算每一个待处理视频数据块在Container Cluster中的初始放置位置并按照计算结果将其加载到Container Cluster中的相应的物理节点进行放置。

视频数据块初始化放置完成以后，Cluster Manager会向集群中的各个节点发送广播消息，通知各个计算节点的Node Manger模块当前已经完成视频数据块初始化放置并请求开始处理视频数据块，Node Manager模块收到Cluster Manager的广播消息后控制Docker计算引擎生成容器计算实例开始读取并处理其节点上放置的视频数据块。

同时，在各个节点读取并处理视频数据块的过程中，Node Manager会控制LDMM模块周期性执行NLDM迁移策略将HDD上待处理视频数据块（尚未被处理）迁移到SSD上进行放置等待后续处理。Cluster Manger会控制GDMM模块周期性执行CLDM迁移策略将高负载节点上视频数据块迁移到合适的低负载节点上进行放置。

最后Container Cluster处理完所有的任务后，Cluster Manager组件会读取任务处理结果并将任务处理结果发送给VS，最后将处理结果呈现给用户。

## 3.4 核心模块设计

为了实现基于混合存储架构下的容器化监控视频离线处理平台中更高效的存储资源利用率，从而进一步提升系统的处理性能，我们对容器化监控视频离线处理平台进行了优化，架构图如图所示，主要添加了与监控视频数据优化分布相关的LDMM模块和GDMM模块。本小节主要介绍LDMM模块和GDMM模块的设计，并同时给出系统核心数据结构设计。

### 3.4.1 本地视频数据迁移模块设计

Cluster Manager控制完成视频数据块初始放置以后，Node Manger开始控制节点的Docker引擎生成容器计算实例读取并处理位于宿主SSD和HDD上的视频数据块，同时，为了进一步优化节点内视频数据分布，提升节点IO性能和高性能出存储媒介SSD利用率，每当处理完一个视频数据块, Node Manager就控制启动本地视频数据迁移模块执行一次节点内视频数据迁移操作。

本地视频数据迁移模块，运行在容器集群中的每一个物理节点上。如图3-3所示，其主要由三个功能子模块组成。



图3-3 LDMM模块示意图

（1）宿主存储资源监控子模块

宿主存储资源监控子模块主要用于获取当前宿主的存储资源使用情况，包括宿主当前SSD的剩余存储资源量，SSD已经使用的存储资源量，HDD的剩余存储资源量和HDD已经使用的存储资源量。然后将获取的宿主存储资源使用情况相关数据以特定的数据格式如JSON等发送给迁移策略制定子模块，迁移策略制定子模块根据获得的资源使用情况数据运行本地视频数据迁移策略LDMA制定合适的迁移计划。

（2）迁移策略定制子模块

迁移策略定制子模块是整个LDMM模块的核心组成部分，其内部实现了我们设计的LDMA算法，当接收到来自宿主存储资源监控子模块的数据后，它会首先基于宿主当前存储资源相关数据初始化LDMA算法，然后运行LDMA算法获取优化宿主视频数据块分布的迁移计划，最后将迁移计划发送给迁移操作执行子模块，迁移操作执行子模块根据获取的迁移计划执行实际的迁移操作。迁移操作定制子模块功能中使用的LDMA算法实现细节将在第四章节给出。

（3）迁移操作执行子模块

迁移操作执行子模块根据获取的迁移计划执行实际的迁移操作将放置在HDD上待迁移的视频数据块迁移到节点的SSD上进行放置等待后续处理。

### 3.4.2 全局视频数据迁移模块设计

全局视频数据迁移模块署运行在Controller节点，主要功能是在视频运行过程中通过将高负载节点上的视频数据块迁移到低负载节点上进行放置来优化集群节点数据分布。

不同于LDMM模块，GDMM主要只包括两个功能子模块：迁移策略制定子模块和迁移操作执行子模块，如图3-4所示。其中迁移策略制定子模块会周期性从后端数据库中获取集群节点信息，各个节点资源使用情况以及各个节点的视频数据块分布相关信息，然后通过运行全局视频数据迁移策略CLDM获取优化集群视频数据块分布的迁移计划，最后将迁移计划发送给迁移操作执行子模块。迁移操作子模块功能和LDMM中类似，在这里不做累述。



图3-4 GDMM模块示意图

### 3.4.3 核心数据结构设计

我们在3.3章节中介绍到当用户通过VS发起视频任务离线处理请求后，VS系统需要将用户请求相关元数据、视频数据块属性等传递给Cluster Manager组件，Cluster Manger组件需要能够从这些信息中获取视频数据块的存储位置以及视频数据块的视频特性等，以便于其从VS中拉取视频数据块并通过我们提出的数据块初始放置策略IDPA将拉取的原始视频数据文件合理地放置到Container Cluster中的各个节点。

为了便于描述视频数据块的相关信息，我们设计了一种描述视频数据块本身属性的数据结构SVDB，如表3-1所示：

表3-1 视频数据块属性描述结构

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 变量名 | SVDB\_Id | MSU\_address | SVDB\_r | SVDB\_f | SVDB\_st |
| 变量类型 | int | int | String | int | double |
| 变量名 | SVDB\_et | SVDB\_job |  |  |  |
| 变量类型 | double | String |  |  |  |

其中，SVDB\_Id是SVDB数据结构的标识符，每一个SVDB的标识是唯一的；MSU\_address用来记录数据块SVDB在MSU中的存储位置，以便Cluster Manager组件能够在MSU找到该SVDB。SVDB\_r表示该视频数据块的分辨率，SVDB\_f表示视频数据块的帧率，SVDB\_st表示视频数据块开始的时间，SVDB\_et表示视频数据块结束的时间，SVDB\_job用来表示当前视频数据块对应的视频任务类型。

由于我们在进行视频任务处理过程中，对于每一个节点需要根据其各个存储媒介的存储资源使用情况以及其视频文件放置情况来制定合理的本地迁移策略，因此针对每一个节点，我们设计维护其各个存储媒介资源使用情况的数据结构STORAGE\_INFO如表3-2所示：

表3-2数据结构STORAGE\_INFO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 变量名 | Store\_type | Store\_used | Store\_avail | Store\_SVDBList |
| 变量类型 | string | double | J | List<SVDB\_id> |

数据结构STORAGE\_INFO中Store\_type表示存储媒介类型，它的值或者为S（表示SSD存储媒介），或者为H（表示HDD存储媒介）；Store\_used表示存储类型为Store\_type的存储媒介的当前已经使用的存储资源量；Store\_avail表示存储类型为Store\_type的存储媒介的当前尚未使用的存储资源量；Store\_SVDBList表示存储类型为Store\_type的存储媒介上放置的视频数据块列表,列表中的元素是SVDB数据结构的唯一ID标识，对应于一个特定的视频文件。

同时，我们设计一个用来维护每个节点信息的数据结构NODE\_INFO,该结构主要提供Cluster Manager模块用来维护每个节点的信息。NODE\_INFO的结构如表3-3所示：

表3-3数据结构NODE\_INFO

|  |  |  |
| --- | --- | --- |
| 变量名 | Node\_Id | Node\_SVDBList |
| 变量类型 | int | List<SVDB\_Id,Store\_type> |

其中Node\_Id是节点标识，每一个节点的标识符是唯一的，Node\_SVDBList是一个列表，对应于放置在节点标识为Node\_Id的节点上的所有的视频数据块，Node\_SVDBList中的每一项都是一个二元组<SVDB\_Id, Store\_type>,表示标识为SVDB\_Id的视频数据块存储在节点Node\_Id的类型为Store\_type的存储媒介中。

最后，我们设计数据结构CLUSTER，用来管理每个节点信息NODE\_INFO,其结构如表3-4所示：

表3-4数据结构CLUSTER

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| NODE\_INFO | NODE\_INFO | NODE\_INFO | …… | NODE\_INFO |

CLUSTER中维护所有计算节点的信息，相当于是实际容器处理集群的一个虚拟映射。方便Cluster Manger组件管理集群中的节点。另一方面，GDMM全局视频数据迁移模块可以基于CLUSTER和STORAGE\_INFO获取整个集群的实时的视频分布状态，便于全局视频迁移策略的制定。

## 3.5 本章小结

本章我们首先对智能视频监控云平台中的视频离线分布式处理系统进行需求分析，然后根据ITU标准介绍了典型的视频监控系统，我们介绍了将要使用的基于混合存储架构的监控视频离线分布式处理系统总体设计和工作流程设计，最后详细叙述了监控视频离线分布式处理系统的核心组件的设计。

# 第四章 基于混合存储架构的云平台中数据分布策略的实现

在上一个章节中，我们已经对智能视频监控云平台中基于混合存储架构的视频离线分布式处理系统架构进行了详细的介绍，但是在目前的智能视频云处理平台中，并没有可以直接运用在基于混合存储架构的视频离线分布式处理系统中的数据分布策略。为了提高基于混合存储架构的视频离线分布式处理系统中高性能存储介质SSD的资源利用率，以提升整个平台的I/O读取性能和视频任务处理效率，同时保证视频任务处理过程中集群的负载均衡。我们提出了一种基于视频数据块处理视频预测模型PTPM的视频数据块初始放置策略IDPA和一种视频数据块迁移策略，该迁移策略包括一种基于线性规划的节点内视频数据块迁移算法NLDM和一种负载感知的跨节点全局视频数据块迁移算法CLDM。本章将首先分析传统的基于Apache Hadoop的视频离线分布式处理系统中默认采用的数据分布策略的不足，然后根据视频数据块的特性提出一种精准的视频数据块处理时间预测模型PTPM，并基于该模型设计并提出一种最小负载优先的视频数据块初始放置策略和监控视频数据块动态迁移策略。

## 4.1 基于Apache Hadoop的监控视频云平台数据分布策略

### 4.1.1 Apache Hadoop默认的数据分布策略

目前大多数的云计算平台采用Apache Hadoop构建，监控视频分布式离线处理平台也不例外，例如Zhang等人[23]就提出了一种基于Apache Hadoop构建的监控视频分布式离线处理系统,该系统将从VS获取的待处理的海量视频数据放置在集群的HDFS中，通过上层的分布式处理框架Spark读取HDFS上存储的海量数据进行分布式计算处理。

然而，当前的Apache Hadoop平台默认采用循环轮询方案（round-robin）作为其数据初始分布策略，而不考虑节点存储介质本身的特性（SSD还是HDD）。并且，HDFS分布式文件系统虽然实现了负载均衡器（Balancer）的守护进程，该进程会检测集群中节点的负载情况，并将负载较大的节点上的数据迁移到负载较低的节点上来保证集群的负载均衡，然而该负载均衡策略只考虑在任务处理过程中各个节点的负载率一致，而不考虑视频任务本身的特性以及节点存储介质异构性，采用hadoop进行视频任务处理，可能会导致将计算敏感性的视频处理任务分配到计算能力弱的节点上，而将IO密集型的视频任务分配到IO能力弱的节点上，从而导致整个视频任务的处理效率低下。最后，HDFS在进行数据存储时，为了实现高容错以及方便上层计算框架如Spark的并行处理等，会将视频文件按照默认的数据块（Block）大小（64M）进行分割备份并保存在不同的计算节点，这导致在进行离线监控视频任务的处理过程中，即使是针对同一个视频文件，也不可避免地需要进行网络传输，增大了网络传输开销。我们知道传输视频数据的代价是很大的，大量的网络传输会导致集群内网络堵塞，浪费大量时间，导致视频任务处理时间大大增加。

### 4.1.2 解决方案

为了实现基于混合存储架构的监控视频离线分布式处理系统中视频数据的优化分布，提升系统高性能存储介质SSD的高效利用，优化系统的IO性能和保证视频任务处理过程中的负载均衡，我们在进行视频数据分布时不仅需要考虑节点的负载情况，还需要综合考虑节点的计算能力和存储能力异构性以及视频任务的资源需求异构性（IO密集型、计算密集型），这就需要我们制定新型的面向基于混合存储架构的监控视频离线分布式处理系统的数据分布策略（算法）。另一方面，为了降低视频任务处理过程中过多的网络开销，在进行视频任务处理过程中，也需要尽量将视频任务分配给存储该任务相关数据的节点上执行，即保证视频数据处理本地化。

虽然目前有很多针对HDFS的优化的数据分布策略的研究[47-51]，然而由于HDFS默认不支持实现自定义的数据放置（分布）方法，这些研究都是通过模拟实验将其实现的优化的数据分布策略和HDFS默认的数据分布策略进行对比，并没有实际集成到HDFS中。另一方面，Apache Hadoop默认也没有提供给用户获取宿主物理资源特性的接口。因此基于Apache Hadoop实现基于混合存储架构的监控视频离线分布式处理平台难以集成自定义数据分布策略，进一步地，也就无法保证混合存储架构下高性能存储介质SSD的高效利用。最后，由于HDFS默认采用基于块（Block）的机架感知的副本放置策略，导致不可避免的在视频任务处理过程中需要进行跨节点的数据读取，增加了视频任务处理过程中的网络开销。

为了实现更高效的面向基于混合存储架构的监控视频离线分布式处理平台的数据分布策略，一方面，在进行系统构建时，我们摈弃了传统的基于Apache Hadoop构建监控视频离线分布式处理平台的方法，而改用Docker容器技术以及google开源的Kubernets容器集群管理器作为实现分布式监控视频离线处理平台的核心技术支撑。基于Docker和Kubernets作为构建分布式监控视频离线处理平台的核心技术，不仅能够保证整个分布式平台在任务管理、资源调度等方面的可靠性，同时基于Docker可直接获取硬件资源信息的特性以及Kubernets任务调度器的插件化实现机制使得更容易集成自定义调度算法和数据分布策略，为提升整个平台视频处理任务的高效性提供了保障。详细的系统架构已经在第三章节进行了详细的描述。

另一方面，我们也实现了一套完整的面向基于混合存储架构的监控视频离线分布式处理平台的视频数据分布策略，主要包括基于视频数据块处理时间预测模型的视频数据放置策略和视频数据迁移策略。本章节后续部分我们将详细介绍我们的基于视频数据块处理时间预测模型的视频数据放置策略和视频数据迁移策略。我们的数据分布策略（包括数据初始放置策略和数据迁移策略）在进行视频数据块放置和迁移时充分考虑了各个节点存储和计算能力异构性以及节点负载的动态性等，优化了视频数据在集群中的分布，有效提高了集群高性能存储介质SSD的利用率，实现了集群的负载均衡以及大大降低了视频任务的完成时间。

## 4.2 视频数据块处理时间预测模型

针对大规模视频任务的分布式离线分析处理场景，在进行监控视频数据分布式处理的过程中，各个节点的处理负载是不断发生变化的，只有能预先根据集群中各个节点视频数据放置情况以及当前的视频任务处理类型等已知条件，精确估算出各个节点的处理负载大小，并基于此分析和制定合理的视频数据分布策略，优化视频数据在集群中的分布，才能进一步保证监控视频云计算集群在执行视频处理任务时的集群的负载均衡，减少集群内部的网络传输开销、降低集群内部的资源消耗以及提升集群中高性能存储介质SSD的资源利用率等。为此我们需要建立一个精准的视频任务处理时间预测。

然而，由于不同的视频文件可能具有不同的帧率、码率、分辨率以及时长，同时，不同类型的视频处理任务对计算资源和存储资源的需求也不同，导致不同的视频处理任务在同一节点的处理时长的差异可能很大，另一方面，由于集群节点本身计算和存储能力的异构性，即使是针对同一类型的视频任务中的同一个视频文件进行处理，在不同节点的处理时长也不相同，因此，如何构建一个针对视频处理任务的精确的处理时间预测模型是一个关键并且困难的技术点。

为此，我们分析了当前常用的智能视频处理算法，总结出了其算法执行过程中涉及的相同操作，并通过大量的历史视频数据进行了实验仿真，最终我们发现影响视频任务处理时间的主要因素主要包括视频数据块的时间长度、视频数据块的分辨率、视频数据块的帧率、集群中节点的计算能力、集群中节点的存储能力（I/O能力）以及视频任务处理类型这六点。下面我们将具体说明我们视频任务处理时间预测模型的建立过程。

由于采集摄像头的种类繁杂，采集到的视频数据的分辨率和视频数据的帧率都是多种多样的，例如摄像头采集到的数据分辨率笼统的划分有1024\*768、1440\*900、1280\*1024、1920\*1080等，摄像头采集到的数据的帧率也不尽相同，例如有20fps、25fps、27fps等等，我们将不同的分辨率定义为集合， 其中*m*表示集合*R*中共有*m*种分辨率，表示集合中第*m*种分辨率；定义集合 表示不同的帧率，其中*n*表示集合*F*中有*n*种帧率，表示集合F中第*n*种帧率。通过定义不同分辨率的集合和不同帧率的集合，我们提出了视频质量的概念，用大写字母*Q*表示，如公式所示：



集合*Q*是集合*R*和*F*的笛卡尔积，例如：,，那么。集合*Q*中的每一个元素，例如或者，表示一个标准视频质量级别，并用*QR*表示。

同时，我们定义表示不同的视频任务类型，如视频浓缩，越界检测等，其中表示集合Z中有种不同的视频任务类型。结合集合Q和Z，我们定义集合*S*如式所示：



集合*S*是集合*Q*和*Z*的笛卡尔积，集合S中的每一个元素代表特定的一种视频任务处理模型，表述为SVPM，比如，,那么。基于集合*S*，我们定义集合*D*如式所示：



对于集合S中的任意一个元素， 集合D中的元素代表其平均每处理单位时长的视频数据所产生的临时中间数据的大小。的值可以通过在集群中运行大量的SVPM为的SVDBs并通过计算每一个SVDB产生的临时数据大小的平均值获得。

假设集群中有N个计算节点，对于任意一个节点,我们定义式：



对于每一个节点,表示其平均每处理单位时长的视频数据所需要的CPU计算时间的大小，的值可以通过在节点上运行大量SVPM为的SVDBs并计算每一个SVDB的平均CPU计算时间的大小获得。

对于任意一个视频数据块SVDB，其在节点的总的完成时间既包括CPU时间也包括IO时间，基于以上的定义，我们可以获取对任意SVDB在节点上处理所需要花费的总的处理时间大小如式所示：



其中对应于第个SVDB的SVPM，表示在节点上处理第个SVDB时，其处理单位长度视频的CPU计算时间，，其中 和 分别是第个SVDB的视频开始时间和结束时间。是第个SVDB的视频体积大小。 和 分别是节点的平均的数据读写速率，其值可以通过常用的磁盘读写速率基准测试工具hdparm[52]获取。对于具有混合存储介质的计算集群，由于每一个节点既有SSD又有HDD，因此其应该是两组值，分别表示位于节点上的SSD的视频数据的平均读写速率和节点上位于HDD存储介质上的视频数据的平均数据读写速率。

一个视频处理任务通常关联多个SVDBs，对于在节点上运行的任意一个视频处理任务，其所花费的处理时间如式所示：



其中是和当前视频任务相关的SVDB的个数。该模型在计算视频任务所需的处理时长时，不仅考虑了CPU的处理能力，不同存储介质的读写速率，还考虑了各个视频文件本身的特性，可以准确计算出一个节点的视频任务所需的处理时长。

## 4.3 监控视频数据初始放置算法

在分布式视频任务处理场景中，一个视频任务的最终完成时间由处理时间最长的节点确定，如果在视频数据初始放置时，不考虑各个节点的负载差异，一方面会造成处理能力弱的节点分配的视频数据量大，而处理能力强的节点分配到的视频数据量偏小，导致延长整个视频任务的处理时间。另一方面，节点初始负载差异过大也会增加后续视频任务处理过程中的数据迁移成本，进一步导致降低整个视频处理任务的处理效率，因此制定合理的视频处理的初始放置策略对于提升整个平台视频任务的处理效率至关重要。

上节我们结合智能视频处理算法的特点构建了PTPM模型。本节我们提出了一种基于PTPM模型的监控视频数据初始放置算法，该算法主要作用于用户提交的作业所包含的数据从NVR传输到监控视频离线分布式处理系统的容器集群中进行初始放置的过程。

### 4.3.1 数据放置算法

在监控视频分布式处理场景下，一个视频处理任务可以划分为多个不同的视频处理子任务。我们假设当前的视频处理任务可以划分为个个待处理的子任务，每一个子任务对应于一组待处理的监控视频数据块SVDBs。例如一个子任务的功能就是通过运行视频浓缩算法对一组来自同一个监控摄像头获取的视频数据块进行处理。我们的初始放置算法的目的是通过将每一组待处理的监控视频数据块SVDBs放置到合适的存储媒介从而保证提升整个平台的处理效率。

我们定义代表整个容器集群中的计算节点集合，对于每一个结算节点，我们表述为。其中和分别表示节点的总的SSD的存储空间大小和HDD的总的存储空间大小。和分别表示节点的当前已经使用的SSD的存储资源和HDD的存储资源，和分别表示SSD和HDD已分配的用于存储视频处理任务产生的临时中间数据的存储空间大小。同时，我们定义作为当前待处理的视频任务处理集合。表示为有序的视频数据块组集合，其中表示一组对应于的待处理的视频数据块。

### 4.3.2 数据放置算法实现

我们的监控视频数据块初始放置算法采用最小负载优先放置策略，也就是在满足节点存储空间约束的前提下，依次从任务集合中选择一个视频处理子任务，并将对应的待处理的一组视频数据块放置到当前负载最小的节点上。为了简化问题和方便表述，我们假设集群各个节点的初始负载大小为0，同时我们定义字符常量代表SSD，字符常量代表HDD，代表节点当前选择的用于放置视频数据块的存储媒介类型。详细的监控视频数据初始放置算法如下：

**步骤1**： 从视频任务处理集合中选取一个视频处理任务并将其从集合中移除。

**步骤2**： 利用公式计算处理所产生的临时视频数据量大小，定义如式：



其中分别代表中第个监控视频数据块SVDB的开始时间和结束时间。表示第个监控视频数据块对应的监控视频处理模式SVPM。代表处理单位长度的SVPM为的监控视频数据块所产生的临时数据大小。

**步骤3：**遍历节点集合，对于每一个节点，分别利用公式和公式计算当把视频数据块集合放置到其SSD和HDD以后，SSD和HDD的剩余存储空间大小。





如果公式或者公式的结果大于等于0，我们认为节点满足任务的存储空间需求，我们将它先加入候选节点集合（初始化为空）。如果公式的结果大于等于0，我们更新变量的值为。如果公式小于0但公式大于等于0，我们更新变量的值为。

**步骤4：**定义代表当前选择的用于处理的节点，初始化为，代表当前用于放置所关联的视频数据块集合（和）的存储媒介类型，并初始化集群当前最小负载（定义为）为无穷大。

**步骤5：**从候选节点集合中选取任意节点并将其从节点集合中移除。

**步骤6：**利用PTPM模型估算节点处理任务的花费时间，并基于公式获取将分配给节点处理所预计即将到来的处理负载大小如式：



如果，更新，。

**步骤7：**重复步骤4~步骤6，直到候选节点集合为空。

**步骤8：**如果等于，我们就将放置到节点的SSD上，并根据公式和公式更新和的值，否则我们将放置到节点的HDD上，并根据公式和公式更新和的值。









**步骤9：**重复步骤1~步骤8，直到集合变为空，也就是所有的视频数据块都已经被放置完成。

结合上述步骤，可得到IDPS算法的伪代码如下所示：

**算法1** IDPS算法

1. **for** **do**
2. 初始化
3. **end for**
4. **while** 不为空 **do**
5. ;
6. 根据公式X计算处理所产生的临时中间数据体积；
7. **for**  **do**
8. 根据公式X计算将关联的视频数据块（文件）放置在节点的SSD后，SSD剩余存储空间大小；
9. **if** 的SSD剩余存储空间大于等于0 **do**
10. 
11. **end if**
12. 根据公式X计算将关联的视频数据块（文件）放置在节点的HDD后，HDD剩余存储空间大小;
13. **if** 的HDD剩余存储空间大于等于0 **do**
14. 
15. **end if**
16. **end for**
17. 初始化；
18. **while**  不为空**do**
19. 
20. 利用PTPM模型估算节点处理任务的花费时间，然而基于公式X获取将分配给节点处理所预计即将到来的处理负载大小；
21. **if**  **then**
22. 更新以及
23. **end for**
24. **end while**
25. **if** 等于**then**
26. 将放置到节点的SSD上，并根据公式和公式更新和的值;
27. **else**
28. 将放置到节点的HDD上，并根据公式和公式更新和的值;
29. **end if**
30. **更新**;
31. **end while**

## 4.4 监控视频数据块迁移策略

在上一节我们介绍了基于最小负载优先放置策略的监控视频数据块初始放置算法。然而，不同于基于单一存储架构的云平台，基于混合存储架构的云计算平台中，每个节点不仅配备有存储容量大、价格低廉的HDD，同时也配备存储容量小但读写性能高的SSD。在视频任务处理过程中，节点内部的SSD以及HDD上待处理的视频文件随着任务进行越来越少，占用的存储资源也越来越少，如果能通过合理的数据迁移方式将节点内HDD上一些满足条件的待处理视频文件迁移到当前节点的SSD上等待后续处理，即可以提高节点SSD的存储资源利用率，同时也能够进一步降低当前节点视频任务的处理时间。但是，由于视频文件从HDD迁移到SSD上也需要花费一定的迁移成本（时间），同时SSD本身可利用的存储空间有限，如果迁移策略不合理，一方面可能导致由于违反SSD存储空间约束而丢失视频文件，另一方面，甚至会出现由于视频数据迁移时间过长而延长整个节点的视频任务的最终处理完成时间。因此，如何制定合理的节点内异构储存媒介之间的监控视频数据块迁移策略，使得在满足SSD存储空资源约束的条件下最大化迁移后的收益值，即使迁移后节点视频任务的总处理时间相对于迁移前的节点视频任务处理总时间的差值最大化十分重要。

另一方面，在视频任务并行处理过程中，各个节点可利用的SSD和HDD的存储空间大小是随着时间不停发生变化的，随着任务的进行，各个节点的SSD以及HDD上待处理的视频文件越来越少，相对的，其可利用的存储资源就越来越多。在不考虑节点间数据迁移的情况下，由于不同节点之间的计算能力和存储能力不同，并且各个节点的初始放置的视频数据量大小也可能不同，在视频任务进行过程中，节点之间的负载差异也就随时间的变化越来越大，最终负载最高的节点会延长整个视频任务的完成时间。通过制定一个合理的全局视频数据迁移策略，使得系统能够在视频任务处理过程中根据各个节点的负载值以及可利用的存储资源状况周期性地将负载高的节点上的视频数据迁移到负载低的节点进行后续处理，以使得在视频任务处理过程中降低整个集群的负载不均衡程度，同时最小化整个视频任务的完成时间，对于提升整个计算平台的处理性能至关重要。

结合以上分析，我们设计并实现了一种基于线性规划的节点内视频数据动态迁移策略简称为NLDM（Node Level Data Migration） 和一种负载感知的全局视频数据迁移策略简称为CLDM（Cluster Level Data Migration）。其中，NLDM策略通过分析节内各个存储媒介（SSD或者HDD）上剩余待处理视频文件的数量，节点内部各个存储媒介的读写速率和剩余存储空间大小、以及节点本身的计算能力等，建立以最小化节点处理负载为目标的0/1线性规划方程，并通过求解方程来选取HDD上需要迁移到SSD上的待处理视频数据块，优化节点内视频数据分布。CLDM该策略会对计算和获取各个节点的负载值，并基于一定的策略将集群节点划分为高负载节点集合 和 低负载节点集合，最后通过综合考虑各个节点可利用的存储资源情况以及各个节点之间的网络传输带宽等因素，将负载高的节点上的合适的视频数据块迁移到负载低的节点进行后续处理，以使得在视频任务处理过程中降低整个集群的负载不均衡程度，同时最小化整个视频任务的处理时间。本章后续章节将详细介绍NLDM策略的实现过程和CLDM策略的实现过程。

### 4.4.1 相关变量定义

对于集群中的每一个节点，我们定义代表节点的SSD存储媒介，代表节点的HDD存储媒介，我们也分别定义和代表节点的SSD的平均的视频数据读速率和平均的视频数据写速率，和代表节点的HDD的平均的视频数据读速率和平均的视频数据写速率，和分别代表节点的SSD的总共的存储空间大小和HDD的总的存储空间大小。最后，我们定义表示处理完所有放置在节点上的SVDBs（包括放置在节点SSD上的SVDBs和放置在节点HDD上的SVDBs）的总共花费的时间, 的值可以通过我们PTPM模型计算获得。

在分布式视频任务的处理过程中，当前视频任务相关的所有的视频数据块SVDBs 被放置在各个节点的SSD或者HDD上。对于每一个节点，我们定义表示放置在它的SSD存储设备上的视频数据块集合，表示放置在它的HDD存储设备上的视频数据块集合，那么则表示放置在节点上的所有的视频数据块。对于每一个视频数据块，我们定义二进制变量作为它的存储媒介类型标识，等于1表示视频数据块被放置在节点的SSD上，等于0表示视频数据块被放置在节点的HDD上。那么，对于每一个视频数据块，其磁盘读操作的花费时间可以通过公式进行估算：



其中表示视频数据块的体积大小。

### 4.4.2 NLDM本地视频数据迁移算法实现

对于每一个节点，为了得到一个详细的节点内数据迁移计划，我们定义二进制变量指示是否将从的HDD迁移到的SSD。等于1表示在当前的本地数据迁移任务中需要将从HDD迁移到SSD，否则的话等于0。注意，表示的是放置在节点的HDD存储媒介中的第个监控视频数据块SVDB，它和的含义不同。

我们定义变量和分别代表在没有进行节点内视频数据迁移任务之前处理完节点的SSD上和HDD上放置的所有视频数据块所花费的时间，结合公式5，和的大小可以通过如下公式和公式获得。





基于公式和公式，在执行本地视频迁移任务之前，节点处理完其放置的所有视频数据所花费的总时间如式所示：



我们定义代表完成当前节点内视频数据迁移任务所需的迁移时间，的大小可以通过如下公式获取：



其中。

同时，我们定义表示处理除去在当前视频迁移任务中需要迁移到SSD上放置的视频数据块（即）的其他所有放置在HDD上的SVDBs所需要花费的时长，代表处理在当前视频迁移任务中到从HDD迁移到SSD放置的所有视频数据块的花费时间，那么的大小可以通过公式获取，的大小可以通过公式获取：





这样，如果我们执行了当前的节点内视频数据迁移任务，对于节点来说，其处理完成所有SVDBs所需花费的总的处理时长大小可以通过如下公式获取：



我们定义代表和之间的差异大小，也就是迁移前后节点的处理负载差异。通过结合公式~公式，的大小可以通过如下公式获得：



这里，，，和分别对应于第个放置在节点的SSD上和HDD上的SVDB的视频时长。

最后，我们将节点内数据迁移问题描述为一个求值的最优化问题，如式所示：



其中，约束XX保证了迁移后节点的SSD上所放置的视频数据块的总的存储空间需求不大于的SSD的总的存储空间大小。很容易注意到上面这个数学优化问题是一个0/1线性规划问题，我们可以通过标准的分支定界法（standard branch and bound algorithm）[53]对它进行求解，获取一组的值用于指示我们从HDD上迁移哪些视频数据块到SSD可以使得迁移前后节点的负载差异最大。同时，我们需要注意如果求得的的最优解大小小于等于0，也就表示当前无法通过从HDD迁移视频数据块到SSD来提升节点的处理性能，此时Node Manager就控制本地数据迁移模型LDMM不执行实际的数据迁移动作。

NLDM算法的主要逻辑如算法2所示，我们首先通过分支定界法获取的值以及（第6行），这里是一组二进制变量组成的列表。LDMM模块根据执行实际的视频数据块迁移操作（第9行）。当前数据迁移任务完成以后，我们为每一个迁移的视频数据块SVDB更新其对应的存储介质标识（第10行~等12行）。注意我们使用一个全局辅助变量标识当前节点是否正在进行迁移任务（节点内迁移任务或者全局迁移任务）。表示节点当前没有运行任何数据迁移任务，表示节点正在运行节点内部异构存储媒介间数据迁移任务（Node Level），表示节点正在进行全局视频数据迁移任务。我们通过使用辅助变量以及进程互斥锁保证统一时刻同一个节点最多运行一个视频数据迁移任务，放置由于多个视频数据迁移任务同时操作一个视频数据块造成的竞态问题。

**算法2** NLDM算法

1. **if**  then
2. **return;**
3. **end if**
4. pLock();
5. **更新**;
6. **pReleaseLock();**
7. **利用分支定界法求解公式X ~ 公式X获取本地迁移计划** 
8. **if**  then
9. **return;**
10. **end if**
11. **根据**执行本地视频数据迁移操作;
12. **for**  do
13. **if**  then
14. **更新**;
15. **end if**
16. **end for**
17. **pLock();**
18. **更新**；
19. **pReleaseLock();**

### 4.4.3 CLDM全局视频数据迁移算法实现

我们定义表示集群当前的平均负载大小，通过公式我们可以首先估算的值，



然后，我们根据的大小，将整个集群的节点分为两部分，其中一部分代表所有高负载节点组成的节点集合，定义为，中的每一个节点的负载都大于, 另一部分代表所有低负载节点组成的节点集合，定义为,中的每一个节点的负载大小都小于。同时，我们定义和分别代表集合中的所有节点负载的平均值和集合中所有节点的负载的平均值，然后我们定义用于判断集群当前的负载是否均衡的指示变量（注意表示的绝对值大小），如式所示：



当时，我们认为集群当前是负载均衡的，否则我们认为集群当前是负载不均衡的，我们需要执行我们的数据迁移算法从负载高的节点选取并迁移合适的视频数据块SVDBs到负载低的节点的合适的存储介质（SSD或者HDD）上进行放置等待后续处理。其中是一个给定的常量。

详细的视频数据迁移算法的步骤如下：

**步骤1：**对于每一个节点的视频数据块集合，我们分别计算将其中的任意一个视频数据块迁移到节点进行处理所需的预期完成时间，的值可以根据公式计算。



其中代表将视频数据块放置在节点上进行处理所需的花费时间。

代表在开始被节点处理之前的等待时长。我们规定只有在节点的当前剩余的存储空间大小大于视频数据块的存储空间需求时，才可以将迁移到节点进行后续处理，也就是说如果节点的当前的SSD剩余的存储空间大小和HDD剩余空间大小都不满足的存储空间需求的话，将不能将迁移到节点进行放置和处理，相应的我们设置对应的，。对于任意节点，我们定义和分别代表节点的SSD和HDD的剩余存储空间大小，和的值可以分别通过公式和公式获得。





其中表示位于节点的SSD上的第个视频数据块SVDB的体积大小，表示位于节点的HDD上的第个视频数据块SVDB的体积大小。

在节点满足的存储空间需求的情况下，的大小可以通过后面描述的PTPM模型中的公式进行计算。我们采用SSD优先考虑的迁移放置策略，也就是说，如果节点的SSD的剩余空间大小满足的存储空间需求，我们就考虑优先将迁移并放置到节点的SSD上，此时，公式中的等于，否则我们考虑将视频数据块迁移并放置到节点的HDD上，此时，等于。

我们可以通过公式获取的大小



其中代表将从节点迁移到节点的总的花费时间，它的计算公式如公式所示：



主要包括三部分：第一部分是从节点的存储设备上读取的花费时长，第二部分是通过网络将从节点传输到节点的网络传输时长，第三部分是将数据块写入到节点的存储设备（SSD或者HDD）上的总的花费时长。其中的大小我们可以通过公式获取，的大小我们可以通过公式进行估算：



其中表示节点和节点之间的网络传输速率，的值的大小可以通过使用网络性能测试工具如[54]进行获取。在我们的方法里，我们通过利用网络性能测试工具周期性地获取的值并通过计算这些的值的平均值定义为作为实际的公式中的的大小，以降低测量误差。的大小我们可以通过公式进行计算：



**步骤2：** 我们从所有中选取最小值，表示将视频数据块从高负载节点迁移到低负载节点进行处理可以获取当前最小的预期完成时间。我们同时从集群的低负载节点集合中选取当前负载最小的节点定义为。对于节点，我们通过公式获取将视频数据块从高负载节点迁移到上进行处理的最小预期完成时间：



**步骤3：**根据公式计算最小相对完成时间比率，根据公式计算最小相对负载比率：





如果的值大于等于，说明将视频数据块迁移到节点进行处理和将视频数据块迁移到节点进行处理导致的预期的最小完成时间的差异大于节点和节点之间的负载差异，这种情况下，我们需要将视频数据块从节点迁移到节点进行处理。如果上的SSD的剩余空间的大小满足的存储空间需求，我们将从节点迁移到节点的SSD上存储，否则我们将从节点迁移到节点的HDD上存储。

如果的值小于，说明将视频数据块迁移到节点进行处理和将视频数据块迁移到节点进行处理导致的预期的最小完成时间的差异小于节点和节点之间的负载差异，这种情况下，我们需要将视频数据块从节点迁移到节点上进行处理。如果上的SSD的剩余空间的大小满足的存储空间需求，我们将从节点迁移到节点的SSD上存储，否则我们将从节点迁移到节点的HDD上存储。

**步骤4：**更新集群节点集合中的每一个节点的负载大小，根据公式重新计算和的大小，根据公式重新计算值，重复步骤1~3，直到满足为止。

结合上述步骤可得到CLDM算法的伪代码，如下所示。

**算法3** CLDM算法

1. 初始化 ****
2. **while  do**
3. 根据公式分别计算****和****的值，根据公式计算****的值
4. **if**  **then**
5. ****
6. **for  to do**
7. **for  to do**
8. **for  to do**
9. 根据公式计算的大小
10. **end for**
11. **end for**
12. **end for**
13. 获取当前负载最小的节点
14. 分别获取和的值
15. 根据公式和公式分别计算和的大小
16. **if** **then**
17. **if** 的SSD剩余存储空间大小满足的存储空间需求 **then**
18. 将节点的视频数据块迁移到节点的SSD上存储
19. 更新
20. **else**
21. 将节点的视频数据块迁移到节点的HDD上存储
22. 更新
23. **end if**
24. **else**
25. **if** 的SSD剩余存储空间大小满足的存储空间需求 **then**
26. 将节点的视频数据块迁移到节点的SSD上存储
27. **更新**
28. **else**
29. 将节点的视频数据块迁移到节点的SSD上存储
30. **更新**
31. **end if**
32. **end if**
33. 
34. **end while**

在我们的算法执行过程中，我们会通过变量控制整个算法的迭代次数，以防止由于的值设置过小导致算法无法退出。

## 4.5 本章小结

本章首先分析了传统的基于Apache Hadoop构建的监控视频云计算平台中数据分布策略的不足，然后详细叙述了监控视频数据块处理时间预测模型建立的理论基础以及建立的过程，最后，详细介绍了基于PTPM时间预测模型的监控视频数据块初始放置算法和监控视频数据块迁移算法的思路以及伪代码实现。

# 第五章 系统实现及测试

## 5.1 系统环境配置

### 5.1.1 系统开发环境

文本设计的视频监控云平台的开发环境为：

（1）操作系统 Ubuntu14.04.3 LTS桌面版；

（2）程序开放环境：Vim，Atom，Pycharm，Docker，g++；

（3）计算机视觉库：OpenCV 2.4.9

### 5.1.2 系统硬软件配置

本系统硬件部分由12台物理服务器组成，其中控制节点1个，命名为controller，工作节点12个（注意我们将控制节点也同时配置为工作节点），除去controller节点，其他节点依次命名为node1~node11。每台物理节点上都安装有Ubuntu 14.04.1 LTS(GNU/Linux 3.13.0-32-generic x86 64)操作系统，安装版本为1.11.1版本的Docker Engine以及用于进行视频数据块存储的存储媒介SSD和HDD。整个集群物理服务器配置如下：

表5-1 物理服务器配置列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | CPU | SSD容量 | HDD容量 | 数量 |
| 1 | 16core | 64GB | 1T | 2 |
| 2 | 16core | 64GB | 2T | 3 |
| 3 | 24core | 64GB | 2T | 2 |
| 4 | 24core | 64GB | 2T | 2 |
| 5 | 32core | 64GB | 1T | 2 |
| 6 | 32core | 64GB | 2T | 1 |

我们通过hdparm获取每个节点配置的SSD存储媒介的平均读写速率分别是1149.66（M/Sec）和865.54（M/Sec）,每个节点配置的HDD存储媒介的平均读写速率分别是199.82（M/Sec）和197.78（M/Sec）。所有的物理节点通过两台千兆交换机进行互联，任意两个节点之间的网络传输带宽可以通过Netperf获得。

## 5.2 系统功能实现

### 5.2.1 视频数据初始放置功能实现

我们采用Kubernets作为整个容器集群的底层支撑组件，负责整个集群的资源管理和任务调度相关功能。虽然Kubernets默认没有实现与数据放置相关的功能模块，但是其基于插件化的模块功能实现机制却为开发者提供了便利的接口，用户可以很方便地集成自定义的算法和功能对其进行扩展。

开发者可以使用go语言实现自定义的功能函数或类，然后将对应的文件放置在Kubernets安装路径下的plugin/cmd/scheduler/algorithm/algorithmprovider目录下即可。Kubernets启动过程中会通过app/server.go中的SchedulerServer对象的CreateFromProvider方法从algorithmprovider目录下加载用户自定义的功能文件并执行。

本文为了实现第四章提出的数据初始放置策略，首先基于该初始放置策略思想实现了名为InitPlacementForSVDB的类，然后将对应的文件放置在所述的algorithmprovider目录，Kubernets启动后会最终调用InitPlacementForSVDB类的runIDPS方法进行视频数据的初始放置。

### 5.2.2 本地视频数据迁移模块实现

如第三章3.4.1小节所述，LDMM模块主要由宿主资源监控子模块、迁移策略定制子模块以及迁移操作执行子模块三个功能模块组成，主要通过运行本地视频数据迁移子策略NLDM实现节点内视频数据块的优化分布，其基本的流程如图5-1所示：



图5-1 LDMM模块流程图

宿主资源监控子模块主要用于获取宿主节点SSD和HDD的存储资源使用情况。由于LDMM模块本身是直接部署运行在各个计算节点，同时，每个计算节点安装的操作系统环境都是Ubuntu 14.04.1 LTS，因此我们可以直接基于原生liunx命令实现存储资源监控子模块的核心功能。其核心代码如下：

|  |
| --- |
| #/bin/bash  OLD\_IFS=$IFS  IFS=”\n”  # 初始化变量  HDDResourceUsed=0; HDDResourceAvail=0;  SSDResourceUsed=0; SSDResourceAvail=0;  #获取节点存储资源使用相关数据  disks=`df –TPh | awk ‘{print $0, $3, $4}’`  #遍历分别计算SSD和HDD相关存储资源使用数据（已使用和剩余量）  for disk in `echo “$disks”`  do  name = `echo $disk | awk ‘{print $1}’`  if [[ $name =~ ‘dev/mapper’ ]]; then  used=`echo $disk | awk ‘{print $4}’`  avail=`echo $disk | awk ‘{print $5}’`  if [[ $name =~ ‘ssd’ ]];then  SSDResourceAvail=`expr $SSDResourceAvail + $avail`  SSDResourceUsed=`expr $SSDREsourceUsed + $used`  else  HDDResourceAvail=`expr $HDDResourceAvail + $avail`  HDDResourceUsed=`expr $HDDResourceUsed + $used`  fi  fi  done |

该功能代码首先通过linux原生的磁盘资源查看命令df获取整个宿主的物理磁盘使用情况，然后将获取的结果按行扫描，依次读取磁盘分区名称，当前分区已使用的存储容量大小和当前分区剩余可利用存储容量大小等信息，并通过linux正则表达式对磁盘分区名进行模式匹配，判断当前磁盘分区属于SSD还是HDD，如果是SSD，则将当前获得的存储资源使用量和剩余可利用存储量分别累加到SSDResourceAvail和SSDResourceUsed变量上，否则累加到HDDResourceAvail和HDDResourceUsed。最终，SSDResourceAvail和SSDResourceUsed对应的便是当前节点SSD存储媒介剩余的总的可用的存储资源量和已经使用的存储资源量，HDDResourceAvail和HDDResourceUsed对应的便是HDD存储媒介剩余可使用的存储资源量和已经使用的存储资源量。同时为了便于整个LDMM模块的部署，我们基于我们平台的优势，将宿主资源监控子模块基于Dockerfile文件进行构建，生成用于进行宿主存储资源监控的功能镜像并命名为monitorStorageResouce。用于进行monitorStorageResouce镜像构建的Dockerfile代码如下：

|  |
| --- |
| FROM ubuntu:latest  MAINTAINER gyy [gyyzyp@163.com](mailto:gyyzyp@163.com)  RUN mkdir –p /build /output  WORKDIR /build  ADD storageResouece.sh #添加用于资源监控的shell脚本到构建目录  RUN rm –rf /build && rm –rf ~/.sh/\*  VOLUME /output  CMD [“/bin/bash”, ”storageResourceMonitor.sh”] |

Dockerfile编写完成可以通过以下命令制作镜像并上传到镜像仓库

|  |
| --- |
| docker build –t controller1:5000/monitorStorageResource //controller1:5000为视频服务的仓库地址。  docker run –ti –rm controller1:5000/monitorStorageResource //运行测试看看制作的镜像是否可用  docker push controller1:5000/synopsis //将镜像上传到镜像仓库 |

迁移策略定制子模块的主要功能主要是通过运行LDMA算法制定本地迁移策略并将结果输出到迁移操作执行子模块。关于LDMA算法的具体实现我们已经在第四章给出了详细的说明，这里不做累述。为了便于整个LDMM模块的部署，整个迁移策略制定子模块也同样采用Dockerfile文件进行构建生成对应的功能镜像命名为makeLocalMigrationPlan，其对应的Dockerfile文件的代码如下：

|  |
| --- |
| FROM ubuntu:latest  MAINTAINER gyy [gyyzyp@163.com](mailto:gyyzyp@163.com)  RUN mkdir –p /build  WORKDIR /build  ADD src . #将本地迁移策略定制子模块源码文件拷贝到容器中的/build目录  RUN rm –rf /build  CMD [“python”, ”startMakePlan.py”] #运行主函数，执行LDMA算法 |

迁移操作执行子模块主要是根据从迁移策略制定子模块获取的迁移计划执行实际的迁移操作。由于只是单个节点不同存储媒介之间的数据迁移，我们可以基于linux原生的mv命令实现迁移操作。例如需要将位于HDD上的视频文件A迁移到SSD上放置的操作指令如下：

|  |
| --- |
| #pathA 是A文件的绝对路径  #SSDMountDir 是SSD的挂载目录  mv pathA SSDMountDir |

同上，迁移操作执行子模块功能也通过Dockerfile文件封装成功能镜像，以便于后续在各个节点部署，其镜像名称命名为doMigrationOperation。

最后，我们通过Docker编排技术对整个LDMM模块的三个子模块进行服务编排，其对应的编排配置如下：

|  |
| --- |
| version: ‘2’  services:  monitor:  image: gyy/monitorStorageResouce  migPlan:  image: gyy/makeMigrationPlan  links:   * monitor:monitor   depends\_on:   * monitor   doMig:  image: gyy/doMigrationOperation  depends\_on:   * migPlan |

然后通过以下命令我们就可以轻松地在各个节点实现LDMM的部署：

|  |
| --- |
| docker-compose up |

### 5.2.3 全局视频数据迁移模块实现

全局视频数据迁移模块主要用于全局视频迁移策略的实现，负责在视频任务处理过程中周期性地将高负载节点上的合适的视频数据块迁移到低负载节点上进行放置，优化视频数据块分布，均衡各个计算节点的处理负载。如第三章3.4.3小节所述，全局视频数据模块主要分为迁移策略定制子模块和迁移操作执行子模块。其中迁移操作定制子模块主要是对CLDM数据迁移算法的实现。

CLDM算法需要首先获取集群各个节点存储资源使用情况以及各个节点当前放置的视频数据块集合相关信息进行算法相关参数以及变量的初始化，初始化完成以后执行算法核心流程获取全局视频数据迁移计划。所实现和使用的主要方法如表5-3所示：

表5-3 资源监控所需方法列表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 方法名 | 所属对象 | 参数 | 返回类型 | 描述 |
| getNodeList | Cluster | null | List<NODE\_INFO> | 获取集群节点列表,返回NODE\_INFO类型的节点对象集合 |
| getSSDInfo | Node | null | STORAGE\_INFO | 获取节点的SSD相关信息 |
| getHDDInfo | Node | null | STORAGE\_INFO | 获取节点的HDD详细信息 |
| getSVDBs | Node | null | List<SVDB> | 获取节点当前放置的SVDB列表 |
| runCLDM | Global | null | List<SVDBId, srcNodeId, targetNodeId, storeType | 返回迁移计划 |

其中Cluster和Node分别是CLUSTER数据结构类型和NODE\_INFO数据结构类型对应的实例对象。runCLDM方法是GDMM模块中的一个全局方法,负责全局数据迁移算法的执行并返回迁移计划，其中详细的算法实现已经在第四章4.4.3章节进行了详细的描述。runCLDM方法返回一个四元祖列表，其中每一项可表述为<SVDBId, srcNodeId, targetNodeId, StoreType>，其表示的含义为将标识符为SVDBId的视频数据块SVDB从标识符为srcNodeId的节点迁移到标识符为targetNodeId的节点的类型为storeType的存储媒介中进行放置。

runCLDM方法返回四元组列表后，GDMM模块中的迁移操作执行子模块遍历扫描四元组列表执行实际的数据迁移操作。为了实现任意两个节点间的数据迁移，我们首先通过TCP Socket机制实现GDMM模块和任意工作节点之间的远程消息通信（假设每一个工作节点都默认开启一个TCP服务器进程并监听在8080端口），然后利用linux原生的scp命令实现任意两个计算节点之间的视频数据块的传输。例如将位于计算节点A的SSD上的视频数据块a迁移到计算节点B的HDD上进行放置的实际工作流程如下：

（1）GDMM模块中的迁移操作执行子模块向节点A发送一个请求报文段告知其需要进行跨节点视频数据迁移操作。

（2）节点A收到GDMM模块的请求报文后进行解析获取待迁移的视频数据块a的标识符SVDBId,目标节点B的targetNodeId以及目标节点用于当放置待迁移视频数据块a的存储媒介类型storeType。

（3）节点A通过SVDBId标识符查询数据块a对应的SVDB实例对象的相关信息，然后遍历STORAGE\_INGO类型的实例对象获取数据块a在节点B上的绝对路径（假设为abs\_A\_a\_path）。同时，节点A通过targetNodeId获取节点B对应的具体的NODE\_INFO类型的实例对象信息，然后基于storeType获取节点B的存储类型为storeType的存储媒介所挂载的目录的绝对路径（假设为abs\_B\_storeType\_path）。

（4）节点A执行scp命令将视频数据块a迁移到节点B的HDD上放置，具体执行命令如下：

|  |
| --- |
| scp absA\_a\_path root@ipOfB:abs\_B\_storeType\_path |

其中ipOfB为节点B的ip地址。

（5）节点A向GDMM模块发送通知报文，告知已经完成数据块a的迁移操作。

### 5.2.4 视频浓缩服务镜像实现

为了测试本文实现的基于PTPM模型的视频数据块放置策略和视频数据块迁移策略，我们基于实验室先前工作中实现的视频浓缩算法实现了针对视频浓缩服务请求的功能镜像。当用户请求视频浓缩离线处理服务时，Cluster Manager会首先根据视频初始放置算法将视频文件从VS拉取并放置在容器集群中的各个节点，初始放置完成后，各个节点的Docker计算引擎会通过远程镜像仓库拉取视频浓缩服务镜像文件生成容器计算实例开始本地视频数据的处理。

由于我们先前工作中实现的视频浓缩算法的源代码是通过Maven进行构建的，因此我们需要首先在Maven项目的根目录下创建Dockerfile文件，然后以Maven的官方镜像作为基础镜像进行镜像的构建，同时为了减小镜像文件体积，提升从仓库拉取镜像的速率，我们在构建镜像的过程中需要将编译环节生成的中间文件通过系统命令进行删除。整个Dockerfile文件的代码如下：

|  |
| --- |
| FROM maven:3  MAINTAINER gyy [gyyzyp@163.com](mailto:gyyzyp@163.com)  RUN mkdir –p /build /input /output  WORKDIR /build  ENV TASK synopsis.jar  ADD pom.xml .  ADD src src  RUN mvn package && mvn test  RUN cp target/$TASK / && rm –rf /build && rm –rf ~/.m2/\*  VOLUME /output  CMD [“java”, “-jar”, “/synopsis.jar”, “$@”] |

Dockerfile编写完成后通过以下脚本文件完成镜像构建和上传。

|  |
| --- |
| docker build –t controller1:5000/synopsis  docker push controller1:5000/synopsis |

## 5.3 系统功能与算法效果测试

本节我们主要基于5.2.3小节实现的视频浓缩服务功能对系统性能以及算法效果进行验证。

### 5.3.1 系统功能验证

首先，为了验证系统视频浓缩服务镜像的功能的有效性，我们准备了大约8GB的监控视频数据，这些监控视频数据均来自中国福州部署的监控视频系统，视频本身采用H.264进行编码，视频分辨率为1920\*1080，码率为25fps，总时长为1730分钟。我们将8GB的视频文件随机放置在Controller节点和node1节点上，启动Docker计算引擎拉取视频浓缩服务镜像生成视频浓缩服务容器计算实例读取视频数据进行处理，最后生成的浓缩后的视频文件总大小约为1.2GB，时长约为240分钟。



图5-2 系统功能侧视图

如图5-2所示，左侧为原视频文件对应的视频画面，可以看到每个画面存在大量的冗余信息，而右边为经过处理获得的浓缩后的视频文件的画面，可以看到相对于原始画面，经过视频浓缩服务容器计算实例处理得到的浓缩文件除去了大量的冗余信息，大大提高了视频中有效信息的密集程度。从而验证了系统视频浓缩服务功能的有效性。在后续小节中，我们基于该视频浓缩服务功能镜像对系统性能进行测试。

### 5.3.2 PTPM准确性验证试验

为了验证PTPM模型的准确性，我们选择了一个包含3200个SVDBs的监控视频数据块集合进行离线视频浓缩处理任务，并通过对比预测时间、PTPM模型预测时间以及实际任务的处理时间来验证模型的准确性，其中PTPM时间预测模型是只根据节点CPU时间进行视频任务处理时间的估算。我们选择的监控视频数据块集合中每一个SVDB的码率都是25fps。这些监控视频数据块集合包含三种类型分辨率的SVDB，分别是L(640\*360),M(1280\*720)和H(1920\*1080)，其中L代表低分辨率，M代表中分辨率，H代表高分辨率。每个SVDB的视频时长或者为1320s或者为2640s。按照PTPM模型的规定，这些监控视频数据块集合对应的视频质量Q为{L25, M25, H30}, 视频处理模型集合SVPM={(L25, V), (M25, V), (H30, V)}，其中V表示视频任务处理类型为视频浓缩。我们首先通过在容器集群的各个计算节点分别离线处理2000个SVDBs来进行模型参数的初始化，然后我们利用PTPM时间预测模型预测剩余的1200个SVDBs在相关节点上处理需要消耗的时间，同时为了减小预测误差，对于每一类视频质量相同以及视频时长相同的视频数据块，我们通过求它们的预测时间的平均值作为该视频质量和视频时长的SVDB的预测时间。然后，我们又通过CTPM模型预测这些SVDB的处理时长，并同样计算每一类视频质量相同并且视频时长相同的SVDB的平均值作为实际预测时长。最后，我们实际运行这些SVDB并获取每一类视频质量相同且视频时间相同的SVDB的处理时长，计算其对应的平均值。

图5-3 PTPM模型的预测结果和实际结果对比

如图5-3所示，我们对这三种平均值进行了对比。图中横坐标表示每种视频质量和时间长度的类型，例如M25(1320s)表示视频质量为M，视频帧率为25，视频时长为1320s的SVDB，纵坐标表示每种视频质量相同且视频时长相同的SVDB的平均执行时间，单位为秒。通过对比这三种平均值，我们可以看到我们的PTPM模型在不同类型和视频数据和不同实验配置下能够十分精确的预测不同类型SVDB的所需的处理时间，相比于CTPM具有更高的预测准确率，相比于时间运行时间，误差大小仅为1.89%左右。

### 5.3.3 IDPS初始数据放置算法效果验证试验

为了验证我们提出的IPDS初始放置算法的效果，我们选取了4个有代表性的监控视频数据集合，每个视频数据集合包含不同数量和不同食品质量的SVDB。Dataset 1集合包含总共830个SVDBs，其中L25类型的SVDB有280个，M25类型的SVDB有300个，H25类型的SVDB有250个；Dataset 2集合包含总共860个SVDBs，其中L25类型的SVDB有300个，M25类型的SVDB有310个，H30类型的SVDB有250个；Dataset 3集合包含总共920个SVDBs,其中L25类型的SVDB有300个，M25类型的SVDB有350个，H25类型的SVDB有270个；Dataset 4包含总共970个SVDBs，其中L25类型的SVDB有320个，M25类型的SVDB有350个，H25类型的SVDB有300个。

图5-4 IDPS和HDFS默认的数据放置策略运行时间对比

我们同Apache Hadoop的默认的数据放置策略进行了对比。我们首先从VS中分别按照IDPA数据放置策略和Apache Hadoop默认的数据放置策略拉取并放置数据到容器集群中的各个节点，然后利用容器集群对视频数据进行处理，对比两种放置策略对视频处理任务中时间的影响。为了避免其他不稳定因素对实验的影响，我们保证实验过程中没有其他任务负载，并且我们将该实验在相同的实验环境下运行了8次并将8次计算结果的平均值作为每种数据放置策略下视频任务处理时间的最终值。实验结果如图5-4所示。

图中横坐标代表不同的监控视频数据集合，纵坐标表示每种监控视频数据集合在不同放置策略下的平均处理总时长，单位为秒。可以看到，我们的数据放置算法相比于Apache Hadoop默认的放置策略可以显著地减少任务的处理时间。

图5-5 IDPS和HDFS默认的数据放置策略下节点SSD资源利用率对比

同时，为了验证我们的初始放置算法在SSD利用率上的有效性，我们在node3节点处理dataSet3和dataSet4的过程中，周期性地读取其SSD的资源使用情况并计算其节点SSD的利用率。如图5-5所示，相比于Apache Hadoop的默认的数据放置策略，我们的数据初始放置策略导致节点具有更高的SSD利用率。

### 5.3.4 LADM数据迁移算法效果验证试验

这个实验我们主要验证我们的LADM数据迁移算法的效果。本次实验所使用的数据集和5.3.4的数据集是一样的。我们和默认HDFS数据放置策略以及DRA算法进行对比，其中DRA算法会在视频任务处理过程中周期性地将最高负载节点上的视频数据块迁移到最低负载的节点上放置来优化视频数据分布，但该方法不考虑节点存储介质异构性。同样本次实验也是进行多次实验取平均值。为了观察不同的初始数据放置策略下的实验结果，我们首先使用Apache Hadoop默认的数据放置策略作为平台的初始放置策略，然后使用我们提出的IDPA初始放置策略作为平台的初始数据放置策略,并观察针对不同数据集，在不同初始放置策略下，使用DRA迁移算法和LADM迁移算法所导致的任务处理总时长。

图5-6 HDFS默认数据放置策略下的LADM和DRA算法运行时间对比

图5-7 IDPS初始放置策略下的LADM和DRA算法运行时间对比

图5-6和图5-7分别展示了不同初始放置策略下使用LADM算法和DRA算法进行数据迁移在视频浓缩任务处理时间上的对比，其中图5-6为初始放置策略采用Apache Hadoop默认数据放置策略的实验结果，图5-7为初始放置策略采用IDPS初始数据放置策略的实验结果，通过实验结果可以看出不论采用哪种类型的初始放置策略，我们的LADM迁移算法相比于DRA算法均能有效地缩短视频任务的完成时间。

图5-8 集群负载均衡程度对比

在系统处理每一个监控视频数据集的过程中，我们同时记录每个节点处理完其所放置的视频数据子集所花费的处理时间（即每个节点的处理负载大小），然后我们通过公式计算变量的值，通过来衡量系统的负载均衡程度。



图5-8显示了采用IDPA初始放置算法前提下，处理不同的监控视频数据集对应的的大小。其中横坐标为不同的监控视频数据集，纵坐标为的值，单位为秒。从图中可以看出，相比于DRA算法，我们的LADM数据迁移算法可以更好地均衡系统的负载，提升系统负载均衡程度。

图5-9 dataSet2处理过程中SSD利用率对比

图5-10 dataSet4处理过程中SSD利用率对比

最后，为了验证我们的算法具有高的SSD存储资源利用率，我们在节点node2（其配置如表5-1中的实例3所示）处理dataSet2和dataSet4的过程中周期性采集其高性能存储媒介SSD的存储资源使用情况。图5-9和图5-10分别显示了节点node2处理dataSet2和dataSet4过程中的SSD的资源利用率。其中红色虚线表示采用DRA数据迁移算法所对应的SSD资源利用率，蓝色实线表示采用我们的LADM数据迁移算法对应的SSD资源利用率。从图中可以看出，不论是处理dataSet2还是dateSet4，相比于DRA算法，采用我们的LADM视频数据迁移算法都会导致更高的SSD存储空间利用率。

## 5.4 本章小结

本章首先详细介绍了基于混合存储架构的监控视频离线处理系统实现，主要是对LDMM功能模块和GDMM功能模块的实现细节进行了详细的介绍，然后介绍了基于Docker容器技术的离线视频浓缩服务镜像的实现，之后对系统基本功能进行了验证，最后通过实验验证了本文所提出的PTPM时间预测模型的准确性、IDPS数据块初始放置策略和LDAM数据迁移策略的效果，证明了基于PTPM模型的IDPA和LDAM策略，相比于当前的数据放置和迁移策略，能够有效提升基于混合存储架构的监控视频分布式处理系统中的高性能存储媒介SSD的利用率，平衡各个节点间的处理负载以及大大减小视频任务的处理总时间。

# 第六章 总结与展望

## 6.1总结

随着视频监控系统朝着大规模、海量数据和智能化的方向发展，基于云计算和智能视频分析技术构建监控视频离线分布式处理平台成为今年来解决海量视频数据计算需求的主流方案之一。同时，考虑到大多数智能视频处理任务属于资源消耗型，其任务处理过程中不仅涉及大量的逻辑运算，同时还伴随大量的读写操作，构建基于SSD+HDD混合存储架构的监控视频离线分布式处理平台，通过综合利用HDD的高容量、低成本以及SSD的高性能、低延迟特性来优化平台整体的读写性能对于提升监控视频处理平台的任务处理性能至关重要。然而，传统的视频监控云处理平台在进行海量视频数据的分布式处理过程中，既没有没有考虑海量视频任务处理场景中可利用存储资源以及节点负载的动态性特点，也没有考虑视频任务的资源需求特性，无法保证高性能存储媒介SSD的存储资源的高效利用，也很容易出现节点负载不均衡情况，进一步地增加视频任务的处理时间。本文深入分析基于混合存储架构的监控视频离线分布式处理平台中分布式视频任务处理特点，提出了一种基于视频任务处理时间预测模型（PTPM）的数据初始放置策略IPDS和数据迁移策略LADM来解决分布式视频任务处理过程中SSD利用率低以及集群节点负载不均衡等问题，相比于当前云平台中常用的数据分布策略，有效提升了系统高性能存储媒介SSD的资源利用率和保证了集群节点间的负载均衡。之后，本身设计并实现了基于容器技术的视频浓缩算法功能镜像对PTPM、IDPS、LADM算法进行了效果验证。本文主要研究内容有以下4个部分：

（1）监控视频离线处理任务时间预测模型

本文通过分析当前主流的智能视频处理算法，发现目前主流的智能视频处理算法都是建立在像素处理的基础上进行功能扩展的，得出影响这些算法处理时间的主要因素主要是监控视频数据块的分辨率、帧率等属性，在实验的基础上提出了一种监控视频离线处理任务时间预测模型（PTPM），该预测模型通过结合视频数据块本身的特征，例如分辨率、帧率、时间长度等，视频处理任务的类型如视频浓缩、视频摘要提取、行为检测等，以及集群中各个节点的计算能力和存储能力等，在基于大量历史任务数据分析基础上，通过自适应的调整参数，建立时间估计函数，进而估算出视频数据块在不同计算节点上处理的花费时间，从而计算得出监控视频离线处理任务时间。

（2）一种视频数据初始化放置策略IDPS

基于PTPM时间预测模型，研究并实现了一种最小负载优先的视频数据初始放置算法。该数据初始放置算法将同属于一个视频处理任务的视频数据块作为基本的放置单位，对于每一个待放置的视频数据块，通过每次选取当前负载最小的计算节点进行放置来降低集群中各个节点的负载差异。

（3）一种视频数据动态迁移策略LADM

随着视频处理任务的进行，各个节点待处理的视频数据块就越来越少，相应的，其节点的高性能存储媒介SSD的可利用存储空间就越来越多。另一方面，由于不同节点之间的计算能力和存储能力不同，并且各个节点的初始放置的视频数据量大小也可能不同，在视频任务进行过程中，节点之间的负载差异也就随时间的变化越来越大。为了有效利用各个节点的SSD的存储资源以及保证视频任务处理过程中集群的负载均衡，我们提出了一种视频数据动态迁移策略LADM,该策略包括一种基于线性规划的节点内视频数据迁移子策略NLDM和一种负载感知的全局视频数据迁移子策略CLDM。其中NLDM是一种基于0/1线性规划的本地视频数据迁移策略。该迁移子策略在节点进行视频数据处理的过程中，通过周期性分析节点内各个存储媒介（SSD或者HDD）上剩余待处理视频文件的数量，节点内部各个存储媒介的读写速率和剩余存储空间大小等因素，建立以最小化节点处理负载为目标的0/1线性规划方程，通过求解方程来选取HDD上需要迁移到SSD上的待处理视频数据块集合，通过将该数据块集合迁移到SSD放置，来提升节点SSD的存储资源利用率和节点视频任务处理效率。CLDM策略则会周期性对各个节点的负载大小分析，将集群节点划分为高负载节点集合 和 低负载节点集合，并通过综合考虑各个节点可利用的存储资源情况以及各个节点之间的网络传输带宽等因素，将负载高的节点上的合适的视频数据块迁移到负载低的节点进行放置等待后续处理，以降低视频任务处理过程中整个集群的负载不均衡程度，同时提升了整个视频任务的处理效率。

（4）基于Docker容器技术的离线视频浓缩算法验证系统

基于以上提出的数据放置和迁移策略，实现了一种基于Docker容器技术的视频浓缩算法功能镜像，使用交通车辆监控视频作为数据源，对所述的视频数据初始放置算法和视频数据迁移算法进行了验证。该系统使用SSD + HDD的混合存储架构作为整个计算集群的存储后端，上层采用Docker容器技术实现视频计算任务的分布式处理。当用户提交视频摘要任务后，各个节点会通过视频浓缩算法功能镜像生成容器计算实例读取本地放置视频数据进行处理，同时系统会按照基于PTPM时间预测模型的视频数据初始放置策略IDPS和视频数据迁移策略LADM进行数据的放置和移动，以优化集群中的数据分布，保证了集群任务处理过程中的负载均衡和高性能存储媒介SSD的高效利用，大大减少视频浓缩离线任务处理的总时间。

## 6.2 展望

本文通过综合分析传统视频监控云处理平台以及常用数据分布策略的不足之处，结合离线分布式场景下视频任务处理特性以及平台存储介质异构性等特点，提出面向基于混合存储架构的监控视频离线分布式处理平台的视频数据放置以及迁移策略，最后通过基于Docker容器技术的视频浓缩算法功能镜像对所述数据分布策略进行了验证，可有效提升系统高性能存储介质资源利用率以及视频任务处理效率，具有一定的实用性。但是本文还认为存在一些可以进一步研究的地方：

（1）PTPM时间预测模型优化。目前的PTPM时间处理预测模型主要基于对大量历史数据的离线分析，通过均值求解获取模型各个相关参数值大小。并且，模型没有考虑视频片段之间的关联性以及视频数据块所处时间段对处理时间的影响，未来考虑通过利用如逻辑回归、SVM等机器学习方法进行模型求解，实现模型在线学习，同时增加如视频片段关联度等信息到模型中，进一步提升模型求解的实时性和准确性。

（2）探索不同混合存储架构下监控视频云处理平台性能优化问题。本文主要关注基于SSD + HDD混合存储架构的监控视频离线分布式处理平台的性能优化问题，其中SSD和HDD是作为同层级的存储设备。然而，由于还存在许多其他类型的混合存储架构，比如将SSD用作HDD的缓存设备的存储架构，需要研究更多类型存储架构下的监控视频云平台的数据分布优化问题，并进一步实现更通用的面向视频监控云处理平台的视频数据分布策略。

（3）更丰富的智能视频服务功能镜像。文本主要实现了视频浓缩功能镜像来验证所述监控视频分布式处理平台以及所提出的数据分布策略及算法的有效性，后期应该学习如越界检测、车牌识别等视频智能分析算法，实现更多的视频服务功能镜像来进一步完善系统功能。

# 参考文献

1. Xiong Y H, Wan S Y, He Y, et al. Design and implementation of a prototype cloud video surveillance system [J]. Journal of Advanced Computational Intelligence and Intelligent Informatics, 2014, 18(1): 40-47.
2. Zhao X M, Ma H D, Zhang H T, et al. HVPI: Extending Hadoop to Support Video Analytic Applications [C]. // The 8th IEEE International Conference on Cloud Computing, IEEE, 2015: 789-796.
3. 刘秉煦. 云存储环境下的混合存储算法研究与实现 [D]. 上海交通大学, 2015.
4. Tan W, Fong L, Liu Y. Effectiveness Assessment of Solid-State Drive Used in Big Data Services [C]. // 2014 IEEE International Conference on Web Services, IEEE, 2014: 393-400.
5. Shi H, Arumugam R V, Foh C H, et al. Optimal Disk Storage Allocation for Multitier Storage System [J]. IEEE Transation on Magnetics, 2013, 49(6): 2603-2609.
6. Huang X, Huang Y Z, Liu Y, et al. A strip level data layout strategy for heterogeneous parallel storage systems [C]. // The 11th International Conference on Natural Computation, IEEE, 2015: 1085-1091.
7. Fan Y, Wu W, Cao H, et al. A heterogeneity-aware data distribution and rebalance method in Hadoop cluster [C]. // The 7th ChinaGrid Annual Conference, IEEE, 2012: 176-181.
8. Xu X, Cao L, Wang X. Adaptive Task Scheduling Strategy Based on Dynamic Workload Adjustment for Heterogeneous Hadoop Clusters [J]. IEEE Systems Journal, 2016, 10(2): 471-482.
9. Yan W, Li C, Du S, et al. An Optimization Algorithm for Heterogeneous Hadoop Clusters Based on Dynamic Load Balancing [C] // 2016 17th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT), IEEE, 2016: 250-255.
10. Chen F, Koufaty D A, Zhang X. Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems [C]. // The 25th ACM International Conference on Supercomputing, ACM, 2011: 22-32.
11. Zhang X, Davis K, Jiang S. iTransformer: Using SSD to Improve Disk Scheduling for High-performance I/O [C] // 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IEEE, 2012: 715-726.
12. Chang H P, Luo J C, Chang D W. A Load-Balancing Data Caching Scheme in Multi-tiered Storage Systems [C] // 2016 IEEE 18th International Conference on High Performance Computing and Communications, IEEE, 2016: 124-127.
13. Wan L, Lu Z, Cao Q, et al. SSD optimized workload placement with adaptive learning and classification in HPC environments [C] // The 30th Symposium on Mass Storage Systems and Technologies (MSST), IEEE, 2014: 1-6.
14. 郭伟. 云计算环境中数据放置及复制策略研究 [D]. 山东大学, 2015.
15. Bernstein D. Containers and Cloud: From LXC to Docker to Kubernetes [J]. IEEE Cloud Computing, 2014, 1(3): 81-84.
16. 何松林. 基于Docker的资源预调度策略构建弹性集群的研究 [D]. 浙江理工大学, 2017.
17. Gao Y, Wang H, Huang X. Applying Docker Swarm Cluster into Software Defined Internet of Things [C]. // 2016 8th International Conference on Information Technology in Medicine and Education (ITME), IEEE, 2016: 445-449.
18. 浙江大学SEL实验室. Docker:容器与容器云 [M]. 人民邮电出版社, 2015.
19. Bernstein D. Containers and cloud: From lxc to docker to kubernetes [J]. IEEE Cloud Computing, 2014, 1(3): 81-84.
20. Dadi C, Yi P, Fei Z, et al. A New Block-Based Data Distribution Mechanism in Cloud Computing [C]. // 2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing, IEEE, 2016: 54-59.
21. Poonthottam V P, Madhu Kumar S D. A Dynamic Data Placement Scheme for Hadoop Using Real-time Access Patterns [C]. // 2013 2nd International Conference on Advances in Computing, Communications and Informatics, IEEE, 2013: 225-229.
22. Kayyoor A K, Deshpande A, Khuller S. Data Placement and Replica Selection for Improving Co-location in Distributed Environments [J/OL]. arXiv: 1302.4168, 2013-02-18.
23. Zhang H T, Xu B, Yan J, et al. Proactive Data Placement for Surveillance Video Processing in Heterogeneous Cluster [C]. // 2016 8th IEEE International Conference on Cloud Computing Technology and Science, IEEE, 2016: 206-213.
24. Kgil T, Mudge T. FlashCache: a NAND flash memory file cache for low power web servers [C]. // International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, IEEE, 2006: 103-112.
25. Wan L, Lu Z, Cao Q, et al. SSD-optimized workload placement with adaptive learning and classification in HPC environments [C]. // 2014 30th Symposium on Mass Storage Systems and Technologies (MSST), IEEE, 2014: 1-6.
26. Chen F, Koufaty D A, Zhang X. Hystor: Making the Best Use of Solid State Drives in High Performance Storage Systems [C]. // Proceedings of the International Conference on Supercomputing, ACM, 2011: 22-32.
27. Yang Q, Ren J. I-CASH: Intelligently Coupled Array of SSD and HDD [C]. // Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture, IEEE, 2011: 278-289.
28. Shi H, Arumugam R V, Foh C H, et al. Optimal Disk Storage Allocation for Multitier Storage System [J]. IEEE Transation On Magnetics, 2013, 49(6): 2603-2609.
29. 李文龙. 基于Docker集群的分布式爬虫研究与设计 [D]. 浙江理工大学, 2017.
30. Docker Inc. Docker Swarm [EB/OL]. https://www.docker.com/docker-swarm, 2016-12-24.
31. Vohra D. Kubernetes Microservices with Docker [M]. New York City: Apress, 2016: 1432.
32. Acuña P. Kubernetes [M]. New York City: Apress, 2016: 1-127.
33. Seongjin L, Seokhui C, Haesung K,et al. Performance analysis of SSD/HDD hybrid storage manager [C]. // The 16th North-East Asia Symposium on Nano, Information Technology and Reliability, IEEE, 2011: 136-139.
34. 杨濮源, 金培权, 岳丽华. 一种时间敏感的SSD和HDD高效混合存储模型 [J]. 计算机学报, 2012, 35(11): 2294-2305.
35. 陈震, 刘文洁, 张晓, 等. 基于磁盘和固态硬盘的混合存储系统研究综述 [J]. 计算机应用, 2017, 37(05): 1217-1222.
36. Zhou J, Xie W, Noble J, et al. SUORA: A Scalable and Uniform Data Distribution Algorithm for Heterogeneous Storage Systems [C] // 2016 IEEE International Conference on Networking, Architecture and Storage (NAS), IEEE, 2016: 1-10.
37. Xie W, Zhou J, Reyes M, et al. Two-mode data distribution scheme for heterogeneous storage in data centers [C] // 2015 IEEE International Conference on Big Data (Big Data), IEEE, 2015: 327-332.
38. 贺昱洁. 负载均衡的大数据分布存储方法研究与实现 [D].上海交通大学, 2015.
39. Ghemawat S, Gobioff H, Leung S T. The Google File System[C]. // The Nineteenth ACM Symposium on Operating Systems Principles. ACM, 2003: 29-43.
40. Borthakur D. HDFS architecture guide [EB/OL]. http://hadoop.apache.org/d-ocs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html, 2017-11-18.
41. Burns A J, Lora K D, Martinez E, et al. Building a Parallel Cloud Storage System using OpenStacks Swift Object Store and Transformative Parallel I/O [R/OL]. http://www.osti.gov/scitech/servlets/purl/1048678, 2012-07-30.
42. Decandia G, Hastorun D, Jampani M, et al. Dynamo: Amazon’s Highly Available Key-value Store[C]. // The 21th ACM SIGOPS Symposium on Operating Systems Principles, ACM, 2007: 205-220.
43. Boyer E B, Broomfield M C, Perrotti T A. GlusterFS One Storage Server to Rule Them All [R/OL]. http://www.osti.gov/scitech/servlets/purl/1048672, 2012-07-30.
44. Weil S A, Brandt S A, Miller E L, et al. Ceph: A Scalable, High-performance Distributed File System [C]. // The 7th Symposium on Operating Systems Design and Implementation, ACM, 2006: 307-320.
45. Weil S A. Ceph: Reliable, scalable, and high-performance distributed storage [D]. UNIVERSITY OF CALIFORNIA SANTACRUZ, 2007.
46. ITU-T H.626-2011, Architectural requirements for visual of surveillance [S].
47. Dai W, Ibrahim I, Bassiouni M. An Improved Replica Placement Policy for Hadoop Distributed File System Running on Cloud Platforms [C]. // 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud), IEEE, 2017: 270-275.
48. Lin C Y, Lin Y C. A Load-Balancing Algorithm for Hadoop Distributed File System [C]. // 2015 18th International Conference on Network-Based Information Systems, IEEE, 2015: 173-179.
49. Poonthottam V P, Madhu K S D. A Dynamic Data Placement Scheme for Hadoop Using Real-time Access Patterns [C]. // 2013 International Conference on Advances in Computing, Communications and Informatics (ICACCI), IEEE, 2013: 225-229.
50. Du Y, Xiong R, Jin J, et al. A Cost-Efficient Data Placement Algorithm with High Reliability in Hadoop [C]. // 2017 Fifth International Conference on Advanced Cloud and Big Data (CBD), IEEE, 2017: 100-105.
51. Fan Y, Wu W, Cao H, et al. A Heterogeneity-aware Data Distribution and Rebalance Method in Hadoop Cluster [C]. // 2012 Seventh ChinaGrid Annual Conference, IEEE, 2012: 176-181.
52. hdparm [EB/OL]. https://sourceforge.net/projects/hdparm, 2017-05-01.
53. H. P. Williams. Logic and integer programming [M]. Springer, 2009.
54. Hewlett Packard Enterprise. Netperf [CP/OL]. https://github.com/HewlettPac-kard/netperf, 2010-12-22.

# 致谢

2014年10月初，我与北京邮电大学计算机学院智能通信软件与多媒体结下了难得的缘分，那时我还只是南方一所兄弟院校的大四本科生，有幸获得了保研资格，为了更美好的未来，用我稚嫩的经历但执着的追求，成功获得了马华东老师的青睐，由此拉开了我与实验室各位老师、同学的缘分，更成为我人生重大的转折点，使我从执着读博留校的人生规划中看到了更多不一样的世界，不再害怕求职就业，不再畏惧不熟悉的领域。这两年多的时光，我从浮躁、散漫的大学生慢慢成长为负责、踏实、敢于挑战自己、更加成熟稳重、逻辑严谨的硕士研究生。这一切都得益于身边老师、同学、朋友和家人的影响，在此，对每一位帮助指导过我的老师，耐心倾听我的同学朋友，贴心关心鼓励我的家人表示衷心的感谢。

感谢我的导师马华东老师。感谢您在我的学习和生活中给予无微不至的帮助和关怀。感谢您对我的每一次耐心的指导。您对待工作的敬业精神和严谨的治学态度都将激励我在今后的学习工作中不断努力，不敢懈怠。

感谢张海涛老师对我的培养和认可。他就像一位严厉的大哥哥，耐心指导我的工作和学习。他虽有很多工作要做，但仍坚持每周督促我的学习，认真仔细地与我探讨学术上的细小问题，关心我的改变，他对工作的这种负责任、从不拖延、安排有序的做事风格深深影响着我，间接督促我改掉拖延的不良习惯。

十分感谢实验室的同窗好友们。感谢付广平师兄、杨贤达师兄以及姜哲师兄对我工作和学术上的指导，你们对待科研的严谨态度和为人处世的随和教会我很多，而且你们散发的个人魅力也影响着我，激励我努力充实自己，成为和你们一样的人。也感谢张老师小组的唐炳昌师弟、杨宁学妹和实验室的其他同学们，大家一起学习，一起团建，互帮互助，与各位共度人生美好的年华，使我视野开阔了许多，谢谢各位的理解和支持。

感谢相伴我七年风风雨雨的你——朱彦沛，因为有你，我才更加安心学术。学习上的问题总是你陪在身边开解我，甚至压缩自己的时间用实际行动帮助我，生活上有你的支持和陪伴，让我不再感到孤单。谢谢你的包容和谅解，感谢有你一路同行。也因为你，我结识了更多志趣相投的伙伴——孙栎倩、周宇、孙康，在找工作的关键时期，大家一起努力，相互鼓励，资源共享，感谢你们的信任。

衷心感谢我亲爱的母亲大人，你给了我无限的动力。每当工作、学习遇到障碍的时候，都有你在一旁耐心开导，用你的豁达和乐观感染着我，鼓励着我。每当工作或学习取得了小小的成就，也是你与我分享小成就带来的喜悦和骄傲，无论我作出什么决定，你都在背后默默地支持着我，谢谢你给我一个坚强的后盾和温暖的港湾，让我没有后顾之忧，一心拼搏。

毕业论文的写作是一个痛并快乐着的过程，要直面庞大的研究课题，总结两年来的工作，并尽可能用科学的表述方式将我们的思想保留下来，是个痛苦的需要跳出舒适圈的工作，但一步步完善各个章节的过程却充满着骄傲和欣喜，不止是篇幅一点点接近了要求，更多的是对自己两年生涯的无憾无悔，对自己走过的这些岁月沉淀出的东西的珍视。感谢有这样的机会表达和记录自己。

最后，万分感谢各位评审老师和专家在百忙之中抽出宝贵的时间审阅我的文章，谢谢你们的宝贵意见和建议！

# 作者攻读学位期间发表的学术论文目录

以第一作者已发表的论文如下：

[1] Two-Stage Data Distribution for Distributed Surveillance Video Processing with

Hybrid Storage Architecture [C]. // 2017 IEEE 10th International Conference on

Cloud Computing (CLOUD), Hawaii, USA, 2017. (CCF C 类)

[2] A Load-Aware Data Migration Scheme for Distributed Surveillance Video

Processing with Hybrid Storage Architecture [C]. // 2017 IEEE 19th International Conference on High Performance Computing and Communications (HPCC),Bangkok, Thailand, 2017. (CCF C 类)

其它已发表论文如下：

[3] Container based Video Surveillance Cloud Service with Fine-Grained Resource Provisioning[C]// 2016 IEEE 10th International Conference on

Cloud Computing (CLOUD),  San Francisco, USA, 2017. (CCF C 类)