**密级：保密期限：**



**硕士学位论文**



**题目：****监控视频云计算平台中数据放置策略**

**研究与应用**

**学号：2014110709**

**姓名：许彬**

**专业：计算机技术**

**导师：李文生**

**学院：计算机学院**

**年月日**

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

学位论文作者完全了解北京邮电大学有关保留和使用学位论文的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属北京邮电大学。学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，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，可以允许采用影印、缩印或其它复制手段保存、汇编学位论文。（保密的学位论文在解密后遵守此规定）

保密论文注释：本学位论文属于保密在年解密后适用本授权书。非保密论文注释：本学位论文不属于保密范围，适用本授权书。

本人签名：日期：

导师签名：日期：

监控视频云计算平台中数据放置策略研究与应用

摘要

随着社会对安防的重视，监控摄像头遍布城市的各个角落，而且数量还在不断增长，由此产生了海量的监控视频数据，分布式处理监控视频数据成为一种趋势。传统的监控视频离线处理系统在分布式处理大规模视频数据时，集群内节点存在负载不均衡的问题，集群内节点之间往往会产生大量的网络数据传输，从而降低了分布式视频处理任务的计算效率。

传统的分布式监控视频离线处理系统存在的问题表明视频大数据的分布式存储策略与分布式计算框架的协作显得尤为重要。首先，本文通过分析当前主流的智能监控视频处理算法和分布式计算框架，在实验的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（CTPM），该预测模型利用了视频数据块的特征，例如分辨率、帧率、时长以及集群中节点的计算能力等，能够预测视频处理任务在不同节点上所需要的处理时间。其次，本文提出一种基于CTPM时间预测模型的监控视频数据块初始放置算法（IDPA），该策略将同属于一个处理任务的视频数据块集合DBG作为基本的放置单元，通过CTPM模型预测每个DBG的处理时间，然后基于负载均衡原则将数据块在集群中合理放置。考虑到混合集群中节点的计算能力会动态变化，导致IDPA算法出现较大的误差，因此本文提出一种基于CTPM时间预测模型的数据迁移算法（DRA），对某些计算能力发生显著变化的节点进行数据迁移，保证集群负载均衡，提高集群的资源利用率。最后，为了验证本文提出的监控视频数据放置策略，本文实现了基于Spark分布式计算框架的离线视频浓缩算法，使用交通监控视频作为数据源，实验结果表明，IDPA和DRA算法能够显著地降低监控视频分布式处理任务的完成时间，使集群内各节点负载均衡，提高了集群的资源利用率。

关键词：数据放置策略，数据迁移策略，离线视频处理，云计算

**RESEARCH AND APPLICATION OF DATA PLACEMENT STRATEGY IN SURVEILLANCE VIDEO CLOUD COMPUTING PLATFORM**

ABSTRACT

With the society's emphasis on security, the number of surveillance cameras in all corners of the city is growing. This results inthe massive monitoring video data. Distributed large scale surveillance video processing becomes a trend. The nodes within the cluster may become load imbalance when we use the traditional distributed video off-line processing systems to processthelarge-scale video data. At the same time, there is no cooperation betweenthe data distribution and the computing in the traditional distributed data processing system, and this leads to more data transmission cost in the cluster. All the reasons above will reduce the computational efficiency of video off-line processing systems.

The problem of the traditional distributed video processing system shows that the collaboration between the distributed storage strategy and the distributed computing is very important. First of all, this paper presents a ComputingTime Prediction Model (CTPM) for monitoring video data blocks based on the analysis of the current mainstream intelligent video processing algorithms and distributed computing framework. The prediction model utilizes the video data block’s feature, such as resolution, frame rate, duration, and the computing power of the nodes in the cluster, and it can predict the processing time required by the video processing tasks at different nodes. Secondly, this paper proposes an Initial Data Placement Algorithm (IDPA) for video data blocks based on CTPM, which uses theData Block Group(DBG) of video data blocks belonging to a processing task as the basic placement unit, and predicts each DBG’s processing time by CTPM, and then places these video data blocks in the cluster by IDPA. Considering that the computing power of the nodes in the cluster will change dynamically which results in a greater error of the IDPA algorithm. Therefore, this paper proposes a Data Rebalance Algorithm (DRA) based on the CTPM. For some nodes with significant changes in computing power, DRA can ensure the cluster load balancing and improve the cluster resource utilization. Finally, in order to verify the video data placement strategy proposed in this paper, an off-line video synopsisalgorithm based on Spark framework is implemented. Traffic monitoring video is used as the data source. Experimental results show thatIDPA and DRA algorithms can significantly reduce the completion time of the distributed surveillance video processing tasks, so that all nodes within the cluster are load-balanced and improve the cluster resource utilization.

KEY WORDS: data placement, data rebalance, off-line surveillance video processing, cloud computing

**目录**

[第一章 绪论 1](#_Toc476847695)

[1.1 研究背景及意义 1](#_Toc476847696)

[1.2 国内外研究现状 2](#_Toc476847697)

[1.3 论文的主要研究内容 3](#_Toc476847698)

[1.4 论文组织结构 4](#_Toc476847699)

[第二章 相关技术介绍 6](#_Toc476847700)

[2.1 分布式计算框架 6](#_Toc476847701)

[2.1.1 MapReduce计算框架 6](#_Toc476847702)

[2.1.2 Spark计算框架 7](#_Toc476847703)

[2.2 HDFS分布式文件存储系统 9](#_Toc476847704)

[2.3 数据放置技术 10](#_Toc476847705)

[2.3.1 数据放置策略分类 10](#_Toc476847706)

[2.3.2 HDFS分布式文件存储系统中的数据放置策略 12](#_Toc476847707)

[2.4 视频浓缩相关技术 13](#_Toc476847708)

[2.4.1 目标检测技术 13](#_Toc476847709)

[2.4.2 视频浓缩算法 14](#_Toc476847710)

[2.5 本章小结 15](#_Toc476847711)

[第三章 监控视频离线分布式处理系统总体设计 16](#_Toc476847712)

[3.1 需求分析 16](#_Toc476847713)

[3.2 视频监控系统介绍 16](#_Toc476847714)

[3.3 监控视频离线分布式处理系统架构设计 17](#_Toc476847715)

[3.3.1 总体架构设计 18](#_Toc476847716)

[3.3.2 监控视频离线分布式处理系统工作流程设计 18](#_Toc476847717)

[3.4 核心模块的设计 20](#_Toc476847718)

[3.4.1 Controller组件设计 20](#_Toc476847719)

[3.4.1 核心数据结构设计 21](#_Toc476847720)

[3.5 本章小结 24](#_Toc476847721)

[第四章 云平台中视频数据放置策略的实现 25](#_Toc476847722)

[4.1 基于HDFS的监控视频数据放置策略 25](#_Toc476847723)

[4.1.1 HDFS默认放置策略 25](#_Toc476847724)

[4.1.2 解决方案 26](#_Toc476847725)

[4.2 视频数据块处理时间预测模型 26](#_Toc476847726)

[4.2.1 模型的建立 26](#_Toc476847727)

[4.2.2 模型的初始化和参数更新 28](#_Toc476847728)

[4.3 监控视频数据初始放置算法 29](#_Toc476847729)

[4.3.1 数据放置算法 29](#_Toc476847730)

[4.3.2 数据放置算法实现 30](#_Toc476847731)

[4.4 视频数据块迁移算法 31](#_Toc476847732)

[4.4.1 视频数据块迁移算法 31](#_Toc476847733)

[4.4.2 视频数据块迁移策略实现 32](#_Toc476847734)

[4.6 本章小结 33](#_Toc476847735)

[第五章 系统实现及测试 34](#_Toc476847736)

[5.1 基于Spark分布式计算框架的视频离线处理系统实现 34](#_Toc476847737)

[5.1.1 HDFS数据放置过程的实现 35](#_Toc476847738)

[5.1.2 HDFS数据放置迁移的实现 35](#_Toc476847739)

[5.2 视频浓缩算法实现 35](#_Toc476847740)

[5.3 测试系统功能与算法效果验证 38](#_Toc476847741)

[5.3.1 测试环境 38](#_Toc476847742)

[5.3.2 测试系统功能验证 38](#_Toc476847743)

[5.3.3 CTPM准确性验证实验 39](#_Toc476847744)

[5.3.4 IPDA初始数据放置算法的效果验证实验 40](#_Toc476847745)

[5.3.5 DRA数据迁移算法的效果验证实验 41](#_Toc476847746)

[5.4 本章小结 42](#_Toc476847747)

[第六章 总结与展望 43](#_Toc476847748)

[6.1 总结 43](#_Toc476847749)

[6.2 展望 44](#_Toc476847750)

[参考文献 46](#_Toc476847751)

[攻读硕士学位期间的主要研究成果 50](#_Toc476847752)

[致谢 51](#_Toc476847753)

# 绪论

1.1研究背景及意义

随着全球范围内安全形势的日益严峻，全球各国都加大了对视频监控的部署，粗略估计全球范围内至少部署了1亿个摄像头，每天产生监控视频数据至少有200T。面对如此大规模的视频监控数据，传统基于人工分析视频监控信息的方式已经不能满足实际需求。各种智能视频处理技术应运而生，例如目标检测与跟踪、监控视频增强、视频浓缩[3]等，这些智能视频处理技术的出现大大提高了人们从监控视频中获取信息速度。但是在摩尔定律破灭后，基于单机的监控视频离线处理平台越来越难以应对当今计算量巨大的视频大数据处理任务。如何快速地处理这些监控视频数据，成为监控视频离线处理平台的挑战之一。近年来随着分布式计算以及其他相关技术的飞速发展，基于各种分布式计算框架的监控视频云计算平台逐渐成为监控视频离线处理平台的一种解决方案。例如文献[4]、文献[5]分别提出了基于Hadoop和Spark分布式计算框架的监控视频处理方案。

监控视频云计算平台将分散的、异构的、单机计算能力较弱的计算资源整合起来，形成强大的集群处理能力来应对监控视频大数据所带来的巨大计算量的问题。任务管理、任务调度和资源管理是传统的分布式计算中三个基本功能。如何方便的管理任务、如何调度任务使集群负载均衡、如何高效的管理资源是监控视频云计算平台同样需要面对问题，同时由于视频数据在传输过程中耗费的网络资源巨大，监控视频云计算平台还要考虑视频处理任务和任务所要的数据之间的协同问题，尽量减少监控视频任务在处理过程中的数据传输量。

在处理大数据，尤其是视频数据时，数据放置和数据计算的协同问题非常重要。云计算平台中的数据放置策略问题，也是研究的热点问题。如果数据放置策略不合理，则在任务执行的过程中会进行跨节点读取数据，由于传输视频数据会占用大量的网络传输资源，导致集群的性能降低。同时由于视频任务不同于传统的文本数据在使用传统的任务调度机制调度时，会导致集群内负载不均衡的情况发生，进一步降低集群的性能，增加视频处理任务处理的总时间。

随着HDFS分布式文件系统以及MapReduce、Spark等分布式计算框架的不断发展和完善，基于这些技术实现的分布式云计算平台，在任务管理和资源管理方面的表现很好。但是处理监控视频数据时，由于HDFS默认的数据放置策略和分布式计算框架默认的任务调度策略不能很好协同工作以及视频任务相对文本任务的特殊性导致集群内节点负载不均衡以及集群内产生大量网络传输的情况发生，导致云计算集群利用率降低，使监控视频离线处理任务耗费的总时间大大增加。本文针对上问题，利用Spark、MapReduce等分布式计算框架在使用HDFS作为其文件系统时，支持数据本地化（Spark或者MapReduce的任务调度机制会将任务分配给存储该任务所需数据的节点处理）的功能，本文提出一种针对监控视频云计算平台的数据放置策略从而解决上述问题。

1.2国内外研究现状

云计算平台中数据放置策略问题，随着大数据，尤其是视频大数据时代的到来，面临着巨大的挑战。众多研究者对云计算平台中数据放置及数据迁移策略等问题进行了的相关研究。

文献针对云计算平台中跨数据中心工作流的高效执行问题，提出了新的数据放置策略，该策略在工作流的初始化阶段就把有关联的数据放置到相同的数据中心；然后尽量将任务调度到相关数据中心去执行。通过实验验证，该策略有效减少了数据中心之间的数据传输量，大大提高了科学工作流的运行效率。

文献提出一种使用实时访问模式的Hadoop动态数据放置策略，这种新数据放置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。

文献提出一种新的数据放置策略自适应地平衡每个节点中存储的数据量，以实现提高的数据处理性能。实验结果表明，在异构Hadoop集群中执行数据密集型应用程序之前，这种数据放置策略可以通过跨节点重新平衡数据来提高MapReduce性能。

文献提出一种基于任务计算比数据放置策略并通过两个标准的MapReduce应用程序WordCount和Grep进行实验，通过实验验证了该文献提出的数据放置策略显著提高了MapReduce的执行效率。

文献提出了一个称为RCFile（记录列文件）及其在Hadoop系统中的实现的大数据布局策略。通过强化实验，验证了RCFile及其在Hadoop系统中的实现的大数据布局策略相比Hadoop默认的设置能够更加快速地进行数据加载、快速地查询处理、更加高效的存储空间利用、对高度动态工作负载模式的更强适应性等。

文献中，提出了一种用于大规模异构Hadoop集群的新型snakelike数据放置机制（SLDP）。SLDP采用异构感知算法首先将各个节点划分为若干个虚拟存储层（VST），然后根据数据的热度，在每个VST中跨节点放置数据块。此外，SLDP使用热度成比例复制以减少磁盘空间消耗，并且还具有有效的功率控制功能。两个真实数据密集型应用的实验结果表明SLDP更加节能和节省空间，能够显着提高异构Hadoop集群中的MapReduce性能。

通过对上述文献研究成果的分析和总结，我们发现目前关于数据放置策略的研究，很少有针对监控视频处理领域数据放置策略的研究。由于视频数据传输的代价很大，监控视频处理领域的数据放置策略十分重要，初始的放置策略十分重要，能够大大减少处理过程中的数据传输量，减少集群间的网络传输，提高集群的利用率。同时监控视频数据的存储与处理技术的协同也必须考虑，要保证视频数据合理的存储到集群节点上，使得在任务运行过程中保证集群内节点的负载均衡，在集群内节点的计算能力的发生变化的时候能够及时的进行数据迁移操作，避免集群出现负载不均衡的情况，提高集群的运行效率，降低任务完成的总时间。

1.3论文的主要研究内容

传统的监控视频视频处理平台，在分布式处理海量视频数据时，集群内节点容易出现负载不均衡的情况或者集群内部产生大量网络传输的情况，进而导致集群使用率降低，增加任务的处理时间。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（CTPM）的数据初始放置策略和数据迁移策略，并实现了基于Spark计算框架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（CTPM），该预测模型根据视频数据块的本身的一些特征，例如分辨率、帧率、时间长度、以及集群中节点计算能力等，通过自适应的调整参数，建立时间预测模型，能够预测视频数据块在不用计算节点上的所需要的处理时间，从而计算得出监控视频离线处理任务时间。

1. 基于CTPM时间预测模型的视频数据块初始放置算法

研究并实现基于CTPM时间预测的数据初始放置算法（IDPA）。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，为了减少放置策略消耗的时间，提出一种新的数据放置算法，能够快速的计算出基本放置单位的放置位置。

1. 基于CTPM时间预测模型的视频数据块迁移算法

研究并实现基于CTPM时间预测模型的数据迁移算法（DRA）。由于集群内计算节点的计算能力受多种因素的影响，例如随着该节点上的负载情况不断的变化而变化，需要对这些计算能力发生变化的节点，及时进行数据迁移，保证集群负载均衡，提高集群的利用率。DRA能够就是获取集群中节点的负载变化，及时将集群中负载过大的节点的上的数据迁移到负载较小的节点上，从而保证集群负载均衡，提高集群的资源利用率。

1. 基于Spark计算框架的离线视频浓缩算法验证系统

基于以上提出的数据放置策略和迁移策略，实现了基于Spark分布式计算框架的离线视频浓缩算法，使用车辆监控视频作为数据源，对IDPA和DRA算法进行了验证。该系统使用HDFS作为分布式存储系统，通过使用HDFS-385和HDFS-2576的Patch实现了IDPA和DRA算法，Spark作为分布式计算框架，当用户提交视频摘要任务后，会按照基于CTPM时间预测模型的IDPA数据放置策略和DRA迁移策略进行数据的放置和移动，由于SPARK支持数据本地化，会优先将任务分配给存储该任务数据的节点，大大降低了集群内节点之间的网络传输，同时保证了集群负载均衡，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。本章主要介绍监控视频云计算平台中常用的分布式计算框架、分布式文件系统、数据放置策略以及视频浓缩等相关技术。

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（CTPM），并详细介绍了建立模型的过程。然后介绍了基于CTPM时间预测模型的初始数据放置策略（IDPA），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Spark计算框架的离线视频浓缩算法验证系统中各个组成部分的详细实现；接着论述了基于Spark计算的视频浓缩算法的实现；之后介绍了测试环境配置；最后对本文提出的CTPM模型的准确性、IDPA和DRA算法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放置策略未来的研究趋势和分析了本文提出的数据放置策略进一步的优化方向。

# 相关技术介绍

本章主要介绍监控视频云计算平台中常用的分布式计算框、HDFS分布式文件存储系统、数据放置策略以及视频浓缩算法所涉及的相关技术。

2.1分布式计算框架

2.1.1MapReduce计算框架

MapReduce 是一个高性能的批处理分布式计算框架，由谷歌在 2004 年发表的论文中提出。该计算框架最初被 Google 设计用于处理海量网页的索引问题，解决了搜索引擎面临的海量数据处理扩展性差的问题。受到函数式编程的启发，MapReduce将大数据处理任务拆分成多个可以并行执行的子任务，然后将这些子任务分发到集群中各个节点上去运行，再由多个合并任务将上述子任务产生的中间结果合并，最后获得任务执行结果文件。编程人员在使用该计算模型编程处理大数据任务时，不需要再关注分布式计算中的任务调度，资源管理、容错机制、通信机制等问题，只需要关注如何编写 Map 和 Reduce 函数即可，大大降低了编写分布式计算任务的难度[18]，有效的提高了编程人员的工作效率，同时降低了工程的复杂性。MapReduce 的执行流程如图 2-1所示：

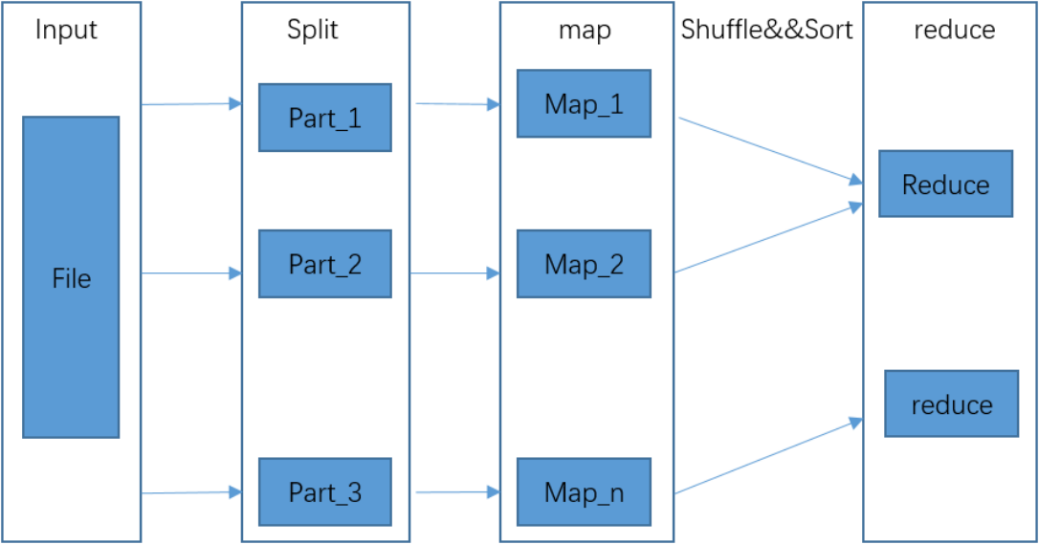


图 2-1 MapReduce 流程图

1. Input 阶段：为 MapReduce 任务设置输入文件的位置，同时会将输入文件进行分割，默认大小为64MB并将这些数据块以键值对的格式传送给Map阶段。
2. Map 阶段：接收 Input 阶段的键值对，调用 Map 函数中用户实现的代码，处理输入数据，产生新的 key-value 键值对。
3. Shuffle 和 Sort 阶段：为了保证 key 值相同的中间数据尽量由同一个 Reduce 处理以及对Map 产生的键值对进行排序。在 Shuffle 阶段，为了将相同 key 的中间结果尽可能汇集到同一节点上，通过函数映射完成混排交换数据；而在 Sort 阶段，为了后续过程处理的简便性，需要将按照 key 的值对Map输出的key-value键值对进行分组并排序，同时为了处理的方便，需要将 key值相同的key-value键值对放在数据节点相同的分区中。
4. Reduce 阶段：Reduce主要是将Map阶段产生的数据按照用户的要求进行拼装，并把结果写入对应的输出目录中。

由于每一次Map操作或者Reduce操作，其中间结果都要持久化到硬盘当中，导致处理的延迟很大，所以MapReduce一般用于批量的离线任务处理，不适用于实时的数据流处理。同时MapReduce计算框架也支持数据本地化的功能，即当MapReduce的任务调度系统在分配Map或者Reduce任务的时候，会优先将任务分配给存储该任务所需要数据的节点来处理，这样可以减少在处理任务的过程中频繁的移动数据导致集群内产生大量的网络传输量，降低集群的处理效率。

2.1.2Spark计算框架

Spark是一个大数据分布式编程框架，由Zaharia M在2013发表的论文中首次提出，Spark不仅实现了MapReduce的算子map函数和reduce函数及计算模型，还提供更为丰富的算子，如过滤（filter）、连接（join）等算子。Spark将分布式内存抽象为弹性分布式数据集（RDD），RDD是一个只读的记录分区集合，实现了分布式内存计算，大大提高Spark计算框架的计算速度。同时Spark实现了基于RDD实现各种数据操作、任务调度、RPC等。Spark的源代码使用函数式语言Scala编写，同时提供了Java、Python等编程语言使用的编程接口。Spark中的数据操作有两种。一种是Transformation操作，另一种是Action操作。Transformation操作不会触发任务的执行，直到遇到Action操作，才会根据RDD上的各种算子生成有向无环图（DAG），然后根据DAG进行分布式任务调度和处理，如图2-2所示。

Spark分布式计算框架采用了分布式计算中的典型的主从模型。主节点是集群中维护Master进程的节点，主节点主要控制整个计算集群，协调集群内个组件的稳定运行。从节点即Slave节点的主要职责是对进行计算，同时向Master节点反馈自己的状态和任务处理的进程；Executor线程是Slave节点上具体负责任务处理的线程；用户通过Client提交Spark应用，Driver组件用于管理Spark应用的执行，如图2-3所示。

下面详细介绍Spark的架构中的基本组件。

·ClusterManager：负责监控和管理整个集群。

·Worker：Slave节点，负责控制计算节点，启动Executor或Driver。

·Driver：Driver主要负责运行Spark Application并创建SparkContext。

·Executor：具体负责执行spark任务的线程池。

·SparkContext：负责控制Spark应用的生命周期。

·RDD：弹性数据分布式数据集，是一个只读的分区记录集合。

·DAG Scheduler：将作业划分为DAG，然后将子任务提交给TaskScheduler。

·TaskScheduler：负责调度子任务给Executor执行。



图 2-2 Spark的任务处理流程图



图 2-3 Spark架构图

Spark整体的工作流程：用户通过Client提交应用后，主节点通过相关算法指定一个工作节点启动Driver，主节点或者资源管理器分配给Driver资源后，然后Driver组件将用户提交的Spark作业转换为RDD依赖图，之后DAGScheduler将各个子任务交给任务调度器，任务调度器根据调度算法将各个任务分配给相关的Executor执行。

2.2HDFS分布式文件存储系统

HDFS是Hadoop Didtribute File System的简称，即Hadoop分布式文件存储系统。HDFS的主要设计思想是存储超大文件、一次写入、多次读取、运行在廉价的服务器上。HDFS的基本框架如图2-5所示，其组成部分包括NameNode、DataNode和Cilent三个部分组成。

1. NameNode主要职责是管理整个文件系统的元数据信息，元数据指的是File system namespace、Replication factor、Mapping of blocks to DataNodes。任何客户端或者DataNode的数据迁移、目录操作都是由NameNode来完成的。
2. DataNode主要职责是存储文件块、服务响应Client的文件读写请求、执行文件块的创建、删除和复制。
3. Client提供了面向应用编程语言的一致API，大大简化的各种编程语言编程的复杂性。同时针对读文件的操作提供缓存、针对写操作通过缓冲批量的方式提高访问HDFS的性能。



图 2-5 HDFS架构图

HDFS的优点：

1. 擅长处理超大文件

HDFS擅长处理GB、TB级大小的文件。在目前的实际应用中HDFS已经可以用来管理PB级大小的数据了。

1. 擅长流式的访问数据

HDFS的设计思想是“有一次写入、多次读写”，基于这个有原则，数据一旦写入HDFS中，会被复制多份副本存储在不同的节点上。在通常情况下，任务处理都会涉及到整个数据集，所以使用HDFS读取整个数据集比单独读取一条记录更加高效。

1. 运行在廉价的机器上

HDFS由于对数据容错性很高，所以HDFS对硬件设备的配置要求较低，可以使用廉价的商用机器搭建HDFS分布式文件系统

HDFS的缺点：

1. 存储小文件时效率很低

在HDFS中每一个文件都要占据NameNode节点中大约150字节的内存空间，当存储大量的小文件时，会占据Namenode大量的存储空间，导致存储效率相比存储大文件大大降低。

1. 数据访问延迟较高

HDFS设计的目的是处理大型数据集任务的，为了达到很高的数据吞吐量，就必须牺牲数据访问的延迟，导致数据访问延迟较高。

2.3数据放置技术

2.3.1数据放置策略分类

近年来，随着大数据处理技术的兴起，在处理数量庞大的数据是，如何高效快速的处理成为研究的热点问题，尤其是数据存储和分布式计算协同问题。数据存储和分布式计算协同问题中数据放置策略的研究主要是在分布式处理数量海量数据时，尽量减少数据在不同的节点之间的传输，降低网络传输大量数据消耗的时间。与此同时，尽量保证集群中节点之间负载均衡，提高集群的利用效率。

通常数据放置策略主要可以分为三类：顺序放置策略、随机分配策略、基于P2P网络的放置。下面分别介绍这三类数据放置策略：

1.顺序放置策略

使用顺序放置策略的存储系统，一般将所有的存储结点在逻辑上视为有顺序的，并将所有的数据进行编号，同时将每个数据的副本也进行排序，然后按照一定规则将所有的数据存储在相关的节点上。例如，采用顺序放置策略的分布式虚拟磁盘系统Petal中，它利用典型的菊花链式结构实现了高效的数据容错机制，只有当连续的存储节点出现问题时才会导致数据丢失，同时当单独的节点数据发生丢失时，能够很快恢复该数据。

文献在此基础上定义了一种通用的基本顺序放置策略，假设系统中存在若干个存储结点，且该系统中存储节点的都按照一定的顺序进行编号，同时有若干个带存储的数据块，也按照一定的顺序进行编号。为了保证存储系统的容错性，需要存储数据块的若干个副本。则顺序数据放置策略就是要将某个数据块以及其副本放置按照某种策略放置到该存储系统的各个节点上的过程，具体的放置策略为：

1. 按照数据块的编号和系统中存储节点的个数取模之后，按照模值存放到对应编号的存储节点上。
2. 将该数据块的其他副本依次放置到该数据块放置节点之后的存储节点上。

使用这种数据放置策略能够通过数据块的副本数来控制系统的冗余度，从而实现数据的容错机制。只有当存储系统连续的*m*个节点发生故障时（*m*为数据块的副本数），才会导致数据丢失，从而为系统提供了不错的容错能力。但是如果某个存储节点发生故障，那么与其相邻的*m*个节点不但要恢复该数据，还要迁移数据的源端，导致存储系统中产生大量的网络传输，如果存储系统多个节点发生发生故障，那么存储系统中的网络带宽会被占用殆尽，极大地降低了系统的恢复错误数据的速度，降低了系统的性能。

2.随机放置策略

使用随机放置策略的系统中，在确定数据放置节点时会产生一个随机值，这个随机值就是该数据在系统的存放位置。下面我们来介绍随机放置策略的优缺点。

1. 随机放置策略能够是数据近似均匀的被放置在系统中的存储节点上，这样有利于存储系统的负载均衡，尤其是对于一些经常需要对数据进行随机访问的场景时。但是现实中对数据访问一般都是局部相关的，在这样的场景下，随机数据放置策略，会导致系统的负载不均衡，从到导致系统的性能降低。
2. 单个节点出现故障时的快速恢复能力，当发生单点故障时，存储系统可以很快的从相关的节点上读取数据来恢复丢失的数据。同时相比顺序存储策略，随机存储策略再发生单点故障时不需要进行大量的数据进行迁移，从而不需要占用系统的网络传输资源，大大提高了系统的性能。同时在容错方面相比顺序放置策略，随机放置策略的容错能力较差。

3.基于P2P网络中的数据放置策略

P2P网络中的数据放置策略本质上是一种顺序放置策略，但是在P2P这种复杂的网络环境下进行数据放置策略的研究，需要考虑许多在传统的数据系统中不需要考虑的问题。P2P网络中的存储设备是多种多样的，设备的稳定性、可靠性一般都很差，同时网络中存储界节点的数据很多，对数据的管理和使用更加具有挑战性。而且网络中存储节点流动性较强，可能有节点会频繁的加入存储系统，同时一些节点从存储节点中消失。所以有必要将这种恶劣环境的数据放置策略单独作为一个研究方向。

目前在使用较为广泛的网络存储系统中，简单的使用顺序放置的策略，很难实现系统的稳定性和可靠性。目前通用的一种做法是通过分布式哈希表（DHT）来维护网络中的存储节点。DHT一般只存放数据的元数据信息，主要作用是能够通过DHT锁定相关数据在存储节点上的存储地址。但是网络中的存储设备多种多样，很难保证设备的可靠性，需要实现有效的数据容错机制，以保证系统的可靠性。相比于传统系统的数据放置策略和随机放置策略，P2P网络中的数据放置策略需要维护一个DHT用来索引数据，增加了额外的存储空间，同时降低了数据读写的速度。为了保证系统可靠性，需要复杂度更高的算法来实现系统的容错能力。

2.3.2HDFS分布式文件存储系统中的数据放置策略

HDFS分布式文件系统运行在由跨越大量机架的节点组成的集群上。属于不同机架上的两个节点需要通过网络交换机实现数据传输，显然，位于同一个机架上的存储节点在进行数据传输时相比位于不同机架上的节点会耗费更少的网络的资源，从而提高系统性能。同时相比机架内两个节点同时发生故障的概率远远大于集群中两个机架发生故障的概率，所以为了保证系统容错性，将数据的不同副本放在来两个机架上会降低数据丢失的概率。所以HDFS中每个节点都会维护自己机架的ID，同时向NameNode的节点报告自己的所属机架的信息。

为了提高数据的容错性，同时能够快速读取数据，HDFS数据放置策略的思路将数据的各个副本放置到两个的机架上，这样就避免了机架故障时，存储在该机架上所有数据都会丢失的可能性，同时读取系统中的数据时，还可以利用不同机架的带宽加快数据读取速度。HDFS的具体的数据放置策略如下：

1. 将数据块的第一个副本放置提交该数据块的那个节点上
2. 将数据块的第二个副本随机放置到与放置第一个副本相同机架上的另外一个节点上。这样能够保证在一个数据块丢失是，能够听过本机架内的数据恢复，加快数据恢复的速度。
3. 将数据块的第三个副本随机放置到与前两个放置节点位于不同机架上的节点。这样做是为了提高数据的容错性，避免将所有数据放置到同一个机架上，发生数据丢失的可能性。

HDFS的这种数据放置策略大大减少了机架间的网络传输量，同时充分利用了机架故障的概率远小于节点故障的概率，巧妙的实提高了系统的数据容错性。同时由于随机的放置数据块，在概率上将数据平均分布在系统中，降低了系统负载不均衡的概率，但是这种放置策略也大大增加了系统的冗余性以及在写入数据时的网络传输量。

HDFS的数据放置策略本质上是个随机放置策略，尤其是当数据中心只有一个机架时。这时候数据可靠性能够很难保证，同时HDFS放置数据块时没有考虑到数据存储节点的负载情况，可能导致某个节点的负载很高，某个节点的负载很低。HDFS分布式文件系统虽然实现了均衡器（Balancer）的守护进程，该进程会检测集群中节点的负载情况，将负载较大的存储节点上的数据迁移到负载较低的节点上，来保证系统在发生负载不均衡的情况下，进行的负载平衡策略，但是该数据平衡策略是滞后的，同时在集群内进行大量数据迁移时，会导致大量的网络传输，消耗网络资源。

同时HDFS默认的数据放置策略没有和分布式计算的协同问题，即HDFS的数据放置策略仅仅是面向存储数据考虑的，没有考虑到如何将数据放置策略和分布式计算协同起来。这样可能导致在对任务进行分布式处理时，集群内会产生大量的网络传输，同时节点之间出出现负载不均衡的情况，这一点尤其在处理视频大数据时十分明显。使用基于HDFS分布式计算平台处理大量视频数据时，可能会导致集群内产生大量的网络传输，并且集群负载也会出现不均衡的情况，大大降低了集群的处理效率。

2.4视频浓缩相关技术

本文的主要数据放置策略主要针对监控视频离线处理，为了验证本文提出的数据放置策略的效果，我们使用典型的视频浓缩任务来验证算法的有效性。本节我们主要分析视频浓缩算法所涉及的目标计算检测等技术。

2.4.1目标检测技术

目标检测技术是智能视频监控系统中最为关键技术的之一，也是其他智能视频处理算法，例如目标跟踪、视频浓缩、越界检测等更高层次处理算法的基础。目标检测是为了能够将用户关心的对象从视频数据中提取出来，然后将检测的结果提交给更高层次的视觉算法做进一步的分析处理。目标检测的结果直接影响到后续算法处理的效果。在智能视频监控系统中，我们可以从目标检测的结果中得到很多有价值的信息。目标检测技术在视频浓缩算法实现的基础。我们大体介绍下目标检测算法的实现方式

帧间差分法、背景减除法和光流法是目前运动目标检测常用的三种方法：

1. 帧间差分法

帧间差分法是通过对视频图像序列中连续两帧或三顿之间的残差图像进行腐蚀来检测运动的区域。帧间差法相比背景减除法和光流法的算法复杂度最小，计算速度很快，而且能够检测背景发生变化时的运动目标检测。但是帧间差分法检测出的运动对象的轮廓不够理想，优势甚至不能够准确检测到运动目标的轮廓。

1. 背景减除法

背景减除算法通常需要对视频中若干画面进行学习，从而建立起视频的背景模型，然后通过对比当前帧与背景的模型之间的差值进行运动目标检测。这种方法计算的复杂度很高，同时需要耗费大量的内存。而且背景经常发生变化的视频或者运动对象移动缓慢的场景，这种方法的效果也不是很理想。常用的背景建模算法包括VIBE建模算法、基于高斯混合模型的背景建模等。

1. 光流法

光流法的原理是通过计算光流方程得出每个像素的运动矢量，从而检测出运动对象的大致区域。相比于帧间差分法和背景减除法，光流法能够在背景变化、摄像头移动时也能够检测出运动对象，但是该方法的时间复杂度非常高，没有特殊计算设备的支持很难做到实时检测。同时该方法对光照变化也是十分的敏感，会导致检测的准确性。基于光流法目标检测，检测出来的目标的轮廓很不完整，所以不适用于要求轮廓精确的场景。

以上三种方法，背景减除法是目前使用最为广泛的目标检测技术的实现方式。本文也是使用这种方法实现的活动目标检测的。

2.4.2视频浓缩算法

随着互联网的迅猛发展，产生了大量的多媒体数据。数字视频在多媒体数据中占的比重不断提升，已经成为多媒体数据的主要载体。数字视频虽然使人们的生活以及工作变得丰富多彩，但是却给视频存储、传输、归档和检索带来巨大压力。尤其是近年来随着监控视频的海量增加，视频浓缩技术应运而生。视频浓缩是对通过原有的视频进行分析，检测出其中的运动对象，将这些运动对象重新按照一定的规则进行组合的过程。

本文主要研究对安防监控视频的浓缩，将在时间和空间上存在大量冗余信息的监控视频行浓缩，如图2-6所示。图2-6左边的场景是原始的监控视频，存在大量的冗余信息；右边部分是经过视频浓缩以后，可以看到整个视频帧几乎被活动对象占满。

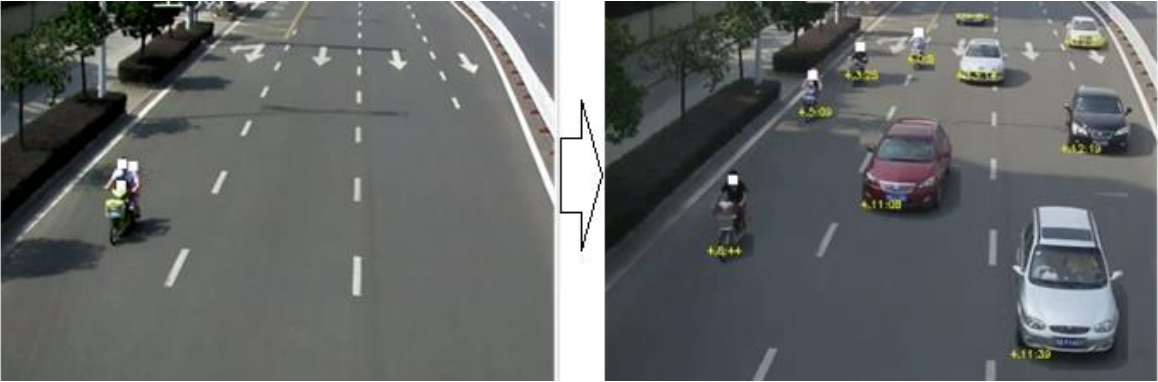


图 2-6视频浓缩前后效果对比图

目前视频浓缩算法一般实现流程分为两个部分，第一步是通过各种背景建模算法建立背景模型，然后通过背景差分算法检测出监控视频中的活动对象（人、车等），并根据目标跟踪算法将同一个活动对象的运动序列保存起来；第二部分主要是根据第一部分生成的活动对象运动序列，按照一定的规则拼接起来，形成浓缩视频。

2.5 本章小结

本章主要分四小节，第一小节主要介绍监控视频云计算平台中常用的分布式计算框：MapReduce分布式框架和Spark分布式框架技术，介绍的内容包括它们的主要工作原理和关键概念。第二小节介绍了分布式文件存储系统HDFS，其组成部分以及其优缺点。第三小节主要介绍了常用的数据放置策略介绍和HDFS中默认的数据放置策略，第四小节介绍了视频浓缩技术。

# 监控视频离线分布式处理系统总体设计

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的需求分析，然后介绍了我们将要使用的监控系统的组成以及各组成部分的作用，接着是监控视频离线分布式处理系统的架构设计及工作流设计，然后是详细说明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

随着近年国际安全形式的不断恶化，特别是恐怖事件在各国引起了强烈的反应，各国纷纷加强了安防措施。视频监控成为主要的安防措施，主要是视频监控能够代替人类持续关注某个区域，并能够如实记录该区域发生的各种事件，而且能够几乎完美复现。随着科技的不断发展，视频监控也从模拟监控向数字监控发展，同时随着互联网的飞速发展，远程监控成为可能。大规模的监控视频管理平台随着涌现，例如中国电信的全球眼智能管理平台，仅仅在上海电信企业监控系统规模，就已经达到了2万路以上。

如此多的监控摄像头每天都会产生海量的监控视频数据，如何避免全人工的从海量的监控视频中获取人们关注的信息成为一大热点研究问题。为了解决这一问题，智能视频处理算法应运而生，例如视频浓缩、越界检测、目标跟踪等等。这些智能视频处理算法能够自动分析监控视频，大大降低了人工查询海量的监控视频信息的难题。但是在摩尔定律破灭后，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计算量巨大的视频大数据处理任务。如何快速地处理这些监控视频数据，成为监控视频离线处理平台的挑战之一。近年来随着分布式计算以及其他相关技术的飞速发展，基于各种分布式计算框架的监控视频云计算平台逐渐成为监控视频离线处理平台的一种解决方案。

监控视频云计算平台将分散的、异构的、单机计算能力较弱的计算资源整合起来，形成强大的集群处理能力来应对监控视频大数据所带来的巨大计算量的问题。监控视频云计算平台的解决方法不仅解决了计算能力不足的问题，而且搭建监控视频云计算平台的硬件成本也十分廉价，是目前应对视频大数据处理问题的主流解决方案。

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电信系统，用于远程捕获多媒体（例如视频，图像和各种报警信号），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保证质量，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），智能视频监控系统的架构由中央管理单元（CMU），业务控制单元（SCU），媒体分配单元（MDU），媒体存储单元 MSU），房屋单元（PU）和客户单元（CU）。

智能视频监控系统的架构如图3-2所示：



图3-2 视频监控系统

中央管理单元CMU主要用于系统集中管理，服务操作管理等。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监控视频离线分布式处理系统架构设计

本节我们主要介绍监控视频离线分布式处理系统架构的总体架构设计以及核心组件的设计。

3.3.1 总体架构设计

如图3-2所示，该系统根据其各自的功能的不同主要划分为两个部分，分别是视频监控系统（VS）和监控视频处理群集（SVPC）。

VS部分的就是我们在3.2提到的视频监控系统。在VS中，MSU由众多的网络视频记录器（NVR）组成，NVR从PU接收数字视频流并将视频存储在其物理存储设备上。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SVPC发送视频处理任务的请求来分布式的处理的监控视频处理任务。

SVPC可以基于通用的分布式计算框架，例如：Spark和MapReduce分布式计算架构，使用HDFS分布式文件系统的。主要提供高效的分布式监控视频处理功能。SVPC由Controller组件和分布式计算集群组成。Controller组件负责从VS接收服务请求，控制视频数据从集群中的MSU加载到HDFS中分布式计算集群处理，初始化Hadoop操作，并将Hadoop生成的处理结果转发到VS。分布式处理集群由安装了Spark或者MapReduce分布式计算框架和HDFS的低成本异构服务器组成，它的职责是分布式的处理智能监控视频处理任务。



图3-2 智能监控视频云计算平台架构

3.3.2监控视频离线分布式处理系统工作流程设计

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在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，通过用户在任务请求页面设置该离线任务处理请求所涉及几个重要参数，例如视频处理任务的类型、该请求所要求的数据在MSU中的存储位置、这些数据之间的相关性等数据，由VS系统分装这些信息后发动给Controller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、每个画面最大同时出现的目标个数等。同时选择处理哪些数据，由VS这些数据在MSU中的存储位置等信息并对这些信息封装成约定的消息格式发送给Controller组件。



图3-3监控视频离线分布式处理系统工作流程图

然后Controller组件接收到来自VS的处理请求消息时，它分析MSU中要处理的视频数据的地址和视频任务类型等信息，同时获取SVPC中可用节点的信息然后启动监控视频任务处理时间模型（CTPM），利用视频数据信息和集群中节点的信息初始化CTPM模型。CTPM初始化以后调用数据块初始放置算法（IDPA）计算数据初始的数据放置位置，并记录这些信息。然后按照计算结果按照数据放置策略将离线视频处理任务从MSU中加载到集群中的HDFS的相关DataNode节点上。

之后启动数据迁移算法（DRA），按照DRA算法的流程监测集群中负载情况，同时按照DRA的负载平衡策略继续从MSU中往HDFS中的DataNode节点中传输数据，供离线处理任务使用。

最后SVPC处理完所有的任务后，SVPC将任务的处理结果发送给Controller组件，Controller组件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核心模块的设计

本节我们主要介绍核心组件Controller和核心数据结构的设计。

3.4.1 Controller组件设计

Controller组件是一个核心组件，主要用来连接VS监控视频系统和SVPC通用计算集群，进行监控视频数据放置的工作。主要职责包括负责接收VS发送的离线视频处理任务就、初始化CTPM时间预测模型、按照数据放置算法分配数据、接收SVPC反馈的实时处理信息、更新CTPM模型、启动数据迁移策略、返回处理结果等等。如图3-4所示：



图3-4Controller组件的职责

1. 接收VS发送的离线视频处理任务

当用户在VS系统，提交离线视频批处理任务后，VS系统将任务和任务所需要的视频数据放置位置、视频属性的参数封装成上文我们提到过的任务集合J和与其对应的DBS数据组合集合，方便Controller进行后期的处理。

1. 初始化CTPM数据

根据从VS系统获取的离线视频批处理任务信息、与任务相对应的数据块信息以及Controller组件本身维护的SVPC中计算节点的信息，按照第四章提到的方法初始化CTPM模型。

1. 初步在SVPC中放置视频数据块

Controller组件初始化CTPM模型后，根据任务所对应的视频数据块的属性信息，调用初始的数据放置算法将任务相关的数据合理的分布在SVPC集群中。

1. 接收SVPC反馈信息

为了避免CTPM模型误差导致的初始数据放置不合理以及SVPC中节点计算能力发生变化时，及时的调用数据迁移策略调整集群中的数据分布，需要利用SVPC反馈的实时信息更新CTPM模型，启动数据迁移策略监控集群中节点的负载情况，及时的通过数据迁移策略，调整集群中节点的负载。接收的信息包括任务的开始时间、结束时间、实际运行时间、预测运行时间等信息。

1. 更新CTPM模型

由于集群中节点的计算能力不是固定不变的，必须通过最近的处理信息，更新CTPM时间预测模型，减少模型的预测误差，提高模型的预测准确度，为后续的数据迁移策略做好准备。

1. 启动数据迁移策略

在任务开始处理以后，根据SVPC反馈的信息不断的更新CTPM时间预测型的参数，然后启动数据迁移策略，寻找集群中负载不均衡的节点，根据最新的CTPM模型重新预测时间，并按照数据策略在集群中进行数据平衡操作，直到所有的任务全部处理完成。

1. 返回结果

所有的任务处理完成以后，Controller组件将结果返回给VS系统，呈现给用户。

3.4.1 核心数据结构设计

在3.3中我们提到了VS系统需要将用户请求、数据块属性等信息传递给Controller组件，以便Controller组件能够使用这些信息初始化CTPM模型，然后利用我们提出的数据块初始放置策略将数据合理的放置在SVPC的分布式文件系统中，之后再根据数据迁移策略合理进行数据迁移操作。

1.为了方便管理任务和其对应的数据块信息以及它们之间的对应关系，我们设计以下数据结构：

由于CTM模型初始化需要VS提供详尽的视频数据块本身的属性，例如视频数据块的分辨率属性、视频数据快的帧率属性、视频数据块的开始和结束属性等等。为此我们设计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 | MSU\_address | SVDB\_r | SVDB\_f | SVDB\_st | SVDB\_et |
| 变量类型 | int | String | int | double | double |

数据结构SVDB使用变量MSU\_address来记录数据块SVDB在MSU中的存储位置，以便Controller组件能够在MSU找到该SVDB。使用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。

为了表示SVDB之间的相关性，我们提出数据块组（Data Block Group）的概念，数据块组即一组相互关联的视频数据块。一个DBG包含多个SVDB，DBG是一个Arraylist<SVDB>的链表结构。

为了方便管理DBG，我们设计有一种DBS的链表用来统一管理和存储DBG，其表述方式为Arraylist<DBG>的List结构。

SVDB、DBG、DBS三者之间的关系如图3-5所示：

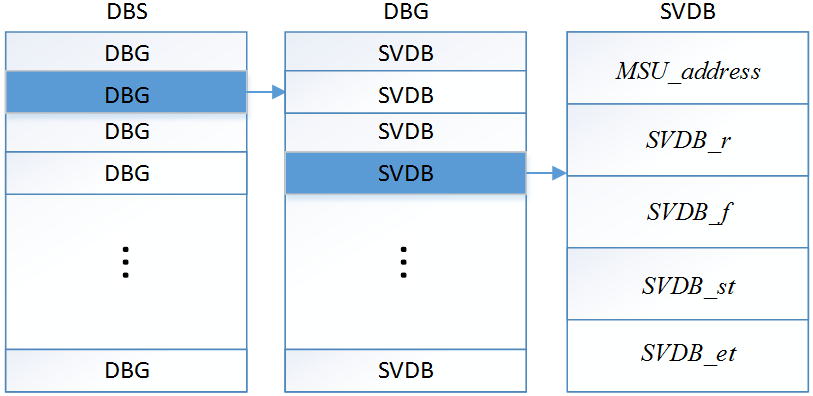


图3-5DBS、DBG、DBS之间的关系

同时我们定义一个描述任务（Job）的结构，包括任务的预测消耗CPU的时间、预测消耗的标准CPU时间时间、任务的开始时间、任务的结束时间等等。

表3-2Job属性的数据结构

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 变量名 | CPU\_t | CPUS\_t | Job\_st | Job\_et | Job\_cput |
| 变量类型 | double | double | double | double | double |

如表3-2所示，我们用CPU\_t表示CTPM模型预测的该任务在某个节点上消耗的CPU处理时间，CPUS\_t表示CTPM模型预测的该任务消耗的标准CPU时间，Job\_st表示该任务在节点上时间开始时间，Job\_et表示该任务在节点上的实际完成时间，Job\_cput表示任务在某个节点上实际消耗的CPU处理时间。

同时为了方便数据放置策略和数据迁移策略对所有的任务进行计算和排序，我们定义任务处理集合J，任务集合中元素由Job组成。

最后我们建立集合J和BDG的集合DBS之间的关系，如图3-6所示，

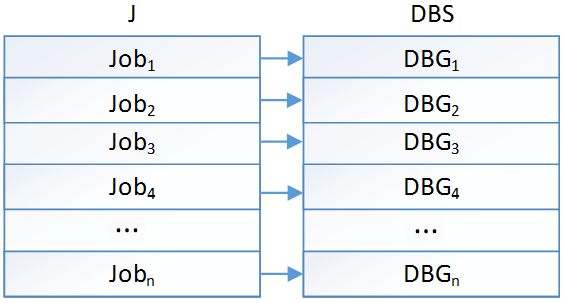


图3-6J、DBS之间的关系

任务集合J中的任务*Jobi*(*i=1,2,…,n*)所需要的数据对应DBS中对应的数据组*DBGi*(*i=1,2,…,n*)。通过这种设计使得任务所需要的数据数据耦合性降低，增加的计算的灵活性。

2.为了方便管理任务和其对应的数据块信息以及它们之间的对应关系，我们设计以下数据结构：

由于SVPC分布式通用计算集群，是混合集群中即集群中的节点是异构的，处理能力个不相同。所以每个节点的都会有独有的DPV集合，用来记录该节点对每种QR（第四章会详细解释其定义）处理能力*dpv*。同时考虑到减少一次计算的误差，我们保留该节点上的每种QR处理能力的最近三次计算数据信息，然后取三次历史数据的信息作为该节点对该QR的*dpv*数据。所以我们定义如表3-3的数据结构QRA：

表3-3数据结构QRA

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 变量名 | QR\_average | QR\_1 | QR\_2 | QR\_3 |
| 变量类型 | double | double | double | double |

数据结构QRA中QR\_average表示三次*dpv*的平均值，QR\_1最近一次的*dpv*计算值，QR\_2表示最近第二次的*dpv*计算值，以此类推。

为了方便管理每种每种QR的QRA，我们设计为每个节点设计一个DPV的数据结构，用来统一管理QRA，数据结构DPV如图3-7所示，DPV实际上一个QRA链表或者数组。

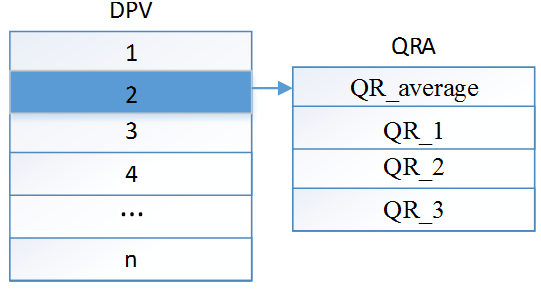


图3-7Controller组件的职责

然后我们设计一个用来维护每个节点信息数据结构NODE\_INFO，该结构主要供Controller用来维护每个节点的信息。NODE\_INFO的结构

表3-4数据结构NODE\_INFO

|  |  |  |  |
| --- | --- | --- | --- |
| 变量名 | DPV\_j | Job\_aassigned | Job\_not\_aassigned |
| 变量类型 | DPV | J | J |

如表3-4所示，变量DPV\_j用来记录该节点处理每种QR的参数信息。Job\_aassigned已经分配给该节点的数据块对应的任务集合，Job\_not\_aassigned为计划尚未分配给节点的数据块对应的任务集合。

最后设计数据结构CLUSTER，用来管理每个节点信息NODE\_INFO。如图3-8所示：



图3-8CLUSTER结构

CLUSTER维护若干个计算的节点的信息NODE\_INFO，相当于实际集群内存中的一个虚拟映射。方便Controller组件管理集群中的节点。

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。

# 云平台中视频数据放置策略的实现

上文已经对智能视频监控云平台的视频离线分布式处理架构进行了详细说明，但是在智能视频监控云平台中，并没有可以直接运用在智能视频监控云平台场景中的高效的数据放置策略，为了能够减少智能视频监控云计算平台中的网络传输，同时能够使集群负载均衡，我们提出了基于视频数据块处理时间预测模型的数据放置策略和数据迁移策略。本章首先分析了HDFS默认的数据放置策略中问题，然后根据视频数据块的特性提出一种视频处理数据块处理时间预测模型，然后基于该模型提出了一种视频数据初始放置算法和一种数据迁移算法。

4.1基于HDFS的监控视频数据放置策略

4.1.1 HDFS默认放置策略

HDFS的数据放置策略本质上是个随机放置策略，尤其是当数据中心只有一个机架时。这时候数据可靠性能够很难保证，同时HDFS放置数据块时没有考虑到数据存储节点的负载情况，可能导致某个节点的负载很高，某个节点的负载很低。HDFS分布式文件系统虽然实现了均衡器（Balancer）的守护进程，该进程会检测集群中节点的负载情况，将负载较大的存储节点上的数据迁移到负载较低的节点上，来保证系统在发生负载不均衡的情况下，进行的负载平衡策略，但是该数据平衡策略是滞后的，同时在集群内进行大量数据迁移时，会导致大量的网络传输，消耗网络资源。

同时HDFS默认的数据放置策略没有和分布式计算的协同问题，即HDFS的数据放置策略仅仅是面向存储数据考虑的，没有考虑到如何将数据放置策略和分布式计算协同起来。这样可能导致在对任务进行分布式处理时，集群内会产生大量的网络传输，同时节点之间出出现负载不均衡的情况，这一点尤其在处理视频大数据时十分明显。使用基于HDFS分布式计算平台处理大量视频数据时，可能会导致集群内产生大量的网络传输，并且集群负载也会出现不均衡的情况，大大降低了集群的处理效率。

同时考虑到我们智能视频监控云计算平台中，即使在放置视频数据块时是均衡的，即每个节点上放置的数据量是差不多的，但是在任务的执行的过程中会产生大量的视频数据移动，导致集群内部产生大量的网络风暴。我们知道传输视频数据的代价很大，大量的网络传输会导致集群内网络堵塞，浪费大量的时间，导致视频处理任务的时间大大增加。另一方面盲目的放置视频数据块，由于Hadoop的默认的任务调度机制是FIFO调度的，会导致智能视频监控云计算平台中节点上任务量相差很大，最终导致集群内节点负载不均，根据“木桶理论”可知会大大增加总体任务完成的时间。

4.1.2解决方案

不论是Spark还是Hadoop分布式计算框架，都需要基于HDFS分布式文件存储系统，作为其文件系统。同时基于HDFS分布式文件系统搭建的Spark或者Hadoop分布式计算集群均支持数据本地化，即在Spark或者Hadoop在分布式计算框架，在给节点分配任务时，会优先将任务分配给存储该任务相关数据的节点，这样可以大大降低从其他节点读取数据浪费的时间。基于此，我们提出一种基于视频数据块时间预测模型的视频数据块放置策略和视频数据块迁移策略，当用户提交智能视频处理任务后，我们根据任务的相关性将监控视频数据块划分成一个个相关数据块组（Data Block Group），我们通过预测处理每一个DBG所需要的处理时间，然后根据每一个节点的所需要处理的所有的DBG的处理时间，作为调度的基本依据，理想状况下保证所有节点同时完成任务。大大提高了集群的利用率，减少了用户等待的时间。

4.2视频数据块处理时间预测模型

为了能够使监控视频云计算集群在执行视频处理任务时能够负载均衡，减少集群内部的网络传输，降低集群内部资源消耗。我们需要建立一个视频数据块处理时间预测模型，然后基于此预测模型，将视频数据块按照最佳的负载均衡策略放置到集群内部的节点上，减少后期再处理过程中，视频数据块迁移的数据量。

为此我们分析了常用的智能视频处理算法，总结了这些算法相同的操作，并进行了大量的实验，最终我们发现了影响智能视频数据处理任务时间的主要因素有四个：视频数据块的时间长度、频数据块的分辨率、视频数据块的帧率以及集群中节点的计算能力。这四个因素决定了监控视频数据块在某个节点上需要的处理时间，下面我们将具体来说明是视频数据块时间预测模型的建立过程。

4.2.1 模型的建立

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，为了能够将方便表述这些不同的属性，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*中分辨率，表示集合中第*m*种分辨率；将不同的帧率定义为集合，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1所示：

(1)

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。

一个监控视频数据块的*QR*属性只属于集合*Q*中的一种，同时我们用*SVDB*来表示监控视频数据块。然后我们提出一个基本长度(Basic Length)的概念，同时定义视频数据块*i*的基本长度为：

(2)

其中表示视频数据块*i*的标准视频质量级别的权重（QR），=，表示视频数据块i的起始时间，表示视频数据块i的终止时间。每一个QR都有一个属于自己的权重，并且初始是这个值是未知的。但是视频数据块*i*的持续时间我们在智能视频监控系统里获取。

一个智能视频处理任务通常包含一个或者多个监控视频数据块（SVDB）。为此，我们提出一个智能视频处理任务作业的基本长度的概念，并用英文字母JBL来表示，并将其定义如下：

(3)

其中*k*表示这个智能视频监控任务所包含的监控视频块（SVDB）的个数。同时，我们定义集群中第*j*个节点的处理该任务的速度为，其计算方式如公式（4）所示：

(4)

其中表示作业基本长度为JBL的任务在云计算集群节点*j*上的执行时间。

考虑到监控视频云计算平台集群中每个节点都有可能在执行其他的任务，可能导致即使处理同一个任务，两次运行的时间也可能大不相同。同时每一个监控视频数据块（SVDB）的QR的权重和每个集群中节点*j*的处理的速度都是未知的，所以我们提出了一个相对速度因子的概念，并用小写字母表示。表示智能视频处理任务所包含的第*i*个任务QR的权重和集群中节点处理该监控视频数据块是的处理速度，并通过公式（5）计算。

(5)

通过公式（2）（3）（4）和（5），我们可以推到出公式(6)：

(6)

为了能够预测监控视频处理任务在监控视频云计算集群中第*j*个节点上的执行时间，我们需要计算出每个QR在监控视频云计算集群中第*j*个节点上的。

。因此我们需要云计算系统能够记录每次任务在每个节点上的执行时间，然后根据云计算系统记录的历史数据，通过公式（7）计算出我们需要的数据

(7)

其中，，。在上面的等式当中，()表示第*i*个任务在集群当中的执行时间，，其中*i=1,2,…,m*且表示第i个视频处理任务中所有QR属性相同的SVDB的起止时间的总和且其QR属性为集合Q中的第k个元素。

当且仅当，非齐次线性线性方程组有唯一解。智能视频云计算系统会记录执行监控视频处理任务的时间，这样我们可以得到一系列的*n*元一次方程，通过求解这些n元一次方程，我们可以得到很多的值，为了保证模型的稳定性，我们使用最近三次的的值平均值作为该属性的值。最后我们建立了如何预测视频任务时间预测模型，如公式（8）所示：

(8)

4.2.2模型的初始化和参数更新

为了方便后面叙述，我们用CTPM（Computing Time Prediction Model）来表示监控视频数据处理任务时间预测模型。现在我们已经建立了CTPM，帮助我们预测一个监控视频处理任务在某个节点上所需要的大概处理时间。我们知道CTPM建立之后很多参数都是未知的，我们提供了一种简单模型初始化方案。在用户第一次使用CTPM模型的时候，提供一个测试用例集合，这个测试用例集合涵盖了本次将要处理的所有任务所包含的SVDB的QR属性。这个测试用例集合里面的元素即单一QR属性的SVDB，通过将这个测试用例集合在监控视频云计算集群内的所有节点上运行，我们便可以初始化CTPM模型所需要的所有参数。

模型建立之后参数不是一成不变的，随着智能监控视频系统反馈的实时处理数据，例如某个监控视频处理所包含的所有SVDB、该任务在哪个节点上被处理的、总共花费了多少时间等，通过这些信息形成上部分提到的n元一次方程，并把这些方程，加入到参数求解队列里面，来计算参数的准确值。这样当监控视频云计算集群中的节点由于各种因数处理速度降低的时候，CTPM模型能够自适应的更新模型参数，减少预测的误差，提高预测的准确性，降低后期数据迁移时的数据量，提高监控视频云计算集群的利用率，减少作业的完成的总时间。

4.3监控视频数据初始放置算法

上节我们根据视频数据的特点，建立了CTPM模型。本节我们提出了一种CTPM模型的监控视频视频数据初始放置算法。为了简述方便我们基于CTPM时间预测模型的视频数据初始放置算法简称为IDPA（Initial Data Placement Algorithm），IDPA数据放置算法主要作用于用户提交的作业所包含的数据从NVR传输到集群中的HDFS系统的过程。

为了简化问题我们假设在IDPA算法作用的集群中当中，假设有*N*个计算节点，并且当中每个节点上存储空间足够大，能够保证分配到该节点上的所有需要处理的数据能够存放的下，不会发生存储空间不够的情况发生。任务队列中有*n*个待处理的视频任务，每个待处理的任务都关联一组SVDB，即每个待处理的视频任务都是由一组SVDB组成的，我们称同属于一组的SVDB是相互关联的。例如，现在用户提交了一个视频浓缩的任务，需要将一星期的监控视频每一天浓缩成一个摘要视频，那么那些将要在同一个节点处理的数据就是相互关联的。IDPA数据放置策略的目的是能够将相互关联的数据尽量放置同一个节点上处理，减少在任务执行的过程中，需要从其他节点获取数据的可能性，同时保证放置在每个节点上的数据相关联的任务量是差不多的，即集群中任务量最大的节点和任务量最小的节点之间的差值是最小的。根据“木桶理论”能够保证完成任务的总时间最少，即负载均衡。

4.3.1数据放置算法

在上文中提到，我们在建立CTPM模型之后，通过运行一个测试用例集合来初始化CTPM模型的参数，所以我们假设现在CTPM模型已经初始化完成，由于每个视频处理任务所包含的SVDB都是不一样的，所以其包含的QR也是不一样的，同时由于在混合集群下，每个计算节点的计算力也是不同的，所以同一个任务在不同的节点上所处理的时间也不一定相同，这点我们在上文中也提到过。虽然每个节点的计算能力有快有慢，但是我们选取一个标准的计算速度，来衡量一批SVDB在标准计算速度下，所需要的标准时间。通过该标准时间来指导我们合理的放置监控视频数据放置。在本文中，为了方便叙述，我们选取所有节点上最小的的值来衡量每个任务所耗费的CPU标准时间(*i=1,2,…,k*)。至此我们工作标准速度的选取，通过CTPM模型计算出了每个任务所要耗费的CPU标准时间。然后根据每个任务所要耗费的CPU标准时间按照从大到小的顺序放在任务队列里。

我们定义一个任务集合，集合中任务按照该任务耗费的标准CPU时间从大大小排序，即用,其中和分别代表任务集合J中任务和,同时满足条件*(i,j=1,2,...,k)*。同时我们定义一个DBS集合，，*(i=1,2,…,k)*代表和*(i=1,2,…,n)*相关联的SVDB的集合。同时我们定义集群中计算节点的集合S，,同时集合中的节点也是按照的值按照从小到大的顺序进行排列的，即如果*i<j,i,j=1,2,…,N*,则,同时也意味着节点*i*的处理能力大于节点*j*的处理能力。也就是的值越小，实际节点的处理能力越强。

4.3.2数据放置算法实现

视频数据块初始放置算法通过多次执行不断迭代获得最终的结果。IDPA每次将N个任务集合中的任务分配到集群中的节点上去。分配的主要原则就是每次将目前任务队列中需要标准处理时间最大的任务分配给集群中目前负载最小的节点，分配完成之后，按照节点上的重新计算该任务在目前节点上的需要耗费的实际时间，加入该节点的负载，重新按照节点上的负载大小从小到大排序。首先，为了简单描述问题，我们假设集群中节点的负载为零，所以每次我们都选择从任务队列选取任务分配给节点集合S中的节点即可。

下面我们将详细介绍具体步骤：

步骤1：我们选取DBS中的前N个数据块组，并将中的视频数据块分配给集群中的节点,其中j=1,2,…,N.

步骤2：任务分配给到节点上之后，按照对应节点上的重新计算该任务在该节点上的实际运行时间，并计入该节点的负载。

步骤3：选择DBS集合中下N个数据块组，即,分别将它们依次分配给按照负载从从小到大排序的节点集合S中的前N个节点。节点的负载定义为分配给该节点的任务在该节点处理需要的总时间。

步骤4：重复步骤2和步骤3直到所有的任务的分配完成为止。

详细的算法步骤，如算法1所示：

**算法1**视频数据块初始放置算法

1. 计算DBS;
2. 计算S;
3. 选取DBS前N个数据块组，记，在DBS中删除;
4. **foreachin do**
5. 分配给节点;
6. 根据CTPM重新计算在节点上的运行时间;
7. .load ;
8. **end for**
9. **while** 集合DBS不为空**do**
10. 从DBS中选取N个数据块组，记，在DBS中删除;
11. **for eachindo**
12. 分配给节点,为S中负载最小的节点;
13. 根据CTPM重新计算在节点上的运行时间;
14. .load ;
15. **end for**
16. **end while**

4.4视频数据块迁移算法

上文我们介绍了视频数据块初始放置算法，通过引入标准CPU计算速度和标准CPU时间，以及相对速度因子成功解决了在混合集群中数据放置的难题，仅仅依靠IPDA算法们虽然能够大大减少集群内部的负载不均衡的状况，减少集群内部节点之间的网络数据传输量，大大提高云计算集群的利用率，但是在任务运行过程中，集群中节点，同时也在接收其他任务或者遇到了其他故障，导致了节点计算能力的降低，根据“木桶效应”相关理论，单个节点完成任务增加，一旦成为最慢的任务，如果不加干涉的话，会导致总体的任务处理时间大大增加，为此在本节我们提出了一种数据迁移策略，为了后文叙述方便我们用字母DRA（Data Block Rebalance Algorithm），来表示本文提出的数据迁移算法，后文出现DRA的地方均表示本文提出的视频数据块迁移算法。

4.4.1视频数据块迁移算法

引入DRA算法还有一个重要的原因就是CTPM本身也是存在一定的误差，会导致基于CTPM模型的IDPA算法在放置视频数块时导致集群内部出现负载不均衡的状况发生。为此我们提出了一种基于CTPM数据迁移算法。

为了解决这个问题，我们采用多阶段作业请求方案，而不是一次提交所有视频处理请求。在我们的系统中，我们首先请求Spark框架根据数据位置运行作业的某些部分。Controller组件监控作业的运行状态，并且可以根据当前系统状态和预测模型CTPM动态调整数据放置。然后，Controller组件请求视频处理作业的另一部分，并重复该处理，直到所有作业都已完成。

初始时我们并不把所有的任务全部分配到云计算集群的节点上，我们根据任务的大小分配一部分，剩下监控视频数据块我们在系统中为每一个节点维护一个虚拟的队列，队列中是即将分配给该节点的监控视频数据，这样可以避免一次性全部分配导致的盲目的数据传输量，我们的原则就是在必要时才使用网络传输资源，降低集群内发生网络风暴的可能性，提高视频云计算集群的利用率。同时我们称位于虚拟节点上任务为待处理任务。

在任务运行过程中，根据监控视频云计算系统反馈的处理信息，CTPM模型会实时更新模型的参数，模型参数更新之后，视频数据块迁移算法会重新计算每个节点上的负载，包括是虚拟节点上即将分配给该节点负载，找出所有节点中负载最大和负载最小的节点，将负载最大的节点上的最小任务所需要的数据移动到负载最小的节点上。移动前必须满足以下前提条件，如果移动前后负载最大的节点和负载最小的节点之间的差值大于阈值的话，则进行移动，否则放弃本次移动。避免出现移动之后负载变化很小的情况的，导致频繁移动数据，降低算法的效率。

4.4.2视频数据块迁移策略实现

假设对应集群中节点的,其中, 是对应的SVDB的数据块集合，即DBS集合中的SVDB数据块组对应任务集合*J*中的任务，即集合中任务的相关的数据块和DBS中SVDB数据块组一一对应。那么DRA算法的具体具体步骤如下：

步骤1：Controller组件分别请求集群中的*N*个节点处理，任务队列中的*x*个任务，*x*为初始设定任务执行数量，同时移除任务队列中的这*x*个任务。

步骤2：Controller组件监控每个任务的运行状况，同时根据监控视频云平台3的反馈的信息实时更新集群每个节点的参数

步骤3：每当集群中节点的的参数发生变化后，都会使用CTPM模型，重新计算即将要分配给每个节点的任务集合，每个节点的任务集合中所有未处理的任务所需要的时间就是这个节点即将到来的负载，记为。

步骤4：假设节点拥有即将到来的最大负载，节点拥有即将到来的最小负载，我们选择节点的任务集合中需要时间最小的任务，记为，同时我们定义公式（9）

(9)

如果我们将任务从节点迁移到节点之后，我们需要重新计算在节点上的所需要的实际时间，记为，之后我们通过公式（10）：

(10)

其中是任务在节点上的运行时间，如果

(11)

其中*thrsh*是给定的阈值，如果调度满足公式（11）的条件，Controller组件则将任务从节点调度到上，同时更新两个节点上负载。

步骤5：重复步骤4，直到没有任务需要进行移动。

步骤6：重复步骤1-5，直到所有的任务执行完毕。

具体实现的伪代码如算法2所示：

**算法2**数据迁移算法

1. **Whille**所有的任务集合不为空**do**
2. 每个节点运行其任务集合中前*x*个任务，然后将这些任务在其任务集合中删除;

3. 更新每个节点参数根据系统反馈的信息;

4. 如果节点更新了参数，则重新计算其任务集合中的任务需要的计算时间;

5. 计算每个节点还未处理的任务需要的计算时间的总和;

6.**repeat**

7.计算集群中负载最大的节点和负载最小的节点的差值*y*根据等式（9）;

8. 将负载最大的节点上的任务计算量最小的任务移动到负载最小的节点上，重新计算负载最小节点上的任务总量和负载最大节点的差值*z*根据等式（10）;

9. **if then**

10. 上的迁移到;

11.11. 更新和;

12. **endif**

13. **until**任意两个节点之间满足等式（11）

14．**endwhile**

4.6本章小结

本章基于智能视频监控云计算平台，首先分析了HDFS默认的数据放置策略的缺陷，同时提出了基于监控视频数据块处理时间预测模型的视频数据块初始放置算法和一种视频数据块迁移算法；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；接着详细介绍了基于CTPM时间预测模型的监控视频数据块初始放置算法，以及伪代码实现；最后介绍了基于CTPM时间预测模型的监控视频数据块迁移算法，以及伪代码实现。

# 系统实现及测试

在第三章监控视频离线分布式处理系统设计以及第四章算法分析的基础上，本章详细介绍了基于Spark分布式计算框架的视频离线处理系统测试系统的实现过程，并对系统功能以及算法效果进行了测试分析。本章首先介绍基于Spark分布式计算框架的视频离线处理系统。之后介绍我们验证算法有效性的视频摘要算法实现，最后对我们提出的数据放置策略通过实验进行了验证。

5.1基于Spark分布式计算框架的视频离线处理系统实现

本系统采用12台物理服务器组成，这些服务器的都配备了Xeon CPU，具体的配置见表格5-1所示，每台服务器上都安装有Ubuntu 14.04.1 LTS(GNU/Linux 3.13.0-32-generic x86 64)操作系统，安装了Spark 1.5.0版本和Hadoop 1.5.0版本，同时安装了HDFS-385和HDFS-2576两个patch保证HDFS支持自动定义的数据放置策略和指定节点存放数据块的功能，同时为了保证Spark将任务分配给存储该任务相关数据的节点，我们将所有节点的主机名添加到spark的配置文件*$SP ARK HOME/conf/slaves*中。

表5-1物理服务器配置

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | CPU | 内存 | 硬盘容量 | 数量 |
| 1 | 16core | 64G | 10TB | 2 |
| 2 | 16core | 32G | 8TB | 2 |
| 3 | 8core | 32G | 4TB | 4 |
| 4 | 8core | 16G | 6TB | 3 |
| 5 | 4core | 16G | 10TB | 1 |

离线视频批处理集群是基于Spark分布式计算架构和HDFS分布式存储系统实现的，如图5-1所示，Spark的计算节点和HDFS的存储节点是对应的。由于Spark也支持数据本地化的概念（Data Locality），和MapReduce的Local Task概念差不多，Spark会根据任务所需要的数据在HDFS系统存储节点的位置，分配给存储该数据的节点去执行该任务。这样我们只需要根据本文提出的视频数据放置策略和迁移策略算法在HDFS放置数据即可。Spark分布式计算框架会根据数据的分布情况自动分配任务，从而间接达到了我们希望集群内节点负载均衡的目的，同时也降低了集群内部节点间的数据网络传输量。



图5-1基于Spark的视频离线处理系统结构

5.1.1 HDFS数据放置过程的实现

默认的HDFS是不支持自己实现数据块放置方法，我们通过使用HDFS-385 API（2013年5月发布的1.2.0版本）实现我们的数据块放置方法。这是一个高级接口，用于希望自定义布局算法以覆盖HDFS默认布置策略的开发人员。这个API使HDFS块布局算法可以实现自定义数据块放置方法。新的块放置策略必须实现BlockPlacementInterface.java中指定的接口。我们BlockPlacementInterface.java实现本文提出的视频数据块初始放置策略。

5.1.2 HDFS数据放置迁移的实现

为了简化数据迁移过程的复杂性，同时减低在数据迁移过程的网络数据传输量，我们在第三章为每个节点设计了待分配任务对应的DBG序列，数据迁移算法根据每个任务运行所需要的实际时间，重新计算并更新CTPM时间预测模型。根据DRA算法调用addBlock（）方法和create（）文件方法中的偏好节点参数，以使得客户端能够向NameNode提供关于该块的每个副本的位置的提示。

addBlock（）方法和create（）是在HDFS-2576 API中，通过这两个方法可以设置数据块的偏好节点参数，以使得客户端能够向NameNode提供关于该块的每个副本的位置的提示。提示NameNode优先将该数据块放置在该数据块的偏好节点。

5.2视频浓缩算法实现

为了测试基于CTPM模型的视频数据块放置策略和视频数据块迁移效果，我们实现了视频浓缩算法。

视频浓缩算法的概念我们已经在相关技术技术中介绍过，本文实现的视频摘要算法分为两个部分，视频浓缩算法的处理流程如图5-2所示：



图 5-2视频浓缩流程图

视频浓缩算法的第一部分主要是通过 Vibe 背景建模算法，利用读入的视频帧建立视频数据块背景模型，然后利用背景差分算法将视频数据块中移动的车和人检测出来，并记录该车或人在监控视频中出现的帧号、顶点坐标、宽高信息，根据目标跟踪算法判断当前检测出来的运动对象分别将属于哪个活动对象的运动序列。最终将监控视频中所有的运动对象都检测出来，形成活动对象的运动序列。如图 5-3所示：

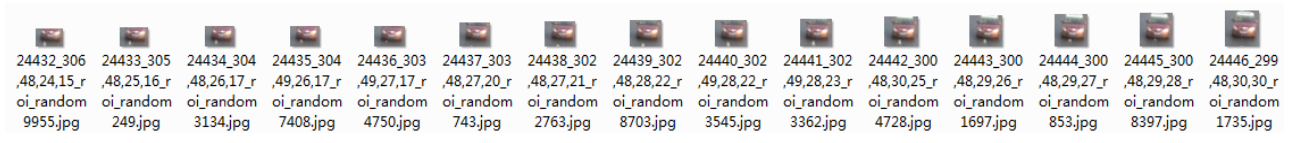


图5-3活动对象的运动序列图

视频浓缩算法的第二部分是根据第一部分生成的活动对象的运动序列，按照一定的规则，例如在拼接的过程中需要保证运动对象的轨迹不能重合、浓缩视频每一帧中最多活动对象的个数等等，生成浓缩视频。

Spark计算框架最初是被Google设计用来处理文本数据的，首先我们需要对Spark的接口进行扩充，方便我们后续进行图片的处理；然后在Map阶段实现背景差分、多目标跟踪生成活动对象的运动序列；为了减少集群内部的网络传输量，在Combine阶段将活动对象的运动序列拼接成子浓缩视频；最后在Reduce阶段将子浓缩视频进行合并输出，MapReduce各个阶段的输入输出数据如图5-4所示：

1. ImageWritable类的实现

首先我们添加ImageWritable类，该类继承Writable类，并覆盖其readFields()以及write()方法，以便能够从将数据流转换成可操作的BufferedImage类型。

1. VideoRecordReader类的实现

添加VideoRecordReader类，该类继承RecordReader类，并覆盖getCurrentKey()、getCurrentValue()、getProgress()、initialize()、nextKeyValue()、close()等方法。该类使用了Xuggle开发库。Xuggle是一个Java开发库，可以几乎实时的解码、处理、编码任何类型的视频文件。该类主要是利用Xuggle从存储在本地节点上的视频数据块中获取解码并获取视频帧，转换成ImageWritable类型。

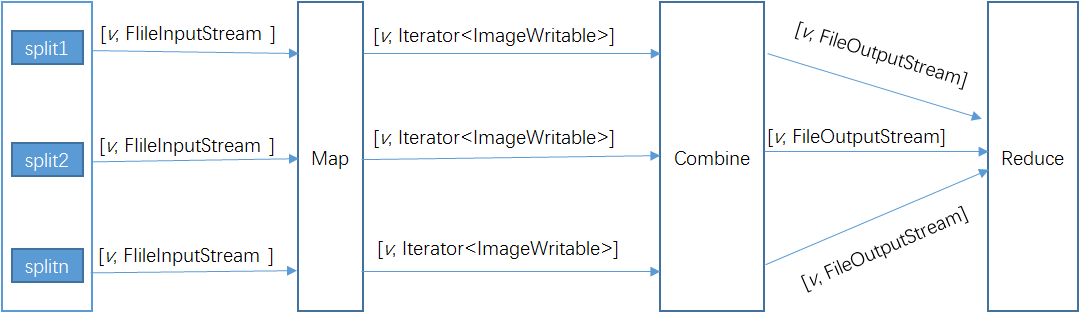


图5-4 基于Spark的视频浓缩处理流程图

1. VideoInputFormat类的实现

添加VideoInputFormat类，该类继承FileInputFormat类，并覆盖createRecordReader()以及getSplits()方法。该类主要是用户从HDFS上获取视频数据块，利用VideoRecordReader类将存储在HDFS上的视频数据块，转换成ImageWritable类型，以供Map阶段使用。由于在由于视频浓缩过程对视频连续性要求较高，并且考虑到单个节点的负载能力，视频数据块分割的大小要合适。兼顾这两方面，我们将大于128M视频数据块，分割成每块128M大小进行实验。由于分割过程中可能会将同一个活动对象分割到两个视频数据块里，为此在分割点附件有300帧的重合部分，保证活动对象的完整性。

1. Map阶段的实现

如图5所示，Map阶段的输入数据为RDD[key,ImageWritable]，其中key是从视频帧的帧号，可以从ImageWritable获取BufferedImage数据，供视频浓缩算法使用。该阶段维护一个Vibe背景模型和一个多目标跟踪器，最终从视频数据块中检测出每个活动对象的活动轨迹以及并将活动对象的运动序列作为输出。

1. Combine阶段的实现

由于Map节点和Reduce节点需要网络传输数据，Map产生的大量小文件，会消耗大量的网络资源。为避免发送大量的小文件，本文实现了Combine过程，在Map节点进行活动对象运动序列的拼接过程生成子浓缩视频，输出RDD[key,FileOutputStream]，大大减轻了网络传输的压力。

1. Reduce阶段的实现

Reduce阶段仅仅对Map节点传输过来的视频数据块进行简单的拼接，然后将拼接好的浓缩视频放入指定的目录当中。

具体实现如算法3 所示：

算法1 视频浓缩算法实现

1. **Input**：*RDD[v, FileInputStream]*
2. **Output**：*RDD[v,**FileOutputStream]*
3. **for all***(v, FileInputStream)* **in***RDD[v, FileInputStream]* **parallel do**
4. **Map stage**:
5. *FFileInputStream;* //使用xuggle
6. *VIBE;F//*使用VIBE背景建模
7. *Iterator<ImageWritable>result RDD[v,**Iterator<ImageWritable>];*
8. **Combine Stage**:
9. *RDD[v,Iterator<ImageWritable>]RDD[v,FileOutputStream];*
10. **Reduce Stage**:
11. *RDD[v, FileOutputStream]RDD[Iterator<v>, FileOutputStream];*
12. **end for**

5.3测试系统功能与算法效果验证

5.3.1测试环境

本实验的测试环境基于5.1介绍的测试系统，实验所用的监控视频数据均来自在中国福州部署的监控视频系统。

5.3.2测试系统功能验证

为了验证基于Spark分布式计算框架的离线视频浓缩算法验证系统的功能，我们准备选取了大约 10GB 的监控视频数据，视频采用 H.264 编码格式，分辨率为1920\*1080， 25fps，监控视频时间总长大约 2200 分钟，本次任务生成了大约 300 分钟的浓缩视频。



图5-5测试系统功能测试图

最终生成的浓缩视频如图 5-5所示，左侧为未浓缩之前的视频画面，存在大量的冗余信息；右侧为浓缩之后的视频画面，可以看出该算法去除了原始视频在时间和空间上的冗余信息，大大提高了有效信息的密集程度，提高了用户查看视频监控的效率。从而验证基于Spark分布式计算框架的离线视频浓缩算法验证系统能够满足我们测试的需要。

5.3.3CTPM准确性验证实验

为了验证CTPM时间预测模型的准确性，我们选取一个监控视频数据集合进行离线视频浓缩任务来对比预测时间和实际运行的时间的误差，通过对比该误差验证CTPM的准确性。该监控视频数据集合包含2700个SVDB数据块，这些视频数据块的帧率都是30FPS，同时这是视频数据块有三种分辨率组成，分别是L(640\*360)、M(1280\*720)和H(1920\*1080)。我们分别用字母L、M、H分别代表3种低、中、高分辨率。每个SVDB的时间长度为668秒和1336秒。按照我们CTPM模型的规定，可以计算出这个监控视频数据集合的视频质量集合*Q*为{L30，M30，H30}。按照CTPM模型里提出的CTPM初始化策略初始CTPM模型。然后我们用CTPM时间预测模型预测这2700个SVDB在相关节点上的需要消耗的时间，并求出每种视频质量相同且时间长度相同类型下的平均数。之后实际运行这批任务，求出每种视频质量相同且时间长度相同类型下的平均数。最后将这两组平均数进行对比可以得到图5-6的结果。

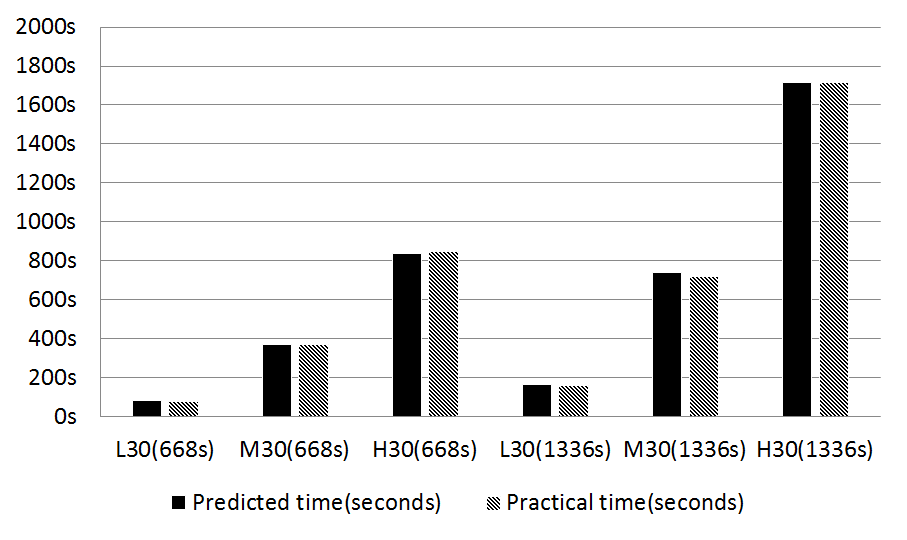


图5-6 CTPM模型的预测结果和实际结果对比

图中纵坐标表示每种视频质量相同且时间长度相同的SVDB的平均执行时间，单位为秒，横坐标表示每种视频质量和时间长度的类型，例如L30（660s）表示视频质量为L，视频帧率为30，视频长度为668秒的SVDB，直方图中蓝色的表示CTPM预测的该类型的SVDB需要的运行时间，黄色的表示该类的SVDB实际的运行时间。从图中我们可以得出CTPM模型在不同类型的数据下和不同的实验配置下能够十分准确的预测不同类型的SVDB需要的运行时间，平均误差大约是2.05%左右。

5.3.4 IPDA初始数据放置算法的效果验证实验

为了验证IPDA初始放置算法的效果，我们选取了4个有代表性的视频数据集合，每个视频数据集合包含不同数量的SVDB，同时包含不同视频质量的类型。如表5-2所示，Dataset 1包含类型为L30的SVDB总数为80，类型为M30的SVDB总数为30，类型为H30的SVDB总数为10；Dataset 2包含类型为L30的SVDB总数为100，类型为M30的SVDB总数为50，类型为H30的SVDB总数为10；Dataset 3包含类型为L30的SVDB总数为100，类型为M30的SVDB总数为70，类型为H30的SVDB总数为20；Dataset 4包含类型为L30的SVDB总数为120，类型为M30的SVDB总数为70，类型为H30的SVDB总数为30。

表5-2视频数据集合

|  |  |  |  |
| --- | --- | --- | --- |
| QR | L30 | M30 | H30 |
| Dataset 1 | 80 | 30 | 10 |
| Dataset 2 | 100 | 50 | 10 |
| Dataset 3 | 100 | 70 | 20 |
| Dataset 4 | 120 | 70 | 30 |

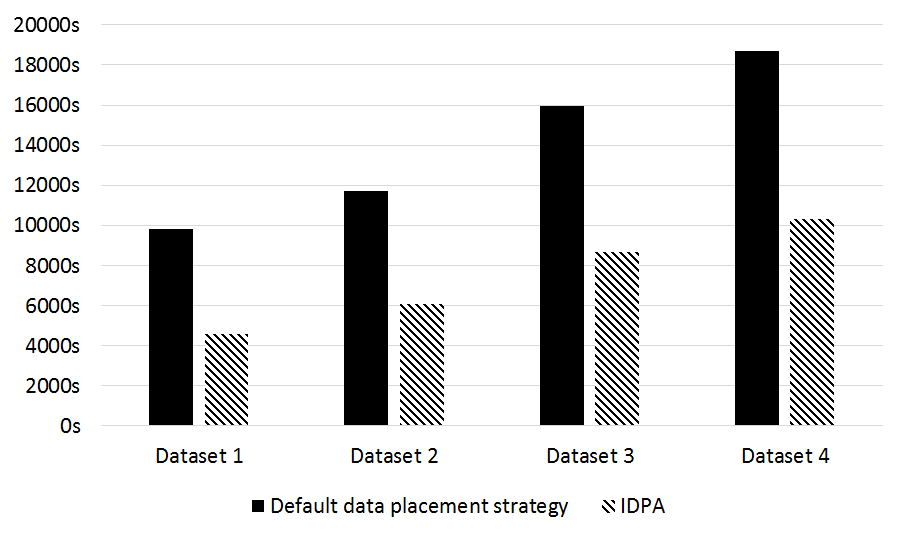


图5-7 IDPA和HDFS默认的数据放置策略运行时间对比

我们同HDFS默认的数据放置策略进行对比实验。我们从NVR中分别按照IPDA数据初始放置算法和HDFS默认的数据放置策略拉取数据并在SVPC中处理，对比着两种数据放置策略对任务总时间的影响。CTPM模型按照第四章提到的初始化方法进行初始化，同时集群中的服务器没有其他的负载任务。为了避免实验的偶然性，我们将该实验在相同的环境下进行了10次，并计算10次平均值分别作为每种数据放置策略的处理结果。实验结果如图5-7所示：

图中横坐标表示每种不同的数据集合分别是Dataset 1、Dataset 2、Dataset 3、Dataset 4，纵坐标是对应数据集合完成10次任务耗费时间的平均值，其中蓝色的部分代表HDFS默认的数据放置策略，黄色的部分代表IDPA数据放置策略。从图中我们可以得出IDPA可以显著的减少任务总的处理时间，相对HDFS默认的数据放置策略，IDPA数据放置算法在四个数据集上分别减少了53.2%、48.3%、45.5%、44.8%的处理时间，平均减少了47.95%的处理时间。

5.3.5 DRA数据迁移算法的效果验证实验

这个是实验我们主要用来验证DRA数据迁移算法的效果，由于实际的集群中每个节点的计算能力不是一成不变的，可能因为故障或者负载的增加导致一些节点的计算能力突然发生变化。由于各种原因导致的集群中原本均衡在负载变得负载不均衡了，为了模拟这种场景，我们通过随机在SVPC中提交一些其他的视频处理任务，随机地使集群中某些节点负载增加。本次实验所使用的数据集和5.3.2的数据集是一样的。我们和默认HDFS数据放置策略以及仅适用IDPA算法和使用IPDA和DRA算法的实验结果进行对比。同样本次实验也是进行多次实验取平均值。

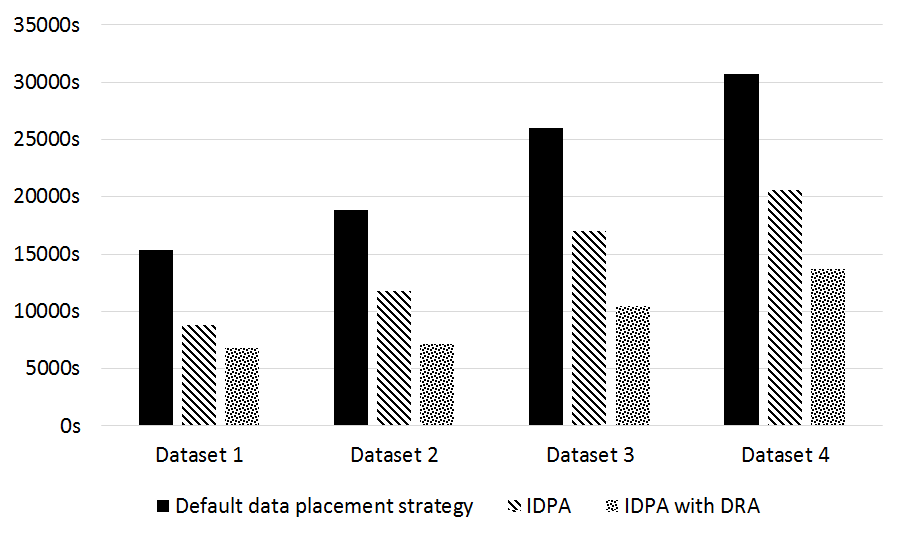


图5-8 HDFS默认策略、IDPA、IDPA with DRA三种策略运行时间对比

实验结果如图5-8所示，图中横坐标表示每种不同的数据集合，分别是Dataset 1、Dataset 2、Dataset 3、Dataset 4，纵坐标是对应数据集合完成多次任务耗费时间的平均值，其中蓝色的部分代表HDFS默认的数据放置策略，黄色的部分代表IDPA数据放置策略，灰色的部分代表同时使用IDPA和DRA策略。从图中我们可以看出在四个测试数据集上相比仅使用IDPA算法分别减少了22.3%、38.7%、38.5%、33.4%的处理时间，平均减少了33.23%的处理时间，相比HDFS默认的数据放置策略平均减少了50%以上的时间，大大减少总任务的处理时间，减少了用户等待的时间。

5.4本章小结

本章首先详细介绍了基于Spark分布式计算框架的视频离线处理系统实现，主要是是对HDFS的自定义数据放置方法的实现，然后介绍了基于Spark分布式计算框架的离线视频浓缩算法的实现，之后详细介绍了测试环境，最后通过实验验证了CTPM时间预测模型的准确性、IDPA数据块初始放置策略和DRA数据迁移策略的效果，证明了基于CTPM模型的IDPA和DRA策略，能够有效平衡SVPC监控视频离线处理集群的负载，减少集群内部网络的传输，相比HDFS默认的数据放置算法大大减少了任务处理的总时间，提高了集群计算资源的利用率，降低了用户等待离线视频处理结果的等待时间。

# 总结与展望

6.1总结

随着社会对安防的重视，监控摄像头遍布城市的各个角落，数量呈爆发式增长。由此产生了海量的监控视频数据，本文对传统的监控视频离线处理系统进行详细的研究。传统的监控视频离线处理系统在分布式处理大规模视频数据时，存在集群内节点负载不均衡的问题。同时集群内节点之间产生大量的网络传输，进一步加剧了集群内负载不均衡的状况，从而导致处理任务时间大大增加，降低了分布式处理集群的效率。本文通过深入分析当前主流分布式视频数据离线计算框架的设计思想以及视频处理过程，提出基于CTPM视频数据块时间预测模型的IDPA初始数据放置算法以及DRA数据迁移算法来解决分布式视频离线处理集群负载不均衡等问题，相比HDFS默认的数据放置算法，大大减少了任务总耗时，提高了集群的负载均衡状况，大大降低了集群内部的网络传输量。之后，本文设计了基于Spark分布式计算框架的离线视频浓缩算法对CTPM、IDPA、DRA算法进行了效果验证。本文主要工作包括以下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混合集群下的监控视频离线处理任务时间预测模型，该预测模型根据视频数据块的本身的一些特征，主要是监控视频数据块的分辨率、监控视频数据块的帧率、监控视频数据块的时间长度、以及混合集群中节点计算能力四个特征，通过自适应的调整参数，建立时间预测模型CTPM来预测监控视频离线处理任务在不用计算节点上所需要的处理时间。

1. 基于CTPM时间预测模型的视频数据块初始放置算法

本文在研究并实现混合集群下监控视频离线处理任务时间预测模型的基础上，提出并实现了基于CTPM时间预测模型的初始视频数据块初始放置算法。该算法将同属于一个视频处理任务的视频数据块所组成的集合DBG作为基本的放置单位，通过CTPM时间预测模型，预测每个DBG对应任务的在标准CPU上的标准CPU消耗时间，按照该消耗时间，从大到小进行排序。然后依次将所对应DBG包含的SVDB分配给集群中负载最小的节点，指导所有的任务的分配完成为止。

1. 基于CTPM时间预测模型的视频数据块迁移算法

本文提出并实现了基于CTPM时间预测模型的视频数据块迁移算法（DRA）。由于集群内计算节点的计算能力可能由于该节点上的负载情况不断的变化，需要对某些计算力突然变化的节点，进行数据迁移，保证集群负载均衡，提高集群的利用率。DRA算法工作计算集群中每个节点负载，找到集群中负载最大和负载最小的节点，并计算二者之间的差值的绝对者，然后假设将集群中负载最大的节点上CPU消耗时间最少的任务迁移到集群中负载最小的节点上之后，重新计算两个节点的负载，并再次计算二者的差值。如果两次差值的差值的绝对值大于预设的阈值，就进行数据迁移操作，否则放弃，然后继续寻找符合要求的节点，直到所有任务运行结束。

1. 基于Spark分布式计算框架的离线视频浓缩算法验证系统

基于以上提出的数据放置算法和迁移算法，实现了基于Spark分布式计算框架的离线视频浓缩算法，使用车辆监控视频作为数据源，对IDPA和DRA算法进行了验证。该系统使用HDFS作为分布式存储系统，通过使用HDFS-385和HDFS-2576的Patch实现了使用IDPA和DRA算法，Spark作为分布式计算框架，当用户提交视频摘要任务后，会按照基于CTPM时间预测模型的IDPA数据放置策略和DRA迁移策略进行数据的放置和移动，由于SPARK的数据本地化支持，优先将任务分配给存储该任务的节点，大大降低了集群内节点之间的网络传输，是集群负载均衡，大大减少任务处理的总时间。

6.2展望

本文提出的数据放置算法可以优化的方向包括但不限于以下几点：

1. CTPM时间预测模型的优化：

目前CTPM模型仅仅考虑了视频数据块分辨率，视频数据块的帧率、视频数据块的时间长度、混合集群中节点的运算能力等四个因素的影响，未来靠考虑加入任务类型、视频数块所处时间段等因素对视频数块处理时间的影响，进一步增强CTPM时间预测模型的鲁棒性。

1. IDPA数据初始放算法的优化：

目前IDPA仅仅在一种对节点存储空间要求较高的要求下实现的，未来可以考虑在节点存储空间有限的情况的下的放置研究，可以放置在距离该节点网络传输代价最小的节点的放置等解决方案，进一步增强IDPA算法的鲁棒性。

目前随着视频大数据时代的到来，分布式处理视频数据成为一种趋势。如何高效的利用集群成为研究热点，分布式处理平台下数据放置策略的能够很好的解决的分布式集群负载均衡的问题和集群内部大量网络传输的问题，成为提高集群利用率的一种重要解决方案。

# 参考文献

1. 巴丽娟. 云存储虚拟化技术在视频监控中的应用[J]. 中国安防, 2013(8):59-62.
2. 黄铁军, 郑锦, 李波,等. 多媒体技术研究:2013——面向智能视频监控的视觉感知与处理[J]. 中国图象图形学报, 2014, 19(11):1539-1562.
3. Mahapatra A, Sa P K, Majhi B. A MULTI-VIEW VIDEO SYNOPSIS FRAMEWORK[C]// IEEE International Conference on Image Processing. IEEE, 2015:31-44.
4. Xiaomeng Zhao, Huadong Ma, Haitao Zhang, Yi Tang, Yue Kou, “HVPI: Extending Hadoop to Support Video Analytic Applications”, IEEE 8th International Conference on Cloud Computing (IEEE Cloud 2015), pp. 789 - 796.
5. Haitao Zhang, Jin Yan, Yue Kou, “Efficient Online Surveillance Video Processing based on Spark Framework”, International Conference on Big Data Computing and Communications (Bigcom 2016), 2016.
6. 周艳慧, 张凯. 新的分布式任务调度算法[J]. 计算机系统应用, 2008, 17(10):40-42.
7. Agrawal D, Abbadi A E, Antony S, et al. Data Management Challenges in Cloud Computing Infrastructures[C]// Databases in Networked Information Systems, International Workshop, Dnis 2010, Aizu-Wakamatsu, Japan, March 29-31, 2010. Proceedings. 2010:1-10.
8. Wei Q, Veeravalli B, Gong B, et al. CDRM: A Cost-Effective Dynamic Replication Management Scheme for Cloud Storage Cluster.[C]// IEEE International Conference on CLUSTER Computing. IEEE, 2010:188-196.
9. 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.
10. 刘少伟, 孔令梅, 任开军,等. 云环境下优化科学工作流执行性能的两阶段数据放置与任务调度策略[J]. 计算机学报, 2011, 34(11):2121-2130.
11. Poonthottam V P, Madhu Kumar S D. A Dynamic Data Placement Scheme for Hadoop Using Real-time Access Patterns[C]// International Conference on Advances in Computing, Communications and Informatics. 2013:225-229.
12. Xie J, Yin S, Ruan X, et al. Improving MapReduce performance through data placement in heterogeneous Hadoop clusters[J]. IEEE International Symposium on Parallel & Distributed Processing Workshops &Phd Forum, 2010:1 - 9.
13. Ubarhande V, Popescu A M, Gonz&#, et al. Novel Data-Distribution Technique for Hadoop in Heterogeneous Cloud Environments[C]// International Conference on Complex. IEEE, 2015:217-224.
14. He Y, Lee R, Huai Y, et al. RCFile: A fast and space-efficient data placement structure in MapReduce-based warehouse systems[C]// IEEE, International Conference on Data Engineering. IEEE Computer Society, 2011:1199-1208.
15. Xiong R, Luo J, Dong F. SLDP: A Novel Data Placement Strategy for Large-Scale Heterogeneous Hadoop Cluster[C]// International Conference on Advanced Cloud & Big Data. IEEE, 2014:9-17.
16. Dean J, Ghemawat S. MapReduce: Simplified Data Processing on Large Clusters[J]. In Proceedings ofOperating Systems Design and Implementation, 2004, 51(1):107-113.
17. 宋杰,刘雪冰,朱志良等. 一种能效优化的 MapReduce 资源比模型[J]. 计算机学报,2015,38(1):59-73.
18. 李成华，张新访，金海等. MapReduce：新型的分布式并行计算编程模型[J]. 计算机工程与科学,2011,190 33(3):129-135.
19. 许彬, 张海涛“一种高效地视频浓缩分布式处理实现方法”中国科技论文在线, 2016
20. Yang S, Wu B. Large Scale Video Data Analysis Based on Spark[C]// International Conference on Cloud Computing and Big Data. IEEE, 2015:209-212.
21. Zaharia M. An Architecture for Fast and General Data Processing on Large Clusters[M]. Association for Computing Machinery and Morgan & Claypool, 2013.
22. 高彦杰．Spark大数据处理:技术应用与性能优化[M]．北京:机械工业出版社,2014:25-30
23. 陈惟康,杜松. 分布式存储中数据放置策略的研究[J]. 计算机应用与软件, 2009, 26(1):6-8.
24. Fagin R,Lotem A,Naor M．Optimal Aggregation Algorithms for Middleware[C]．In Proc．PODS,2001．
25. Zhipeng Gao, Yinghao Qin, Kun Niu. An effective merge strategy based hierarchy for improving small file problem on HDFS[C]// 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS). 2016.
26. 袁国武. 智能视频监控中的运动目标检测和跟踪算法研究[D]. 云南大学, 2012.
27. Zhu F, Zhao C, Cheng J. ECLC: Edge character and latency connection enhanced inter-frame difference algorithm for real-time outdoor surveillance[C]// Sixth International Conference on Distributed Smart Cameras. 2012:1-6.
28. Barnich O, Van D M. ViBe: a universal background subtraction algorithm for video sequences[J]. IEEETransactions on Image Processing A Publication of the IEEE Signal Processing Society, 2011, 20(6):1709-24.
29. T. Bouwmans, F. El Baf, and B. Vachon, “Statistical backgroundmodeling for foreground detection: A survey,” in Handbook of PatternRecognition and Computer Vision (Volume 4). Singapore: WorldScientific, Jan. 2010, ch. 3, pp. 181–199.
30. S. Elhabian, K. El-Sayed, and S. Ahmed, “Moving object detection inspatial domain using background removal techniques—State-of-art,”Recent Pat. Comput. Sci., vol. 1, pp. 32–54, Jan. 2008.
31. Xu J, Li J. Moving Target Tracking Algorithm Based on Scale Invariant Optical Flow Method[C]// International Conference on Information Science and Control Engineering. IEEE Computer Society, 2016:468-472.
32. Wen-Ping J I, Guo B L, Ding G G. Detecting Traffic Volume Statistics Based on Virtual Coil with Optical Flow Method[J]. Computer Simulation, 2004, 21(1):109-117.
33. 黄凯奇,陈晓棠,康运峰等. 智能视频监控技术[J]. 计算机学报,2015,38(6): 1093-1118.
34. 王娟,蒋兴浩,孙锬锋. 视频摘要技术综述[J]. 中国图象图形学报,2014, 19(12): 1685-1695.
35. 宋飞斌. 电信全球眼业务智能化部署方案探讨[J]. 电信技术, 2015, 1(7):75-80.
36. ITU-T H.626 standard, “Architectural requirements for visualof surveillance,” 2011.
37. Haitao Zhang, Bin Xu, Jin Yan, Lujie Liu, Huadong Ma. “Proactive Data Placement for Surveillance Video Processing in Heterogeneous Cluster,”2016 IEEE 8th IEEE International Conference on Cloud Computing Technology and Science, IEEE, 2016
38. Design a pluggable interface to place replicas of blocks in HDFS,https://issues.apache.org/jira/browse/HDFS-385,2016
39. Namenode should have a favored nodes hint to enable clients to have control over block placement,https://issues.apache.org/jira/browse/HDFS-2576,2016

# 攻读硕士学位期间的主要研究成果

1. 许彬, 张海涛“一种高效地视频浓缩分布式处理实现方法”中国科技论文在线, 2016
2. Haitao Zhang, Bin Xu, Jin Yan, Lujie Liu, Huadong Ma. “Proactive Data Placement for Surveillance Video Processing in Heterogeneous Cluster,”2016 IEEE 8th IEEE International Conference on Cloud Computing Technology and Science, IEEE, 2016
3. 许彬, 张海涛, 胡豆豆”云计算平台中监控视频摘要任务调度方法研究” 计算机应用与软件，录用待发表
4. Yi Tang, Haitao Zhang, Bin Xu, “Metadata Organization and Retrieval withAttribute Tree for Large-Scale TrafficSurveillance Videos”, The 1st International Conference on Big Data Computing and Communications (Bigcom 2015), 2015

# 致谢

时光匆匆，转眼间两年半的研究生生活即将结束，回想拿着研究生录取通知书，第一次走进北京邮电大学的校园，徜徉在学校主干道上时的喜悦，仿佛就像昨天。借此论文完成之际，向帮助和关心过我的老师、同学和家人表示由衷的感谢。

首先感谢我的学校北京邮电大学，为我提供了良好的学习环境和科研氛围。在这离别之际，尽管依依不舍，却很珍惜，因为这是在我的生命中最后的一段校园时光。

感谢我的导师李文生老师。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。正是在张老师的悉心指导和帮助下，论文的研究和撰写顺利完成。感谢张老师，让我在研究生期间有如此多机会学习和掌握更多的新理论、新知识，能力得到了极大的提高。

感谢实验室师兄和师姐们对我的指导、鼓励和帮助。感谢付广平、唐毅、赵梦琪、姜哲、骆亮亮等同学的帮助。

感谢父母以及家人一直以来对我的帮助，感谢你们竭尽所能地为我提供良好的学习环境，给予了我健康的体魄，诚朴正直的品质，这些都是我受益终身的财富。