**密级： 保密期限：**



**硕士学位论文**



**题目： 异构视频监控云计算平台中任务调度**

**方法的研究与应用**

**学 号： 2016140476**

**姓 名： 唐炳昌**

**专 业： 计算机技术**

**导 师： 张海涛**

**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：

异构视频监控云计算平台中任务调度方法研究与应用

**摘 要**

随着智能视频监控系统的发展，越来越多的视频监控应用迁移到云平台之上。同时，为了加速视频任务的处理，降低任务的处理时间，许多视频处理任务都开始采用GPU来加速运算。当前主流的基于虚拟机和CPU的视频监控云平台存在资源利用率低、任务处理时间长、平台吞吐量小的问题。此外，由于虚拟机存在启动时间长、不可编程等问题，造成平台的运维困难，影响服务质量。

本文研究基于容器技术的异构视频监控云平台。首先，本文设计并实现了基于Docker和Kubernetes技术的异构视频监控云平台。利用Docker轻量级的特点，降低了平台的额外消耗，提高了平台的资源利用率。利用Kubernetes提供的计算、存储资源管理的能力，有效的管理异构平台的资源。为了方便与视频监控系统对接和方便用户访问，本文采用Golang语言实现了平台中心管理单元模块，该模块在提供可视化操作界面的同时，对外暴露RESTful风格的接口。其次，为了进一步提高平台的资源利用率，充分发挥异构平台的性能，本文提出了基于深度强化学习的两阶段任务调度器CS-NS。该调度器利用深度强化学习在连续决策问题中所表现出来的优势，结合异构视频监控云平台中任务特点、数据特点和资源特点。实时地学习最优的调度策略，选择合适的工作节点和计算设备。最后，实验表明本文设计的异构视频监控云平台能够显著地提高平台的资源利用率，增加平台的吞吐量。

**关键字:** 容器技术、Kubernetes、异构视频监控云、强化学习、任务调度

**Research and Application of Task Scheduling Method in Heterogeneous Video Surveillance Cloud Computing Platform**

ABSTRACT

With the development of intelligent video surveillance systems, more and more video surveillance applications have migrated to the cloud platform. At the same time, in order to speed up the processing of video tasks and reduce the processing time of tasks, many video processing tasks begin to use GPU to accelerate computing. The current mainstream video surveillance cloud platform based on virtual machine and CPU isomorphism has the problems of low resource utilization, long task processing time, and small platform throughput. In addition, due to the long startup time and non-programmability of the virtual machine, the operation and maintenance of the platform is difficult, which affects the service quality.

This paper studies heterogeneous video surveillance cloud platform based on container technology. First of all, this paper designs and implements a heterogeneous video surveillance cloud platform based on Docker and Kubernetes technology. Utilizing the lightweight features of Docker, the additional consumption of the platform is reduced, and the resource utilization of the platform is improved. Utilize the computing and storage resource management capabilities provided by Kubernetes to effectively manage the resources of heterogeneous platforms. In order to facilitate the docking of the video surveillance system and user-friendly access, this paper uses the Golang language cluster management unit module, which exposes the RESTful-style interface while providing a visual operation interface. In addition, in order to further improve the resource utilization of the platform and give full play to the performance of heterogeneous platforms, this paper proposes a two-stage task scheduler CS-NS based on deep reinforcement learning. The scheduler utilizes the advantages of deep reinforcement learning in continuous decision-making problems, combines the task characteristics, data characteristics and resource characteristics of heterogeneous video surveillance cloud platform, learns the optimal scheduling strategy in real time, and selects the appropriate working node and computing device. Experiments show that the heterogeneous video surveillance cloud platform designed in this paper can significantly improve the resource utilization of the platform and increase the throughput of the platform.

**KEY WORDS:** container technology, Kubernetes, heterogeneous video surveillance cloud, reinforcement learning, task scheduling

目 录

[**第一章 绪 论** 1](#_Toc2157182)

[**1.1 研究的背景和意义** 1](#_Toc2157183)

[**1.2 国内外研究现状** 2](#_Toc2157184)

[**1.3 论文的主要研究内容** 4](#_Toc2157185)

[**1.4 论文的组织结构** 5](#_Toc2157186)

[**第二章 相关技术** 6](#_Toc2157187)

[**2.1 Docker相关技术简介** 6](#_Toc2157188)

[**2.1.1 Docker简介** 6](#_Toc2157189)

[**2.1.2 Docker与虚拟机对比** 8](#_Toc2157190)

[**2.2 Kubernetes大规模容器管理技术** 9](#_Toc2157191)

[**2.2.1 Kubernetes基本概念** 9](#_Toc2157192)

[**2.2.2 kubernetes 架构** 11](#_Toc2157193)

[**2.2.3 Kubernetes优势** 12](#_Toc2157194)

[**2.3 深度强化学习相关技术** 13](#_Toc2157195)

[**2.3.1 强化学习简介** 13](#_Toc2157196)

[**2.3.2 Deep Q-Network** 13](#_Toc2157197)

[**2.4 本章小结** 14](#_Toc2157198)

[**第三章 异构视频监控云平台设计与实现** 15](#_Toc2157199)

[**3.1 视频监控系统** 15](#_Toc2157200)

[**3.2 异构视频监控云平台总体架构** 16](#_Toc2157201)

[**3.3 异构视频监控云平台工作流程** 17](#_Toc2157202)

[**3.3.1 视频服务的创建流程** 18](#_Toc2157203)

[**3.3.2 平台服务的访问流程** 19](#_Toc2157204)

[**3.3 分布式资源管理单元** 20](#_Toc2157205)

[**3.4 镜像仓库** 23](#_Toc2157206)

[**3.5 集群管理单元** 24](#_Toc2157207)

[**3.5.1 集群管理单元的总体功能** 25](#_Toc2157208)

[**3.5.2 RESTful接口设计** 26](#_Toc2157209)

[**3.5.3 集群管理单元的开发** 27](#_Toc2157210)

[**3.6 服务访问网关** 28](#_Toc2157211)

[**3.8 平台优化** 30](#_Toc2157212)

[**3.8.1 平台稳定性优化** 30](#_Toc2157213)

[**3.8.2 平台便捷性优化** 31](#_Toc2157214)

[**3.9 本章小结** 31](#_Toc2157215)

[**第四章 异构视频监控云平台中任务调度策略** 32](#_Toc2157216)

[**4.1 异构视频监控云平台上的视频处理任务** 32](#_Toc2157217)

[**4.2 异构视频监控云平台调度问题描述** 32](#_Toc2157218)

[**4.3 现有的集群调度方法** 33](#_Toc2157219)

[**4.3.1 传统的集群调度算法概述** 33](#_Toc2157220)

[**4.3.2 传统集群调度算法缺陷** 33](#_Toc2157221)

[**4.4 基于深度强化学习的调度策略** 34](#_Toc2157222)

[**4.4.1 两级任务调度器的设计** 34](#_Toc2157223)

[**4.4.2 集群级别调度** 35](#_Toc2157224)

[**4.4.3 节点级别调度** 36](#_Toc2157225)

[**4.5 调度算法实现** 36](#_Toc2157226)

[**4.5.1 算法描述** 36](#_Toc2157227)

[**4.5.2 深度神经网络模型描述** 37](#_Toc2157228)

[**4.5.3 调度算法中使用的技巧** 38](#_Toc2157229)

[**4.6 两级调度器的应用** 38](#_Toc2157230)

[**4.7 本章小结** 39](#_Toc2157231)

[**第五章 平台功能及性能测试** 40](#_Toc2157232)

[**5.1 平台基础环境** 40](#_Toc2157233)

[**5.1.1 开发环境** 40](#_Toc2157234)

[**5.1.2 集群环境** 40](#_Toc2157235)

[**5.1.3 视频监控数据** 41](#_Toc2157236)

[**5.1.4 视频处理子任务实现** 41](#_Toc2157237)

[**5.2 异构视频监控云平台的功能测试** 42](#_Toc2157238)

[**5.3 异构视频监控云平台的调度器性能测试** 45](#_Toc2157239)

[**5.3.1 模型参数重要参数的设置** 45](#_Toc2157240)

[**5.3.2 模型的收敛性** 45](#_Toc2157241)

[**5.3.3 与其它方法的对比** 46](#_Toc2157242)

[**5.4 本章小结** 50](#_Toc2157243)

[**第六章 总结与展望** 51](#_Toc2157244)

[**6.1 总结** 51](#_Toc2157245)

[**6.2 未来工作展望** 52](#_Toc2157246)

[**参考文献** 54](#_Toc2157247)

[**攻读硕士学位期间的主要研究成果** 58](#_Toc2157248)

[**致谢** 59](#_Toc2157249)

# **第一章 绪 论**

## **1.1 研究的背景和意义**

近年来，随着全球安全意识的提升和智慧城市的发展，视频监控在全球范围内广泛的部署。传统的基于人工的视频监控方式无法满足监控数据分析的实际需求。许多智能视频处理技术应运而生，如视频摘要、目标跟踪、入侵检测等。随着监控设备的广泛部署、视频监控数据的快速增长，基于单机部署的视频监控数据处理平台已经无法满足当前海量视频数据处理的需求。如何快速高效的处理视频监控数据成为视频监控领域新的挑战。随着分布式计算和其它相关技术的发展，云计算成为该领域新的选择，越来越多的监控视频处理应用被转移到云端。然而，基于CPU同构条件下的视频监控云平台并不善长处理计算量巨大的监控数据分析任务。由于GPU支持大量的并行计算、具有更高的访存速度和更高的浮点运算能力，基于CPU-GPU异构环境的视频数据分析平台迅速成为研究热点[1][2][3]。

异构视频监控云平台通过把分散的、异构的资源整合起来统一集中管理，具有超大规模、廉价、高可用、按需服务等特点。将视频监控系统部署于配置有CPU和GPU的异构计算设备的物理集群，综合利用CPU强大的逻辑分析能力以及GPU高带宽、多线程的并行计算能力可进一步实现大规模智能视频分析任务的高效处理。然而，面对数据的多样性（图片、视频）[4]、资源异构性（CPU、GPU）[5]、视频服务复杂多变性（目标检测、视频浓缩、车牌识别）[6]，视频监控云平台仍然面临如下问题。

问题一，如何高效的管理平台资源。资源管理整合平台中的计算存储资源统一管理，是所有云计算平台首要解决的问题[7]。异构视频监控云平台也不例外。当前主流的云计算平台采用虚拟机的形式提供资源服务[5][8][9][10]。然而，虚拟机本身占用的资源多，资源的分配粒度大，容易造成平台资源的浪费[11][12]。因此，研究使用新的虚拟化技术构建异构视频监控云平台十分必要。

问题二，如何与传统的视频监控系统对接。传统的视频监控服务是一个关注视频或音频应用技术的电信服务[13]，主要被用来捕获远端的多媒体并呈现给终端用户。异构视频监控云平台需要与传统的视频监控系统对接，接受传统视频监控系统的请求，获取并处理视频监控数据。

问题三，如何设计高效的任务调度器，充分发挥异构平台的优势。调度器依据调度算法，在云计算平台中为任务分配合适的计算和存储资源，以提高整个平台的资源利用率。现有的许多分布式系统上的任务调度方法大多是针对CPU同构环境下单的任务调度，或是针对CPU-GPU异构单接点上的任务调度[14][15][16][17]，或者是没有考虑到视频数据和视频处理任务的特点[18][19][20][21][22]。这些方法无法充分发挥异构视频监控云平台任务处理能力，无法有效的提高平台的资源利用率。异构视频监控云平台中的任务调度更是受多方面因素的影响：视频处理任务具有多样性，有的是计算密集型，有的是IO密集性；视频处理数据也具有多样性，视频数据的帧率、码率、数据的大小等都各有不同；计算和实时存储设备的异构性，包括CPU、GPU、内存和显存等。如何设计高效的任务调度器来提高异构视频监控云平台上的资源利用率，增加平台的吞吐量需要进一步研究。

综上，设计高效的异构视频监控云平台还需要很多工作要做。近年来，Docker[23]容器技术成为云计算领域的热门话题，许多容器平台管理技术应运而生。Kubernetes[24][25]更是为容器管理提供丰富的接口和监控运维解决方案，同时还支持CPU-GPU异构资源的管理。当前，还没有利用这些新技术搭建异构视频监控系统的案例。随着深度学习[26]和强化学习的发展，结合了二者的深度强化学习在连续决策问题中也表现出了极大的优势[27][28]。许多学者也开始研究利用强化学习解决任务调度和资源管理问题[20][29][30][31][32]。不过针对视频监控领域，很少有人研究使用强化学习来解决平台调度问题。本文在研究以上新技术以及针对目前异构视频监控云平台存在的问题，提出基于Kubernetes构建异构视频监控云平台，并设计了基于深度强化学习的任务调度器。此外，为了与传统的视频监控对接，本文实现了异构视频监控云平台管理系统，以RESTful[33]接口的形式对外提供服务。

## **1.2 国内外研究现状**

当前，任务调度方法吸引了工业届和学术界的广泛研究，国际上涌现出了大量相关工作。文献[18]针对大规模显微镜图像数据集，将图像处理任务分为粗粒度与细粒度两种操作，设计分级处理Pipeline，提出基于异构平台的图像并行分析方法。此外，作者优化了数据预取和异步数据拷贝，最小化数据迁移带来的开销。然而该方法针对显微镜图像数据分析，任务比较单一，并不适合监控视频中任务多样化的特点。文献[29]为了适应任务类型的多样性和到达时间的不确定性，提出一种基于机器学习的任务调度方法，该方法使用预分配样例作为训练集，不断的调整策略，能够自动获取知识，预测任务的到达时间和任务类型，从而提高了调度器性能，降低了调度过程的开销。该方法针对一般的任务调度，考虑到了任务的多样性和不确定性，然而，并没有考虑到视频监控数据对计算时间的影响，也没有考虑异构设备下的任务调度。文献[32]针对激光探测数据预处理算法，设计数据预处理Pipeline，将任务划分为数据输入，数据注册，数据过滤三个子任务。并提出了GPU/CPU异构设备下，基于强化学习的实时任务调度方法。同时，作者在调度器的设计中加入任务的限制与偏好，更好的适应不同任务的需求。该方法针对单节点异构设备，对多主机集群并不适用。文献[33]提出了一种基于云的分布式图像编码架构，提高了图像编码的效率。Pereira等[34]提出了一种使用动态资源配置的方法来减少视频任务处理时间的架构。然而，文献[33][34]没有考虑到视频任务处理平台环境的动态变化，无法根据环境的变化自适应的调整调度策略，容易造成资源负载的不均衡。文献[35]提出了一种在线的视频处理系统的架构，该架构能够提供可扩展、可靠、高效的视频监控计算能力。Ma等[36]针对海量遥感影响数据，提出了一种基于任务树的动态DAG调度方法，该方法将任务间的依赖性与复杂的并行处理过程分离开来，提高了平台处理大规模数据的能力。Yang等[37]针对实时的视频流处理，提出了一种新的调度算法，该算法依据数据的位置和运行时信息来调度任务。文献[38]提出了一种基于能量优化的云视频监控中心任务调度方法，该方法通过减少虚拟机的数量来降低能耗。然而文献[35][36][37][38]都没有考虑到资源的异构型，所提出的方法无法充分利用异构计算平台的计算能力。

目前，国内有关云平台的研究也得到了广泛的关注，出现了一大批理论研究成果。文献[39]针对IaaS平台,建立了一种基于资源图和层次化资源向量的资源描述模型，提出了一种基于向量分解排序的单节点资源向量调度算法RVS，提高了多维资源协同调度的响应速度。在RVS算法之上，提出了一种基于粒子群算法的多节点需求资源图调度算法 VCE-PSO，优化了多个节点协同调度的响应速度和资源效益。文献[40]针对云计算环境下的动态资源调度问题，参考Amazon EC2的实例资源配置类型及定价模型，提出了两种实例配置方案: 按需实例方案和联合方案。同时考虑到应用数据产生的随机性以及信道状态的随机性，提出了基于带约束马尔可夫决策过程的无线网络选择和上行数据传输调度的联合优化算法。文献[39][40]提出的方法针对一般的IaaS平台，考虑了任务，资源等特点，然而并没有考虑到计算设备的异构性。

可见，国内外为了解决高效利用基础资源问题，或者通过优化调度的方式提高资源利用率，或者通过动态调整资源的方式提高资源利用率，目前异构视频监控领域，很少研究采用深度强化学习和容器技术来提高资源利用率、增加集群吞吐量。本文设计的基于Kubernetes的视频监控云平台和基于深度强化学习的任务调度器，结合容器轻量级和深度强化学习自适应的特点，将有效的提高资源利用率，增加集群吞吐量。

## **1.3 论文的主要研究内容**

上文提到构建一个高效的视频监控云平台至少需要解决以下三大问题：(1)如何高效的管理平台资源；(2)如何与传统的视频监控系统对接；(3)如何设计高效的任务调度器。本文提出基于Docker容器技术和Kubernetes大规模容器集群管理技术的异构视频监控云平台，完成的工作主要包括以下两个部分：

1. 基于Kubernetes的异构视频监控云平台设计与实现

本文利用Docker和Kubernetes相关技术，搭建了视频任务处理平台。Docker采用Linux内核提供的Namespace 和Cgroup技术[41]实现操作系统级别的虚拟化，相对于传统的虚拟机，Docker容器没有其他的额外的性能开销，并且可以直接进行硬件资源如CPU、内存、网络带宽等。Kubernetes是Docker的上层架构，提供了一个全新的基于容器技术的分布式架构解决方案。Kubernetes具有完备的集群管理能力,提供了完善的管理工具。利用Docker和Kubernetes搭建视频监控云平台，解决了传统的基于虚机的视频监控云平台资源利用率低，监控运维困难的问题。

此外，为了与视频监控系统对接，方便用户使用，本文实现了视频监控云平台管理系统，该系统采用Golang语言开发，提供服务管理、资源配额管理、镜像管理、用户管理等核心功能，同时以RESTful的方式为用户提供访问服务，具有简单、松耦合等特点。

1. 基于深度强化学习的异构视频监控云平台任务调度策略

目前基于容器技术的资源调度方法没有考虑资源的物理特性和任务本身的特性，只是简单的根据资源数量的多少进行调度，优先选择GPU资源充裕且满足其计算需求的计算节点进行，并不能满足当前大规模异构视频监控云平台的需求。本文基于深度强化学习提出一个异构智能视频监控云平台上的两级任务调度算法，该方法基于深度强化学习，综合考虑任务类型(计算密集型，IO密集型)、数据特性(分辨率、像素、时长)、资源异构性(CPU、GPU)、数据传输时延等，实时的学习任务调度策略，调度任务到合适的工作节点。此外，本文成功的将该算法应用到搭建的视频监控云平台之上，提升平台的服务处理性能以及资源利用率，增加了集群的吞吐量。

## **1.4 论文的组织结构**

本文共分为六章，各个章节的内容组织如下：

第一章：绪论。本章节首先介绍了本文的研究背景和意义，详细的描述了异构视频监控云平台需要解决的问题。然后介绍了与本文相关的国内外的研究现状，并指出了当前研究的内容的不足指出。最后，简单的描述了本文的研究内容和相关工作。

第二章：相关技术。本章节主要详细介绍了本文所使用的相关技术架构和原理。首先介绍了经典的容器技术Docker，然后介绍了大规模容器管理技术Kubernetes。最后，介绍了平台调度器所使用的深度强化学习技术。为后续使用这些技术奠定了理论基础。

第三章：异构视频监控云平台的设计与实现。本章节首先介绍了传统的视频监控系统的架构和各个功能模块，然后详细介绍了基于Docker技术和Kubernetes大规模容器集群管理技术的异构视频监控云平台的整体架构以及各个模块的功能和实现方法。之后，又详细介绍了本文基于Golang实现的视频监控云平台管理系统的架构和各个模块功能。最后，本文从便捷性和平台的稳定性方面，对平台的安装部署进行了优化。

第四章：异构视频监控云平台中任务调度策略的研究。本章节首先简单介绍了两个异构视频监控云平台上典型的视频处理任务Pipeline，然后详细的描述了异构视频监控云平台上的任务调度问题以及现有的解决方法。之后，基于此问题，提出了基于深度强化学习的两级调度模型CS-NS，并详细的描述了该模型的工作流程和原理。此外，还简单的描述了如何将该模型应用到本文搭建的异构视频监控云平台中。最后，通过实验为该模型选取合适的参数并验证了该模型的收敛性。

第五章：平台的功能及性能测试。本章节从功能和性能两个方面对本文搭建的异构视频监控云平台进行测试。首先介绍了平台的搭建环境，然后介绍了相关的视频处理任务的实现方法，最后分别从功能和性能两个方面进行测试。功能上采用流媒体服务验证了平台功能的完整性，性能上对比了当前主流的调度算法，证明了本文提出的调度方法明显优于其它方法。

第六章：总结与展望。总结了本文的研究内容以及创新成果，并展望未来，提出了未来可以就研究的方向。

# **第二章 相关技术**

## **2.1 Docker相关技术简介**

### **2.1.1 Docker简介**

Docker是由dotCloud公司于2013年开源的容器管理引擎。它采用Golang语言编写，自开源以来一直备受关注[23]。Docker的目标是实现操作系统级别的虚拟化解决方案。它可以让我们在一个资源隔离的进程中运行应用及其依赖项。运行应用程序所必需的组件都将打包成一个镜像并可以复用。执行镜像时，它运行在一个隔离环境中，并且不会共享宿主机的内存、CPU 以及磁盘，这就保证了容器内进程不能监控容器外的任何进程。

图2.1为Docker的架构图，其核心组件包括Docker客户端和服务端、Docker镜像、镜像仓库和Docker容器。

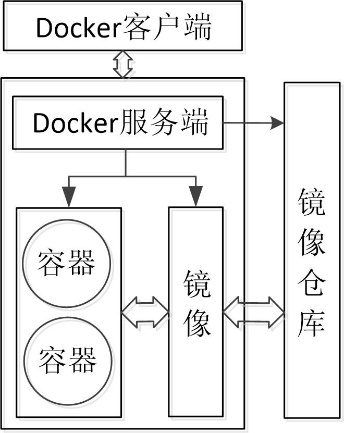


图2.1 Docker架构图[42]

Docker客户端和服务器： Docker是典型的C/S模式的架构，服务端以守护进程的方式运行，以接口的形式对外提供服务。客户端通过接口与服务端通信，完成容器的创建、运行以及镜像管理等操作。客户端可以与服务端在同一个主机，也可以通过远程通信实现跨主机访问。

Docker镜像：镜像是Docker的基础，用户基于镜像来运行容器。镜像就像是容器的“源代码”，基于同一个镜像可以运行多个容器。镜像一般很小，易于存储、分享和更新。镜像的来源一般有三种方法，包括基于Dockerfile文件构建、基于本地模板导入、基于以有的容器创建。

镜像仓库: 是用来集中存放用户构建的镜像。镜像仓库分为共有和私有两种形式，最大的公开镜像仓库是Docker hub,由Docker公司维护。用户可以在Docker Hub注册账户，分享并保存自己的镜像。此外，Docker提供私有的镜像仓库，用户可以借此搭建自己的私有镜像仓库以满足一些特殊的需求。本文设计的视频监控云平台就使用私有的镜像仓库来保证视频处理的镜像的安全。

容器：Docker 利用容器（Container）来运行应用。容器是从镜像创建的运行实例。每个容器都是相互隔离的、保证安全的平台。可以把容器看做是一个简易版的 Linux 环境（包括root用户权限、进程空间、用户空间和网络空间等）和运行在其中的应用程序。Docker借鉴了集装箱的概念。每个容器都包含一个软件镜像，也就是容器的“货物”，并且与真正的货物一样，容器的软件镜像可以进行一些操作，如启动、开始、停止、删除。与集装箱一样，Docker在执行上述操作时，并不关心容器中到底塞进了什么，也不关心用户要把容器运到何方。使用容器，可以快速构建一个应用程序服务器、一个消息总线、一套使用的工具、一个持续集成环境、或者任意一种应用程序、服务或工具。

Docker支持将应用打包进一个可以移植的容器中，重新定义了应用的开发、测试和部署过程。其特点包括：

持续部署与测试：容器消除了线上线下的环境差异，保证了应用生命周期的环境一致性标准化。开发人员使用镜像实现标准开发环境的构建，开发完成后通过封装着完整环境和应用的镜像进行迁移，由此，测试和运维人员可以直接部署软件镜像来进行测试和发布，大大简化了持续集成、测试和发布的过程。

跨云平台支持：容器带来的最大好处之一就是其适配性，越来越多的云平台都支持容器，用户再也无需担心受到云平台的捆绑，同时也让应用多平台混合部署成为可能。目前支持容器的IaaS云平台包括但不限于亚马逊云平台(AWS)、 Google云平台(GCP)、微软云平台( Azure)、 Open Stack等，还包括如Chef、 Puppet、 Ansible等配置管理工具。

高资源利用率与隔离：容器没有管理程序的额外开销，与底层共享操作系统，性能更加优良，系统负载更低，在同等条件下可以运行更多的应用实例，可以更充分地利用系统资源。同时，容器拥有不错的资源隔离与限制能力，可以精确地对应用分配CPU、内存等资源，保证了应用间不会相互影响。

### **2.1.2 Docker与虚拟机对比**



图2.2 虚拟机与容器的实现对比图[43]

如图2.2所示，左图虚拟机的虚拟操作系统层和Hypervisor层在Docker中被Docker Engine层所替代。虚拟机操作系统是一个完整操作系统内核；虚拟机的硬件虚拟化平台层以内核态的驱动存在物理机的操作系统中。虚拟机实现资源隔离的方法是利用独立的操作系统，并利用硬件虚拟化平台虚拟化CPU、内存、IO设备等实现的。对比虚拟机实现资源和环境隔离的方案，Docker就显得简练很多。右图Docker Engine可以简单看成对Linux的Namespace、Cgroup、镜像管理文件系统操作的封装。Docker并没有和虚拟机一样利用一个完全独立的虚拟机操作系统实现环境隔离，它利用的是目前Linux内核本身支持的容器方式实现资源和环境隔离。简单的说，Docker利用namespace实现系统环境的隔离，利用Cgroup实现资源限制，利用镜像实现根目录环境的隔离。

对比虚拟机，Docker具有如下优势：  
（1）Docker有着比虚拟机更少的抽象层[43]。

由于Docker不需要硬件虚拟化平台实现硬件资源虚拟化，运行在Docker容器上的程序直接使用的都是实际物理机的硬件资源。因此在CPU、内存利用率上Docker将会在效率上有优势。  
（2）Docker的启动时间更快[44]。

由于Docker利用的是宿主机的内核，而不需要虚拟操作系统。当新建一个容器时，Docker不需要和虚拟机一样重新加载一个操作系统内核。我们知道，引导、加载操作系统内核是一个比较费时费资源的过程，当新建一个虚拟机时，虚拟机软件需要加载新的操作系统，这个新建过程是分钟级别的。而Docker由于直接利用宿主机的操作系统，则省略了这个过程，因此新建一个Docker容器只需要几秒钟。另外，现代操作系统是复杂的系统，在一台物理机上新增加一个操作系统的资源开销是比较大的，因此，Docker对比虚拟机在资源消耗上也占有比较大的优势。

相比于虚拟机容器拥有更高的资源使用效率，因为它并不需要为每个应用分配单独的操作系统。实例规模更小，创建和迁移速度也更快。这意味相比于虚拟机，单个操作系统能够承载更多的容器。本文采用Docker容器作为视频任务的运行实例，利用Docker计算性能损耗低、快速创建和销毁的特点，能够减低平台额外的消耗，提升平台资源利用率。

## **2.2 Kubernetes大规模容器管理技术**

### **2.2.1 Kubernetes基本概念**

Kubernetes是一个用于容器集群的自动化部署、扩容以及运维的开源平台。通过Kubernetes，可以快速有效地响应用户需求；快速而有预期地部署你的应用；极速地扩展你的应用;无缝对接新应用功能；节省资源，优化硬件资源的使用。为容器编排管理提供了完整的开源方案。在Kubernetes的生态环境中，有许多重要概念和术语，其中本文相关的一些重要的概念和术语[45]如下：

Pod：是Kubernetes管理的最小单元，也是所有业务的基础。如图2.3所示，每个Pod都有一个特殊的被称为“根容器”的Pause容器。除了Pause容器之外，每个Pod还包含一个或多个相关的业务容器。这些容器共享pause容器的网络命令空间、共享Pause挂在的volume。在Pod中，所有的容器被统一调度、共享一个IP，容器之间可以通过localhost进行访问。这样，即简化了密切相关的容器之间的网络通信问题，有很好地解决了他们之间文件的共享问题。

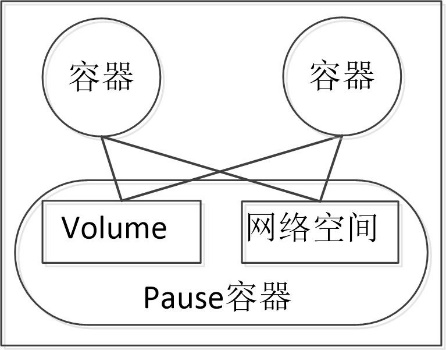


图2.3 Pod组成图

Label和Label Selector：是一个键值对，label可以附加到各种资源对象上，比如Pod、Service。同时，一个资源对象也可以定义任务数量的Label。通过给资源定义Label，然后使用Label Selector查询和筛选拥有某些Label的资源对象。Label和Label Selector是Kubernetes系统中最核心的应用模型，使得被管理对象能够被精细地分组管理，同时实现整个集群的高可用性。如图2.4所示，当选择标签为”env=dev”的pod的时候，pod A和pod B两个pod被选中；当选择标签为”Version=v2”的pod的时候，pod B 和 pod D两个pod被选中。

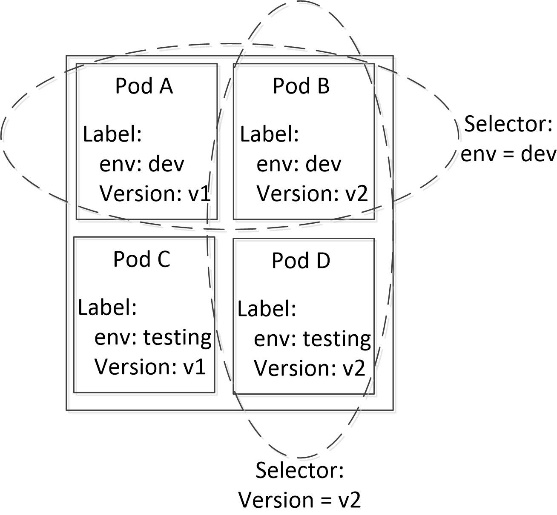


图2.4 Label Selector 作用范围

Service: 如图2.5所示，Service是一个服务的访问入口，Service与其后端Pod副本集群之间是通过Label Selector来实现对接。Service作为Pod路由代理抽象，访问端只需要知道service的地址，由Service来提供代理，保证了pod的动态变化对访问端透明。

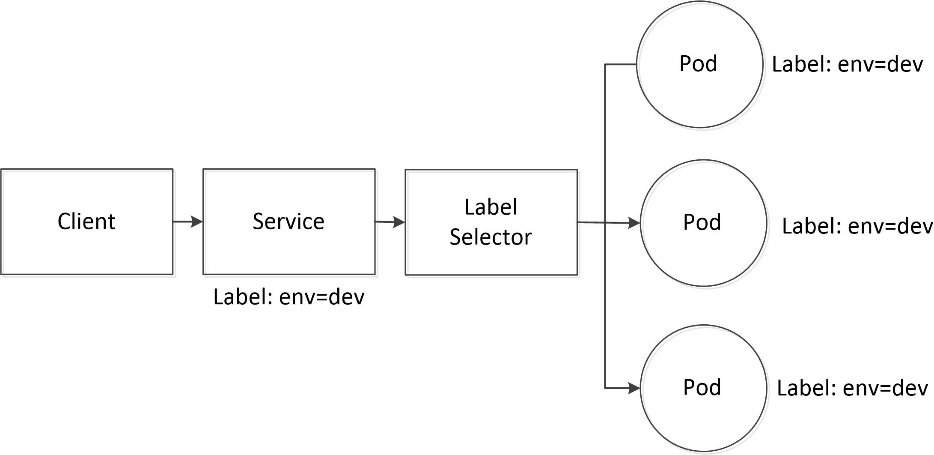


图2.5 service与pod的关系

### **2.2.2 kubernetes 架构**

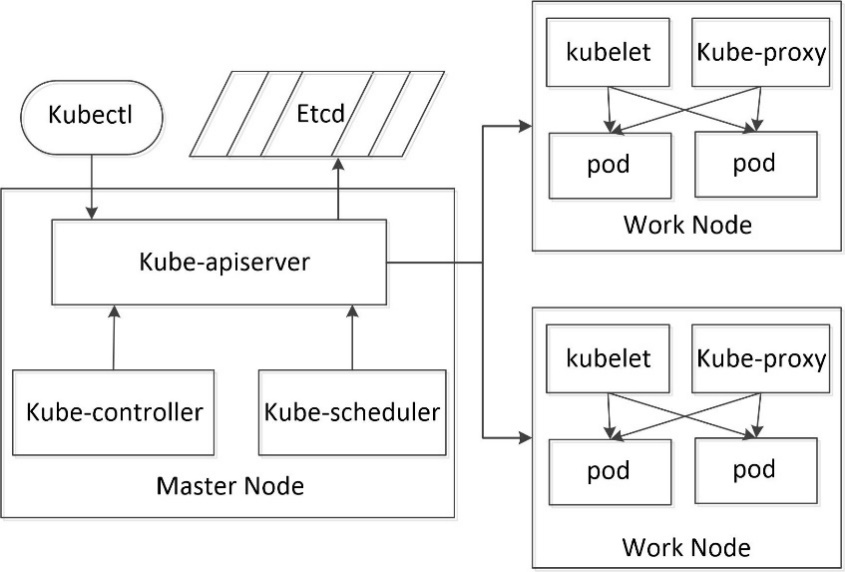


图2.6 Kubernetes架构图[25]

如图2.6 所示，Kubernetes将集群中的机器划分为一个Master节点和一群工作节点。其中，Master节点上运着集群管理相关的组件kube-Apiserver、Kube-Controller和Kube-Scheduler，这些进程实现了整个集群的资源管理、Pod调度、弹性伸缩、安全控制、系统监控和纠错等管理功能。其各个组件具体的功能如下：

**Kube-apiserver:** 提供了HTTP Rest接口的关键服务进程，是Kubernetes里所有资源的增删该查等操作的唯一入口，也是集群的入口进程。其他所有的组件都必须通过它提供的API来操作资源数据。

**Kube-controller:** Kubernetes里所有资源对象的自动化控制中心，实现了集群故障检测和恢复的自动化工作。

**Kube-scheduler：**是负责资源调度的进程。它接收来自于kube-controller-manager触发的调度操作请求，然后根据请求规格、调度约束、整体资源情况等因素进行调度计算，最后将任务发送到目标节点。

Node作为集群中的工作节点，运行真正的应用程序，在Node上运行着Kubernetes管理的最小单元Pod。Node上运行着节点管理相关的组件Kubelet、Kube-Proxy，这些组件负责Pod的创建、删除、启动、监控、销毁，以及实现Pod间访问的负载均衡。其各个组件的具体功能如下：

**Kubelet:**负责Pod的创建、启停等任务。此外，Kubelet会在Kube-Apiserver上注册节点自身的信息，定期向Kube-Apiserver汇报节点资源的使用情况。

**Kube-proxy：**主要负责Service Endpoint到pod实例请求转发及负载均衡的规则管理。

Etcd是分布式的键值数据存储，可以通过RESTful API去执行CRUD的操作，Kubernetes使用Etcd作为主要数据存储。Kubernete集群中的集群状态信息全部存储在Etcd中。

Kubectl是一个用于操作Kubernetes集群的命令行接口，提供操作集群的一系列子命令，是在使用Kubernetes中非常常用的工具。

### **2.2.3 Kubernetes优势**

在Kubernetes集群中，一个服务可以自我扩展、自我诊断，并且容易升级和扩缩容。在收到服务扩容的请求之后，Kubernetes会触发调度流程，最终在选定的目标节点上启动相应数量的服务实例副本，这些副本在启动成功之后会自动加入负载均衡器中并生效，整个过程无需额外的人工操作。另外，Kubernetes会定时巡查每个服务的所有实例的可用性，确保服务实例的数量始终保持为预期的数量，当它发现某个实例不可用时，会自动重启该实例或者在其他节点重新调度、运行一个新实例。这样，一个复杂的过程无需人工干预即可全部自动化完成。

Kubernetes吸收了google内部多年的运行经验和源自于开源社区的最佳实践，形成了一个大的生态环境。其关键特性包括：

自动化装箱：在不牺牲可用性的条件下，基于容器对资源的要求和约束自动部署容器。同时，为了提高利用率和节省更多资源，将关键和最佳工作量结合在一起。

失败恢复：当容器失败时，会对容器进行重启；当所部署的Node节点有问题时，会对容器进行重新部署和重新调度；当容器未通过监控检查时，会关闭此容器；直到容器正常运行时，才会对外提供服务。

水平自动扩缩容：通过简单的命令、用户界面或基于CPU的使用情况，能够对应用进行扩容和缩容。

服务发现和负载均衡：开发者不需要使用额外的服务发现机制，就能够基于Kubernetes进行服务发现和负载均衡。

滚动升级：Kubernetes能够程序化的发布应用和相关的配置。如果发布有问题，Kubernetes将能够回归发生的变更。

存储管理：自动挂接存储系统，这些存储系统可以来自于本地、公共云提供商（例如：GCP和AWS）、网络存储(例如：NFS、iSCSI、Gluster、Ceph、Cinder和Floker等)。

## **2.3 深度强化学习相关技术**

### **2.3.1 强化学习简介**

强化学习是机器学习的一个重要分支，主要用来解决连续决策问题。强化学习可以在复杂的、不确定的环境中学习如何实现我们设定的目标[46]。强化学习的应用场景非常广，几乎包括了所有需要做一系列决策的问题，比如控制机器人、玩游戏等。一个强化学习的问题包含三个主要的概念：环境状态，行动和奖励。强化学习的目标是获取最多的累计奖励。强化学习模型根据环境状态、行动和奖励，学习出最佳的策略，并以最终结果为目标，不能只看某个行动当下带来的利益，还要看到这个行动未来能带来的价值。它不像无监督学习那样完全没有学习目标，也不像监督学习那样由明确的目标。强化学习的目标一般是变化的、不明确的，甚至可能不存在绝对正确的标签。

Policy-Based和Value-Based是强化学习中重要的两类方法,其主要区别在于Policy-Based的方法直接预测在某个环境状态下应该采取的动作，而Value-Based的方法则是预测某个环境状态下所有动作的期望值，之后选择期望值最高的动作执行。而结合深度学习后，Policy-Based的方法就成了策略网络，而Value-Based的方法则成了估值网络。

### **2.3.2 Deep Q-Network**

Q-Learning的目标是求解函数，即根据当前环境状态，估算Action的期望价值。Q-Learning训练模型的过程中，以（状态、行为、奖励、下一个状态）构成的元组为样本进行训练。其中为当前状态，a为当前状态下执行的Action，为在执行Action后获得的奖励，为下一个状态。其中特征是，而学习的目标则是，这个学习目标是当前Action获得的Reward加上下一步可获得的最大期望价值。学习目标中包含了Q-Learning的函数本身，所以这其中使用了递归求解思想。下一步可获得的最大期望价值被乘以一个，即衰减系数，这个参数决定了未来奖励在学习中的中的重要性。如果折扣因子为0，那么模型将学习不到任何未来的奖励信息，将会变得短视，只关注当前的利益；如果折扣因子大于等于1，那么算法将无法收敛。Q-Learning的学习过程可以写成公式(1)，将旧的Q-Learning函数想着学习目标（当前获得的Reward加上下一步可获得的最大期望价值）按照一个较小的学习速率a学习，得到新的Q-Learning函数。如果用来学习Q-Learning的模型是深度神经网络，这样的到的模型即是Deep Q-Network（DQN）[27]。

 (1)

## **2.4 本章小结**

本章介绍了本文相关的技术。首先介绍了Docker容器技术，简单介绍了Docker的架构和特点，对比了Docker与虚拟机的不同和优势。然后介绍了大规模容器管理技术Kubernetes，简单介绍了Kubernetes相关的基本概念、原理结构以及优势。最后，介绍了深度强化学习的相关知识，包括策略网络和估值网络等。

# **第三章 异构视频监控云平台设计与实现**

## **3.1 视频监控系统**

传统的视频监控服务是一个关注视频或音频应用技术的电信服务，主要被用来捕获远端的多媒体（包括视频、音频、图片和各种报警信号等）并呈现给终端用户。如图3.1所示，传统的视频监控系统主要包括6大部分[13]：中心管理单元（CMU）、服务控制单元(SCU)、媒体数据存储单元(MSU)、媒体分发单元(MDU)、数据采集单元(PU)、数据消费单元(CU)。

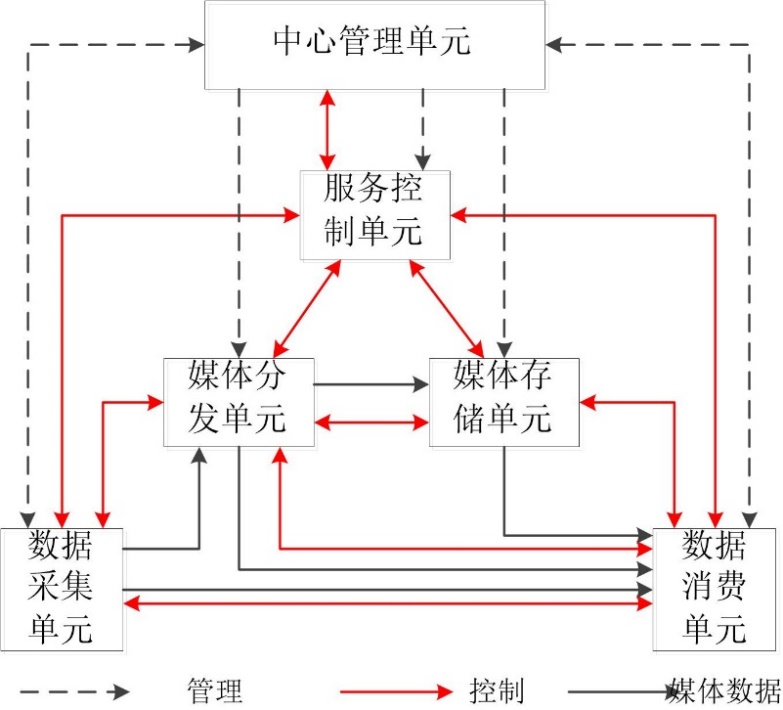


图3.1 视频监控系统[13]

中心管理单元（CMU）：负责中心系统管理、服务操作管理等。其主要功能包括管理内容、服务、网络、终端设备；访问权限控制；监听PU和CU的状态；

服务控制单元(SCU)： 用于访问PU和CU之间的服务控制和信号呼叫控制。SCU的主要功能包括接收来自CU和PU的访问注册请求，并将其注册到注册管理服务器；接收来自CU和PU的访问认证请求，并且将其作为认证服务器功能；识别CU和PU，并将身份与用户数据和终端数据进行比较以确定匹配结果；授权CU和PU访问权限；调解与PU和CU的呼叫控制会话，建立和维护媒体传输路径信息（媒体路由表）等。

媒体数据存储单元(MSU)：用来检索、存储媒体数据，并提供流媒体服务的能力。其功能主要包括从PU或MSU中获取多媒体数据（支持HTTP和RSTP协议）；存储多媒体数据；提供流媒体服务；检索媒体数据；

媒体分发单元(MDU)：MDU用来从PU到CU传输媒体数据。其主要功能包括从PU接收媒体数据；处理媒体数据（包括编码、解码等）；查找PU与CU之间的路由信息，确定媒体数据的传输路径；传播媒体数据到多个请求的CU；转发媒体数据到其他MDU；复制媒体数据等。

数据采集单元(PU)：用来采集多媒体数据。其主要功包括捕获多媒体数据；多媒体数据编码；向外部设备输出警告信息；解析控制信息并操作监控设备；传输多媒体数据到其他实体。

数据消费单元(CU)：CU是视频监控系统中的消费者，向终端用户呈现多媒体数据。主要功能包括多媒体数据的解码；展示视频和音频数据；提供操作视频监控系统的接口。

## **3.2 异构视频监控云平台总体架构**

本文基于Docker容器技术，设计异构视频监控云平台，实现传统视频监控系统的云服务化。并提出基于Kubernetes的统一资源管理系统，解决了传统视频监控云平台中资源粒度大，利用率低的问题。如图3.2所示，本文设计的视频监控云平台主要包括视频监控系统、镜像仓库、集群管理单元、资源管理单元、云化资源池、虚拟视频处理单元、异构物理服务器和服务访问网关。其中视频监控系统在3.1小结中详细说明。

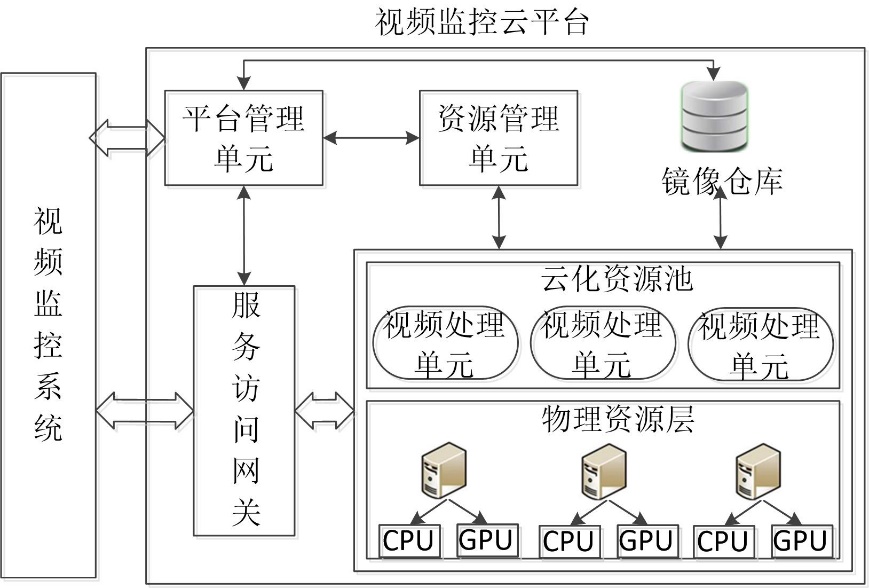


图3.2 视频监控云平台总体设计图

**镜像仓库**：镜像仓库是监控云平台镜像存储中心，保存着平台中所有已构建的视频微服务镜像，同时，当创建新的微服务时为计算节点提供镜像服务。

**集群管理单元**：集群管理单元是视频云服务平台的管理中心，包括集群管理、镜像仓库管理、视频微服务管理、用户管理等。同时视频云服务管理器是整个集群的入口，用户通过集群管理单元与平台进行交互，为用户提供可视化界面方便查看资源的使用情况和监控视频服务的运行状态。主要负责负责安全认证、接受用户请求、协调各个组件工作、返回服务结果信息等。为了方便与传统的视频监控系统对接和功能扩展，该单元对外提供RESTful风格的API。

**资源管理单元**：云资源管理单元是集群资源的管理中心，负责监控资源集群资源、服务状态保证服务稳定的运行。同时资源管理单元接收集群管理单元的任务请求，分配资源并调度视频处理任务等。

**云化资源池**：云化资源池是逻辑模块，实际的物理架构中并不存在，为了表示方便而设计，包括所有的虚拟视频处理单元的集合。

**虚拟视频处理单元**: 虚拟视频处理单元是视频微服务的运行实例，每个虚拟视频处理单元对应一个容器。负责完成具体的视频任务。虚拟视频处理单元运行在具体的物理节点上，由节点上的agent负责管理和回收。

**服务访问网关**: 服务访问网关是用户访问平台提供的视频服务的入口，该模块主要包括用户登陆认证、负载均衡和服务发现的功能。

**异构物理节点**: 异构物理节点是平台的物理计算单元，包括CPU/GPU异构计算单元，每个节点上部署节点管理agent和Docker容器管理引擎，为视频监控云平台服务。

## **3.3 异构视频监控云平台工作流程**

在本部分中，我们专注于视频监控云平台的工作流程。客户端(传统的视频监控系统或者平台用户)通过集群管理单元提供的界面或API与视频监控云平台对接。客户端提交视频任务处理请求到集群管理单元，集群管理单元进行一系列的安全认证（用户的合法性，任务的合法性，用户的权限信息等），然后重新封装任务请求到资源管理单元，资源管理单元接受请求，并根据当前的集群资源状态和视频处理任务的特点，调度任务到合适的物理节点，物理节点上的节点管理agent收到任务处理信息，便去视频镜像仓库中拉取相应的视频任务处理镜像运行。运行结束后，向资源管理单元返回结果信息，资源管理单元向集群管理单元返回信息，集群管理单元向用户返回信息，用户收到返回信息后，可以通过返回的信息访问视频处理服务。以下从平台上视频服务的创建和服务的访问两个方面说明平台的工作流程。

### **3.3.1 视频服务的创建流程**

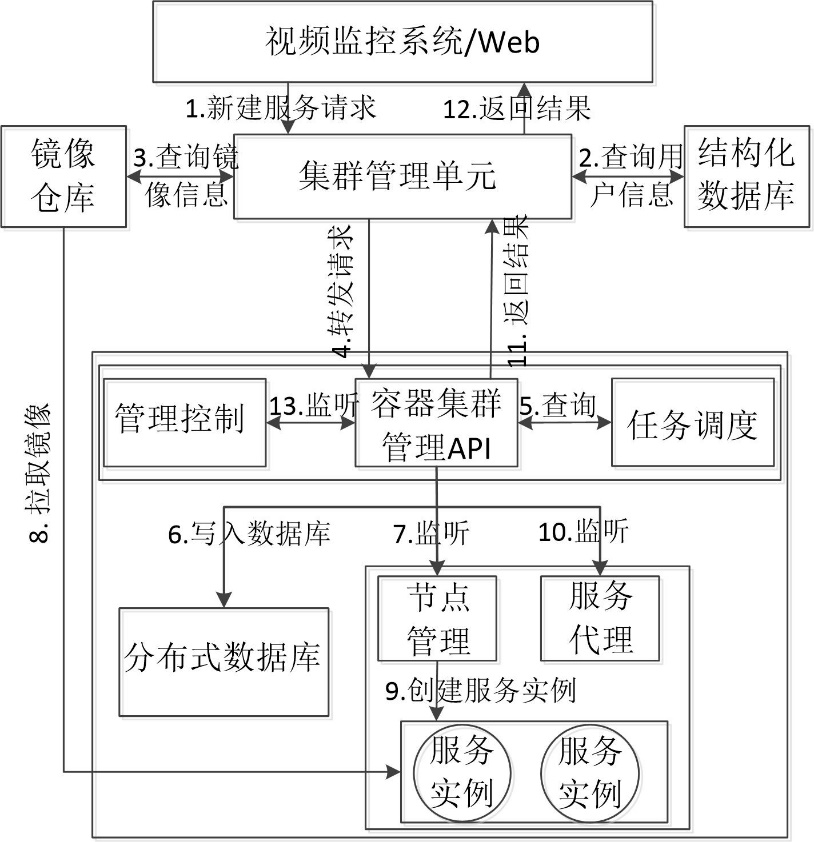


图3.3视频监控云平台服务创建流程图

本文设计的视频监控云平台，客户端发起服务创建请求，集群管理单元接受请求并协调其他组件完成视频服务的创建。如图3.3所示，具体的服务创建流程如下：

1. 视频监控系统或者web平台向视频监控云平台管理API发送新建服务实例的请求；
2. 视频监控云平台管理API接收用户请求，并进行安全认证，向结构化数据库查询用户信息和服务信息；
3. 视频监控云平台管理API向镜像仓库查询相关镜像；
4. 视频监控云平台管理API向容器云平台转发创建服务实例的请求；
5. 容器云平台接受请求并向调度器查询合适的工作节点；
6. 容器集群管理API收到服务实例与工作节点的绑定信息，并将信息写入分布式数据库；
7. 节点管理组件通过容器集群管理API监听到绑定到本节点的服务实例，并获取服务实例启动的相关信息；
8. 节点管理组件去镜像仓库拉取服务镜像；
9. 节点管理组件使用服务镜像创建服务实例；
10. 服务代理组件监听启动的服务，为服务的访问做端口映射；
11. 容器云平台向视频监控云平台管理API返回服务启动结果信息；
12. 视频监控云平台向视频监控系统或web平台返回结果信息；
13. 管理控制组件监听集群中服务实例的个数，保证服务实例满足用户需求。

### **3.3.2 平台服务的访问流程**

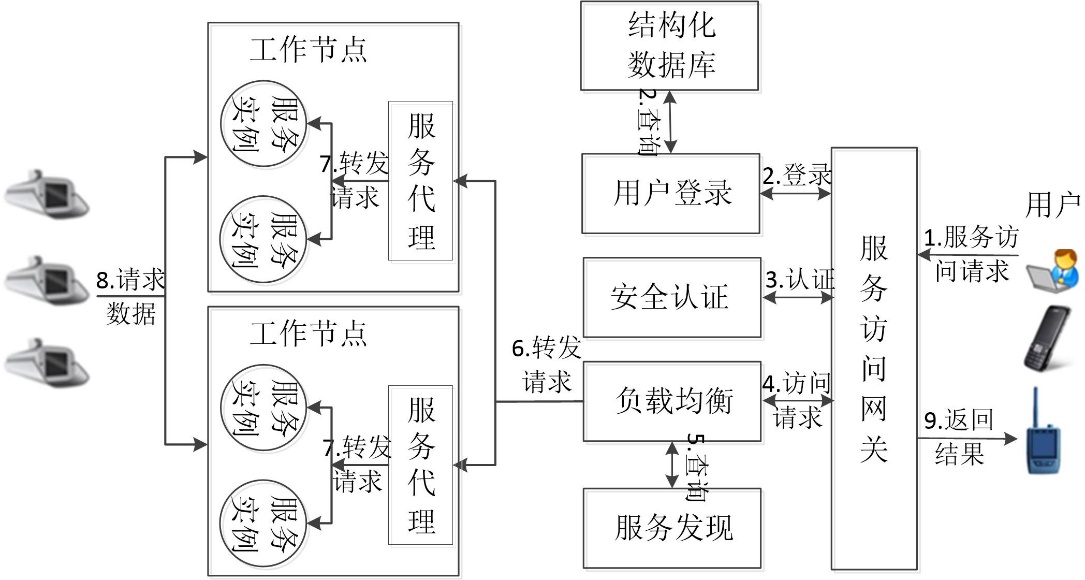
****

图3.4 视频监控云平台服务访问流程

本文设计的视频监控云平台中，用户通过服务访问网关访问平台中的视频处理任务，服务访问网关经过登陆、认证，最终转发请求到合适的服务实例中。如图3.4所示，其具体的访问工作流程如下：

1. 用户向服务访问网关发起服务访问请求；
2. 服务访问网关检查用户登录信息；
3. 服务访问网关进行安全认证；
4. 服务访问网关转发用户请求到向负载均衡组件；
5. 负载均衡组件向服务发现组件查询相应的服务提供者列表；
6. 并转发用户请求到特定的服务提供者；
7. 服务提供者上的服务代理组件接受请求并转发请求到相应的实例；
8. 服务实例接受请求，并从网络摄像头获取数据为用户提供服务；
9. 最后服务访问网关向用户返回结果信息。

## **3.3 分布式资源管理单元**

资源管理单元是视频监控云平台的重要部分，当前基于Docker的大规模容器编排调度技术主要有Docker swarm和Kubernetes。考虑到Kubernetes在容器编排调度、集群监控运维等方面的能力，以及具有灵活可扩展、完善的社区生态环境、快速部署、高可用等优势。本文采用Kubernetes作为集群资源管理单元。基于Kubernetes的分布式资源管理单元如图3.5所示。主要包括分布式数据存储、API服务器、控制器、调度器、节点代理和访问代理。各个组件的功能如下：

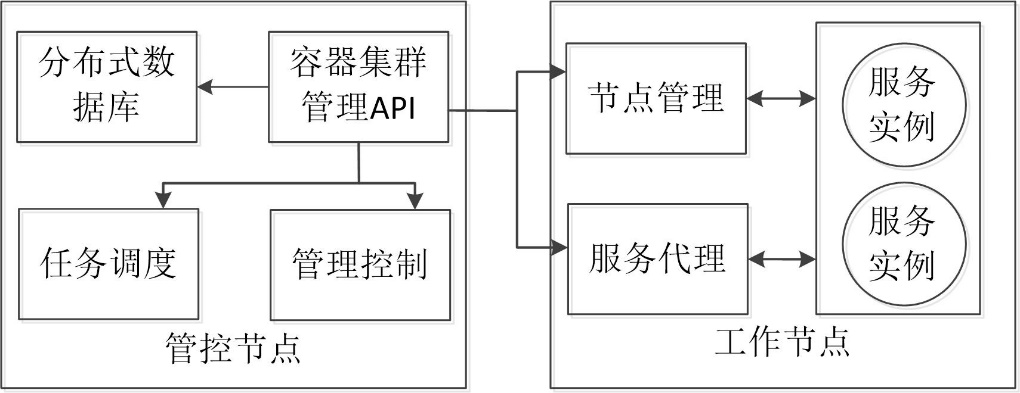


图3.5 分布式资源管理

分布式数据库: 是一个分布式键/值存储集群，是Kubernetes集群的主数据库，为Kubernetes云平台提供存储服务，包括集群的资源状态、集群中应用的运行状态等。本文采用三实例的Etcd集群实现分布式存储。

容器集群管理API：提供了Kubernetes集群上各类资源对象的增、删、改、查和监听的接口，是集群内各个功能模块之间数据交互的和通信的中心枢纽。主要提供一下功能特性:（1）集群管理的API入口；（2）资源配额控制入口；（3）提供完备的集群安全机制。本文使用Kube-Apiserver实现API服务器。

管理控制：控制器是集群的控制中心，控制器通过API服务器提供的接口查询并监听集群的资源和应用的运行状态，保证平台的可用和稳定。当出现服务异常和节点宕机时，即使的调整（包括迁移、重启等）服务实例，以保证服务的稳定可用。本文采用Kube-Controller实现控制器。

任务调度：调度器的作用是待调度的实例按照规定的调度算法和调度策略，绑定到集群的某个合适节点中。本文使用Kube-Scheduler作为集群调度器。

节点管理：节点代理安装到工作节点，负责实时想API服务器汇报节点的资源状态，节点上实例的运行状态。同时，节点代理通过API服务器提供的接口，实时的获取实例的创建或删除指令，创建或删除本节点上的服务实例。本文采用Kubelet实现节点代理。

**服务代理**：是服务的代理兼负载均衡器，负载将对某个服务的请求转发到后端的多个实例上。本文采用Kube-Proxy实现服务访问代理。

接下来介绍，如何基于Kubernetes构建分布式资源管理系统。一共分为如下四个步骤：

（1）生成安全证书

为了保证集群的安全可靠，本文搭建的平台中，Kubernetes相关的组件使用安全端口与Kube-Apiserver通信。为此，须提前为组建的通信生成安全证书。生成证书的脚本如下：

|  |
| --- |
| #!/bin/bash  #CA  openssl genrsa -out dd\_ca.key 2048  openssl req -x509 -new -nodes -key dd\_ca.key -days 10000 -out dd\_ca.crt -subj "/CN=kube-ca"  #key  openssl genrsa -out dd\_server.key 2048  openssl req -new -key dd\_server.key -out dd\_server.csr -subj "/CN=kube-apiserver" -config openssl.conf  #certificate  openssl x509 -req -in dd\_server.csr -CA dd\_ca.crt -CAkey dd\_ca.key -CAcreateserial -out dd\_server.crt -days 10000 -extensions v3\_req -extfile openssl.conf |

脚本中的配置文件openssl.conf如下,其中不同的节点对应不同的节点ip。

|  |
| --- |
| [req]  req\_extensions = v3\_req  distinguished\_name = req\_distinguished\_name  [req\_distinguished\_name]  [ v3\_req ]  basicConstraints = CA:FALSE  keyUsage = nonRepudiation, digitalSignature, keyEncipherment  subjectAltName = @alt\_names  [alt\_names]  DNS.1 = kubernetes  DNS.2 = kubernetes.default  DNS.3 = kubernetes.default.svc  DNS.4 = controller  IP = node ip |

（2）构建分布式的Etcd集群

Etcd是Coreos开发的分布式服务系统，内部采用raft协议作为一致性算法。作为一个高可用的配置共享、服务发现的键值存储系统，Etcd具有简单、安全、快速、可靠等特点。

本文搭建的平台，Etcd集群部署在三个管控节点上。在三个管控节点上下载并安装Etcd v3.1.12。各个节点的Etcd配置如下：

|  |
| --- |
| ETCD\_DATA\_DIR=/var/lib/etcd  ETCD\_NAME=etcd0  ETCD\_INITIAL\_CLUSTER=etcd0=http://master1:2380,etcd1=http://master2:2380,etcd2=http://master3:2380  ETCD\_INITIAL\_CLUSTER\_STATE=new  ETCD\_LISTEN\_PEER\_URLS=http://0.0.0.0:2380  ETCD\_INITIAL\_ADVERTISE\_PEER\_URLS=http://0.0.0.0:2380  ETCD\_ADVERTISE\_CLIENT\_URLS=http://0.0.0.0:2379  ETCD\_LISTEN\_CLIENT\_URLS=http://0.0.0.0:2379  ETCD\_INITIAL\_CLUSTER\_TOKEN=my-etcd-cluster  GOMAXPROCS=24 |

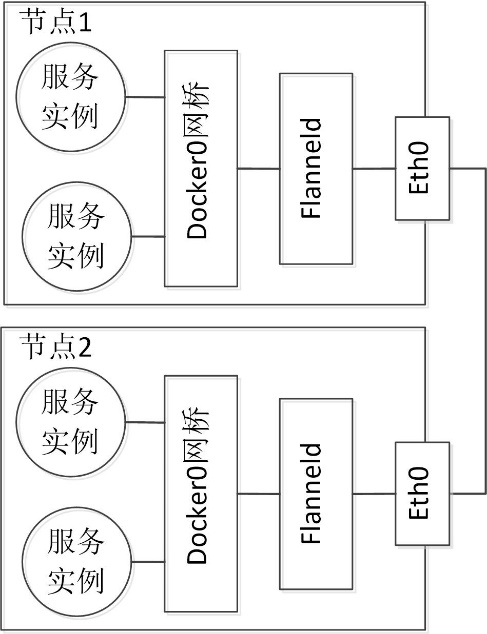
（3）安装Flanneld

本文搭建的平台中，需要保证所有的节点上的Docker容器实例之间能够互相通信。为此，需要达到以下两个要求：

① 所有节点上的容器实例要有不同的IP地址；

② 源实例能够通过路由找到目的实例。

Flannel是CoreOS提供用于解决Dokcer集群跨主机通讯的覆盖网络工具。它的主要思路是：预先留出一个网段，每个主机使用其中一部分，然后每个容器被分配不同的ip；让所有的容器认为大家在同一个直连的网络，底层通过UDP/VxLAN等进行报文的封装和转发。其工作原理如图3.6所示，数据从源容器中发出后，经由所在主机的docker0虚拟网卡转发到flannel0虚拟网卡，这是个P2P的虚拟网卡，flanneld服务监听在网卡的另外一端。源主机的flanneld服务将原本的数据内容UDP封装后根据自己的路由表投递给目的节点的flanneld服务，数据到达以后被解包，然后直接进入目的节点的flannel0虚拟网卡，然后被转发到目的主机的docker0虚拟网卡，最后由docker0到达目标容器。

图3.6 flannel原理图

本文搭建的平台采用flanneld解决平台中的容器实例相互访问的问题，并采用Etcd作为flanneld的后端存储。分别在各个节点上安装flanneld之后，按照如下配置，配置flanneld。

|  |
| --- |
| FLANNELD\_ETCD\_ENDPOINTS="-etcd-endpoints=http://master1:2379,http://master2:2379,http://master3:2379"  FLANNELD\_ETCD\_PREFIX="-etcd-prefix=/coreos.com/network" |

（4）安装Kubernetes相关组件

Kubernetes是大规模的容器管理的技术，共包括5个核心组件。其中Kube-Apiserver、Kube-Controller、Kube-Scheduler是管控面组件，安装在三个管控节点上，Kubelet、Kube-Proxy是共组工作节点组件，安装在工作节点上。Kube-Apiserver与Etcd集群交互，存储集群的状态信息。其余的组件通过Kube-Apiserver相互交互，对Etcd中集群的状态数据进行增删改查。

## **3.4 镜像仓库**

镜像仓库是整个平台的镜像存储中心，保存所有视频处理镜像应用的镜像为了保证镜像的安全，本文采用Docker官方提供的镜像仓库镜像Registry搭建本地镜像存储中心。同时，为了保证镜像仓库的问题，本文利用Kubernetes在监控运维上的能力，使镜像仓库以服务的形式运行在Kubernetes平台中，并搭建NFS服务，挂载磁盘到实例中作为镜像仓库实例后端存储。其部署编排文件如下：

|  |
| --- |
| apiVersion: v1  kind: Pod  metadata:  labels:  app: registry  name: registry  namespace: default  spec:  containers:  - image: registry  name: registry  ports:  - containerPort: 5000  name: mysql  protocol: TCP  volumeMounts:  - mountPath: /var/lib/registry  name: registry-persistent-storage  volumes:  - name: registry-persistent-storage  persistentVolumeClaim:  claimName: registry-pv-claim |

## **3.5 集群管理单元**

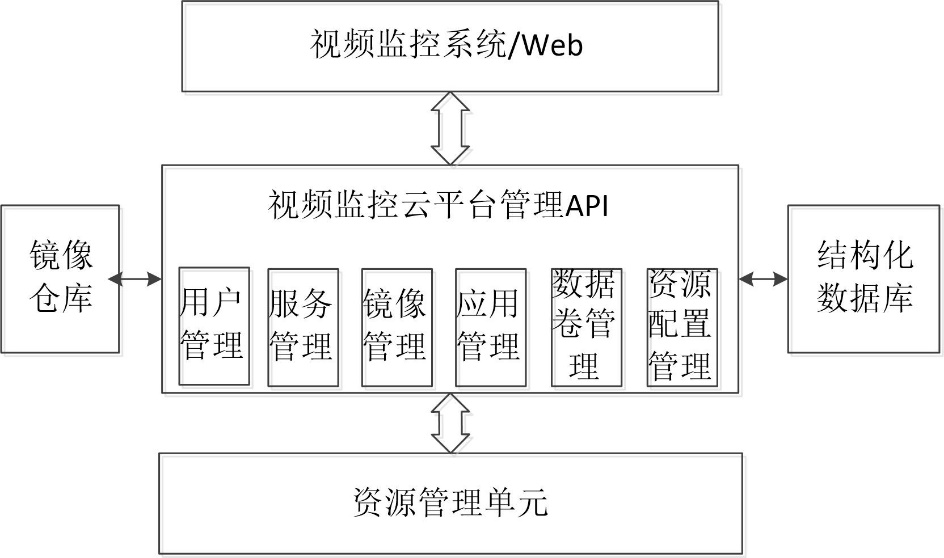
集群管理单元是整个集群的入口，对外提供可视化界面和RESTful风格的访问接口。本文的设计的平台中，集群管理单元RESTful风格的API采用Golang语言开发，可视化界面采用react开发，数据库使用Mysql。该单元主要包括用户管理、镜像管理、应用管理、资源配置管理、数据卷管理、服务管理等。图3.7展示了资源管理的总体架构图。

图3.7集群管理单元的总体架构图

### **3.5.1 集群管理单元的总体功能**

如图3.8所示，集群管理单元包括服务管理、镜像管理、用户管理、应用管理、监控管理、资源配置管理等。



图3.8 集群管理单元总体功能图

**用户管理**：用户管理模块包括用户注册、用户登陆、修改用户信息、删除用户等功能。普通用户与管理员的登陆，采用同一个界面，根据用户角色的不同，跳转到不同的界面，请求到来后，处理之前，受限判断是否登陆过，如果没有登陆过，转到登陆页面，如果登录过，继续判断用户角色，根据用户角色不同，转向不同的处理，请求处理后，判断存储在session中的版本号，选择跳转到不同版本的页面。

**服务管理**：服务管理模块包括新建服务、删除服务、修改服务、查看服务日志、同型创建等功能。服务管理模块接受用户请求，进行一系列的安全认证，包括用户认证，权限认证，请求数据校验等。然后操作Kubernetes集群的服务实例。创建服务，根据用户提交的请求，生成Kubernetes原生API支持的yaml文件，提交并创建服务实例。删除服务调用Kubernetes的删除接口，删除相应的服务实例。修改服务功能提供服务实例升级、资源扩缩容、修改环境变量等。服务日志查看功能提供服务实例的日志查看，为了方便用户排查问题，该功能实现了实时日志查看、历史日志查看以及根据关键字过滤日志信息。同型创建功能能够依据现有服务实例，创建出相同的服务实例提供服务。

**镜像管理**：镜像管理模块包括上传镜像、下载镜像、删除镜像、镜像列表等功能。镜像管理模块是较为单一的模块，该模块主要面对管理员，管理本地镜像仓库Docker Registry。通过调用Docker Registry提供的API接口，实现镜像的增删改查。由于镜像上传和下载都是耗时费力的功能，为了防止过度上传，每次上传时都先校验MD5，一旦发现与服务器端的镜像相同，则可以实现“秒传”。

**应用管理**：应用管理模块负责管理平台所提供的视频处理任务。该模块供平台管理员使用，主要包括平台应用的增删改查，平台中所有的服务实例都对应到相应的视频处理应用。应用是平台所能提供的视频处理服务的描述，主要包括相应的镜像和描述信息等。

**资源配置管理**：资源配置模块提供实例创建过程中所占用的资源配置，包括计算资源、内存资源等。该模块供平台管理员使用，提供资源配置类型的增删改查。普通用户在创建服务实例时，只能查看并选择合适的资源配置信息，为服务实例申请运行所计算和内存资源等。

### **3.5.2 RESTful接口设计**

为了与视频监控系统对接，集群管理单元对外提供RESTful风格的访问接口，提供就集群管理单元的所有功能。接口的设计严格遵循RESTful规则。采用URL定位资源、用HTTP动词(GET、POST、DELETE、PUT等)描述操作。数据采用json形式传输。图3.9展示了部分接口设计描述信息。

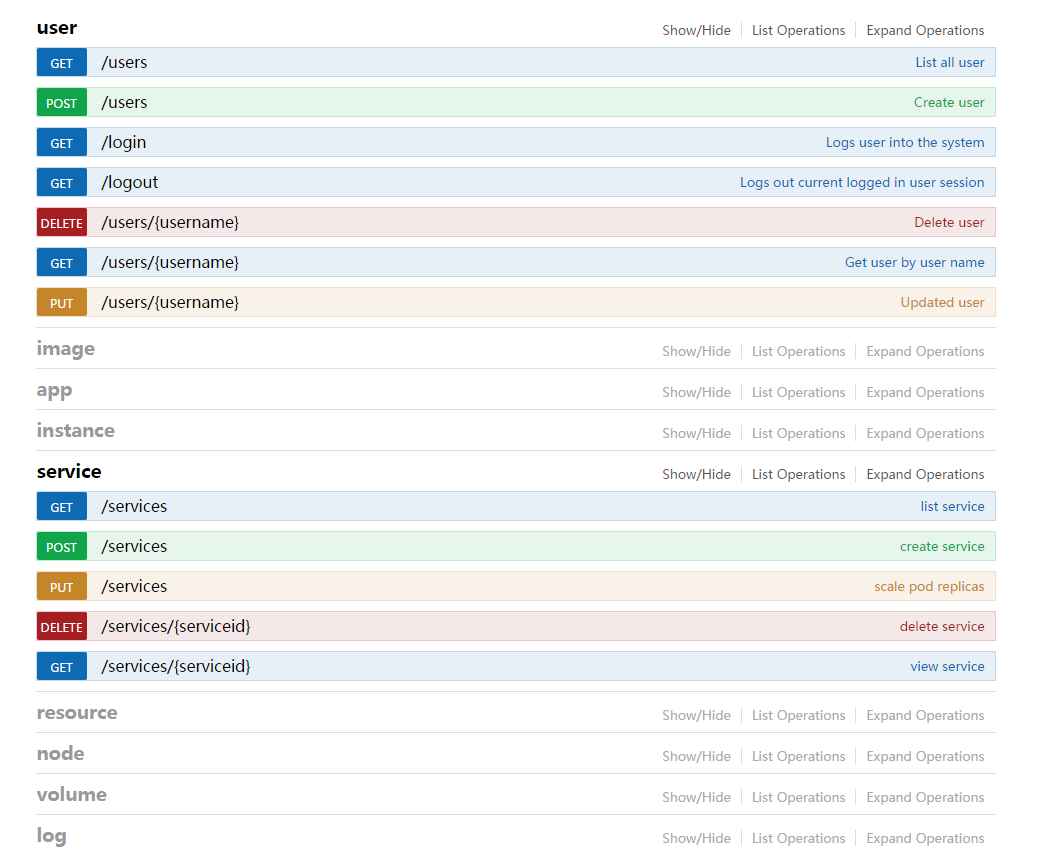


图3.9 集群管理单元RESTful接口设计

### **3.5.3 集群管理单元的开发**

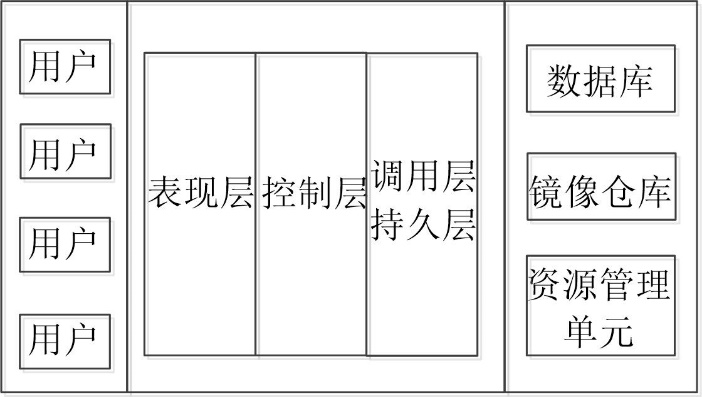
如图3.10所示，该系统的开发采用MVC(Model View Controller)模式，Model层是应用程序中用于处理应用程序数据逻辑的部分，在管理系统中，该层负责在数据库中存储数据，以及与Kubernetes API交互。View层是指用户看到并与之交互的页面，在管理系统中，该层为用户提供一个友好的可视化界面。Controller层处理应用程序中用户交互部分，在管理系统中，该层从View中读取数据，控制用户输入，并向Model层发送请求，Model层接收请求并进行相应的操作。在集群管理单元的开发中，Model层最终调用数据库、镜像仓库和资源管理单元提供的接口完成相应的操作。

图3.10集群管理单元的MVC视图

## **3.6 服务访问网关**

服务访问网管模块负责接收用户的请求，进行一系列的安全认证之后，将请求转发到合适的服务实例提供服务。

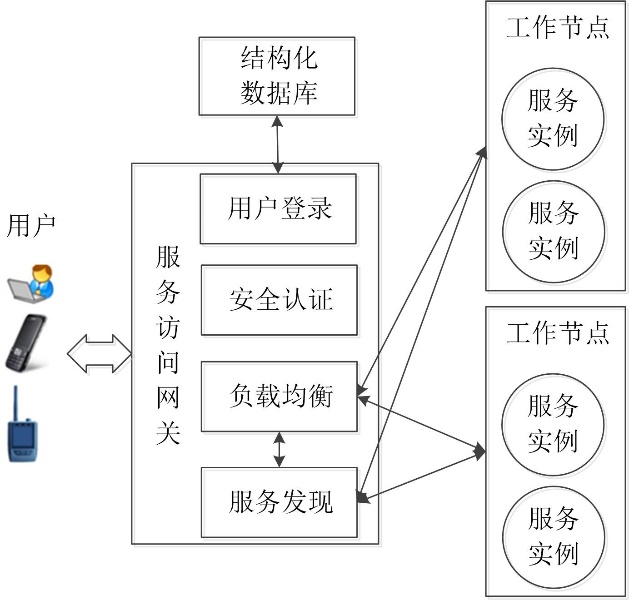


图3.12 服务访问网关

如图3.12所示，其主要功能包括用户登录、安全认证、负载均衡和服务发现。各个功能的具体实现如下：

**用户登录**：用户登录通过查询对比数据库中的用户信息，该用户信息通过3.6小节中集群资源管理单元提供的用户管理模块进行注册。

**安全认证**：安全认证功能主要校验用户提交的信息，本文采用的验证方法为MD5校验，在用户请求字段中添加MD5校验字段，服务端计算并对比MD5校验字段，从而确保请求的正确性。

**服务发现**: 服务发现通过调用资源管理单元提供的接口，实时的获取平台中创建的服务信息，包括服务实例的数量，服务实例的IP和端口信息等。为用户提供服务的查询功能。本文采用Kube-dns来提供服务发现功能，Kube-dns是Kubernetes官方提供服务发现插件，其部属方式类似与Docker镜像仓库，以Pod的形式部署，并对外提供服务。其部属的Yaml文件如下:

|  |
| --- |
| apiVersion: v1  kind: Pod  metadata:  labels:  k8s-app: kube-dns  name: kube-dns  namespace: kube-system  spec:  containers:  - args:  - --domain=cluster.local.  - --dns-port=10053  - --config-dir=/kube-dns-config  - --v=2  image: k8s.gcr.io/k8s-dns-kube-dns-amd64:1.14.8  name: kubedns  ports:  - containerPort: 10053  name: dns-local  protocol: UDP  - containerPort: 10053  name: dns-tcp-local  protocol: TCP  - containerPort: 10055  name: metrics  protocol: TCP  - args:  - -v=2  - -logtostderr  - -configDir=/etc/k8s/dns/dnsmasq-nanny  - -restartDnsmasq=true  - --  - -k  - --cache-size=1000  - --no-negcache  - --log-facility=-  - --server=/cluster.local/127.0.0.1#10053  - --server=/in-addr.arpa/127.0.0.1#10053  - --server=/ip6.arpa/127.0.0.1#10053  image: k8s.gcr.io/k8s-dns-dnsmasq-nanny-amd64:1.14.8  name: dnsmasq  ports:  - containerPort: 53  name: dns  protocol: UDP  - containerPort: 53  name: dns-tcp  protocol: TCP  - args:  - --v=2  - --logtostderr  - --probe=kubedns,127.0.0.1:10053,kubernetes.default.svc.cluster.local,5,SRV  - --probe=dnsmasq,127.0.0.1:53,kubernetes.default.svc.cluster.local,5,SRV  image: k8s.gcr.io/k8s-dns-sidecar-amd64:1.14.8  name: sidecar  ports:  - containerPort: 10054  name: metrics  protocol: TCP |

**负载均衡**：负载均衡接收用户请求，并转发请求到合适的服务实例。本文采用nginx实现负载均衡。

## **3.8 平台优化**

本文在平台搭建之后 ，对平台进行优化，主要从便捷性和稳定性两个方面优化。

### **3.8.1 平台稳定性优化**

平台搭建过程中涉及到的组件较多，组件容易故障停止运行。从而导致节点或者整个集群不能正常工作。为了降低组件意外停止对平台的影响，本文在搭建平台中，将所有相关组件加如操作系统Systemd管理，当组件意外停止时，能够自动重启组件。当物理机意外宕机或者重启时，相关组件也能做到开机自启动。

### **3.8.2 平台便捷性优化**

整个平台在安装部署过程中，涉及到十几台主机，共需要安装几十个组件。安装过成繁琐且容易出错，而且安装过程中对组件的安装顺序有一定的要求。手动部署集群需要花费大量的时间和精力，且对操作者有很高的门槛。在集群调试过程中，需要频繁的安装部署操作。为了降低安装部署的门槛，提高安装部署的效率，本文实现了一套自动化安装部署shell脚本，该脚本只需要配置好所有节点的IP地址和登陆用户名密码，能够一键拉起整个集群。由于整个安装过程繁琐复杂，为了减少脚本实行过程成中出现的不可预料的错误，该脚本在安装前会检测各个节点环境，包括操作系统版本、内核版本、内存和磁盘容量、以及已安装的冲突软件包。此外，为了方便排查安装部署中的问题，该脚本提供安装日志功能，能够快速的定位安装过程中出错的命令和出错的详细信息。

## **3.9 本章小结**

本章主要介绍了基于Docker技术的异构视频监控云平台设设计与实现，首先介绍了传统的视频监控系统，接着介绍了视频监控云平台的总体设计以及如何与传统的视频监控系统对接，以及平台的工作流程。然后详细的介绍了异构视频监控云平台中分布式资源管理单元和集群管理单元的的工作原理和详细的实现过程。最后，从平台的稳定性和便捷性两个方面对平台进行了优化。

# **第四章 异构视频监控云平台中任务调度策略**

为了提高平台的资源利用率，增加台的吞吐量，本节设计了预计深度强化学习的高效的平台任务调度器。

## **4.1 异构视频监控云平台上的视频处理任务**

本文关注异构CPU-GPU云计算平台中的视频处理任务，通过分析各种智能视频处理应用的结构，视频处理任务通常可以有各个更加细粒度的子任务Pipeline组成。每个视频子任务都是根据一个预先定义的工作流程执行的，其中有单独的输入和输出，并且子任务之间有固定的依赖关系。如图4.1所示，视频浓缩应用主要包括以下几个子任务：视频数据输入，GMM背景建模，三帧差分，通道提取，通道优化，拼接、浓缩视频等。如图4.2所示，入侵检测应用主要包括以下几个子任务：视频数据输入，GMM背景建模，三帧差分，预处理，高斯金字塔，DOG金字塔，尺度空间极值检测，提取特征向量，目标跟踪等。

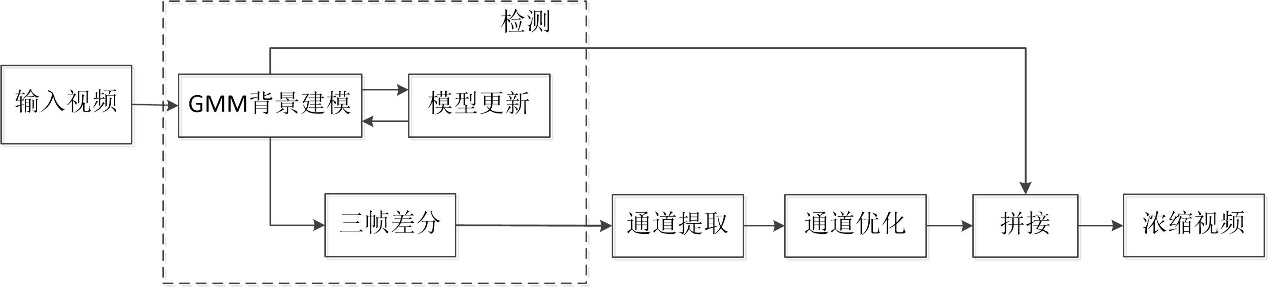


图4.1 视频浓缩摘要

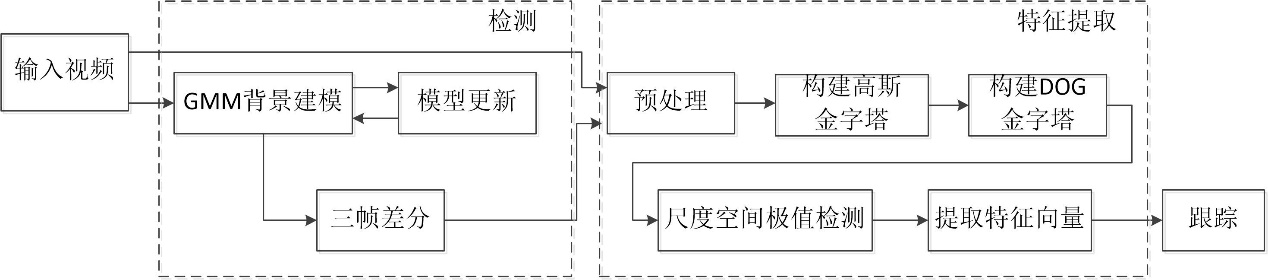


图4.2 目标跟踪

## **4.2 异构视频监控云平台调度问题描述**

本文定义视频处理任务为Ti，根据先前的工作[47]，我们发现视频处理任务的效率主要与视频处理任务的特点和视频数据特点有关，因此，本文采用（tID，tdres，tdsize）三元组代表视频处理任务，其中tID是任务类型的编号，tdres是视频数据的分辨率，tdsize视频数据的大小。更进一步，每一个视频处理任务可以表示为工作流图，其中，是图G的定点，是图G的边。被定义为，其中，是视频处理子任务的数量。每个有向边，代表着子任务依赖子任务。每个视频处理子任务属于，使用三元组(sID,tdres,tdsize)表示，其中，sID是子任务类型的编号。

假设平台有M个工作节点，分别为。每个节点在t时刻的状态使用向量表示，其中分别表示CPU利用率、GPU利用率、节点内存利用率、GPU显存利用率、节点上传带宽、节点下载带宽。假设每个节点有N个计算单元（包括CPU核和GPU），在每个节点S上，每个计算单元在t时刻的状态可以表示为：，其中，分别表示计算资源利用率，内存资源利用率，内存访问速率和数据迁移代价（数据在CPU内存和GPU内存之间的迁移代价）。

在异构视频监控云平台中，视频监控系统提交视频处理任务。调度器的目标是通过有效的并行化运行机制最大化平台的吞吐量。

## **4.3 现有的集群调度方法**

### **4.3.1 传统的集群调度算法概述**

传统的基于虚拟机的云资源调度器，以Openstack[48]为例，当前的调度算法为Filter Scheduler。它的调度主要分为条件过滤、权值计算和主机选择。

当前基于容器的主流调度策略，以Kubernetes为例，包括：

（1）平等优先调度策略: 所有节点同样优先级

（2）最少请求优先调度策略：计算Pods需要的CPU和内存在当前节点可用资源的百分比，具有最小百分比的节点就是最优，得分计算公式：cpu((capacity – sum(requested)) \* 10 / capacity) + memory((capacity – sum(requested)) \* 10 / capacity) / 2 。

（3）最小相同服务优先调度策略：对于属于同一个Service、Replication Controller的Pod，尽量分散在不同的主机上。

### **4.3.2 传统集群调度算法缺陷**

传统的集群调度算法中，大多是依据资源状态进行节点打分，然后依据贪心策略进行选择。这些算法只是简单的考虑平台资源的状态，而且贪心策略只是依据设定好的策略进行调度，没法根据平台的环境变化（比如访问量的变化）做出及时的调整。然而，异构视频监控云平台中，不仅要考虑CPU和内存的状态，还要考虑GPU和显存的状态。视频处理任务也各有特点，有的是IO密集型的，有的时计算密集型的。同时，每次提交过来的任务，对用的视频数据也大有不同，包括视频数据的大小，视频数据的分辨率等。此外视频处理任务还受网络，磁盘IO等因素的影响。传统的只考虑平台资源的任务调度算法无法满足异构视频监控云平台的需求。因此，本文提出了基于强化学习的两级任务调度策略。

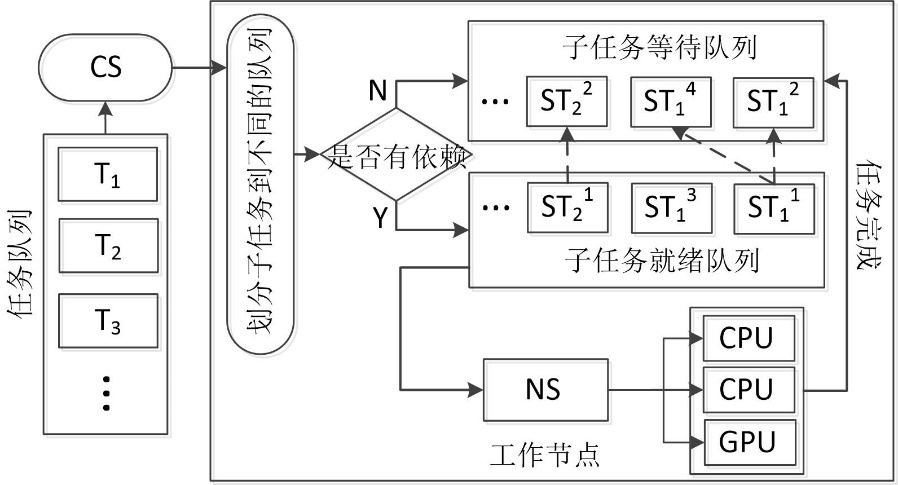
## **4.4 基于深度强化学习的调度策略**

可以看出，目前基于容器技术的资源调度方法没有考虑资源的物理特性和任务本身的特性，只是简单的根据资源数量的多少进行调度。并不能满足当前大规模异构视频监控云平台的需求。针对以上问题，本文提出一个异构智能视频监控云平台上的任务调度策略，综合考虑任务类型(计算密集型，IO密集型等)、数据类型(分辨率、像素、时长等)、资源异构性(CPU、GPU等)、数据传输时延等，实现一个高效的任务调度方法。以提高异构智能视频监控云平台的资源利用率，降低任务响应时延。

### **4.4.1 两级任务调度器的设计**

本文针对视频监控云平台中，资源利用率低、任务响应时延长、平台吞吐量低的问题，提出了一种基于深度强化学习的两级任务调度模型。如图4.3所示，第一级调度为集群级别的调度（CS），选择合适的物理节点处理视频任务pipeline；第二级调度为节点级别的调度（NS），为视频处理pipeline中的子任务选择合适的计算设备。

当视频处理任务pipeline被提交到集群中时，首先进入CS的等待队列中。CS每次从队列的头部取出视频处理任务，调度任务到合适的工作节点。任务到达工作节点之后，首先根据任务pipeline中子任务之间的依赖关系，将子任务划分到子任务等待队列和子任务就绪队列，将有依赖的子任务放入子任务等待队列，将没有依赖的子任务放到任务就绪队列。NS每次从任务就绪队列中调度子任务到合适的计算设备，待子任务完成之后，将子任务等待队列中依赖该任务的子任务移动到子任务就绪队列中。

图4.3两级调度模型

两阶段调度器CS和NS都基于深度强化学习，分别利用集群资源状态（各个节点CPU，GPU，内存，网络带宽等）、视频处理任务特点（任务类型，视频数据大小、分辨率）和节点计算状态(CPU、GPU、内存、显存等)、数据迁移代价等作为强化学习的环境，将集群的吞吐量作为奖励值，实时的训练和学习最优的策略，选择出合适的计算节点和计算设备完成任务。最终提升资源利用率，增加集群的吞吐量。

### **4.4.2 集群级别调度**

在调度的第一阶段，集群级别的调度器CS调度器为任务队列中队首的任务选择合适的工作节点。其工作机制描述如下：

**动作空间**：在集群级别的调度，需要为视频处理任务pipeline选择合适的工作节点。所以，在集群级别的调度过程中，其动作空间定义为所有的工作节点，即。

**状态空间**：强化学习中，每次选择动作时，依据当前的观测值。考虑到平台中视频处理任务的效率与视频任务和资源状态有关，CS调度器采用当前要处理的视频任务的特点，当前集群的资源状态作为当前的环境。因此，当前的观察值表示为：





状态空间表示为观察值和动作的序列：。

**奖励值**：CS调度器的目标是通过执行一系列的动作之后，最大化平台的吞吐量。在当前状态下，执行动作之后，平台将进化到新的状态，并且从环境中收到奖励值。该奖励值为执行动作给平台带来的吞吐量增量。表示为当前的集群吞吐量减去先前的集群吞吐量。

### **4.4.3 节点级别调度**

在调度的第二阶段，节点级别的调度器NS为子任务就绪队列中的子任务选择合适的计算单元。其工作机制描述如下：

**动作空间**：在节点级别的调度，需要为视频处理子任务pipeline选择合适的计算单元。所以，在节点级别的调度过程中，其动作空间定义为节点上的计算单元（包括CPU 核和GPU），即。

**状态空间**：强化学习中，每次选择动作时，依据当前的观测值。考虑到节点中视频处理任务的效率与视频任务和资源状态有关，NS调度器采用当前要处理的视频子任务的特点和各个计算单元的状态作为环境。因此，当前的观察值表示为。

**奖励值**：NS调度器的目标是通过执行一系列的动作之后，最大化节点的吞吐量。在当前状态下，执行动作之后，节点将进化到新的状态，并且从环境中收到奖励值。该奖励值为执行动作给节点带来的吞吐量增量。表示为当前的节点吞吐量减去先前的节点吞吐量。

## **4.5 调度算法实现**

### **4.5.1 算法描述**

现有的基于虚拟机或者容器的调度技术，仅仅考虑到集群中CPU和内存等的资源状态，调度过程中，基本没有考虑异构资源，也没有考虑到要调度的任务的特点。所以，当前的调度算法无法满足异构视频监控云平台的任务调度需求。本文采用基于深度强化学习的两级调度器，充分考虑到了资源的异构性，视频处理任务的特点，视频数据的特点。利用强化学习在连续决策中表现出来的优势，实时的学习最优的调度策略，调度任务到最合适的工作节点和计算单元。

本文采用深度强化学习中较成熟的深度Q-Learning（DQN），其算法的伪代码如算法4-1。DQN中存在两个结构完全相同但是参数却不同的网络，预测Q估计的网络MainNet使用的是最新的参数，而预测Q现实的神经网络TargetNet参数使用的却是很久之前的，Q(s,a;θi)Q(s,a;θi)表示当前网络MainNet的输出，用来评估当前状态动作对的值函数；Q(s,a;θ−i)Q(s,a;θi−) 表示TargetNet的输出，可以解出targetQ，因此当agent对环境采取动作a时就可以根据上述公式计算出Q并根据LossFunction更新MainNet的参数，每经过一定次数的迭代，将MainNet的参数复制给TargetNet。这样就完成了一次学习过程。

|  |
| --- |
| **Algorithm 4-1** 基于DQN的两阶段任务调度算法 |
| 1: Initialize replay memory;  2: Initialize action-value function  with random weights ;  3: Initialize target action-function  with random weights ;  4: for episode = 1, E do  5: Initialize state sequence ;  6: for t = 1, T do  7: With probability  select a random action ;  8: Otherwise select  action ;  9: Execute action  and observe next observation ;  reward ;  10: Store transition  in memory;  11: Sample random mini-batch of transitions  from memory;  12: =  13: Perform a gradient descent step on with the network parameters ;  14: Every  steps, close  to ;  15: end for  16: end for |

### **4.5.2 深度神经网络模型描述**

本文设计的深度Q-Learning采用具有两个隐藏层的多层感知机。该网络模型的输入为环境状态变量。其中，CS的输入为任务特点、集群资源的状态；NS的输入为子任务的特点、节点的资源状态。模型的输出是各个动作的期望值。其中，CS调度器的输出是选择集群中各个节点的期望值，NS调度器的输出是选择各个计算单元的期望值。两个隐藏层的神经元数分别是256和128。考虑到Relu函数具有简单，易收敛等特性，每个神经元均采用Relu函数作为激活函数。此外，模型采用随机梯度下降算法来优化网络模型，并且设置学习率为0.1，batch size的大小为20。

### **4.5.3 调度算法中使用的技巧**

为了结合深度学习和Q-Learning，提高模型的强化学习的准确性，本文在模型的训练过程中，采用了经典的经验回放[31]和双重DQN网络[49]。

经验回放：因为深度学习需要大量的样本，所以传统的中逐一对新样本学习的方式可能不太适合DQN。因此，训练的过程中需要增大样本量，并且进行多个epoch的训练，对样本进行反复训练。经验回放的思想就是存储Agent的样本，并且每次训练完时，随机的抽取一部分样本给网络学习。这样，模型就能够比较稳定的完成学习任务，避免只是短时的学习最新接触的样本，而是综合地、反复地利用过往的大量学习的样本进行学习。本文提出的两级调度模型在训练过程中，创建了一个大为30000的buffer，里面存储比较新的样本。当容量满了以后，会用新的样本替换最老的样本。每次训练时，直接从buffer中抽取batch size的样本给DQN训练。

双重DQN网络：强化学习的学习目标每次都是发生变化的，每次更新模型的参数都会倒是学习目标变化，如果模型的参数更新很频繁、幅度很大，训练的过程就会非常不稳定并且失控。为了让Q-Learning的训练目标保持平稳，本文使用了双重DQN网络：一个用来生成学习目标，称作目标网络；一个用来进行实际训练，称作主网络。首先在主网络上通过最大的Q值选择动作，再去获取这个动作在目标网络上的Q值。我们让目标网络进行缓慢的学习，这样它输出的目标Q值波动也会比较小，可以减小对训练的影响。

## **4.6 两级调度器的应用**

为了提高第三节中搭建的异构视频监控云平台的资源利用率，增加整个平台的吞吐量。我们将本节提出的两级调度策略应用到平台。

对于集群级别的调度器，本文利用Kubernetes提供的自定义调度器接口。在创建应用时，指明使用自定义调度器。Kubernetes对使用自定义调度器的任务，不做调度处理。然后，自定义调度器CS利用模型计算出最有的工作节点，再调用kubernetes提供的Binding类接口，指定任务到对应的工作节点。对于节点级别的调度，NS以服务的形式运行在各个节点之上，暴露相关的接口对外提供服务。每当节点上的视频任务处理pipeline中的子任务执行前，都会调用NS，以获得最优的执行单元来执行子任务。

## **4.7 本章小结**

本节首先分析易购视频监控云平台上视频处理任务的特点和现有的平台调度方法。指出了现有的调度方法不符合异构视频处理云平台的需求。进而提出了基于DQN的两级调度模型，该模型考虑到了平台的异构型，视频处理任务的特点，视频数据的特点，以及资源状态。能够实时的依据平台的变化，自动学习最有的调度策略。然后详细的介绍了两级调度方法的工作原理和算法流程。最终，将本节提出的调度器应用到异构视频监控云平台中。

# **第五章 平台功能及性能测试**

## **5.1 平台基础环境**

### **5.1.1 开发环境**

本文设计的视频监控云平台的开发环境为：

1. 操作系统：centos7 桌面版；
2. 程序开发环境：Sublime Text 3、shell、vim、g++、make、golang;
3. 计算机视觉库：OpenCV 2.4.9;
4. 其他软件库：gorilla。

### **5.1.2 集群环境**

集群环境为：

1. 集群硬件环境

本文设计并构建的视频监控云平台构建在14个物理节点之上，各个节点的配置见表5.1。为了保证集群的高可用和集群的稳定行，其中三个节点作为集群的管控节点，主机名分别为 master1、master2、master3，配置为16核Intel(R) Xeon(R) CPU E5620 @2.40GHz 和16G内存。剩下的11个节点为异构计算节点，主机名分别为work1~9,其中两个配置为：16核Intel(R) Xeon(R) CPU E5620 @2.40GHz，32G内存和12GB显存的GPU；4个配置为：16核Intel(R) Xeon(R) CPU E5620 @2.40GHz，32G内存和8GB显存的GPU；4个配置为：16核Intel(R) Xeon(R) CPU E5620 @2.40GHz，16G内存和6GB显存的GPU；2个配置为：16核Intel(R) Xeon(R) CPU E5620 @2.40GHz，8G内存和6GB显存的GPU。

表5.1 平台硬件环境

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 类型 | CPU数量 | 内存 | 显存 | GPU数量 | 个数 |
| １ | ２ | 32GB | 12GB | 1 | 3 |
| ２ | ２ | 32GB | 8GB | 1 | 5 |
| ３ | ２ | 16GB | 6GB | 1 | 4 |
| ４ | ２ | 8GB | 6GB | 1 | 2 |

1. 集群软件环境

平台构建在64位的Linux操作系统Ubuntu 14.04之上，内核版本为3.10.0-514.el7.x86\_64.所有的节点安装版本为1.11.2-cs3的Docker容器管理引擎。此外，本文采用Kubernetes v1.6.0作为资源管理单元，管控节点上部署Kubernetes的管控组件，工作节点上部署节点管理组件。此外，本文采用分布式数据库Etcd存储集群的运行状态，所使用的版本是3.1.12。为了保证平台的稳定行，本文在三个管控节点上部署Etcd集群。

### **5.1.3 视频监控数据**

实验的视频数据来源于真实的视频监控系统。我们选择的视频监控数据集包括2700视频数据文件。这些文件包括两种帧率，分别时25fps和30fps；包括3中分辨率，分别是L(640\*360)，M(1280\*720),和H(1920\*1080)。包括4中文件大小，分别是668s，1320s，1336s和2640s。

### **5.1.4 视频处理子任务实现**

为了测试平台的性能，我们实现了两个典型的视频处理任务：视频摘要和入侵检测。这两个子任务所包含的子任务和详细的处理流程已在4.1节中介绍。这些视频处理子任务中，一些已经在OpenCV库中存在，一些已经被我们实现。此外，为了适应子任务在异构CPU-GPU节点上运行，一些子任务实现了CPU和GPU两个版本，具体的情况如表5.2。

表5.2 相关视频处理子任务的CPU和GPU版本

|  |  |  |
| --- | --- | --- |
| 子任务 | CPU版本 | GPU版本 |
| GMM背景建模 | OpenCV | 已实现 |
| 帧间差分 | OpenCV | 已实现 |
| 高斯金字塔 | 已实现 | 已实现 |
| DOG金字塔 | 已实现 | 已实现 |
| 尺度空间极值检测 | 已实现 | 已实现 |
| 目标跟踪 | OpenCV | - |
| 通道提取 | 已实现 | - |
| 通道优化 | 已实现 | - |
| 拼接 | 已实现 | - |

## **5.2 异构视频监控云平台的功能测试**

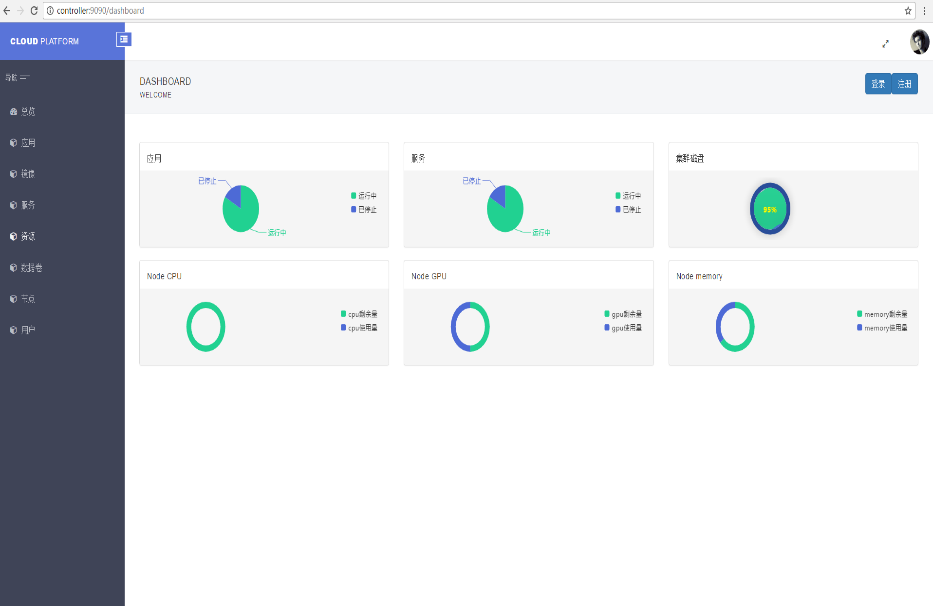
本文搭建基于Kubernetes的异构视频监控云平台，为用户提供了用户管理、应用管理、镜像管理、服务管理、资源配置管理等功能。平台搭建在5.1所描述的环境之上，各个功能模块全部开发完成。各个功能模块的web界面如下：

图5.1 集群管理单元可视化界面

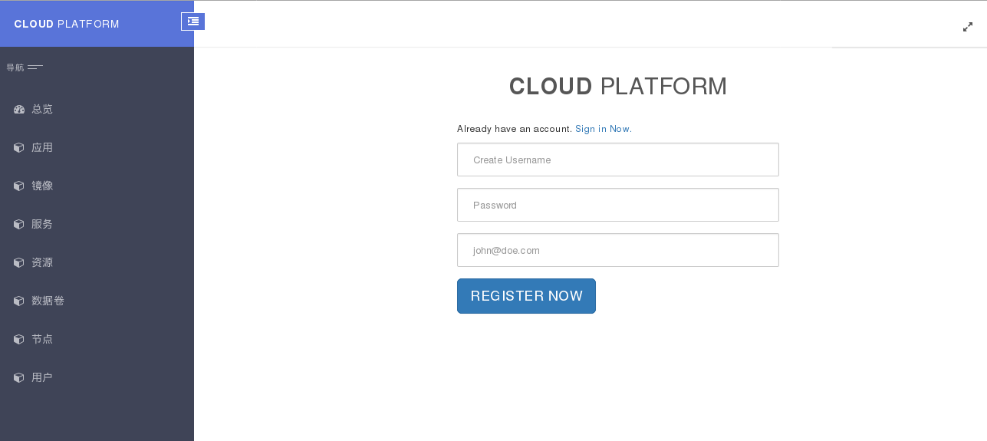
图5.1是平台的主界面，展示了平台的资源总体情况和平台中的服务运行情况。从主界面可以进入各个功能模块。

图5.2 用户管理界面

图5.2展示了用户管理界面，用户管理提供用户的注册、登录、信息修改等功能。

图5.3展示了镜像管理界面，镜像管理提供本地镜像和远程镜像查询、本地镜像删除等功能。

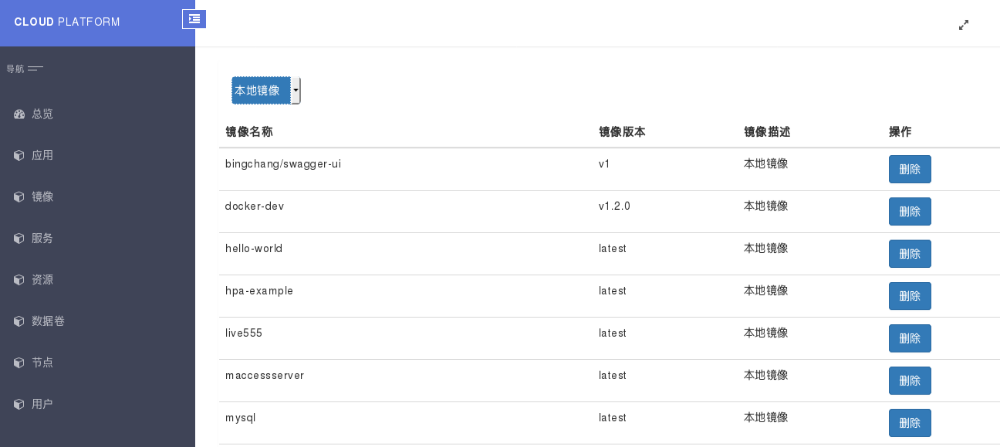


图5.3 镜像管理界面

图5.4 应用管理界面

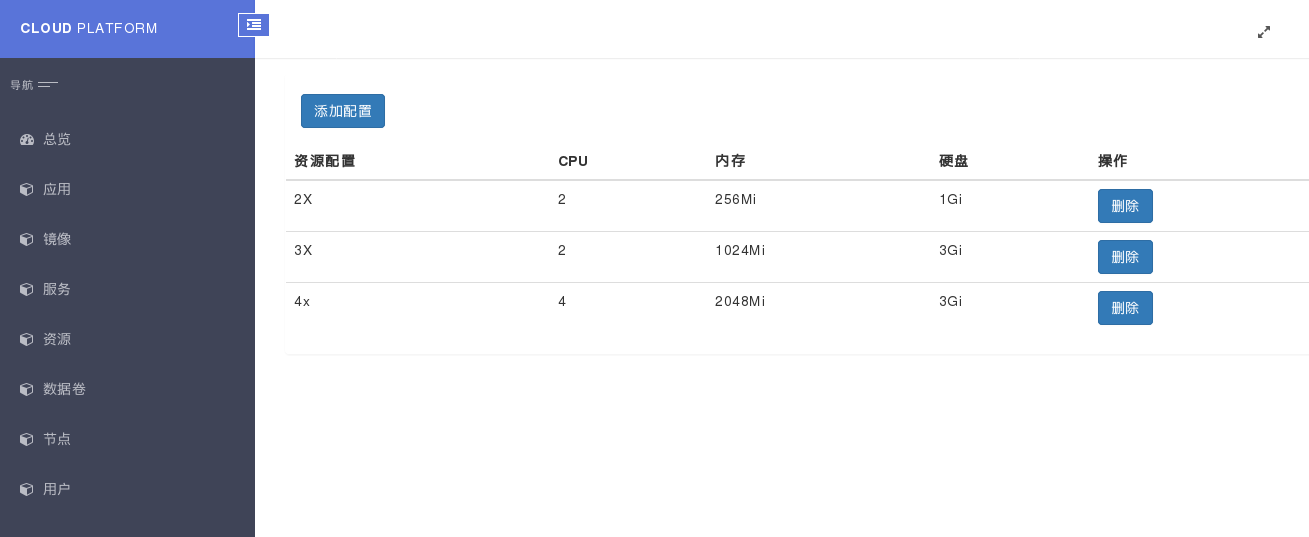
图5.4展示了应用管理界面，应用管理提供应用的创建、删除、详细信息的查看等。

图5.5 资源配置管理界面

图5.5展示了资源配置管理界面，用户根据资源配置，能够快速申请资源。该模块提供新建资源配置、删除资源配置等功能。

图5.6展示了服务管理界面，服务管理提供服务的创建、删除、更新、查看服务状态等功能。

图5.6 服务管理截图

最后，本文在平台中部署了流媒体服务。平台中部署的流媒体服务采用live555流媒体服务器，live555是开源的，目前已有现成的Docker镜像，为krystism/Live555。首先在视频监控云平台管理系统上的应用管理界面，以管理员的身份创建流媒体应用，该应用基于krystism/Live555流媒体服务镜像。然后在视频监控云平台管理系统的服务管理界面，创建3个流媒体服务实例。端口映射配置为容器端口的554映射到主机端口的554。接下来，使用VLC访问媒体数据，效果如图所示5.7。



图5.7 流媒体播放效果图

## **5.3 异构视频监控云平台的调度器性能测试**

### **5.3.1 模型参数重要参数的设置**

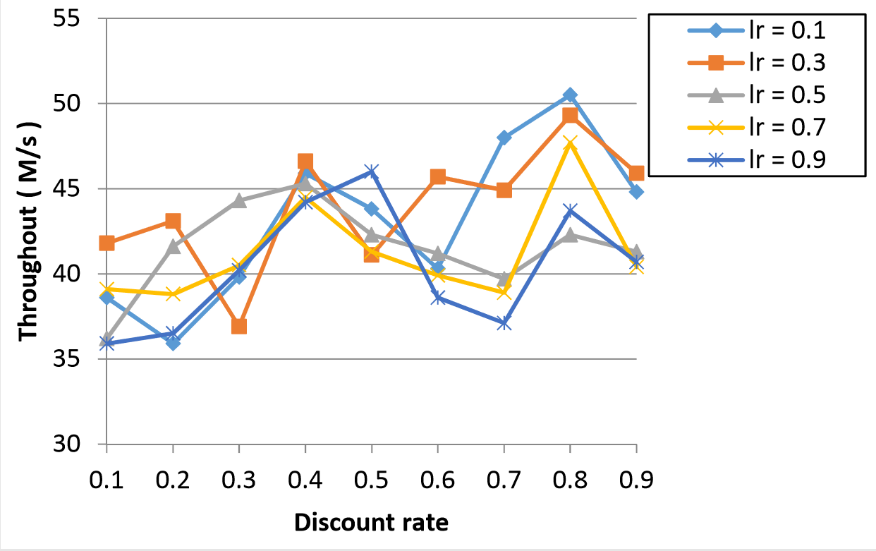
在本问提出的两级调度器中使用的DQN模型，有两个重要的参数会影响集群模型的整体性能：一个是学习率lr，一个是折扣因子r。为了选择合适的参数，我们设置了不同组参数的组合进行多次实验。使用目标检测和视频摘要视频处理任务pipeline来评估，保持平台其他参数不变。最后，我们计算平台的吞吐量。最终的实验结果如图5.8所示，最优的结果是学习率为0.1,折扣因子为0.8。因此，我们使用较小的学习率0.1来避免模型的发散，使用较大的折扣因子0.8来获取更多未来奖励。

图5.8 两个参数对调度器性能的影响

### **5.3.2 模型的收敛性**

收敛性是重要的迭代学习类算法的重要因素。为了评估本文提出的基于DQN的调度模型的收敛性，我们准备3000个入侵检测任务处理不同的视频数据。然后让3000个任务按照每分钟50个的到达率，提交到异构视频监控云平台。图5.9和5.10展示了奖励值随迭代次数的变化曲线。

如图5.9所示，在最开始的300次迭代过程中，集群节点的调度器CS随机的选择工作节点，因此，DQN模型的平均奖励值大概是10。当迭代次数在900次左右时，CS调度模型的奖励值开始快速上升，到迭代次数到达1400此左右时，CS调度器的奖励值逐渐平稳。与CS调度器类似，节点级别的调度器变化如图5.10所示，在最开始的400次迭代中，节点模型的奖励值都比较低，这是因为这段时间节点级别的调度器随机的选择计算单元。当跌代到800次左右的时间，NS模型的奖励值开始快速增长。当大约迭代到1200次时，NS调度模型的奖励值保持稳定。

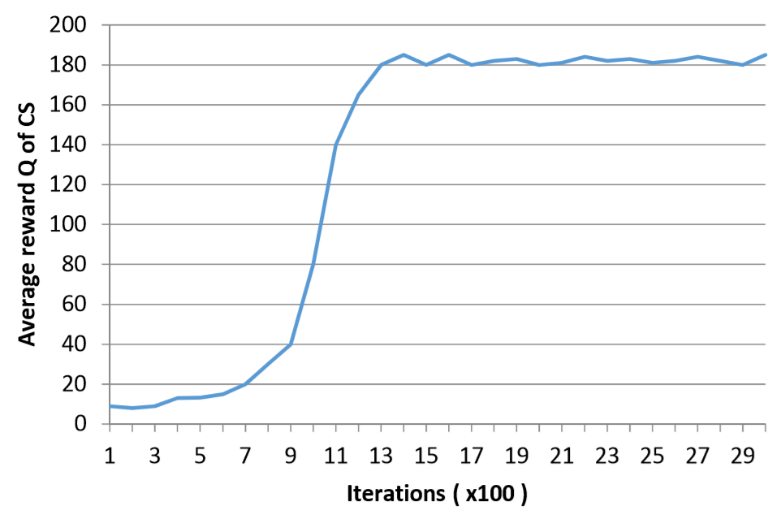
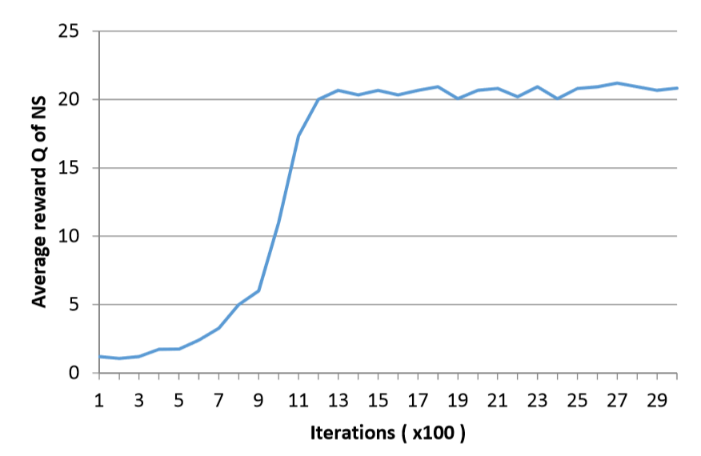
图5.9集群级别调度器的收敛性

图5.10节点级别调度器的收敛性

### **5.3.3 与其它方法的对比**

为了评估本王提出的两阶段调度方法，我们对比了我们的方法和以下几种方法：(1) 随机选择；(2)ECT[14]，该方法中，调度器评估任务在计算单元的最小完成时间，然后调度任务到具有最小完成时间的单元；(3)PATS[22]，该方法调度任务时评估计算单元的负载和性能，调度任务到具有最优性能的计算单元。

首先，我们研究了本文提出的调度器在不同规模的集群中的性能。在不同规模的异构视频监控云平台中，我们持续的提交1000个视频浓缩摘要任务。这1000个任务随机的处理视频监控文件，按照每分钟30个的速率提交大平台中。然后，我们计算整个集群的吞吐量和节点的吞吐量。如图5.11所示，4中调度方法对应的集群吞吐量，都随集群规模的扩大而增长。这是因为随着集群规模的增大，有更多的计算资源可以被使用。从图中可以看我们提出的方法明显由于其他方法。此外，随着集群的规模扩大，我们提出的方法对应集群吞吐量也增长，但是，但集群的工作节点扩大到9时，集群的吞吐量保持稳定。这是因为，任务的到达率一定，当集群的规模扩大到一定程度时，集群有足够的资源来分配使用，任务一旦到达，就能立即被使用。所以，平台的吞吐量几乎保持稳定。图5.12展示了本实验过程中node1节点的吞吐量变化。如图所示，我们提出的方法对应的节点吞吐量在集群规模为3到8个接电视相对稳定。但是，但节点增加到9之后，node1的节点吞吐量逐渐减弱。因为当集群增加大一定程度之后，集群中任务的处理速度大于任务的到达率，造成了节点的资源空闲。此外，从图中可以看出我们的方法明显优于其它方法。

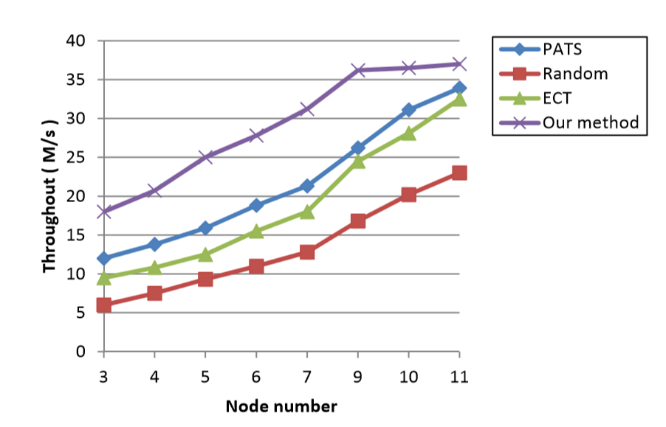
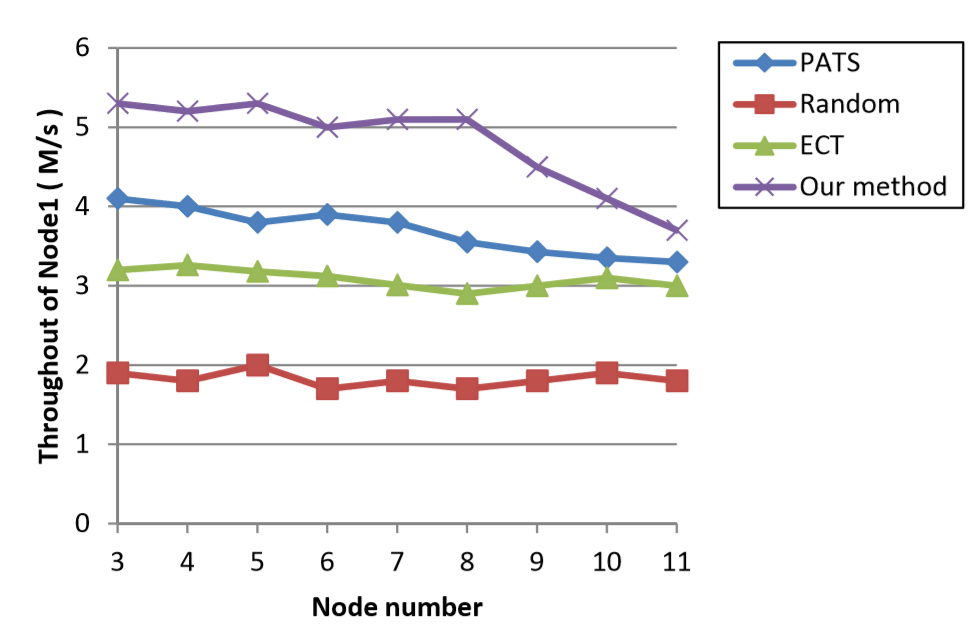


图5.11不同节点数量的集群吞吐量变化

图5.12不同节点数量Node1节点的吞吐量变化

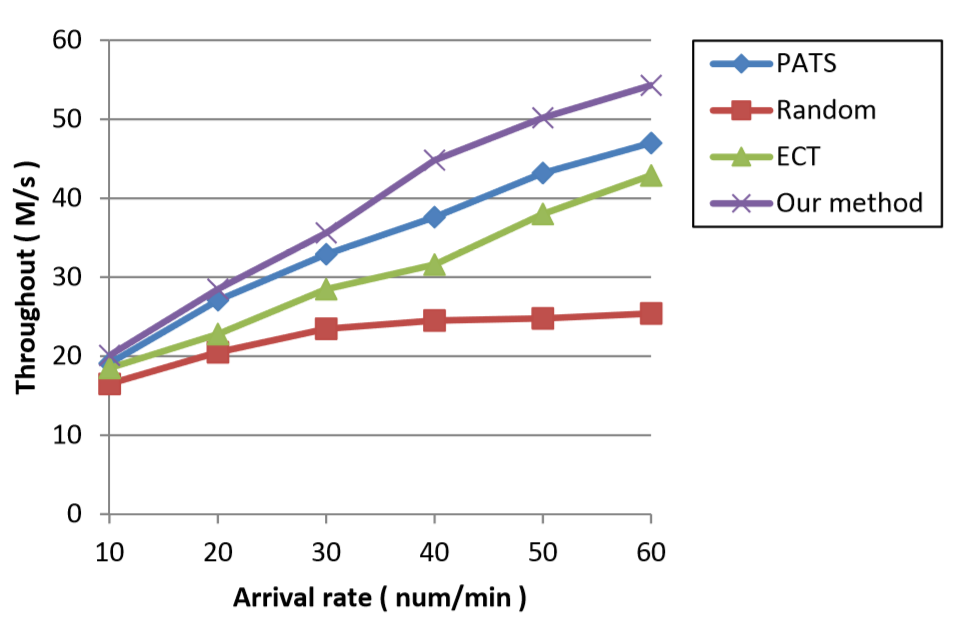
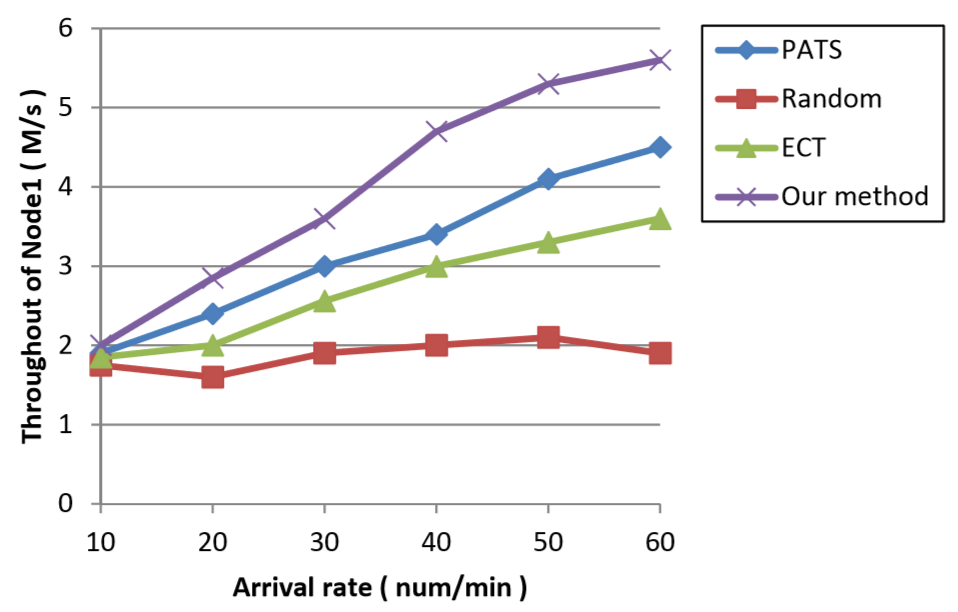
为了测试本文提出的方法在不同任务到达率下的效果，我们使用1000个视频摘要处理任务，每个任务处理不同的视频监控数据。由11个异构物理节点构建集群。在本次实验中，我们设置不同的任务到达率，分别为每分钟10个、20个、30个、40个、50个。然后使用相同的1000个任务测试。当所有的任务处理完成之后，我们计算集群的吞吐量和node1的节点吞吐量。图5.13展示了随着任务到达率的变化集群吞吐量的变化，从图中可以看出，当任务到达率为每分钟10个时，所有调度方法对应的集群吞吐量基本相同，这是因为任务到达率太小时，无论使用哪种调度方法，任务的处理速度都大于任务的到达速度，刚到来的任务能够被迅速处理。当任务到达率增加时，集群的吞吐量也随之增加，但是，我们提出的方法明显优于其他方法。图5.14展示了本实验中node1的吞吐量变化，如集群级别的吞吐量变化相似，本文提出的方法明显优于其它方法。

图5.13不同的任务到达率对应的集群吞吐量变化

图5.14不同的吞吐量对应的Node1吞吐量变化

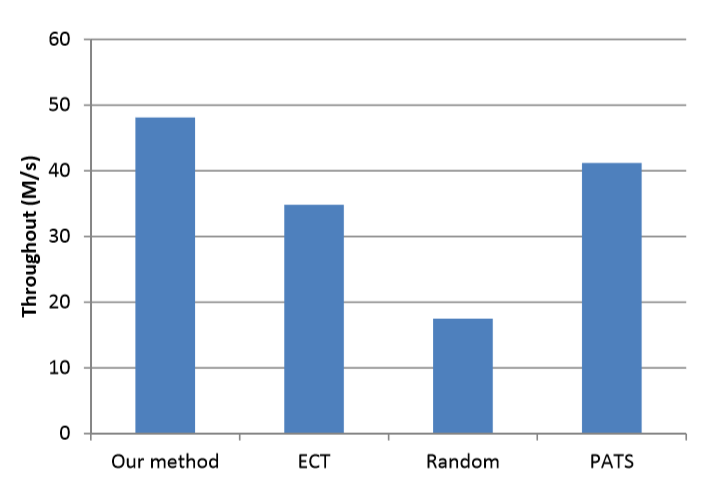
为了测试不同的视频处理任务pipeline对本文提出的调度器性能的影响。我们使用1000个视频处理任务，其中包括500个视频摘要任务和500个入侵检测任务。这50个任务按照每分钟50个的任务到达率提交到平台中。当任务执行完成之后，我们计算集群的吞吐量和node1的节点吞吐量。其中，图5.15展示了不同调度方法的集群吞吐量，图5.16展示了不同调度方法的节点吞吐量。与之前的实验相比，各个方法的性能均有所改变，这是因为不同的视频任务处理pipeline包含不同的视频处理子任务，对集群的吞吐量有一定的影响。但是，从图中可以看出，我们提出的方法仍有更能好的性能。

图5.15混合任务时不同的调度方法集群吞吐量对比

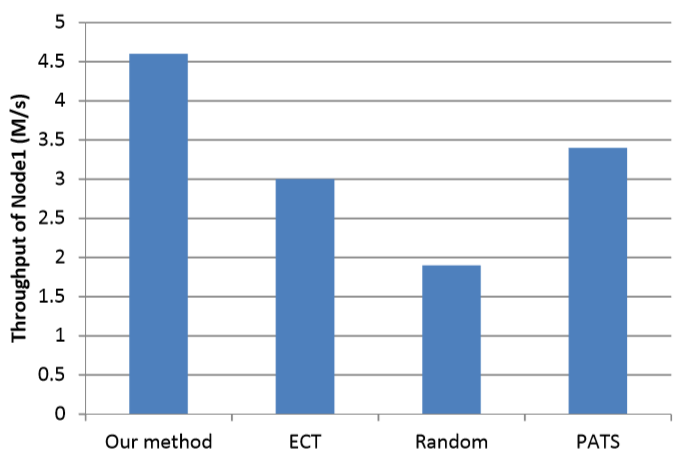


图5.16混合任务时不同的调度方法节点吞吐量对比

## **5.4 本章小结**

本章节首先介绍了平台搭建的环境，包括平台硬件环境、软件环境以及平台核心组件的开发环境。然后介绍了实验用到的两个视频处理算法的实现情况以及视频监控数据的特点。最后，从功能和性能上对平台进行测试，从不同的角度上对比了本文提出的方法与其它调度方法性能。

# **第六章 总结与展望**

## **6.1 总结**

近年来，随着智能视频监控技术的发展，越来越多的视频应用被迁移到云平台之上。随着视频监控数据的海量增长，传统的基于CPU同构的云计算平台逐渐不能满足快速处理视频监控数据的需求，基于CPU-GPU异构的云计算平台成为新的选择。然而，构建一个高效的异构视频监控云平台并不容易，至少需要解决一下三大问题：（1） 如何高效的管理平台资源;（2）如何与传统的视频监控系统对接；（3）如何设计高效的任务调度器，充分发挥异构平台的优势。针对以上三个问题，本文提出了基于Docker容器技术和kubernetes大规模容器集群管理技术的异构视频监控云平台，完成的工作主要包括了两大部分，每个部分围绕异构视频监控云台的难点展开。第一个部分围绕难点一和难点二，第二个部分围绕难点三。这两个部分包括：

1. 基于Kubernetes的异构视频监控云平台设计与实现

本文针对难点一和难点二，设计并实现了基于Kubernetes的异构视频监控云平台，该平台利用Docker容器快速启动、高效的资源利用率等特点来降低平台的额外开销，提高平台资源的利用率。利用Kubernetes在容器平台资源管理和监控运维方面的优势（包括自动化装箱、失败恢复、自动扩缩容、升级回滚、支持各种存储方案、支持GPU以及提供完善的接口）来整合平台资源统一管理，满足平台异构性的需求的同时，保证了平台的稳定性。同时，为了保证监控视频处理应用的安全和减少镜像传输的时间。该平台设计了本地私有镜像仓库，用来存储监控视频处理镜像。最后，为了方便与传统的视频监控系统对接，本文搭建的平台中设计了集群管理单元，该单元针对异构视频监控云平台，采用Golang语言开发，在提供可视化交互界面的同时，提供RESTful风格的接口供视频监控系统调用。

1. 基于深度强化学习的异构视频监控云平台的任务调度策略

本文针对难点三，提出了基于深度强化学习的两级任务调度策略CS-NS。CS为集群级别的调度器，在集群中选择合适的节点。NS为节点级别的调度器，选择合适的计算设备。利用深度强化学习在连续决策问题中表现出来的优势，可以在复杂的、不确定的环境中学习如何实现我们设定的目标的特点，来设计和实现平台的任务调度器。该调度策略综合考虑到了视频处理任务的特点、监控视频数据的特点以及资源的异构性等特点，能够实时的学习最优的调度策略，选择最合适的节点运行视频处理任务。最终，本文将设计的调度器应用到视频监控云平台中，利用视频浓缩摘要和入侵检测两个视频处理任务pipeline来验证本文提出的任务调度器的效率，实验表明，该调度器能够提高集群资源利用率，增加集群的吞吐量。

## **6.2 未来工作展望**

本文基于Docker和Kubernetes技术搭建的异构视频监控云平台，并基于深度强化学习设计了视频监控云平台两级任务调度器。显著提高了平台的资源利用率，增加了平台的吞吐量。为了方便与视频监控系统对接，本文设计并实现了异构视频监控云平台的管理单元，在提供可视化界面的同时对外暴露RESTful风格的访问接口。该平台在实验室有一定规模的部署，具有一定的实际应用性，但本文存在几个地方进一步研究和探索：

1. 进一步提高平台的稳定性

本文搭建的平台能够利用Kubernetes管控组件的失败重启机制来保证服务实例的稳定性，为用户提供稳定可靠的服务。然而，Kubernetes的管控组件本身是单实例部署，一旦Kubernetes管控组件异常终止或者管控节点宕机，集群的稳定性将无法保证，甚至造成整个集群无法提供服务。后续可以通过高可用方式部署Kubernetes管控组件，利用冗余机制和负载均衡相结合，来进一步提高集群的稳定性。

1. 高效的负载均衡策略

外部用户访问平台中的视频服务时，首先通过服务访问网关中的负载均衡组件Nginx选择合适的节点，然后经过节点上的Kube-Proxy提供的负载均衡策略来转发请求到合适的实例。Nginx和Kube-Proxy原生支持的负载均衡策略都比较简单，或是轮询，或者是最小连接等。并没有考虑视频处理应用的特点和视频处理数据的特点，视频服务实例间容易出现负载不均衡，从而造成资源的浪费。后续可以针对视频监控云平台，研究高效的负载均衡策略，进一步提高平台的资源利用率。

1. 合理的扩缩容策略

本文搭建的平台中，能够根据服务实例的资源使用情况，进行自动的服务实例扩缩容。该扩缩策略是Kubernetes原生支持的，其做法比较简单，通过设定资源使用率的上限和下限，当服务实例的资源使用率达到上限时进行扩容，当服务实例的使用率达到下限时进行缩容。在扩缩容得过程中，需要预先设定好服务实例个数得范围，在范围内进行扩缩容。然而，在异构视频监控云平台中，GPU是很重要得计算资源，Kubernetes原生得扩缩容策略，并没有考虑GPU的状态，容易倒是扩缩容得时机不当，从而造成资源得浪费，或是降低服务质量。后续可以针对异构视频监控云平台，提出合理得服务实例扩缩容策略，保证平台得服务质量。

# **参考文献**

1. Zhang H T, Xu B, Yan J, et al. “Proactive Data Placement for Surveillance Video Processing in Heterogeneous Cluster[C]”. [2016 IEEE International Conference on Cloud Computing Technology and Science](https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7823365), 2016, pp. 206–213.
2. Ilic A, Momcilovic S, Roma N, and Sousa L. “FEVES: Framework for Efficient Parallel Video Encoding on Heterogeneous Systems[C]”. [2014 43rd International Conference on Parallel Processing](https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6955014), 2014, pp. 20–29.
3. Fung J, and Mann S. “Using Graphics Devices in Reverse: GPU-based Image Processing and Computer Vision[C]”. [2008 IEEE International Conference on Multimedia and Expo](https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4599583), 2008, pp. 9–12.
4. 黄凯奇,陈晓棠,康运锋等. “智能视频监控技术综述[J]”.计算机学报, 2015, vol. 38, no. 6, pp. 1093-1118.
5. Hossain M S, Hassan M M, Qurishi M A, et al. “Resource allocation for service composition in cloud-based video surveillance platform[C]”. 2012 IEEE International Conference on Multimedia and Expo Workshops (ICMEW), 2012, pp. 408-412.
6. Yang C T, Huang K L, Chu W C, et al. “Implementation of video and medical image services in cloud[C]”. 2013 IEEE 37th Annual Computer Software and Applications Conference Workshops (COMPSACW), 2013, pp. 451-456.
7. Chaisiri S, Lee B S, and Niyato D. “Optimization of resource provisioning cost in cloud computing[C]” . IEEE Transactions on Services Computing, 2012, vol. 5, no. 2, pp. 164-177.
8. Neal D, and Rahman S M. “Video surveillance in the cloud-computing[C]”. 2012 7th International Conference on Electrical & Computer Engineering (ICECE), 2012, pp. 58-61.
9. Limna T, and Tandayya P. “Video surveillance as a service cost estimation and pricing model[C]”. 2015 12th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2015, pp. 174-179.
10. Ardagna D. “Cloud and multi-cloud computing: current challenges and future applications[C]”. the Seventh IEEE International Workshop on Principles of Engineering Service-Oriented and Cloud Systems, 2015, pp. 1-2.
11. A Vouk M. “Cloud computing–issues, research and implementations[C]”. CIT. Journal of Computing and Information Technology, 2008, vol. 16, no. 4, pp. 235-246.
12. Liu D, and Zhao L. “The research and implementation of cloud computing platform based on docker[C]”. 2014 IEEE 11th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP), 2014, pp. 475-478.
13. ITU-T H.626 standard, “Architectural requirements for visual of surveillance,” 2011.
14. Ciznicki M, Kurowski K, and Weglarz J. “Energy Aware Scheduling Model and Online Heuristics for Stencil Codes on Heterogeneous Computing Architectures[J]”. Cluster Computing, 2017, vol. 20, no. 3, pp. 2535–2549.
15. Fu D P, Xiong Y H, Lu C D, et al. “A Task Scheduling Method for Energy-Efficient Cloud Video Surveillance System Using A Time-Clustering-Based Genetic Algorithm[C]”. [2016 IEEE 22nd International Conference on Parallel and Distributed Systems](https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7822825), 2016, pp. 661–668.
16. Bleuse R, Hunold S, and Kedad-Sidhoum S, et al. “Scheduling Independent Moldable Tasks on Multi-Cores with GPUs[J]”. IEEE Transactions on Parallel and Distributed Systems, 2017, vol 28, no. 9, pp. 2689–2702.
17. Naithani A, Eyerman S and Eeckhout L. “ReliabilityAware Scheduling on Heterogeneous Multicore Processors[C]”. 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017, pp. 397–408.
18. Teodoro G, Pan T, Kurc T M, et al. “High-Throughput Analysis of Large Microscopy Image Datasets on CPU-GPU Cluster Platforms[C]”. [2013 IEEE 27th International Symposium on Parallel and Distributed Processing](https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6569024), 2013, pp. 103–114.
19. Liu J W, and Shen H Y. “Dependency-Aware and ResourceEfficient Scheduling for Heterogeneous Jobs in Clouds[C]”. 2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2016, pp. 110–117.
20. Hussin M, Lee Y C, and Zomaya A Y. “Efficient Energy Management using Adaptive Reinforcement Learning-based Scheduling in Large-Scale Distributed Systems[C]”. [International Conference on Parallel Processing](https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6046213), 2011, p. 385–393.
21. Zhang L X, Li K L, et al. “Maximizing Reliability with Energy Conservation for Parallel Task Scheduling in a Heterogeneous Cluster[J]”. Information Sciences, 2015, vol 319, no. 21, pp. 113–131.
22. Teodoro G, Kurc T M, Pan T, et al. “Accelerating Large Scale Image Analyses on Parallel, CPU-GPU Equipped Systems[C]”. [2012 IEEE 26th International Parallel and Distributed Processing Symposium](https://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6266782), 2012, pp. 1093–1104.
23. 浙江大学SEL实验室.“Docker：容器与容器云[M]”, 人民邮电出版社,2015
24. Bernstein D. “Containers and cloud: From lxc to docker to kubernetes[J]”. IEEE Cloud Computing, 2014, vol. 1, no. 3, pp. 81-84.
25. [龚正](https://www.amazon.cn/s/ref=dp_byline_sr_ebooks_1?ie=UTF8&text=%E9%BE%9A%E6%AD%A3&search-alias=digital-text&field-author=%E9%BE%9A%E6%AD%A3&sort=relevancerank). “Kubernetes权威指南：从Docker到Kubernetes实践全接触[M]”, 电子工业出版社, 2017.
26. Krizhevsky A, Sutskever I, and Hinton G E. “ImageNet Classification with Deep Convolutional Neural Networks[C]”. Advances in neural information processing systems , 2012, pp. 1097-1105.
27. Mnih V, Kavukcuoglu K, Silver D, et al. “Playing Atari with Deep Reinforcement Learning[C]”. arXiv preprint arXiv:1312.5602, 2013.
28. Silver D, Huang A, Maddison C J, et al. “Mastering the Game of Go with Deep Neural Networks and Tree Search[J]”. Nature, 2016, vol. 529, no. 7587, pp. 484–489.
29. Tong Z, Xiao Z, K. Li. “Proactive Scheduling in Distributed ComputingA Reinforcement Learning Approach[J]”. Elsevier Journal of Parallel and Distributed Computing, 2014, vol. 74, no. 7, pp. 2662–2672.
30. Cheng M X, Li J, and Nazarian S H. “DRL-Cloud: Deep Reinforcement Learning-based Resource Provisioning and Task Scheduling for Cloud Service Providers[C]”. Proceedings of the 23rd Asia and South Pacific Design Automation Conference. 2018, pp. 129–134.
31. Mao H Z, Alizadeh M, Menache I, and Kandula S. “Resource Management with Deep Reinforcement Learning[C]”. Proceedings of the 15th ACM Workshop on Hot Topics in Networks, 2016, pp. 50–56.
32. Zhang T, and Li J. “Online Task Scheduling for LiDAR Data Preprocessing on Hybrid GPU/CPU Devices: A Reinforcement Learning Approach[J]”. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 2015, vol. 9, no. 1, pp. 386–397.
33. Song X D, Peng X L, et al. “Cloud-based Distributed Image Coding[J]”. IEEE Transactions on Circuits and Systems for Video Technology, 2015, vol. 25, no. 12, pp. 1926– 1940.
34. Pereira R, Azambuja M, Breitman K, and Endler M. “An Architecture for Distributed High Performance Video Processing in the Cloud[C]”. 2010 IEEE 3rd international conference on cloud computing, 2010, pp. 482–489.H.
35. Zhang T, Yan J, and Kou Y. “Efficient Online Surveillance Video Processing Based on Spark Framework[C]”. International Conference on Big Data Computing and Communications, 2016, pp. 309–318.
36. Ma Y, Wang L Z, et al. “Task-Tree based Large-Scale Mosaicking for Massive Remote Sensed Imageries with Dynamic DAG Scheduling[J]”. IEEE Transactions on Parallel and Distributed Systems, 2014, vol. 25, no. 8, pp. 2126–2137.
37. Yang H B, Guo J H, et al. “An Optimization of the Delay Scheduling Algorithm for Real-Time Video Stream Processing[C]”. Frontier Computing, 2016, pp. 173–183.
38. Xiong Y H, Wan S Y,et al. “An Energy-Optimization-based Method of Task Scheduling for a Cloud Video Surveillance Center[J]”. Journal of Network and Computer Applications, 2016, vol. 59, pp. 63-73.
39. 王春光.“面向云服务的超算中心资源综合调度关键技术研究与实现[D]”, 国防科技大学,2015.
40. 冉泳屹. “云环境下基于随机优化的动态资源调度研究[D]”, 中国科学技术大学, 2014
41. Bacis E, Mutti S, Capelli S, et al. “DockerPolicyModules: Mandatory Access Control for Docker containers[C]”, 2015 IEEE Conference on Communications and Network Security (CNS), 2015, pp. 759-750.
42. [James Turnbull(著)](https://www.amazon.cn/s/ref=dp_byline_sr_book_1?ie=UTF8&field-author=%E8%A9%B9%E5%A7%86%E6%96%AF%C2%B7%E7%89%B9%E6%81%A9%E5%B8%83%E5%B0%94+%28James+Turnbull%29&search-alias=books), [李兆海](https://www.amazon.cn/s/ref=dp_byline_sr_ebooks_2?ie=UTF8&text=%E6%9D%8E%E5%85%86%E6%B5%B7+%E5%88%98%E6%96%8C+%E5%B7%A8%E9%9C%87&search-alias=digital-text&field-author=%E6%9D%8E%E5%85%86%E6%B5%B7+%E5%88%98%E6%96%8C+%E5%B7%A8%E9%9C%87&sort=relevancerank)(译),“第一本Docker书[M]”,人民邮电出版社,2016
43. Docker. [https://docs.docker.com/get-started/](https://docs.docker.com/get-started/（43)
44. Felter W, Ferreira A, Rajamony R, et al. “An updated performance comparison of virtual machines and linux containers[C]”. 2015 IEEE international symposium on performance analysis of systems and software (ISPASS), 2015, pp. 171-172.
45. Kubernetes. [https://kubernetes.io/docs/concepts/](https://kubernetes.io/docs/concepts/（45)
46. [黄文坚](https://www.amazon.cn/s/ref=dp_byline_sr_book_1?ie=UTF8&field-author=%E9%BB%84%E6%96%87%E5%9D%9A&search-alias=books), [唐源](https://www.amazon.cn/s/ref=dp_byline_sr_book_2?ie=UTF8&field-author=%E5%94%90%E6%BA%90&search-alias=books). “Tensorflow实战-黄文坚[M]”, 电子工业出版社, 2017
47. Gao Y Y, Zhang H T, et al. “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.2017, pp. 563–570.
48. Openstack. [https://docs.openstack.org/rocky/](https://docs.openstack.org/rocky/（48)
49. Van Hasselt H, Guez A, and Silver D. “Deep Reinforcement Learning with Double Q-learning[C]”. Thirtieth AAAI Conference on Artificial Intelligence, 2016.

# **攻读硕士学位期间的主要研究成果**

1. **发表论文4篇**
2. Zhang H, **Tang B**, Geng X, et al. “Learning Driven Parallelization for Large-Scale Video Workload in Hybrid CPU-GPU Cluster,” Proceedings of the 47th International Conference on Parallel Processing (ICPP), 2018. (CCF-B, 学生一作)
3. Gao Y, Zhang H, **Tang B**, et al. “Two-Stage Data Distribution for Distributed Surveillance Video Processing with Hybrid Storage Architecture”. 2017 IEEE 10th International Conference on Cloud Computing (CLOUD), 2017. pp. 616-623. (CCF-C，3作)
4. Gao Y, Zhang H, Zhu Y, **Tang B**, et al. “A Load-Aware Data Migration Scheme for Distributed Surveillance Video Processing with Hybrid Storage Architecture”. 2017 IEEE 19th International Conference on High Performance Computing and Communications (HPCC), 2017, pp. 563-570. (CCF-C，4作)
5. Zhang H, Yang N, Xu Z, **Tang B**, et al. “Microservice Based Video Cloud Platform with Performance-aware Service Path Selection,” 2018 IEEE International Conference on Web Services (ICWS), 2018, pp. 306-309. (EI，4作)
6. **申请专利和软著共7项**
7. 异构集群中视频任务并行化方法、装置及异构集群系统. 申请号：201810931427.1（2作）
8. 一种微服务路径选择方法及装置. 申请号：201810496770.8（4作）
9. 一种视频云服务平台架构及实现方法. 申请号：201811024237.8（4作）
10. 一种分布式计算平台中视频数据迁移方法及装置. 申请号：201710231948.1（5作）
11. 一种基于混合存储的计算系统中的数据迁移方法及装置. 申请号：201710501299.2（4作）
12. 一种分布式计算平台中视频数据分布方法及装置. 申请号：201711147816.7（4作）
13. 睿视容器云服务平台管理系统. (中国软件著作， 2作)

# **致谢**

岁月如梭如歌，转眼间三年的研究生求学生活就要结束。站在毕业的门槛上，回首往昔奋斗和辛劳成为丝丝的记忆，甜美与欢笑也都尘埃落定。北京邮电大学以其优良的学习风气、严谨的科研氛围教我求学，以其博大包容的情怀胸襟、浪漫充实的校园生活育我成人。在这离别之际，尽管依依不舍，却很珍惜。借此论文完成之际 我谨向所有关心、爱护、帮助我的人们表示最诚挚的感谢与最美好的祝愿。

在此，我要深深的感谢我的导师张海涛老师，感谢张老师这两年半的培养、教育和关怀。张老师严于律己、宽以待人的风范，认真负责、踏实勤奋的工作态度都潜移默化的影响着我。在研究生生活中，不仅教会了我做学问的技巧，还传授了做人的准则，这些必将使我受益终身。研究课题和毕设论文的完成，离不开导师的精心指导和帮助，在每个阶段里，张老师都耐心的审阅我的汇报，及时提出修改意见，保证了课题的顺利进行。在毕业论文阶段，张老师更是细心尽责，指导我克服困难与疑惑。衷心的感谢导师对我学习和生活上的关怀、支持和鼓励。

本论文的完成也离不开其他各位老师、同学和朋友的关心与帮助。在这样一个温馨、朝气蓬勃、团结友爱的大家庭里面，使我能够轻松快乐的学习和生活。感谢薛玉磊、杨宁同学，在我学习上遇到困难的时候积极与我交流、帮助我解决问题。感谢徐政钧师弟、耿欣师妹在论文完成过程中付出的努力。正是有了大家的共同努力，才营造了实验室良好的学习氛围，同时也带来了实验室的欢声笑语。

最后，感谢我的家人，家人是我最坚强的后盾，让我在硕士的三年里衣食无忧，专心科研。他们在生活和学业上给予我的鼓励和关心是我前进的最大动力。