密级：　　保密期限：

北京郵電大學

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，论文中不包含其它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它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

视频监控系统的普及产生了海量的视频数据，分布式处理监控视频数据成为一种趋势。智能视频分析任务具有高的数据读写需求，利用 SSD + HDD 的混合存储架构构建监控视频离线分布式处理系统十分必要。传统的监控视频离线分布式处理系统大多以节点存储资源数量为约束条件进行数据的优化分布，未考虑集群节点存储介质异构性和视频任务处理过程中节点负载、可用存储资源的动态性，无法保证集群高性能存储媒介 SSD 的高效利用和节点间的负载均衡，进一步增加了视频任务的处理时间。

如何进行监控视频数据的优化分布，实现一个高性能的基于混合存储的监控视频离线分布式处理系统是本文的研究重点。首先，本文分析当前主流智能视频处理算法，提出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特征和集群节点的存储、计算能力等，预测视频任务在不同节点上的处理时长。其次，在 PTPM 模型基础上，提出一种监控视频数据初始分布策略 IDDS，该策略考虑节点存储介质异构性和节点负载差异性，在满足节点存储资源约束条件下，实现最小化集群节点间初始负载差异。同时，提出一种视频数据动态迁移策略 LADM，该策略考虑视频任务分布式处理过程中节点可用存储资源和负载的动态性特点，通过进行视频数据重分布，降低任务处理阶段节点间的负载差异和进一步提高节点存储资源利用率。最后，本文实现了基于 Docker 容器技术的验证系统，验证所提出的视频数据初始分布策略和视频数据迁移策略，实验结果表明本文所提出的策略能有效提升 SSD 的资源利用率，保证集群的负载均衡，大大提高视频任务的处理效率。

关键词：云计算，混合存储，数据分布，数据迁移，离线视频处理

RESEARCH AND APPLICATION ON DATA DISTRIBUTION STRATEGY IN SURVEILLANCE VIDEO CLOUD COMPUTING PLATFORM WITH HYBIRD STORAGE

ABSTRACT

The popularization of video surveillance system has contributed to the explosive growth of surveillance video data， distributed processing of surveillance video data has become a trend. There are many I/O operations in the processing of intelligent video analysis, and it is necessary to build a surveillance video offline distributed processing system with hybrid storage architecture using SSD and HDD. The traditional surveillance video offline distributed processing system mostly takes the number of the storage resources as the constraint to optimize the data distribution of cluster, without considering the storage media heterogeneity of cluster, as well as the dynamic of available storage resources and node load during the video task processing. Using default data distribution approach for large-scale video processing will lead to the low utilization of the high-performance devices and unbalanced cluster load, further defer the overall video task completion time.

This thesis focuses on how to optimize the distribution of surveillance video data for surveillance video offline distributed processing system based on hybrid storage architecture and realize a high-performance surveillance video offline distributed processing system. Firstly, this thesis proposes a Processing Time Prediction Model (PTPM) for surveillance video data blocks based on the analysis the current mainstream intelligent video processing algorithms. The PTPM considers the characteristics of video data, the storage and computing power of nodes. It can predict the processing time required by the video processing tasks at different nodes. Secondly, based on the PTPM model, this thesis proposes an Initial Data Distribution Strategy (IDDS). The IDDS considers the heterogeneity of the node storage media as well as the load difference between nodes, and can minimize the initial load difference between nodes while stratifying the

storage resources constraints. Thirdly, considering the dynamic characteristics of available storage resources and loads during the processing of video tasks, this thesis proposes a Load Aware Data Migration (LADM) strategy. The LADM strategy can re-distribute the video data of cluster during the process of video task distributed processing for reducing the load difference between nodes and further improving the node storage resource utilization. Finally, we implement a Docker based surveillance video offline distribution processing system, and conduct the extensive performance experiments for verifying the proposed IDDS strategy and LADM strategy. The experimental results show that our methods can effectively improve the resource utilization of SSDs, ensure the load balancing of clusters and further improve the processing efficiency of surveillance video processing tasks.

KEY WORDS: cloud computing, hybrid storage, data distribution, data migration, offline surveillance video processing

目录

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，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，固态硬盘 SSD（Solid State Drive），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。然而相比于机械硬盘 HDD（Hard Disk Drive），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本过高。通过结合 HDD 和 SSD，利用 HDD 的大容量、低成本和 SSD 低能耗、高性能、低延迟等特性，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]。考虑智能视频分析任务的高数据读写需求，利用 SSD + HDD 混合存储架构构建监控视频离线分布式处理系统十分必要。

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，保障平台中计算以及存储资源尤其是 SSD 存储资源的高利用率，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和视频处理任务相对于文本处理任务的复杂性，制

定高效的针对基于混合存储的监控视频离线处理系统的数据分布策略更加具有挑战性，主要面临以下两个难点:

（1）如何制定视频数据的初始分布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，进一步导致视频任务的处理效率降低[14]，因此，制定合理的视频数据初始分布策略对于提升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视频数据迁移策略，以实现集群的负载均衡。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，设计一个动态的视频数据迁移策略，根据各个节点的负载值和可利用的存储空间大小，在视频任务处理过程中动态地将负载高的节点上的视频数据迁移到负载低的节点处理，从而降低各个节点之间的负载不均衡程度，最小化整个视频任务的完成时间，对于提升云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（Hadoop Distributed File System）作为底层的存储系统用于存储海量应用数据，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，并未考虑节点存储介质本身的特性[4]，并且，Hadoop 默认采用的负载均衡策略只考虑任务处理过程中各个节点的负载率一致，而未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 I/O 密集型的视频任务分配到 I/O 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。

因而，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]。并且，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，而且 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价值。

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（包括数据初始分布策略和数据迁移策略），优化云平台中的数据分布，对于提升云平台计算性能至关重要，国内外有很多针对云计算平台中的数据分布策略的研究。

（1）云平台中数据初始分布策略研究

Dadi[20]等人主要考虑云平台中数据分布对于数据交付效率（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策略，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及其对应的存储节点位置，从而实现数据交付过程效率的最优化。Poonthottam 等人[21]提出了一种使用实时访问模式的 Hadoop 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。Kayyoor 等人[22]关注云平台中数据副本分布对查询能耗的影响，通过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待查询数据项的副本个数以及放置位置，优化数据副本分布，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，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（Data Blocks），通过每一次选择一个 DB 放置在当前负载最小的节点上处理，降低节点间初始负载差异，提升视频任务的处理效率。

然而，以上这些研究提出的数据分布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，使用以上这些数据分布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，进而降低视频任务的处理性能。本文设计的视频数据块初始分布策略，综合考虑节点存储媒介异构性及初始负载差异，可有效提高集群节点 SSD 的存储资源利用率和节点任务处理性能。

（2）云平台中数据迁移策略研究

近年来，随着混合存储架构在云平台中使用越来越广，基于混合存储的云计算平台中数据迁移策略的研究也得到了大家的关注，取得了一些研究成果。Wu 等人[24]考虑数据访问模式的不对称性，提出了一种面向混合存储的偏置对象存储策略（Biased Object Storage Strategy），该策略基于对数据访问模式的分析，将读密集型数据迁移到 SSD 放置，而将写密集型数据迁移到 HDD 放置从而减少 SSD 写入，提升系统性能。Wan 等人[25]考虑用户自定义的数据放置规则对数据分布的影响，提出了一种基于用户规则（User Polices）约束的数据分布策略，通过基于马尔科夫链的数据分类模型对数据访问信息进行实时分析，在满足用户规则约束前提下，将预测访问频率高的数据迁移到 SSD 上进行放置，有效提升了系统数据访问性能。Chen 等人[26]提出了一种基于 SSD + HDD 的混合存储系统 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，将更新频率低并且读取频繁的参考数据块存储在 SSD 上，而将与参考数据块相对应的涉及频繁 I/O 操作的增量数据块存储在 HDD 上，通过利用数据块的强正则特性和内容局部性特点，使得一次 HDD 的磁盘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，提升存储系统的读取性能。

然而，这些方法主要考虑通过提升高性能存储媒介如 SSD 的利用率来提高集群读写性能，而没有考虑大规模视频任务处理过程中的节点负载和可利用存储

资源动态性特点。本文第四章设计针对海量视频数据分布式处理场景中的数据迁移策略，通过综合考虑节点存储介质异构性和节点负载动态性特点，实现视频任务处理的高效率。

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系统通过综合利用 HDD 的高容量、低成本和 SSD 的高性能、低延迟特性来优化平台的读写性能，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重要。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和节点负载的动态性特点，也没有考虑视频任务的资源需求和存储介质异构性，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（Processing Time Prediction Model）的视频数据初始分布策略和视频数据动态迁移策略，并实现了基于 Docker 容器技术的验证系统验证算法的有效性。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，基于大量历史任务数据离线分析，通过自适应的调整参数，建立时间估计函数，获取视频数据块在不同计算节点上的处理时间，从而计算得出监控视频离线处理任务时间。

（2）面向监控视频云计算平台的视频数据初始分布策略

当前主流的基于 Apache Hadoop 构建的监控视频处理平台在进行数据初始分布时主要保证各个节点数据存储量一致，而没有考虑节点存储媒介异构性以及节点计算负载差异性。本文基于 PTPM 时间预测模型，研究并实现了一种视频数据块初始分布策略 IDDS（Initial Data Distribution Strategy）。该策略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集合，在考虑节点存储媒介异构性基础上，每次选取当前负载最

小的计算节点放置，降低集群中各个节点的初始负载差异，保障集群节点的高资源利用率和高视频任务处理性能。

（3）基于负载感知的监控视频数据块迁移策略

传统的监控视频离线处理平台在进行视频任务分布式处理过程中或者没有考虑视频任务处理过程中节点负载和可利用存储资源动态性，或者没有考虑集群存储资源异构性特点，容易造成集群存储资源利用率低和负载不均衡。本文通过分析分布式场景下视频任务处理特点，综合考虑节点负载动态性和存储资源异构性特点，提出一种负载感知的监控视频数据块迁移策略 LADM（Load Aware Data Migration），该策略可以制定合理的视频数据块迁移计划用于指导集群进行视频数据的优化分布。通过实验验证，该策略能有效提升集群 SSD 存储资源利用率，降低集群负载不均衡程度，进一步提升视频任务处理效率。

（4）基于 Docker 容器技术的验证系统

传统的监控视频离线分布式处理系统采用 Apache Hadoop 构建，难以获取节点物理资源特性和扩展自定义功能。本文采用 Docker 集群技术构建监控视频离线分布式处理系统，并按照本文设计的视频数据初始分布策略和视频数据动态迁移策略扩展实现了数据分布相关功能模块用于负责视频数据的优化分布。最后，通过视频浓缩服务验证了系统高效的任务处理效率。

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平台中的数据分布策略的研究现状，最后介绍了本文的主要研究内容。

第二章：相关技术。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。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，最后详细介绍监控视频离线分布式处理系统的总体架构设计、工作流设计以及核心组件设计。

第四章：云平台中视频数据分布策略研究，这部分是本文的核心。首先详细介绍了监控视频处理任务时间预测模型 PTPM，然后介绍了基于 PTPM 时间预测模型的视频数据初始分布策略 IDDS，并详细介绍该策略的实现过程，最后介绍负载感知的视频数据动态迁移策略 LADM，并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 IDDS、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，分析本文提出的数据分布策略进一步的优化方向。

第二章 视频监控云平台相关技术

本章主要介绍了后续章节涉及的一些基础知识，主要包括视频监控云平台相关的云环境技术、混合存储体系结构相关技术以及云平台数据分布相关技术。

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，仅仅三年的时间就成为时下最流行的技术，为云计算注入了新的活力，至今已经成为主流的云计算技术之一[18]。Docker 底层是基于 Linux 提供的 LXC 轻量级虚拟化技术（如 Namespace、CGroup 等）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。

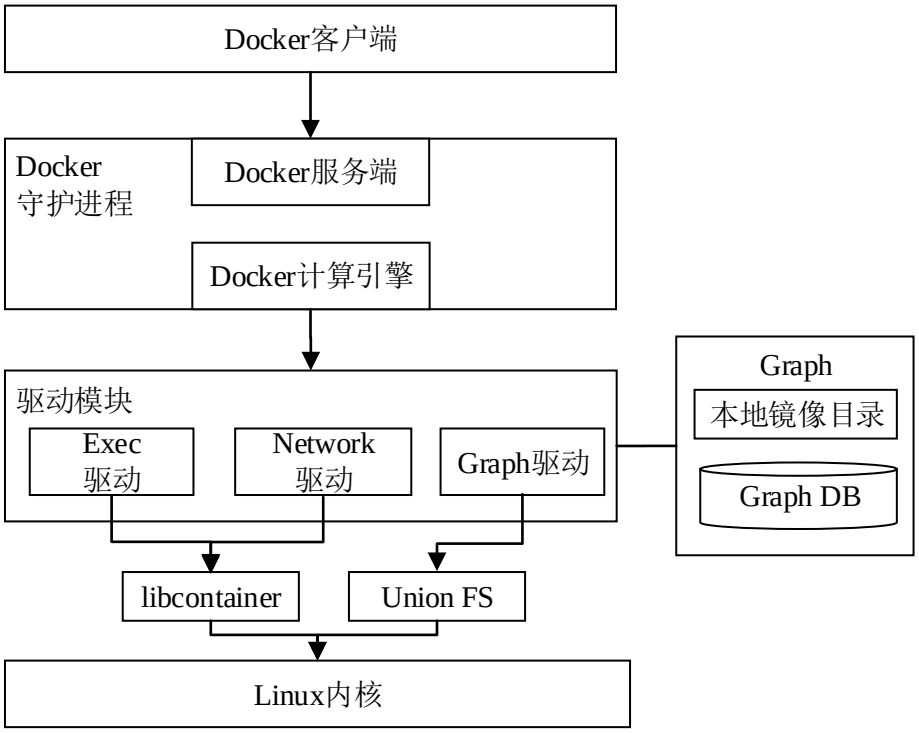

图 2-1 Docker 框架图

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 客户端、Docker 守护进程、驱动模块、Graph、libcontainer 等 5 大模块。

（1）Docker 客户端

Docke 客户端是 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 客户端负责与 Docker 守护进程之间建立通信，并将用户服务请求转发给 Docker 守护进程。当 Docker 守护进程处理完成当前的请求后，再由 Docker 客户端负责将请求结果返回给用户。Docker 客户端主要通过 tcp://host:port 的方式和 Docker 守护进程之间建立通信,其中 host 和 port 分别为 Docker 守护进程进程所在宿主的 IP 以及 Docker 守护进程的监听端口，默认为 2735。

（2）Docker 守护进程

Docker 守护进程是整个 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 客户端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 守护进程本身主要由 Docker 服务端和 Docker 计算引擎两个子模块组成，其中 Docker 服务端主要负责接收和解析 Docker 客户端请求，将解析后的内容发送给 Docker 计算引擎。Docker 计算引擎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）驱动模块

驱动模块是 Docker 架构中负责底层系统调用的核心部分。Docker 守护进程接收到用户请求后，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 守护进程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 Exec 驱动,用于网络管理相关的 Network 驱动以及用于文件存储相关操作的 Graph 驱动。

（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 和本地镜像目录两部分构成。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。本地镜像目录则是一个本地目录，用于实

际存储全部下载完成的镜像文件信息，这些信息包括：容器镜像代表的具体 rootfs 数据、容器镜像文件的大小、容器镜像元数据等。

（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 而不需任何其它依赖。

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 镜像具有可编程的特性，从而帮助开发人员打通应用线上与线下之间的环境壁垒，确保应用生命周期环境的一致和标准。借助这一特性，开发人员借助 Docker 镜像构建标准的开发环境，封装环境和应用的完整镜像完成后续迁移。那么，在应用的测试和运维阶段，大大简化了持续集成、测试和发布的过程，只需直接部署软件镜像即可进行测试和发布。

（2）可跨平台移植。Docker 不仅可编程，而且适配性良好，几乎可以在当下的所有平台中运行，这一特性使得应用开发者不再受云平台的限制捆绑，而且还可实现应用的异构混合部署。目前支持 Docker 的包括 AWS、GCP、Azure、OpenStack 等大型 IaaS 云平台，还包括 Chef、Ansible、Puppet 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，因而，Docker 的系统负载更低，性能也更加优良，相较于其它技术在相同的条件下可运行更多的应用实例，从而使得系统资源得到更充分的利用。与此同时，Docker 又具有良好的资源隔离、资源限制能力，能够精确分配 CPU、内存等资源给各个应用实例，确保应用之间互不影响。

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）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 开源的 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源管理和调度，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 的服务，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，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 HDD 和固态硬盘 SSD 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它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，存在写前擦除限制。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

前面两个小节，本文分别对两种主流的存储媒介 SSD 和 HDD 进行了简单的介绍，下面主要介绍基于 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 + 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 + 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，介绍 SSD 作为 HDD 缓存的混合存储架构，其架构如图 2-2 所示。

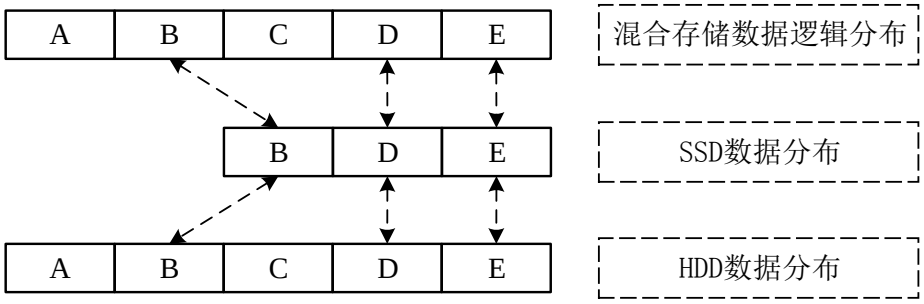

图 2-2 SSD 作为 HDD 缓存的混合存储架构示意图

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，则无法有效利用 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。

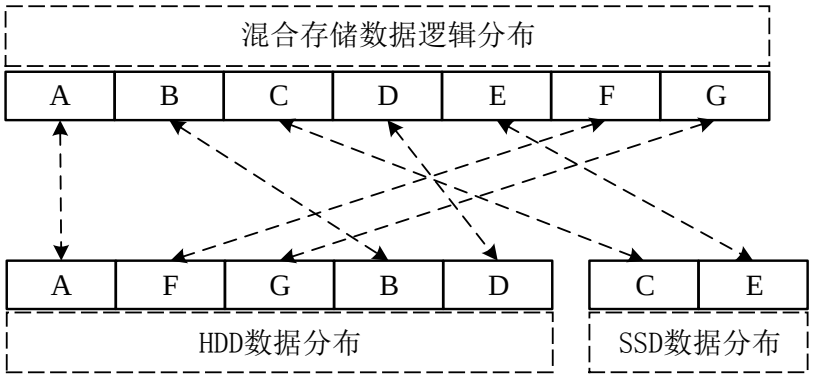

图 2-3 SSD 作为 HDD 同级存储设备的混合存储架构示意图

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不同计算节点之间的网络数据传输量等至关重要。云平台在设计海量数据的分布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[36]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 DataNodes 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。

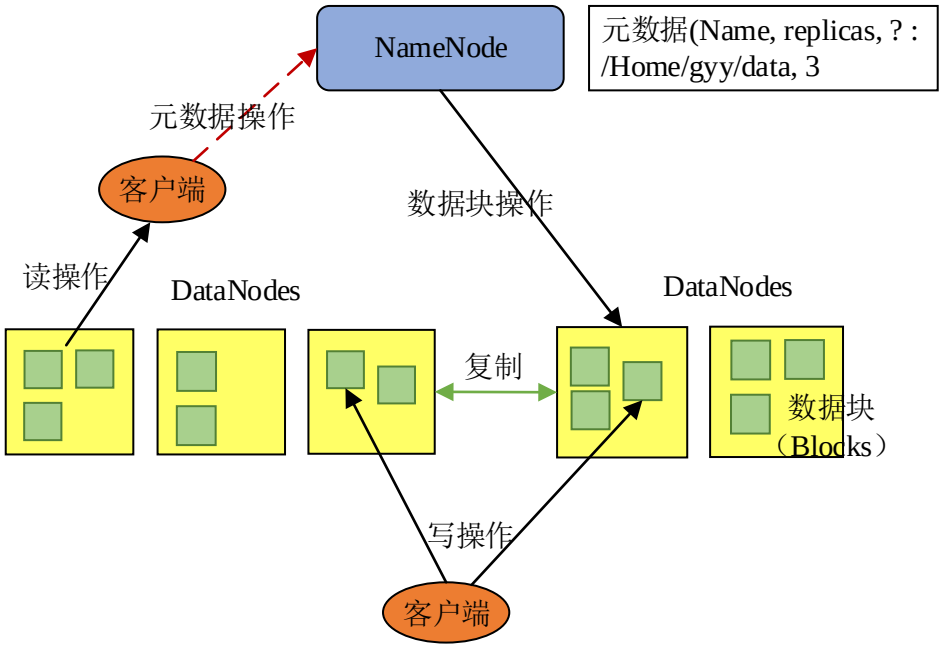

图 2-4 HDFS 数据分布示意图

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，默认为 64MB，然后将这些数据块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它任意一个 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[37,38]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）计算用户数据及其副本的存储位置。系统在进行数据放置时，通过将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首尾相连的圆环，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，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。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，降低已有节点的存储负担。

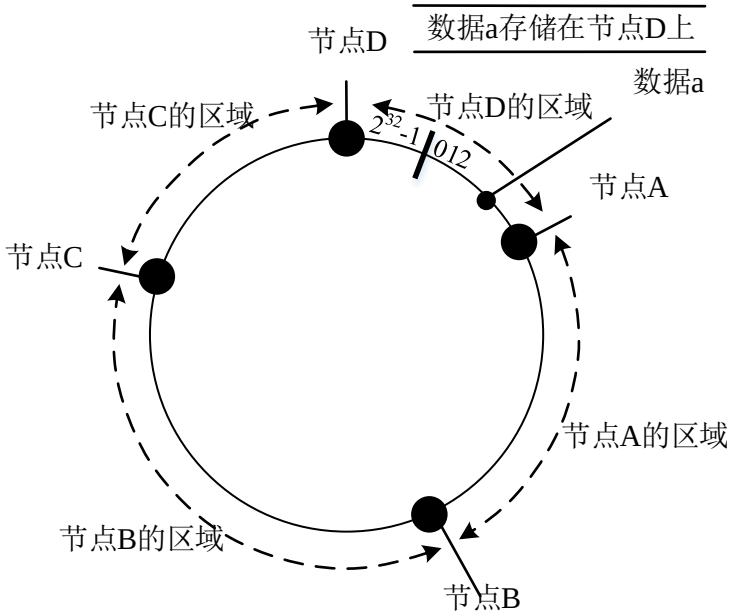

图 2-5 基于一致性 Hash 算法的数据分布示意图

最后，为了保证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，减少了管理以及检索元数据的资源需求，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 + HDD 的混合存储架构，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的研究提供基础。

第三章 基于混合存储的监控视频离线分布式处理系统设计

本章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，然后简单介绍视频监控系统的组成以及各个组成部分的作用，接着提出了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和工作流，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、卫生、城市管理）。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、越界检测等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，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与云平台计算能力相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 + 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、图像及各种报警信号）数据，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，目前已被使用在安防、教育、医疗等各行各业。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。

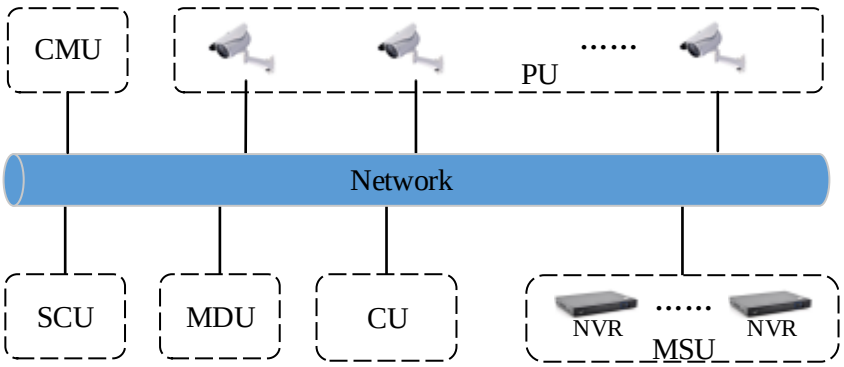

图 3-1 视频监控系统架构

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 主要由网络存储设备 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和吞吐量限制。例如 NVR 仅可以同时从 16 个摄像机接收视频流，并且仅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它系统组件。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总体设计

3.3.1 总体架构设计

本文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[47]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：

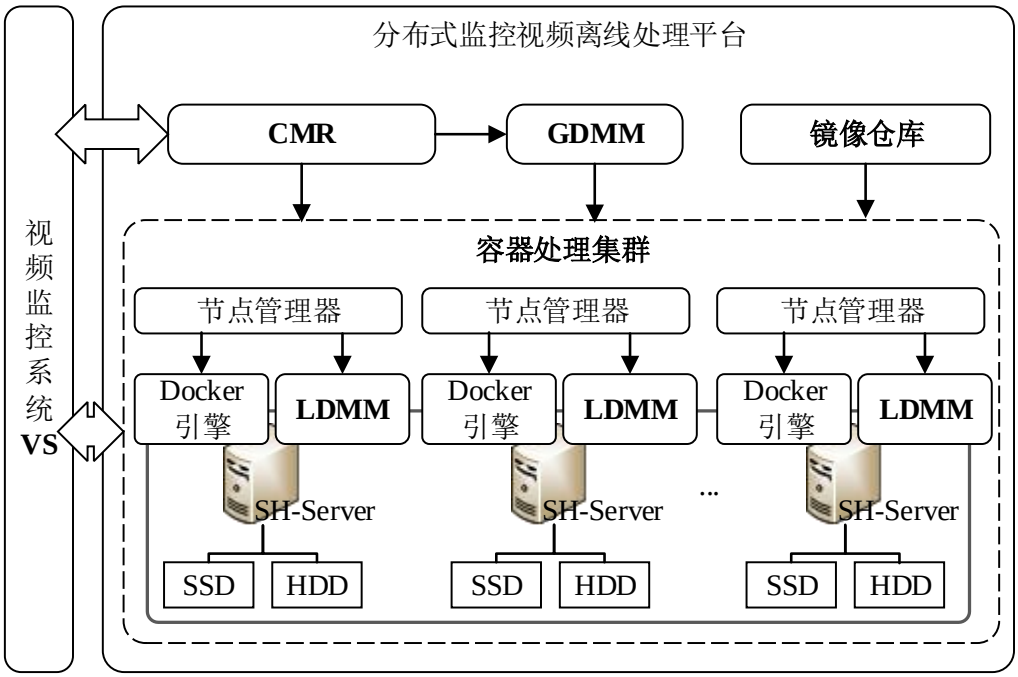

图 3-2 基于混合存储的监控视频离线分布式处理平台架构

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 部分即为 3.2 小节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 CMR 组件发送视频处理任务的请求分布式地处理监控视频任务。

（2）集群管理器 CMR

集群管理器 CMR (Cluster Manager) 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，底层是基于 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，并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 根据集群视频数据迁移算法执行数据视频数据迁移任务，优化集群数据分布，具体的集群视频数据迁移算法将在第四章详细介绍。

（3）容器处理集群

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。此外，每一个物理节点都运行一个节点管理器进程，节点管理器组件主要负责从 CMR 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节点内视频数据迁移模块 LDMM 执行节点内视频数据块迁移算法执行单节点异构储存媒介间视频数据迁移任务，具体的节点内视频数据迁移算法将在第四章详细介绍。

（4）集群视频数据迁移模块 GDMM

集群视频数据迁移模块 GDMM（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集群视频数据迁移算法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，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第 4 小节介绍。

（5）节点内视频数据迁移模块 LDMM

节点内视频数据迁移模块 LDMM（Local Data Migration Module）主要功能是在视频任务处理工程中周期性执行节点内视频数据迁移算法将本地节点 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第 4 小节介绍。

（6）镜像仓库

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，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

本小节将介绍分布式系统中监控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。

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送给 CMR 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 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 CMR 组件。

CMR 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，同时还会获取容器处理集群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，CMR 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，CMR 会从 VS 中的 MSU 中依次拉取待处理监控视频数据块 SVDB（Surveillance Video Data Block）并调用视频数据块初始分布策略 IDDS 计算每一个待处理视频数据块在容器处理集群中的初始放置位置并按照计算结果将其加载到容器处理集群中相应的物理节点上。

视频数据块初始化放置完成以后，CMR 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节点管理器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，节点管理器模块收到 CMR 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，节点管理器会控制 LDMM 模块周期性执行节点内视频数据迁移算法将 HDD 上待处理视频数据块（尚未被处理）迁移到 SSD 上等待后续处理。CMR 会控制 GDMM 模块周期性执行集群视频数据迁移算法将高负载节点上视频数据块迁移到合适的低负载节点上。

最后容器处理集群处理完所有的任务后，CMR 组件会读取任务处理结果并将任务处理结果发送给 VS，VS 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，进一步提升系统的处理性能，本文对容器化监控视频离线处理平台进行了优化，其架构如图 3-2 所示，主要添加了与监控视频数据优化分布相关的节点内视频数据迁移模块 LDMM 和集群视频数据迁移模块 GDMM。本小节主要介绍 LDMM 模块和 GDMM 模块的设计，并给出系统核心数据结构设计。

3.4.1 节点内视频数据迁移模块设计

节点内视频数据迁移模块的执行流程如图 3-3 所示。

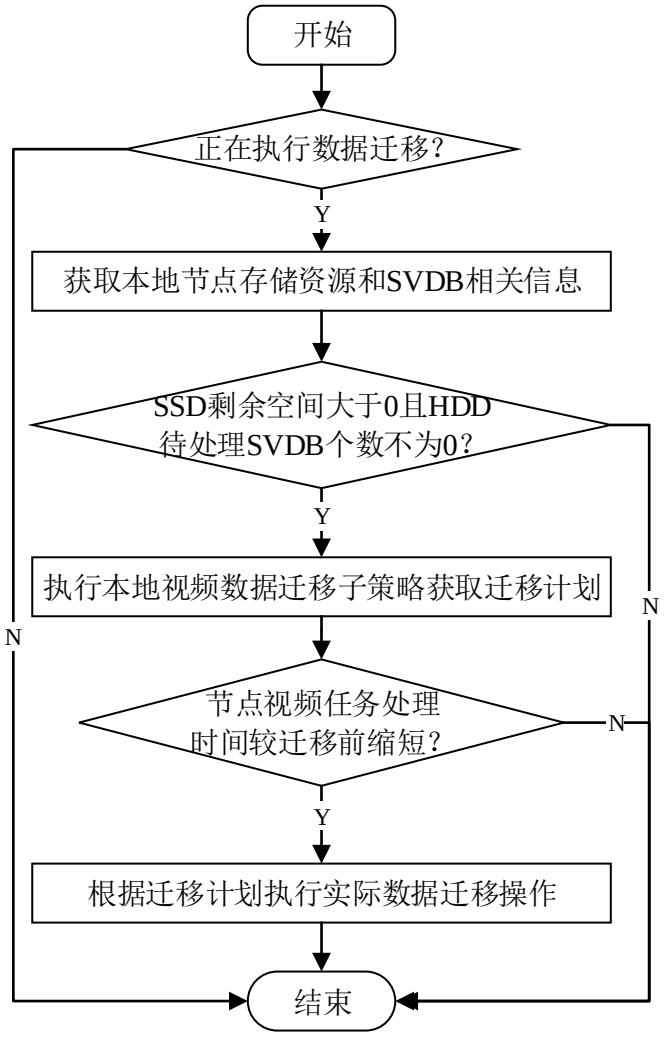

图 3-3 LDMM 模块执行流程

CMR 完成视频数据块初始放置后，节点管理器开始控制节点的 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 I/O 性能和高性能存储媒介 SSD 资源利用率，每处理完一个视频数据块，节点管理器组件就控制启动节点内视频数据迁移模块执行一次节点内视频数据迁移操作。

节点内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 3-4 所示，主要由三个功能子模块组成。

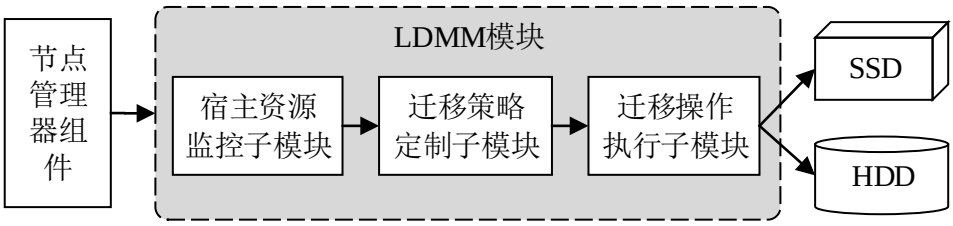

图 3-4 LDMM 模块示意图

（1）宿主资源监控子模块

宿主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 的剩余存储资源量以及 HDD 已经使用的存储资源量等。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数据制定合适的本地视频数据迁移计划。

（2）迁移策略定制子模块

迁移策略定制子模块是 LDMM 模块的核心组成部分，其内部实现了本文设计的视频数据迁移策略中的节点内视频数据迁移算法，它会首先解析从宿主存储资源监控子模块接收到的数据，获取宿主当前存储资源相关数据和宿主视频数据块分布信息等，完成节点内视频数据迁移算法的初始化，然后运行该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节点内视频数据迁移算法的实现细节将在第四章介绍。

（3）迁移操作执行子模块

迁移操作执行子模块的功能比较简单，主要是从迁移策略定制子模块中获取本地视频数据迁移计划，根据所获取的迁移计划执行实际的节点内视频数据迁移操作，将放置在宿主 HDD 上待迁移的视频数据块迁移到节点的 SSD 上放置等待后续处理。

3.4.2 集群视频数据迁移模块设计

集群视频数据迁移模块部署运行在主节点，主要功能是在视频任务处理过程中通过周期性将高负载节点上的视频数据块迁移到低负载节点上从而优化集群数据分布，避免出现节点负载差异过大和节点SSD存储资源利用率过低的情况，其执行流程如图3-5所示。

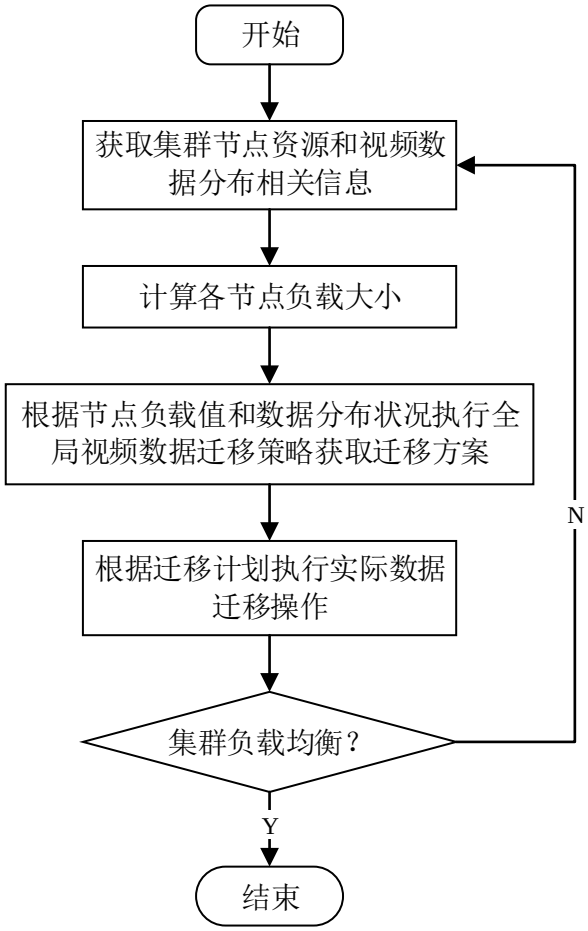

图3-5 GDMM模块执行流程

不同于LDMM模块，GDMM主要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，其基本的模块组成架构如图3-6所示。

其中，迁移策略定制子模块会周期性从后端数据库中获取集群节点信息，包括各个节点的资源使用情况以及各个节点视频数据分布相关信息等，并以获取的集群节点信息为输入完成集群视频数据迁移算法初始化，然后运行集群视频数据迁移算法获取优化集群视频数据分布的全局迁移计划，最后将全局视频数据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，主要根据获取的全局视频数据迁移计划执行实际的迁移操作，其中使用的集群视频数据迁移算法实现细节将在第四章介绍。

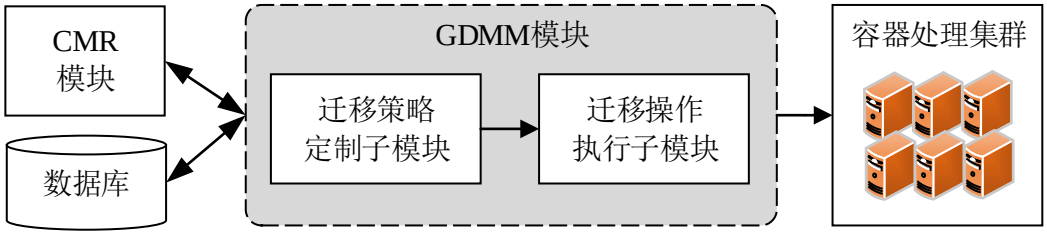

图 3-6 GDMM 模块示意图

3.4.3 核心数据结构设计

在 3.3 小节中介绍了当用户通过 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 CMR 组件，CMR 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视频数据初始分布策略将拉取的原始视频数据文件合理地放置到容器处理集群中的各个节点。

为了便于描述视频数据块的相关信息，本文设计了一种描述视频数据块本身属性的数据结构 SVDB（Surveillance Video Data Block），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	SVDB_address	MSU_r	SVDB_f
变量类型	int	String	int	int
变量名	SVDB_st	SVDB_et	SVDB_job	-
变量类型	double	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 CMR 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 表示当前视频数据块对应的视频任务类型。

由于在进行视频任务处理过程中，每一个节点都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制定合理的节点内迁移策略，因此针对每一个节点，本文设计维护其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	char	double	double	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 SVDB_id，对应于一个特定的视频文件。

同时，本文设计一个维护每个节点信息的数据结构 NODE_INFO,该结构主要提供 CMR 组件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，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 维护所有计算节点的信息，相当于实际容器处理集群的一个虚拟映射，方便 CMR 组件管理集群中的节点。另一方面，集群视频数据迁移模块 GDMM 可以根据 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，根据 ITU 标准介绍了典型的视频监控系统，然后介绍了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核心模块及数据结构的设计。

第四章 基于混合存储的监控视频离线分布式处理系统数据分布策略

在监控视频离线分布式处理过程中，各节点的负载大小和可利用存储资源是动态变化的，因而需要制定合理的数据分布策略，避免出现节点负载差异过大或节点存储资源利用率过低的情况。本章首先分析了传统的基于 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了一种视频数据块处理时间预测模型 PTPM，最后，针对基于混合存储的监控视频离线分布式处理系统，结合 PTPM 模型提出一种视频数据初始分布策略和视频数据动态迁移策略。

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，而未考虑节点存储介质本身的特性。并且，HDFS 分布式文件系统虽然实现了负载均衡器的守护进程用于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，而未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 I/O 密集型的视频任务分配到 I/O 能力弱的节点上，从而导致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大小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。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，从而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 I/O 性能并保证视频任务处理过程中的负载均衡，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（I/O 密集型、计算密集型），这就需要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。另一方面，为了降低视频任务处理过程中过多的网络开销，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[48-52]，然而由于 HDFS 默认不支持实现自定义的数据分布方法，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。此外，Apache Hadoop 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 默认采用基于块的机架感知的副本放置策略，导致不可避免地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 容器技术和 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技术方案。这样不仅能够保证分布式平台在任务管理、资源调度等方面的可靠性，同时由于 Docker 可直接获取硬件资源信息且 Kubernets 任务调度器具有插件化实现机制，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进行了详细的描述。

另一方面，本文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初始分布策略和视频数据动态迁移策略，本章节后续部分将详细介绍。本文提出的视频数据分布策略（包括数据初始分布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，大大缩短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，需要根据集群中各个节点视频数据放置情况和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。为此需要建立一个精准的视频任务处理时间预测。

然而，不同于简单的针对文本数据的处理任务，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，此外，由于集群节点计算和存储能力的异构性，即使是处理同一类型的视频任务中的同一个视频文件，在不同节点的花费时长也不相同，因此，构建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且困难的技术点。

为此，本文分析了当前常用的智能视频处理算法，总结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，最终发现影响视频任务处理时间的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以及视频任务处理类型等。下面将具体说明基于上述六个影响视频任务处理时间的主要因素进行视频数据块处理时间预测模型 PTPM（Processing Time Prediction Module）构建的详细过程。

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划分有 1024*768、1440*900、1280*1024、1920*1080 等，采集到的数据的帧率也不尽相同，例如有 20fps、25fps、27fps 等，我们将不同的分辨率定义为集合 $R=\{r_1,r_2,...,r_m\}$， 其中 m 表示集合 R 中共有 m 种分辨率，r_m 表示集合中第 m 种分辨率；定义集合 $F=\{f_1,f_2,...,f_n\}$ 表示不同的帧率，其中 n 表示集合 F 中有 n 种帧率，f_n 表示集合 F 中第 n 种帧率。根据不同分辨率的集合和不同帧率的集合，本文提出了视频质量的概念，用大写字母 Q 表示，如公式(4-1)所示：

$$Q=R\times F \tag{4-1}$$

集合 Q 是集合 R 和 F 的笛卡尔积，例如：$R=\{r_1,r_2\}, F=\{f_1,f_2\}$，那么 $Q=\{(r_1,f_1),(r_1,f_2),(r_2,f_1),(r_2,f_2)\}$。集合 Q 中的每一个元素，例如 (r_1,f_1) 或者 (r_1,f_2)，表示一个标准视频质量级别，并用 QR 表示。

同时，定义 $Z=\{z_1,z_2,...,z_q\}$ 表示不同的视频任务类型，如视频浓缩、越界检测等，其中 q 表示集合 Z 中有 q 种不同的视频任务类型。结合集合 Q 和 Z，定义集合 S 如式(4-2)所示：

$$S=Q\times Z \tag{4-2}$$

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 $Q=\{(r_1,f_1),(r_2,f_2)\}$，$Z=\{z_1,z_2\}$，那么 $S=\{(z_1,r_1,f_1),(z_1,r_2,f_2),(z_2,r_1,f_1),(z_2,r_2,f_2)\}$。基于集合 S，定义集合 D 如式(4-3)所示：

$$D=\{\overline{d_{s_1}},\overline{d_{s_2}},...,\overline{d_{s_{q\times m\times n}}}\} \tag{4-3}$$

对于集合 S 中的任意一个元素 s_i，集合 D 中的元素 $\overline{d_{s_i}}$ 代表其平均每处理单位时长的 SVPM 为 s_i 的视频数据段所产生的临时中间数据的大小，很容易看到 $\overline{d_{s_i}}$ 值的大小只和视频数据块本身以及视频任务类型有关，而和集群节点的计算和存储能力无关。可以通过在集群中任意一节点离线运行大量的 SVPM 为 s_i 的 SVDBs，获取并计算每一个 SVDB 所产生的临时中间数据大小，并通过进行均值求解获取 $\overline{d_{s_i}}$ 的值的大小。

假设集群中有 N 个计算节点，对于任意一个节点 p_i，定义式(4-4)：

$$E_{p_i}=\{\overline{e_{p_i,s_1}},...,\overline{e_{p_i,s_{q\times m\times n}}}\} \tag{4-4}$$

对于每一个节点 p_i，$\overline{e_{p_i,s_j}}$ 表示其平均每处理单位时长的 SVPM 为 s_j 的视频数据所需要的 CPU 计算时间的大小，$\overline{e_{p_i,s_j}}$ 的值可以通过在节点 p_i 上离线运行大量 SVPM 为 s_i 的 SVDBs，获取每个 SVDB 的 CPU 计算时间值，并求解均值获得。

对于任意一个视频数据块 SVDB，其在节点 p_j 的总的完成时间既包括 CPU 时间也包括 I/O 时间，基于以上的定义，对任意 SVDB，可以通过公式(4-5)计算节点 p_j 上对其处理所需要花费的总的处理时间大小。

$$P_{i,p_j}=\overline{e_{p_j,s_k}}\cdot ts_i+\frac{W_i+\overline{d_{s_k}}\cdot ts_i}{\overline{r_{p_j}}}+\frac{\overline{d_{s_k}}\cdot ts_i}{\overline{w_{p_j}}} \tag{4-5}$$

其中 s_k 对应第 i 个 SVDB 的 SVPM；$\overline{e_{p_j,s_k}}$ 表示在节点 p_j 上处理第 i 个 SVDB 时，每处理单位长度视频片段所需的 CPU 计算时间；$ts_i=t_i^2-t_i^1$，其中 t_i^1 和 t_i^2 分别是第 i 个 SVDB 的视频开始时间和结束时间；W_i 是第 i 个 SVDB 的视频体积大小；$\overline{r_{p_j}}$ 和 $\overline{w_{p_j}}$ 分别是节点 p_j 的平均的数据读写速率，可以通过常用的磁盘读写速

率基准测试工具 hdparm[53]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$(\overline{r_{p_j}, w_{p_j}})$应该是两组值，分别表示位于节点$p_j$的 SSD 的视频数据的平均读写速率和位于节点$p_j$的 HDD 存储介质上的视频数据的平均读写速率。

一个视频处理任务通常关联多个 SVDBs，对于在节点p_j上运行的任意一个视频处理任务，其所花费的处理时间T_{p_j}如式(4-6)所示：

$$T_{p_j} = \sum_{i=1}^{M} P_{i,p_j} \tag{4-6}$$

其中M是节点p_j上放置的当前运行的视频任务相关的 SVDBs 的个数。

综上，可以看出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，可以准确计算一个节点的视频任务所需的处理时长。

4.3 视频数据初始分布策略

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，从而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，进一步降低整个视频任务的处理效率，因此制定合理的视频数据初始分布策略对于提升整个平台视频任务的处理效率至关重要。

上节本文结合智能视频处理算法的特点构建了 PTPM 模型。本节将提出一种基于 PTPM 模型的视频数据初始分布策略 IDDS（Initial Data Distribution Strategy），该策略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。

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，不同的子任务对应不同的视频数据集。假设当前的视频处理任务可以划分为k个待处理的子任务，每一个子任务对应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。初始分布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提升平台的处理效率。

定义 $\rho=\{p_1,p_2,...,p_m\}$ 代表容器集群的计算节点集合，对于每一个计算节点 p_i，表述为 $p_i=\{ssd_i,hdd_i,ssd_i^{used},hdd_i^{used},ssd_i^{tmp},hdd_i^{tmp}\}$。其中 ssd_i 和 hdd_i 分别表示节点 p_i 的总的 SSD 的存储空间大小和总的 HDD 的存储空间大小。ssd_i^{used} 和 hdd_i^{used} 分别表示节点 p_i 的当前已经使用的 SSD 的存储资源和 HDD 的存储资源，ssd_i^{tmp} 和 hdd_i^{tmp} 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，定义 $J=\{job_1,job_2,...,job_k\}$ 作为当前待处理的视频任务处理集合。$DBS=\{db_1,db_2,...db_k\}$ 表示有序的视频数据块组集合，其中 db_i 表示一组对应于 job_i 的待处理的视频数据块。

本文的视频数据初始分布策略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，假设集群各个节点的初始负载大小 $p_\cdot^{load}$ 为 0，同时定义字符常量 S 代表 SSD，字符常量 H 代表 HDD，$selectMedia_j(1\le j\le|P|)$ 代表节点 p_j 当前选择的用于放置视频数据块的存储媒介类型。详细的视频数据初始分布策略 IDDS 如下：

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

步骤 2： 计算处理 job_i 所产生的临时视频数据量大小 db_i^{tmp} 如式(4-7)：

$$db_i^{tmp}=\sum_{j=1}^{|db_i|}\overline{d_{s_k}}\cdot ts_j \tag{4-7}$$

其中 $ts_j=t_j^2-t_j^1$，t_j^2 和 t_j^1 分别代表监控视频数据集合 db_i 中第 j 个监控视频数据块 SVDB 的结束时间和开始时间。s_k 表示第 j 个监控视频数据块对应的监控视频处理模式 SVPM。$\overline{d_{s_k}}$ 代表处理单位时长的 SVPM 为 s_k 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 ρ，对于每一个节点 p_j，定义 ssd_j^{avail} 和 hdd_j^{avail} 分别表示把视频数据块集合 db_i 放置到其 SSD 和 HDD 后，SSD 和 HDD 的剩余存储空间大小。ssd_j^{avail} 和 hdd_j^{avail} 的值可分别利用公式(4-8)和公式(4-9)计算。

$$ssd_j^{avail}=ssd_j-[ssd_j^{used}+db_i+\max(0,db_i^{tmp}-ssd_j^{tmp})] \tag{4-8}$$

$$hdd_j^{avail}=hdd_j-[hdd_j^{used}+db_i+\max(0,db_i^{tmp}-hdd_j^{tmp})] \tag{4-9}$$

如果公式(4-8)或者公式(4-9)的结果大于等于 0，则认为节点 p_j 满足任务 job_i 的存储空间需求，将它先加入候选节点集合 P'（初始化为空）。如果公式(4-8)的结果大于等于 0，则更新变量 $selectMedia_j$ 的值为 S。如果公式(4-8)小于 0 但公式(4-9)大于等于 0，更新变量 $selectMedia_j$ 的值为 H。

步骤 4：定义 *selected* 代表当前选择的用于处理 job_i 的节点，初始化为 *None*，*selectMedia* 代表当前用于放置 job_i 所关联的视频数据块集合（db_i 和 db_i^{tmp}）的存储媒介类型，并初始化集群当前最小负载 $load_{\min}$ 为无穷大。

步骤 5：从候选节点集合 P' 中选取任意节点 p_j 并将其从节点集合 P' 中移除。

步骤 6：利用 PTPM 模型估算节点 p_j 处理任务 job_i 的花费时间 T_{p_j}，并基于公式(4-10)获取将 job_i 分配给节点 p_j 处理所预计即将到来的处理负载大小：

$$p_j^{upload} = p_j^{load} + T_{p_j} \tag{4-10}$$

如果 $p_j^{upload} < load_{\min}$，更新 $load_{\min} = p_j^{upload}$，$selected = p_j$。

步骤 7：重复步骤 4~步骤 6，直到候选节点集合 P' 为空。

步骤 8：如果 *selectMedia* 等于 S，则将 db_i 放置到节点 *selected* 的 SSD 上，并根据公式(4-11)和公式(4-12)更新 $ssd_{selected}^{used}$ 和 $ssd_{selected}^{tmp}$ 的值，否则将 db_i 放置到节点 *selected* 的 HDD 上，并根据公式(4-13)和公式(4-14)更新 $hdd_{selected}^{used}$ 和 $hdd_{selected}^{used}$ 的值。

$$ssd_{selected}^{used} = ssd_{selected}^{used} - ssd_{selected}^{tmp} + db_i + \max(db_i^{tmp}, ssd_{selected}^{tmp}) \tag{4-11}$$

$$ssd_{selected}^{tmp} = \max(db_i^{tmp}, ssd_{selected}^{tmp}) \tag{4-12}$$

$$hdd_{selected}^{used} = hdd_{selected}^{used} - hdd_{selected}^{tmp} + db_i + \max(db_i^{tmp}, hdd_{selected}^{tmp}) \tag{4-13}$$

$$hdd_{selected}^{tmp} = \max(db_i^{tmp}, hdd_{selected}^{tmp}) \tag{4-14}$$

步骤 9：重复步骤 1~步骤 8，直到集合 J 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 IDDS 算法的伪代码如下所示：

算法 1 视频数据初始分布算法

```
1.  while J 不为空 do
2.      job_i ← J.get();
3.      根据公式(4-7)获取临时中间数据体积 db_i^tmp;
4.      for ∀p_j ∈ ρ do
5.          根据公式(4-8)计算 SSD 剩余存储空间大小;
6.          if p_i 的 SSD 剩余存储空间大于等于 0 do
7.              P' = P' ∪ p_j; selectMedia_j = S;
8.              continue;
9.          根据公式(4-9)计算 HDD 剩余存储空间大小;
10.         if p_i 的 HDD 剩余存储空间大于等于 0 do
```

（续）

算法 1 视频数据初始分布算法

11. $P' = P' \cup p_j$;
12. $selectMedia_j = H$;
13. 初始化 $selected = None$, $selectedMedia = S$, $load_{\min} = Infinite$;
14. **while** P' 不为空 **do**
15. $p_j \leftarrow P'.get()$;
16. 利用 PTPM 模型估算节点 p_j 处理任务 job_i 的花费时间 T_{p_j};
17. 基于公式(4-10)获取节点 p_j 预计处理负载大小;
18. **if** $p_j^{upload} < load_{\min}$ **then**
19. 更新 $load_{\min} = p_j^{upload}$, $selected = p_j$ 以及 $selectedMedia = selectMedia_j$;
20. **if** $selectedMedia$ 等于 S **then**
21. 将 db_i 放置到节点 $selected$ 的 SSD 上;
22. 根据公式(4-11)和公式(4-12)更新 $ssd_{selected}^{used}$ 和 $ssd_{selected}^{tmp}$ 的值;
23. **else**
24. 将 db_i 放置到节点 $selected$ 的 HDD 上，并
25. 根据公式(4-13)和公式(4-14)更新 $hdd_{selected}^{used}$ 和 $hdd_{selected}^{tmp}$ 的值;
26. 更新 $p_{selected}^{load} = load_{\min}$;

4.4 视频数据动态迁移策略

上一节介绍了基于最小负载优先放置原则的视频数据初始分布策略。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 上一些待处理视频文件迁移到当前节点的 SSD 上，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，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视频任务的最终完成时间。因此，如何制定合理的节点内异构储存媒介之间的视频数据迁移策略十分重要，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，也就是使迁移后节点视频任务的总处理时间相对于迁移前的差值最大化。

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 和 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。在不进行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。制定一个合理的集群视频数据迁移策略，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，降低集群负载不均衡程度，对于提升云平台的处理性能也至关重要。

结合以上分析，本文设计并实现了一种负载感知的视频数据动态迁移策略 LADM（Load Aware Data Migration），该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简称为 NLDM（Node Level Data Migration）和一种基于最小相对负载和最小相对完成时间比率的集群视频数据块迁移算法简称为 CLDM（Cluster Level Data Migration）。其中，NLDM 算法通过分析节内各个存储媒介上剩余待处理视频文件的数量，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 算法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 算法和 CLDM 算法的实现过程。

4.4.1 节点内视频数据块迁移算法

对于集群中的每一个节点 $p_i(p_i \in \rho)$，定义 s_{p_i} 代表节点 p_i 的 SSD 存储媒介，h_{p_i} 代表节点 p_i 的 HDD 存储媒介，也分别定义 $R(s_{p_i})$ 和 $W(s_{p_i})$ 代表节点 p_i 的 SSD 的平均视频数据读速率和平均视频数据写速率，$R(h_{p_i})$ 和 $W(h_{p_i})$ 代表节点 p_i 的 HDD 的平均视频数据读速率和平均视频数据写速率，$U(s_{p_i})$ 和 $U(h_{p_i})$ 分别代表节点 p_i 的 SSD 总的存储空间大小和 HDD 总的存储空间大小。最后，定义 $L(p_i)$ 表示处理完所有放置在节点 p_i 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，$L(p_i)$ 的值可通过 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 p_i，定义 $DB_{p_i}^s$ 表示放置在它的 SSD 存储媒介上的视频数据块集合，$DB_{p_i}^h$ 表示放置在它的 HDD 存储设备上的视频数据块集合，那么 $DB_{p_i} = DB_{p_i}^h \cup DB_{p_i}^s$ 则表示放置在节点 p_i 上的所有的视频数据块。对于放置在节点 p_i 上的第 j 视频数据块 $db_{i,j}(db_{i,j} \in DB_{p_i}, 1 \le j \le \left|DB_{p_i}\right|, 1 \le i \le |\rho|)$，定义二进制变量 $o_{i,j}$ 作为它的存储媒介类型标识，$o_{i,j}$ 等于 1 表示视频数据块 $db_{i,j}$ 被放置在节点 p_i 的 SSD 上，$o_{i,j}$ 等于 0 表示视频数据块 $db_{i,j}$ 被放置在节点 p_i 的 HDD 上。那么，对于每一个视频数据块 $db_{i,j}(\in DB_{p_i})$，其磁盘读操作的花费时间可以通过公式(4-15)进行估算：

$$R_{db_{i,j}}^e = V_{db_{i,j}} \cdot o_{i,j} \cdot R(s_{p_i}) + V_{db_{i,j}} \cdot (1 - o_{i,j}) \cdot R(h_{p_i}) \tag{4-15}$$

其中 $V_{db_{i,j}}$ 表示视频数据块 $db_{i,j}$ 的体积大小。

对于每一个节点 p_i，为了得到详细的节点内视频数据迁移计划，定义二进制变量 u_i 指示是否将 $db_{i,j}^h(\in DB_{p_i}^h)$ 从 p_i 的 HDD 迁移到 p_i 的 SSD。u_i 等于 1 表示在当前的本地视频数据迁移任务中需要将 $db_{i,j}^h(\in DB_{p_i}^h)$ 从 HDD 迁移到 SSD，否则的话不迁移。注意，$db_{i,j}^h$ 表示的是放置在节点 p_i 的 HDD 存储媒介中的第 j 个监控视频数据块 SVDB，它和 $db_{i,j}$ 的含义不同。

定义变量 t_i^s 和 t_i^h 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 p_i 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(4-5)，t_i^s 和 t_i^h 的大小可以通过公式(4-16)和公式(4-17)获得。

$$t_i^s = \sum_{j=1}^{\left|DB_{p_i}^s\right|} P_{j,p_i}^s \tag{4-16}$$

$$t_i^h = \sum_{j=1}^{\left|DB_{p_i}^h\right|} P_{j,p_i}^h \tag{4-17}$$

基于公式(4-16)和公式(4-17)，在执行本地视频迁移任务之前，节点 p_i 处理完其放置的所有视频数据所花费的总时间如式(4-18)所示：

$$L^{pre}(p_i) = t_i^s + t_i^h \tag{4-18}$$

定义 t_i^m 代表完成当前节点内视频数据迁移任务所需的迁移时间，t_i^m 的大小可以通过如下公式(4-19)获取：

$$t_i^m = \left(\sum_{j=1}^{DB_{p_i}^h} V_{db_{i,j}^h} \cdot u_j\right) / \varphi_i^m \tag{4-19}$$

其中 $\varphi_i^m = \min(R(h_{p_i}), W(s_{p_i}))$。

同时，定义$t_i^{\tilde{h}}$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$u_j=1(1\le j\le\left|DB_{p_i}^h\right|)$）的其它所有放置在 HDD 上的 SVDBs 所需要花费的时长，$t_i^{\tilde{s}}$代表处理在当前视频迁移任务中从 HDD 迁移到 SSD 放置的所有视频数据块的花费时间，那么$t_i^{\tilde{h}}$的大小可以通过公式(4-20)获取，$t_i^{\tilde{s}}$的大小可以通过公式(4-21)获取：

$$t_i^{\tilde{h}}=\sum_{j=1}^{\left|DB_{p_i}^h\right|}P_{j,p_i}^h(1-u_j) \tag{4-20}$$

$$t_i^{\tilde{s}}=\sum_{j=1}^{\left|DB_{p_i}^h\right|}P_{j,p_i}^s\cdot u_j \tag{4-21}$$

这样，如果执行了当前的本地视频数据迁移任务，对于节点p_i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(4-22)获取：

$$L(p_i)=\max(t_i^s,t_i^m)+t_i^{\tilde{h}}+t_i^{\tilde{s}} \tag{4-22}$$

定义 *Diff* 代表$L^{pre}(p_i)$和$L(p_i)$之间的差异大小，也就是迁移前后节点p_i的处理负载差异。通过结合公式(4-18)~公式(4-22)， *Diff* 的大小可以通过如下公式(4-23)获得：

$$Diff=\begin{cases}\sum_{j=1}^{\left|DB_{p_i}^h\right|}\chi_j^1\cdot u_j & \text{if } t_i^s\ge t_i^m\\ t_i^s+\sum_{j=1}^{\left|DB_{p_i}^h\right|}\chi_j^2\cdot u_j & \text{otherwise}\end{cases} \tag{4-23}$$

这里，$\chi_j^1=P_{j,p_i}^h-P_{j.p_i}^s\cdot ts_j^h/ts_j^s$，$\chi_j^2=\chi_j^1-V_{db_{i,j}^h}/\varphi_i^{\min}$，$ts_j^s$和$ts_j^h$分别对应于第 j 个放置在节点p_i的 SSD 上和 HDD 上的 SVDBs 的视频时长。

最后，本文将节点内数据迁移问题描述为一个求 *Diff* 值的最优化问题，如式(4-24)所示：

$$\max\ Diff \tag{4-24}$$

$$s.t.\ \ \sum_{j=1}^{\left|DB_{p_i}^h\right|}V_{db_{i,j}^h}\cdot u_j\le U(s(p_i))-\sum_{j=1}^{\left|DB_{p_i}^s\right|}V_{db_{i,j}^s} \tag{4-25}$$

$$u_j\in\{0,1\},\ 1\le j\le DB_{p_i}^h \tag{4-26}$$

其中，约束(4-25)保证了迁移后节点p_i的 SSD 上所放置的视频数据块的总的存储空间需求不大于p_i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，本文通过标准的分支定界法（standard branch and bound algorithm）[54]对它进行求解，获取一组$u_j(1\le j\le\left|DB_{p_i}^h\right|)$的值用于指示从 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，需要注意如果求得的 *Diff* 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，此时节点管理器就控制节点内数据迁移模型 LDMM 不执行实际的数据迁移动作。

算法 2 节点内视频数据块迁移算法

1. **if** $\varphi_i \neq 0$ **then**
2. return;
3. pLock();
4. 更新 $\varphi_i = 1$;
5. pReleaseLock();
6. 利用分支定界法求解公式(4-24), 获取节点内迁移计划 *migPlan*;
7. **if** $Diff \leq 0$ **then**
8. return;
9. 根据 *migPlan* 执行节点内视频数据迁移操作;
10. **for** *each* $u_j \in migPlan$ **do**
11. **if** $u_j = 1$ **then**
12. 更新 $o_{i,\left|DB_{p_i}^s\right|+j} = 1$;
13. pLock();
14. 更新 $\varphi_i = 0$;
15. pReleaseLock();

NLDM 算法的主要逻辑如算法 2 所示，首先通过分支定界法获取 *Diff* 的值以及 *migPlan*（第 6 行），这里 *migPlan* 是一组二进制变量 $u_j(1 \leq j \leq \left|DB_{p_i}\right|)$ 组成的列表。LDMM 模块根据 *migPlan* 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，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）。注意本文使用一个全局辅助变量 φ_i 标识当前节点 p_i 是否正在进行迁移任务（本地视频数据迁移任务或者集群视频数据迁移任务）。$\varphi_i = 0$ 表示节点 p_i 当前没有运行任何数据迁移任务，$\varphi_i = 1$ 表示节点 p_i 正在运行本地视频数据迁移任务，$\varphi_i = 2$ 表示节点 p_i 正在进行集群视频数据迁移任务。通过使用辅助变量 φ_i 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，解决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。

4.4.2 集群视频数据块迁移算法

定义 L_{avg} 表示集群当前的平均负载大小，通过公式(4-27)可以首先估算 L_{avg} 的值，

$$L_{avg}=\frac{\sum_{i=1}^{|\rho|}L(p_i)}{|\rho|} \tag{4-27}$$

然后，根据L_{avg}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集合，定义为ρ^h，ρ^h中的每一个节点的负载$L(\cdot)$都大于L_{avg}，另一部分代表所有低负载节点组成的节点集合，定义为ρ^l,ρ^l中的每一个节点的负载大小都小于L_{avg}。同时，定义L^h_{avg}和L^l_{avg}分别代表集合ρ^h中的所有节点负载的平均值和集合ρ^l中所有节点负载的平均值，然后定义θ作为判断集群当前负载是否均衡的指示变量（注意$|x|$表示x的绝对值大小），如式(4-28)所示：

$$\theta=\left|L^h_{avg}-L^l_{avg}\right| \tag{4-28}$$

当$\theta\le$G时，则认为集群当前是负载均衡的，否则认为集群当前是负载不均衡的，需要执行集群视频数据块迁移算法 CLDM 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G是一个给定的常量。

详细的集群视频块数据迁移算法 CLDM 的步骤如下：

步骤 1：对于每一个节点的视频数据块集合$DB_{p_i}(1\le i\le\left|p^h\right|)$，分别计算将其中的任意一个视频数据块$db_{i,k}(db_{i,k}\in DB_{p_i})$迁移到节点$p_j(p_j\in\rho^l)$进行处理所需的预期完成时间$C_{i,j,k}$，$C_{i,j,k}$的值可以根据公式(4-29)计算。

$$C_{i,j,k}=E_{i,j,k}+Z_{i,j,k} \tag{4-29}$$

其中$E_{i,j,k}$代表将视频数据块$db_{i,k}$放置在节点p_j上进行处理所需的花费时间。$Z_{i,j,k}$代表$db_{i,k}$在开始被节点p_j处理之前的等待时长。规定只有在节点p_j的当前剩余的存储空间大小大于视频数据块$db_{i,k}$的存储空间需求时，才可以将$db_{i,k}$迁移到节点p_j进行后续处理，也就是说如果节点p_j的当前 SSD 剩余的存储空间大小和 HDD 剩余空间大小都不满足$db_{i,k}$的存储空间需求的话，将不能将$db_{i,k}$迁移到节点p_j进行放置和处理，相应的本文设置对应的$E_{i,j,k}=\infty$，$Z_{i,j,k}=\infty$。对于任意节点$p_j(p_j\in\rho^l)$，定义$\bar{U}(s_{p_j})$和$\bar{U}(h_{p_j})$分别代表节点p_j的 SSD 和 HDD 的剩余存储空间大小，$\bar{U}(s_{p_j})$和$\bar{U}(h_{p_j})$的值可以分别通过公式(4-30)和公式(4-31)获得。

$$\bar{U}(s_{p_j})=U(s_{p_j})-\sum_{m=1}^{DB^s_{p_j}}V^s_{db_{j,m}} \tag{4-30}$$

$$\bar{U}(h_{p_j})=U(h_{p_j})-\sum_{m=1}^{DB^h_{p_j}}V^h_{db_{j,m}} \tag{4-31}$$

其中$V^{s}_{db_{j,m}}$表示位于节点p_j的 SSD 上第m个视频数据块 SVDB 的体积大小，$V^{h}_{db_{j,m}}$表示位于节点p_j的 HDD 上第m个视频数据块 SVDB 的体积大小。在节点p_j满足$db_{i,k}$的存储空间需求的情况下，$E_{i,j,k}$的大小可以通过后面描述的 PTPM 模型中的公式(4-5)进行计算。本文 SSD 优先考虑的迁移放置策略，也就是说，如果节点p_j的 SSD 的剩余空间大小满足$db_{i,k}$的存储空间需求，就考虑优先将$db_{i,k}$迁移并放置到节点p_j的 SSD 上，此时，公式(4-5)中的$\overline{r_{p_i}}$等于$R(s_{p_i})$，否则考虑将视频数据块$db_{i,k}$迁移并放置到节点p_j的 HDD 上，此时，$\overline{r_{p_i}}$等于$R(h_{p_i})$。

通过公式(4-32)获取$Z_{i,j,k}$的大小：

$$Z_{i,j,k}=\max(L(p_j),T^{Migrate}_{i,j,k}) \tag{4-32}$$

其中$T^{Migrate}_{i,j,k}$代表将$db_{i,k}$从节点p_i迁移到节点p_j的总的花费时间，它的计算公式如公式(4-33)所示：

$$T^{Migrate}_{i,j,k}=t^{r_i}_{i,k}+t^{net}_{i,j,k}+t^{w_j}_{i,k} \tag{4-33}$$

主要包括三部分：第一部分是从节点p_i的存储设备上读取$db_{i,k}$的花费时长$t^{r_i}_{i,k}$，第二部分是将$db_{i,k}$从节点p_i传输到节点p_j的网络传输时长$t^{net}_{i,j,k}$，第三部分是将数据块$db_{i,k}$写入到节点p_j的存储设备（SSD 或者 HDD）上的花费时长$t^{w_j}_{i,k}$。其中$t^{r_i}_{i,k}$的大小可以通过公式(4-15)获取，$t^{net}_{i,j,k}$的大小可以通过公式(4-34)进行估算：

$$t^{net}_{i,j,k}=V_{db_{i,k}}/\beta_{i,j} \tag{4-34}$$

其中$\beta_{i,j}$表示节点p_i和节点p_j之间的网络传输速率，$\beta_{i,j}$值的大小可以使用网络性能测试工具如 *Netperf* [55]进行获取。在本文的方法里，通过利用 *Netperf* 网络性能测试工具周期性地获取$\beta_{i,j}$的值并通过计算这些$\beta_{i,j}$的平均值$\overline{\beta_{i,j}}$作为公式(4-34)中$\beta_{i,j}$的实际大小，以降低测量误差。$t^{w_j}_{i,k}$的大小可以通过公式(4-35)进行计算：

$$t^{w_j}_{i,k}=\begin{cases}V_{db_{i,k}}/W(s_{p_j}), V_{db_{i,k}}<\bar{U}(s_{p_j})\\ V_{db_{i,k}}/W(h_{p_j}), V_{db_{i,k}}<\bar{U}(h_{p_j})\ and\ V_{db_{i,k}}>\bar{U}(s_{p_j})\end{cases} \tag{4-35}$$

步骤 2：从所有$C_{i,j,k}(1\le i\le|p^h|,1\le j\le|p^l|,1\le k\le|DB_{p_i}|)$中选取最小值$C_{i_m,j_m,k_m}$，$C_{i_m,j_m,k_m}$表示将视频数据块$db_{i_m,k_m}$从高负载节点$P_{i_m}$迁移到低负载节点$P_{j_m}$进行处理可以获取的当前最小预期完成时间。同时从集群的低负载节点集合ρ^l中选取当前负载最小的节点定义为P_{l_m}。对于节点P_{l_m}，通过公式(4-36)获取将视频数据块从高负载节点迁移到P_{l_m}上进行处理的最小预期完成时间$C_{\tilde{i}_m,l_m,\tilde{k}_m}$：

$$C_{\tilde{i}_m,l_m,\tilde{k}_m}=\min\{C_{i,l_m,k}\mid 1\le i\le|P^h|,1\le k\le|DB_{p_i}|\} \tag{4-36}$$

步骤 3：根据公式(4-37)计算最小相对完成时间比率R_c，根据公式(4-38)计算最小相对负载比率R_l：

$$R_c = \frac{C_{i_m,j_m,k_m}}{C_{\tilde{i}_m,l_m,\tilde{k}_m}} \tag{4-37}$$

$$R_l = \frac{L(p_{j_m})}{L(P_{l_m})} \tag{4-38}$$

如果 R_c 的值大于等于 R_l，说明将视频数据块 db_{i_m,k_m} 迁移到节点 p_{j_m} 进行处理和将视频数据块 $db_{\tilde{i}_m,\tilde{k}_m}$ 迁移到节点 p_{l_m} 进行处理导致的预期的最小完成时间的差异大于节点 p_{j_m} 和节点 p_{l_m} 之间的负载差异，这种情况下，需要将视频数据块 db_{i_m,k_m} 从节点 p_{i_m} 迁移到节点 p_{j_m} 进行处理。如果 p_{j_m} 上的 SSD 的剩余空间的大小满足 db_{i_m,k_m} 的存储空间需求，则将 db_{i_m,k_m} 从节点 p_{i_m} 迁移到节点 p_{j_m} 的 SSD 上存储，否则将 db_{i_m,k_m} 从节点 p_{i_m} 迁移到节点 p_{j_m} 的 HDD 上存储。

如果 R_c 的值小于 R_l，说明将视频数据块 db_{i_m,k_m} 迁移到节点 p_{j_m} 进行处理和将视频数据块 $db_{\tilde{i}_m,\tilde{k}_m}$ 迁移到节点 p_{l_m} 进行处理导致的预期的最小完成时间的差异小于节点 p_{j_m} 和节点 p_{l_m} 之间的负载差异，这种情况下，需要将视频数据块 $db_{\tilde{i}_m,\tilde{k}_m}$ 从节点 $p_{\tilde{i}_m}$ 迁移到节点 p_{l_m} 上进行处理。如果 p_{l_m} 上的 SSD 的剩余空间的大小满足 $db_{\tilde{i}_m,\tilde{k}_m}$ 的存储空间需求，则将 $db_{\tilde{i}_m,\tilde{k}_m}$ 从节点 $p_{\tilde{i}_m}$ 迁移到节点 p_{j_m} 的 SSD 上存储，否则将 $db_{\tilde{i}_m,\tilde{k}_m}$ 从节点 $p_{\tilde{i}_m}$ 迁移到节点 p_{j_m} 的 HDD 上存储。

步骤 4: 更新集群节点集合 ρ 中的每一个节点的负载大小 $L(\cdot)$，根据公式(4-27)重新计算 L_{avg}^h 和 L_{avg}^l 的大小，根据公式(4-28)重新计算 θ 值，重复步骤 1~3，直到满足 $\theta \le \mathrm{G}$ 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 全局视频数据块迁移算法

1. 初始化 $IterCount = 0, MAX\_COUNT = 1000$
2. **while** $IterCount < MAX\_COUNT$ **do**
3. 根据公式(4-27)分别计算 L_{avg}^h 和 L_{avg}^l 的值，根据公式(4-28)计算 θ 的值;
4. **if** $\theta \le \mathrm{G}$ **then** return;
5. **for** $i = 1$ **to** $\left|\rho^h\right|$ **do**
6. **for** $k = 1$ **to** $\left|DB_{p_i}\right|$ **do**
7. **for** $j = 1$ **to** $\left|\rho^l\right|$ **do**
8. 根据公式(4-29)计算 $C_{i,j,k}$ 的大小;
9. 获取当前负载最小的节点 p_{l_m},以及 C_{i_m,j_m,k_m} 和 $C_{\tilde{i}_m,l_m,\tilde{k}_m}$ 的值;
10. 根据公式(4-37)和公式(4-38)分别计算 R_c 和 R_l 的大小;
11. **if** $R_c \ge R_l$ **then**
12. **if** P_{j_m} 的 SSD 剩余存储空间大小满足 db_{i_m,k_m} 的存储空间需求 **then**

（续）

算法 3 全局视频数据块迁移算法

13. 将节点 P_{i_m} 的视频数据块 db_{k_m} 迁移到节点 P_{j_m} 的 SSD 上存储;
14. 更新 $o_{j_m,\left|DB_{p_{jm}}\right|+1}=1$;
15. **else**
16. 将节点 P_{i_m} 的视频数据块 db_{k_m} 迁移到节点 P_{j_m} 的 HDD 上存储;
17. 更新 $o_{j_m,\left|DB_{p_{jm}}\right|+1}=0$;
18. **else**
19. **if** P_{l_m} 的 SSD 剩余存储空间大小满足 $db_{\tilde{i}_m,\tilde{k}_m}$ 的存储空间需求 **then**
20. 将节点 $P_{\tilde{i}_m}$ 的视频数据块 $db_{\tilde{k}_m}$ 迁移到节点 P_{l_m} 的 SSD 上存储;
21. 更新 $o_{l_m,\left|DB_{p_{lm}}\right|+1}=1$;
22. **else**
23. 将节点 $P_{\tilde{i}_m}$ 的视频数据块 $db_{\tilde{k}_m}$ 迁移到节点 P_{l_m} 的 SSD 上存储;
24. 更新 $o_{l_m,\left|DB_{p_{lm}}\right|+1}=0$;
25. $IterCount = IterCount + 1$

在算法执行过程中，会通过 *MAX _ COUNT* 变量控制整个算法的迭代次数，以防止由于G 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，详细介绍了基于视频数据块处理时间预测模型的视频数据初始分布策略和视频数据动态迁移策略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 系统硬件配置

本系统硬件部分由 12 台物理服务器组成，为保证集群的高可用以及容错性，控制节点配置为 2 个，分别命名为 controller1 和 controller2。同时，为了更好地利用集群物理资源，将包括控制节点在内的全部物理节点配置为工作节点，除去两个控制节点，其它节点依次命名为 node1~node10。每一个节点配置有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

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通过 *Netperf* 获得。

5.1.2 软件配置

虽然目前 Docker 已经原生支持在各种操作系统环境如 Linux、Window 以及 Mac 等安装部署，但考虑性能影响以及为了兼容实验室已有的容器云环境，本文选择在 Linux 操作系统环境下进行集群的软件配置和部署。为此，在每台物理节点上都统一安装了内核版本为 GNU/Linux 3.13.0-32-generic x86 64 的 Ubuntu 14.04.1 LTS()操作系统以及版本为 1.11.1 的 Docker Engine。整个系统平台的开发工作是在实验室电脑上进行，具体的软件开发环境为：

（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；

（4）数据库：Mysql， MongoDB

5.2 系统功能实现

5.2.1 视频数据初始放置功能实现

本文采用 Kubernets 作为整个容器集群的底层支撑组件，负责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。

为了实现第四章提出的视频数据初始分布策略 IDDS，首先基于该初始分布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 runIDDS 方法进行视频数据的初始放置。

5.2.2 节点内视频数据迁移模块实现

如 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子模块组成，主要通过运行节点内视频数据块迁移算法 NLDM 实现节点内视频数据块的优化分布。

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，并将获取的资源使用情况相关数据发送给迁移操作执行子模块以便于迁移操作子模块进行算法初始化。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 存储媒介剩余的可用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，本文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
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是通过运行 NLDM 算法制定节点内迁移策略并将结果输出到迁移操作执行子模块。关于 NLDM 算法的具体实现已经在第四章给出了详细的说明，这里不做累述。为了便于 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
RUN mkdir –p /build
WORKDIR /build
ADD src . #将节点内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，执行节点内视频数据算法
```

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，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，通过 Docker 编排技术对 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
```

然后通过以下命令就可以轻松地在各个节点实现 LDMM 模块的部署：

```
docker-compose up
```

5.2.3 集群视频数据迁移模块实现

集群视频数据迁移模块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 3.4.2 小节所述，集群视频数据迁移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视频数据迁移策略 LADM 中的集群视频数据块迁移算法 CLDM 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和变量的初始化，初始化完成以后执行算法核心流程获取集群视频数据迁移计划。所实现和使用的主要方法如表 5-2 所示。

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

责集群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 4.4.3 小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 的节点上类型为 storeType 的存储媒介中放置。

表 5-2 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

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，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 模型的视频数据初始分布策略 IDDS 和视频数据迁移策略 LADM，本文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，CMR 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，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
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

本节主要基于 5.2.4 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，本文准备了大约 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。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 5-1 系统功能测试图

如图 5-1 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，将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 视频数据块处理时间预测模型准确性验证

为了验证 PTPM 模型的准确性，本文选择了包含 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 CTPM 模型[23]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 CTPM 时间预测模型只根据节点 CPU 时间和视频文件相关特性进行视频任务处理时间

的估算，而未考虑磁盘读写时间和视频任务类型。本文选择的监控视频数据块集合中 SVDBs 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, H25}，视频处理模型集合 SVPM={(L25, V), (M25, V), (H25, V)}，其中 V 表示视频任务处理类型为视频浓缩。

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，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，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，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。

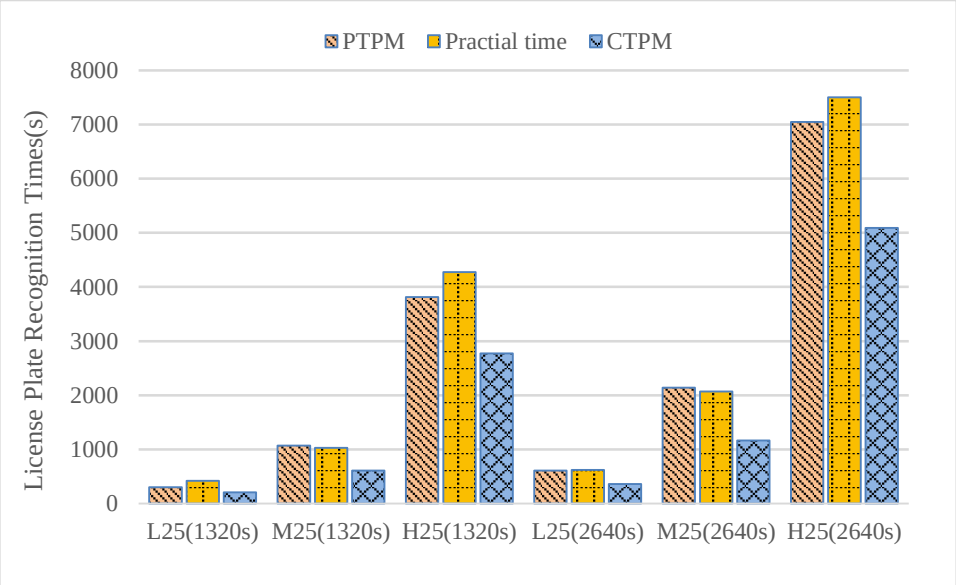

图 5-2 PTPM 模型的预测结果和实际结果对比

如图 5-2 所示，本文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，可以看到本文所提出的 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 监控视频数据块初始分布策略性能验证试验

为了验证本文提出的视频数据初始分布策略 IDDS 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视频质量的 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。

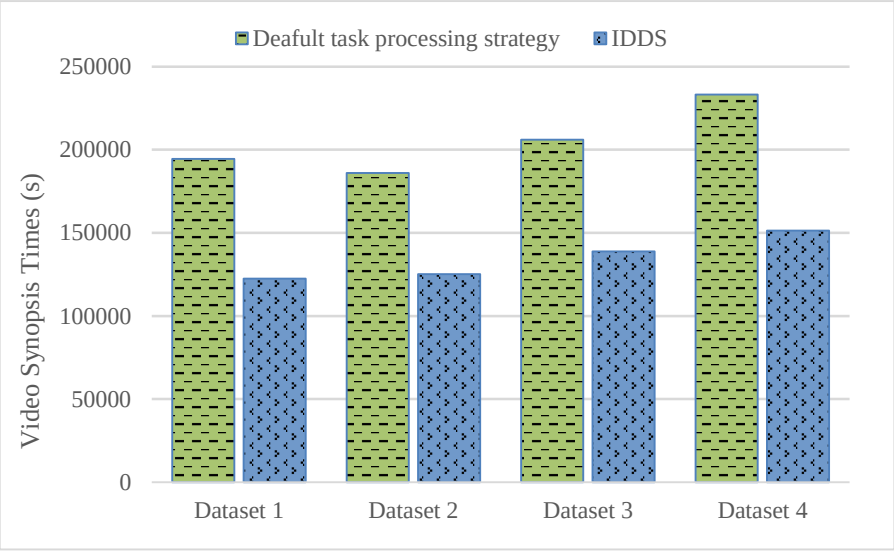

图 5-3 IDDS 和 HDFS 默认的数据放置策略运行时间对比

我们同 Apache Hadoop 的默认的数据放置策略进行了对比。首先从 VS 中分别按照视频数据初始分布策略 IDDS 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，对比两种数据分布策略对视频处理任务中时间的影响。为了避免其它不稳定因素对实验的影响，我们保证实验过程中没有其它任务负载，并且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 5-3 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的平均处理总时长，单位为秒。可以看到，本文提出的视频数据初始分布策略相比于 Apache Hadoop 默认的放置策略可以显著地减少任务的处理时间。

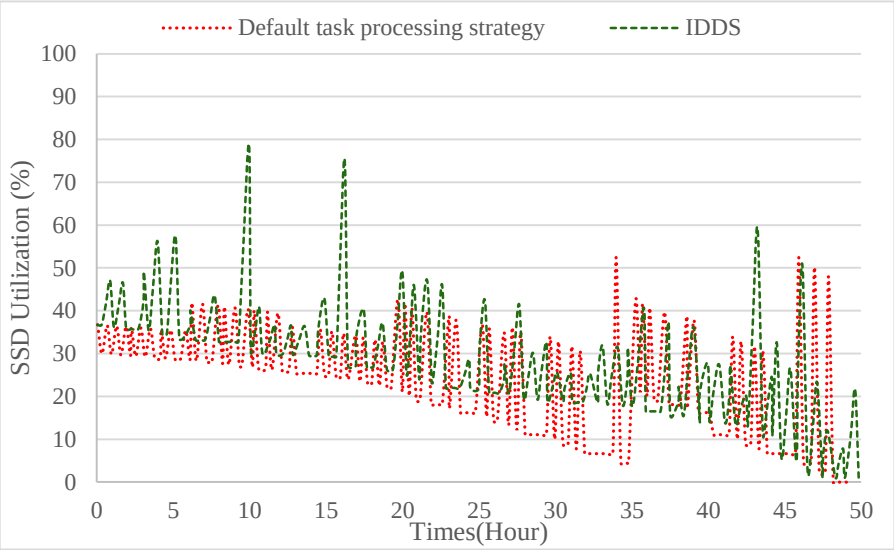

图 5-4 IDDS 和 HDFS 默认的数据放置策略下节点 SSD 资源利用率对比

同时，为了验证本文的视频数据初始分布策略 IDDS 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 5-4 所示，相比于 Apache Hadoop 的默认的数据放置策略，本文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 监控视频数据块迁移策略性能验证试验

接下来，将主要验证本文的视频数据迁移策略 LADM 的效果。本次实验所使用的数据集和 5.3.3 所使用的数据集是一样的。我们和默认 HDFS 数据放置策略以及 DRA 算法（Data Rebalance Algorithm）[23]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，但该方法未考虑节点存储介质异构性以及可利用存储资源动态性。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，本文首先使用 Apache Hadoop 默认的数据放置策略作为平台的初始放置策略，然后使用本文提出的 IDDS 策略作为平台的初始数据分布策略,并观察针对不同数据集，在不同初始分布策略下，使用 DRA 迁移算法和 LADM 迁移策略所导致的任务处理总时长。

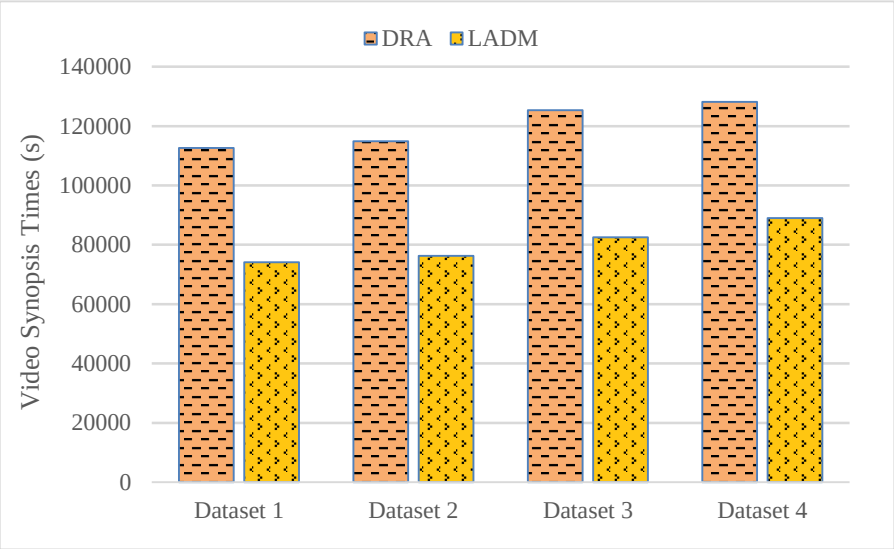

图 5-5 HDFS 默认数据放置策略下的 LADM 和 DRA 算法运行时间对比

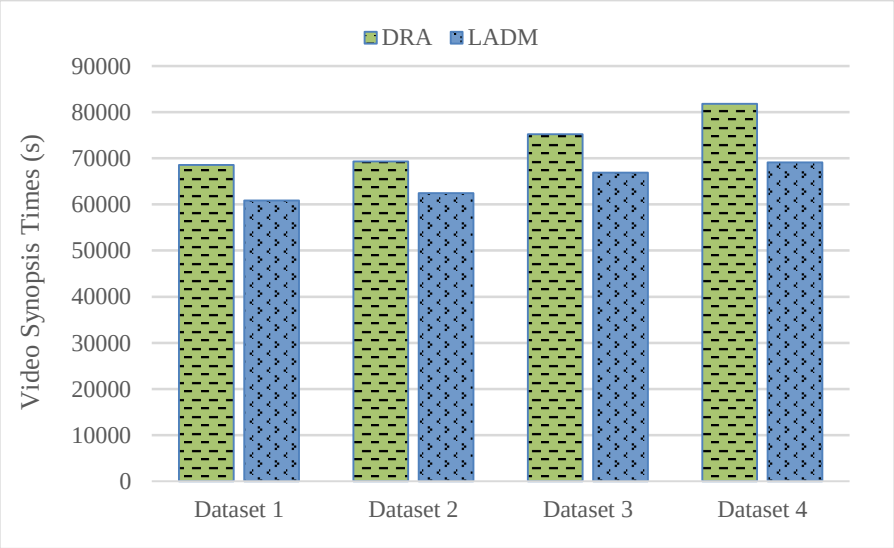

图 5-6 IDDS 初始放置策略下的 LADM 和 DRA 算法运行时间对比

图 5-5 和图 5-6 分别展示了不同初始放置策略下使用 LADM 策略和 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 5-5 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 5-6 为初始放置策略采用 IDDS 初始分布策略的实验结果，通过实验结果可以看出不论采用哪种类型的初始分布策略，本文的视频数据迁移策略 LADM 相比于 DRA 算法均能有效地缩短视频任务的完成时间。

在系统处理每一个监控视频数据集的过程中，本文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），然后通过公式(4-39)计算变量 *Bal* 的值，并通过 *Bal* 来衡量系统的负载均衡程度。

$$Bal = \sqrt{\frac{\sum_{i=1}^{|\rho|}(L(p_i) - L_{avg})^2}{|\rho|}} \tag{4-39}$$

图 5-7 显示了采用 IDDS 初始分布策略前提下，处理不同的监控视频数据集对应的 *Bal* 的大小。其中横坐标为不同的监控视频数据集，纵坐标为 *Bal* 的值，单位为秒。从图中可以看出，相比于 DRA 算法，LADM 数据迁移策略可以更好地均衡系统的负载，提升系统负载均衡程度。

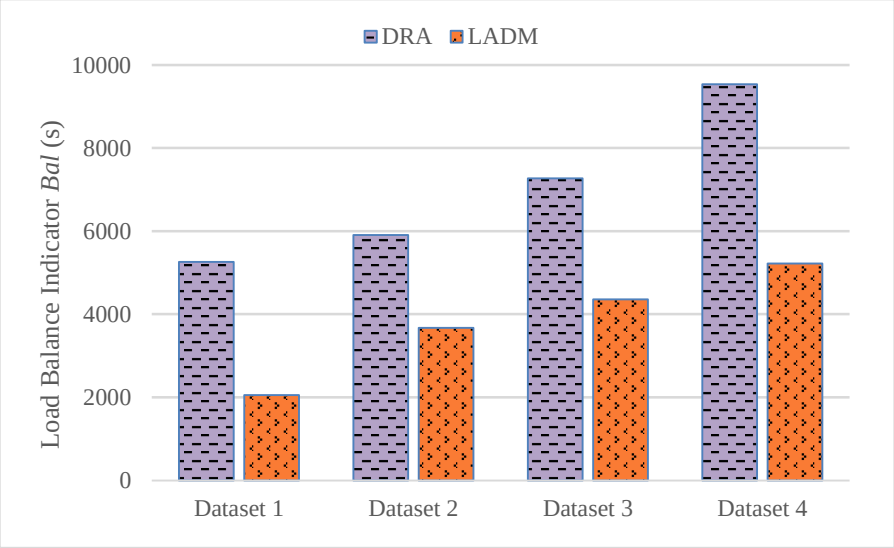

图 5-7 集群负载均衡程度对比

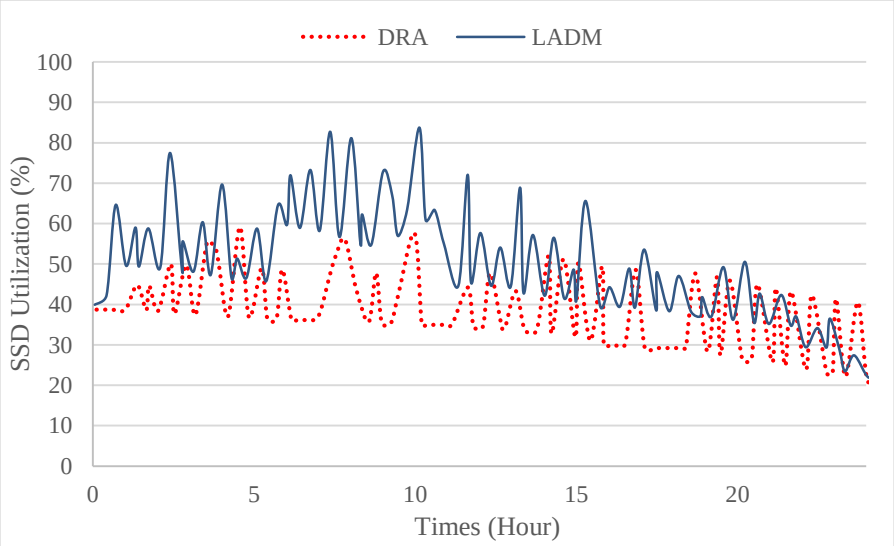

图 5-8 dataSet2 处理过程中 SSD 利用率对比

最后，为了验证本文的视频数据迁移策略 LADM 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-

8 和图 5-9 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，蓝色实线表示采用本文的 LADM 数据迁移策略对应的 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 LADM 策略都会带来更高的 SSD 存储空间利用率。

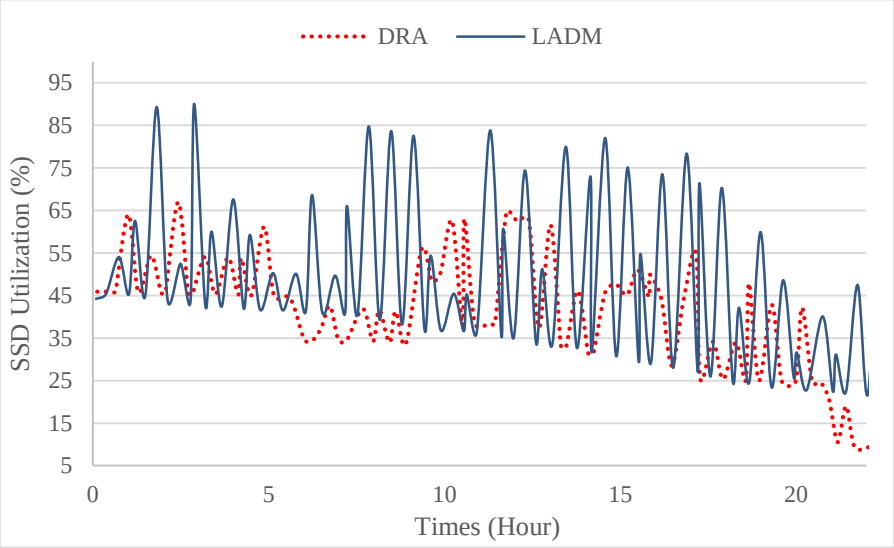

图 5-9 dataSet4 处理过程中 SSD 利用率对比

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视频数据块处理时间预测模型 PTPM 的准确性、视频数据初始分布策略 IDDS 和视频数据迁移策略 LADM 的效果，实验表明基于 PTPM 模型的 IDDS 和 LADM 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并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，基于云计算技术和智能视频分析技术构建监控视频离线分布式处理系统成为近年来解决海量监控视频数据处理需求的主流方案。考虑智能视频分析任务的高的数据读写需求，利用 SSD + HDD 混合存储架构构建监控视频离线分布式处理系统十分必要。然而，当前的监控视频离线处理系统在进行海量视频数据的分布式处理过程中，无法保证集群高性能存储媒介 SSD 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。如何提高基于混合存储的监控视频离线分布式处理系统的存储资源利用率，保证集群的负载均衡，实现一个高性能的监控视频离线分布式处理系统是本文研究的重点。围绕这一目标，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，发现它们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，估算出视频数据块在不同计算节点上处理的花费时间。

（2）面向监控视频云平台的视频数据初始分布策略

传统的云平台在进行视频数据的初始放置时没有考虑节点间的初始负载差异以及节点存储介质异构性。本文基于所提出 PTPM 时间预测模型，研究并实现了一种异构感知和最小负载优先的视频数据初始分布策略 IDDS。该策略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，在考虑节点存储介质异构性的基础上，每次选取当前负载最小的计算节点进行放置，降低了集群中各个节点的初始负载差异，保障了集群节点的高的资源利用率和视频任务处理性能。

（3）基于负载感知的视频数据迁移策略

传统的监控视频离线处理平台在进行视频任务分布式处理过程中或者没有考虑节点负载和可利用存储资源动态性，或者没有考虑集群存储资源异构性特点，容易造成集群存储资源利用率低和负载不均衡。本文通过分析分布式场景下视频任务处理特点，综合考虑节点负载动态性以及存储资源异构性，提出了一种负载感知的监控视频数据块迁移策略，该策略可以制定合理的视频数据块迁移计划用于指导集群进行视频数据的优化分布。通过实验验证，该策略能有效提升集群 SSD 存储资源利用率、降低集群负载不均衡程度并进一步提升视频任务处理效率。

（4）基于 Docker 容器技术的验证系统

传统的监控视频离线分布式处理系统采用 Apache Hadoop 构建，难以获取节点物理资源特性和进行功能扩展。Docker 容器及其集群相关技术成为近年来云计算领域的热门技术，Docker 容器能够访问真实的物理资源并且 Docker 集群技术如 Kubernents 等可以很方便进行自定义功能扩展。本文采用 Docker 技术构建监控视频离线分布式处理系统，并按照本文设计的视频数据初始分布策略 IDDS 和视频数据迁移策略 LADM 扩展实现了数据分布相关功能模块，负责视频任务分布式处理过程中数据的优化分布。最后，通过视频浓缩服务验证了系统高效的任务处理效率。

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，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初始分布以及迁移策略，最后通过基于 Docker 容器技术的视频浓缩服务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。但是本文工作还存在一些可以进一步研究的方向：

（1）PTPM 时间预测模型优化。目前的视频数据块处理时间预测模型 PTPM 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 Logistic Regression、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，由于还存在许多其它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。本文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和算法的有效性，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，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] Wu L, Zhuge Q, Sha E H M, et al. BOSS: An Efficient Data Distribution Strategy for Object Storage Systems With Hybrid Devices [J]. IEEE Access, 2017, 05: 23979-23993.

[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]. // The 25th ACM International Conference on Supercomputing, ACM, 2011: 22-32.

[27] Yang Q, Ren J. I-CASH: Intelligently Coupled Array of SSD and HDD [C]. // 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/docs/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] Zhang H T, Ma H D, Fu G P, et al. Container Based Video Surveillance Cloud Service with Fine-Grained Resource Provisioning [C]. // The 9th International Conference on Cloud Computing, IEEE, 2016: 758-765.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] hdparm [EB/OL]. https://sourceforge.net/projects/hdparm, 2017-05-01.

[54] H. P. Williams. Logic and integer programming [M]. Springer, 2009.

[55] Hewlett Packard Enterprise. Netperf [CP/OL]. https://github.com/HewlettPackard/netperf, 2010-12-22.

致谢

2014年9月底，我与北京邮电大学计算机学院智能通信软件与多媒体北京市重点实验室结下了难得的缘分，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，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它同学们，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

[1] **Yangyang Gao**, Haitao Zhang, Bingchang Tang, Yanpei Zhu, Huadong Ma. Two-Stage Data Distribution for Distributed Surveillance Video Processing with Hybrid Storage Architecture [C]. // 2017 IEEE 10th International Conference on Cloud Computing (IEEE CLOUD 2017), Hawaii, USA, 2017. (CCF C 类)

[2] **Yangyang Gao**, Haitao Zhang, Yanpei Zhu, Bingchang Tang, Huadong Ma. 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 (IEEE HPCC 2017), Bangkok, Thailand, 2017. (CCF C 类)

[3] Haitao Zhang, Huadong Ma, Guangping Fu, Xianda Yang, Zhe Jiang, **Yangyang Gao**. Container based Video Surveillance Cloud Service with Fine-Grained Resource Provisioning[C]// 2016 IEEE 10th International Conference on Cloud Computing (IEEE CLOUD 2017), San Francisco, USA, 2017. (CCF C 类)