(19)国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 119166364 A
(43)申请公布日 2024.12.20

(21)申请号 202411649328.6

(22)申请日 2024.11.19

(71)申请人 中国科学院空天信息创新研究院
地址 100190 北京市海淀区北四环西路19号

(72)发明人 孙小涓 桑冠南 高斌 胡玉新 姜芸珂 牟文浩 孙国庆

(74)专利代理机构 中科专利商标代理有限责任公司 11021
专利代理师 肖慧

(51)Int.Cl.
G06F 9/50(2006.01)
H04L 67/61(2022.01)
H04L 67/10(2022.01)

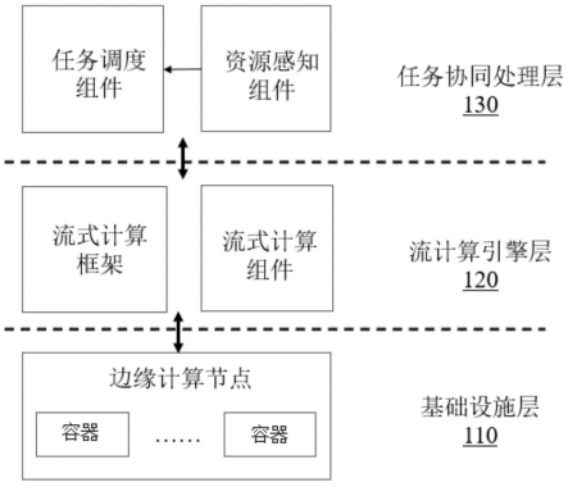
权利要求书1页 说明书13页 附图4页

(54)发明名称

视频流数据处理系统及方法

(57)摘要

本发明提供了一种视频流数据处理系统及方法,涉及边缘计算、云计算、流式计算技术领域。视频流数据处理系统包括:基础设施层,用于构建和管理边缘计算节点,所述边缘计算节点上部署有轻量化的容器集群;流计算引擎层,用于根据待处理任务,调用所述边缘计算节点上的容器集群以并行处理视频流数据;任务协同处理层,用于根据所述边缘计算节点在处理所述视频流数据时的资源利用率,调整所述待处理任务在所述容器集群上的分配。



CN 119166364 A

1.一种视频流数据处理系统,其特征在于,包括:

基础设施层,用于构建和管理边缘计算节点,所述边缘计算节点上部署有轻量化的容器集群;

流计算引擎层,用于根据待处理任务,调用所述边缘计算节点上的容器集群以并行处理视频流数据;

任务协同处理层,用于根据所述边缘计算节点在处理所述视频流数据时的资源利用率,调整所述待处理任务在所述容器集群上的分配。

2.根据权利要求1所述的视频流数据处理系统,其特征在于,所述边缘计算节点通过轻量化容器编排工具部署和管理所述容器集群,其中,所述轻量化容器编排工具采用轻量化数据库进行所述容器集群的数据存储,并通过二进制文件启动所述容器集群。

3.根据权利要求1或2所述的视频流数据处理系统,其特征在于,所述流计算引擎层包括流式计算组件和流式计算框架,其中,所述流式计算框架用于接收所述待处理任务,并通过与所述流式计算组件集成,调用所述容器集群以并行处理所述视频流数据。

4.根据权利要求3所述的视频流数据处理系统,其特征在于,所述流式计算框架包括作业管理器和至少两个并行的任务管理器,所述作业管理器用于对所述待处理任务进行调度和分配,所述任务管理器用于接收和执行所述作业管理器分配的所述待处理任务,并在执行所述待处理任务时与其他至少一个任务管理器之间进行数据交换。

5.根据权利要求1所述的视频流数据处理系统,其特征在于,所述任务协同处理层包括资源感知组件,其中,所述资源感知组件用于采集和分析所述边缘计算节点在处理所述视频流数据时的指标数据,基于所述指标数据获得所述资源利用率。

6.根据权利要求5所述的视频流数据处理系统,其特征在于,所述任务协同处理层还包括任务调度组件,其中,所述任务调度组件用于利用所述资源感知组件获取的所述资源利用率,基于预设的调度算法调整所述待处理任务在所述容器集群上的动态分配,以提高所述资源利用率。

7.根据权利要求6所述的视频流数据处理系统,其特征在于,所述调度算法包括流拓扑协同调度方法,其中,所述流拓扑协同调度方法根据所述资源利用率调整流拓扑的结构,以调整所述待处理任务在所述容器集群上的分配。

8.根据权利要求6所述的视频流数据处理系统,其特征在于,所述任务调度组件还用于调整所述容器集群的数量和容量,以实现所述容器集群的资源弹性伸缩。

9.根据权利要求8所述的视频流数据处理系统,其特征在于,所述任务调度组件通过水平拓展和/或垂直拓展的方式调整所述容器集群的数量和容量。

10.一种视频流数据处理方法,其特征在于,包括:

接收视频流数据和待处理任务;

根据所述待处理任务,调用边缘计算节点上的容器集群并行处理视频流数据;

根据所述边缘计算节点处理所述视频流数据时的资源利用率,调整所述待处理任务在所述容器集群上的分配。

视频流数据处理系统及方法

技术领域

[0001] 本发明涉及边缘计算、云计算、流式计算等技术领域,具体涉及一种视频流数据处理系统及方法。

背景技术

[0002] 在当今数字化时代,视频流应用场景的重要性日益突显。以遥感数据视频流为例,随着航空航天领域的快速发展,遥感卫星数量愈来愈多,无人机遥感技术也蓬勃发展,海量的遥感图像视频流正源源不断地产生。在资源、气象、海洋、环境、国防等方面具有广泛的应用场景。这些视频流应用场景通常对数据处理实时性要求高,现有的视频流处理方式,其计算和存储资源分散,难以应对实时数据处理的挑战,因而阻碍了其发展。

发明内容

[0003] 有鉴于此,本发明提供了一种视频流数据处理系统及方法,以至少部分地解决上述问题。

[0004] 本发明一方面提供一种视频流数据处理系统,包括:基础设施层,用于构建和管理边缘计算节点,边缘计算节点上部署有轻量化的容器集群;流计算引擎层,用于根据待处理任务,调用边缘计算节点上的容器集群以并行处理视频流数据;任务协同处理层,用于根据边缘计算节点在处理视频流数据时的资源利用率,调整待处理任务在容器集群上的分配。

[0005] 根据本发明的实施例,边缘计算节点通过轻量化容器编排工具部署和管理容器集群,其中,轻量化容器编排工具采用轻量化数据库进行容器集群的数据存储,并通过二进制文件启动容器集群。

[0006] 根据本发明的实施例,流计算引擎层包括流式计算组件和流式计算框架,其中,流式计算框架用于接收待处理任务,并通过与流式计算组件集成,调用容器集群以并行处理视频流数据。

[0007] 根据本发明的实施例,流式计算框架包括作业管理器和至少两个并行的任务管理器,作业管理器用于对待处理任务进行调度和分配,任务管理器用于接收和执行作业管理器分配的待处理任务,并在执行待处理任务时与其他至少一个任务管理器之间进行数据交换。

[0008] 根据本发明的实施例,任务协同处理层包括资源感知组件,其中,资源感知组件用于采集和分析边缘计算节点在处理视频流数据时的指标数据,基于指标数据获得资源利用率。

[0009] 根据本发明的实施例,任务协同处理层还包括任务调度组件,其中,任务调度组件用于利用资源感知组件获取的资源利用率,基于预设的调度算法调整待处理任务在容器集群上的动态分配,以提高资源利用率。

[0010] 根据本发明的实施例,调度算法包括流拓扑协同调度方法,其中,流拓扑协同调度方法根据资源利用率调整流拓扑的结构,以调整待处理任务在容器集群上的分配。

[0011] 根据本发明的实施例,任务调度组件还用于调整容器集群的数量和容量,以实现容器集群的资源弹性伸缩。

[0012] 根据本发明的实施例,任务调度组件通过水平拓展和/或垂直拓展的方式调整容器集群的数量和容量。

[0013] 本发明另一方面提供一种视频流数据处理方法,包括:接收视频流数据和待处理任务;根据待处理任务,调用边缘计算节点上的容器集群并行处理视频流数据;根据边缘计算节点处理视频流数据时的资源利用率,调整待处理任务在容器集群上的分配。

[0014] 根据本发明实施例的视频流数据处理系统及方法,通过流计算引擎层调用边缘计算节点上的轻量化容器集群以并行方式处理视频流数据,能极大提高数据处理的速度和效率。并行处理使得大量数据可以同时被多个计算资源处理,减少了整体处理时间。利用轻量化容器技术,通过将应用程序及其依赖项打包成轻量级容器,实现了在边缘设备上的快速部署和灵活扩展,最大限度地利用了边缘设备的资源,提高了系统的灵活性和可扩展性。与此同时,资源感知的任务调度机制能够实时监测边缘设备的资源利用情况,并根据资源的动态变化进行任务调度和资源分配,可以在边缘环境中实现有效的分布式任务处理,应对资源的动态变化,提高系统的任务执行效率和资源利用率。

附图说明

[0015] 通过以下参照附图对本发明实施例的描述,本发明的上述以及其他目的、特征和优点将更为清楚,在附图中:

[0016] 图1示意性示出了根据本发明实施例的视频流数据处理系统的结构图;

[0017] 图2示意性示出了一种可用于本发明实施例的视频流数据处理系统的K3s容器结构图;

[0018] 图3示意性示出了一种可用于本发明实施例的视频流数据处理系统的Flink架构图;

[0019] 图4示意性示出了根据本发明实施例的视频流数据处理系统的处理流程图;

[0020] 图5示意性示出了根据本发明实施例的视频流数据处理系统的资源利用率曲线图;

[0021] 图6示意性示出了根据本发明实施例的视频流数据处理系统的并行处理性能图;

[0022] 图7示意性示出了根据本发明实施例的视频流数据处理方法的流程图;

[0023] 图8示意性示出了根据本发明实施例的适于实现上文描述的方法的电子设备的框图。

具体实施方式

[0024] 以下,将参照附图来描述本发明的实施例。但是应该理解,这些描述只是示例性的,而并非要限制本发明的范围。在下面的详细描述中,为便于解释,阐述了许多具体的细节以提供对本发明实施例的全面理解。然而,明显地,一个或多个实施例在没有这些具体细节的情况下也可以被实施。此外,在以下说明中,省略了对公知结构和技术的描述,以避免不必要地混淆本发明的概念。

[0025] 在此使用的术语仅仅是为了描述具体实施例,而并非意在限制本发明。在此使用

的术语“包括”、“包含”等表明了特征、步骤、操作和/或部件的存在,但是并不排除存在或添加一个或多个其他特征、步骤、操作或部件。

[0026] 在此使用的所有术语(包括技术和科学术语)具有本领域技术人员通常所理解的含义,除非另外定义。应注意,这里使用的术语应解释为具有与本说明书的上下文相一致的含义,而不应以理想化或过于刻板的方式来解释。

[0027] 在使用类似于“A、B和C等中至少一个”这样的表述的情况下,一般来说应该按照本领域技术人员通常理解该表述的含义来予以解释(例如,“具有A、B和C中至少一个的系统”应包括但不限于单独具有A、单独具有B、单独具有C、具有A和B、具有A和C、具有B和C、和/或具有A、B、C的系统等)。

[0028] 在实现本发明的过程中,申请人发现:边缘视频流实时处理技术至少存在以下难点:

[0029] 1.边缘计算资源有限,单一节点难以满足要求。视频流目标检测应用具有较高的处理实时性要求,而单一节点通常具有有限的计算、存储和网络资源,这使得在边缘计算环境中执行大规模数据处理任务变得困难。这种资源瓶颈导致了性能下降、任务延迟增加以及应用程序的可扩展性受限等问题。

[0030] 2.边缘网算资源的可变性,分布式处理难以成功。边缘计算环境中的资源具有高度的动态性和不确定性。边缘设备的上下线、网络带宽的波动、用户需求的变化等因素都会导致资源的动态变化。这种资源的不稳定性使得在边缘环境中实现有效的分布式任务处理变得非常困难。传统的分布式任务调度和执行模型往往无法适应边缘环境中资源的动态变化,导致任务调度效率低下、资源利用率不高、任务执行延迟较大等问题。

[0031] 针对以上问题,本发明提供了一种视频流数据处理系统及方法。该视频流数据处理系统及方法采用轻量化容器技术,实现资源共享池化,统一调度。通过将应用程序及其依赖项打包成轻量级容器,可以实现在边缘设备上的快速部署和灵活扩展。这种轻量化容器技术能够最大限度地利用边缘设备的资源,提高系统的灵活性和可扩展性。同时,该视频流数据处理系统及方法还通过基于流式计算框架的任务调度,实时监测边缘设备的资源利用情况,并根据资源的动态变化进行任务调度和资源分配。当边缘设备的资源发生变化时,系统能够快速制定调度方案,通过流拓扑重建、资源弹性伸缩和任务迁移恢复等措施,保证系统的性能和稳定性。这种资源感知的任务调度方法能够有效地应对边缘计算环境中资源可变性带来的挑战,提高系统的任务执行效率和资源利用率。通过将轻量化容器技术和流式计算的边缘应用任务调度相结合,能够有效地解决当前边缘计算环境中存在的资源限制和动态性问题,提高了应用程序的实时处理吞吐率和系统资源利用率,从而满足了大规模视频流处理应用场景对于高性能、低延迟的需求。下面结合具体的实施例进行详细介绍。

[0032] 图1示意性示出了根据本发明实施例的视频流数据处理系统的结构图。

[0033] 如图1所示,视频流数据处理系统可以包括基础设施层110、流计算引擎层120和任务协同处理层130。

[0034] 基础设施层110用于构建和管理边缘计算节点,边缘计算节点上部署有轻量化的容器集群。

[0035] 基础设施层110也可以称为云基础设施层,是整个系统的基础,提供了边缘计算环境所需的基础资源和服务,例如物理服务器、虚拟化平台、存储系统等。在基础设施层110

上,可以构建和管理边缘计算节点,以支持边缘计算任务的部署和执行。边缘计算节点可以是分布在边缘设备上的计算单元,负责接收、处理和传输本地产生的视频流数据。这些边缘计算节点可以位于距离数据源较近的位置,可以采用体积小、功耗低、性能适中的硬件设备,例如可以是专用的边缘服务器等。边缘设备具备一定的计算和存储能力,以处理来自终端设备的数据。

[0036] 在边缘计算节点上部署有轻量化的容器集群。轻量化的容器集群部署可以采用容器化技术,如Docker、Kubernetes等。容器化技术允许将应用程序及其依赖项打包成一个独立的容器,这些容器可以在任何支持该技术的环境中运行,从而实现了应用的快速部署和迁移。

[0037] 请继续参阅图1,流计算引擎层120用于根据待处理任务,调用边缘计算节点上的容器集群以并行处理视频流数据。

[0038] 流计算引擎层120是系统的核心部分,利用容器集群的并行处理能力,流计算引擎层能够同时处理多个视频流数据任务。流计算引擎层120可以提供高性能的流式数据处理能力,能够处理来自边缘设备的视频流,并实时进行分析、处理和响应。视频流数据例如可以是遥感视频流数据。待处理任务例如可以是视频流识别、目标检测等。

[0039] 任务协同处理层130用于根据边缘计算节点在处理视频流数据时的资源利用率,调整待处理任务在容器集群上的分配。

[0040] 通过上述实施例,视频流数据处理系统采用轻量化容器技术,可以实现资源共享池化,统一调度。这种轻量化容器技术能够最大限度地利用边缘设备的资源,提高系统的灵活性和可扩展性。在基于流式计算框架并行处理视频流数据时,能够通过任务协同处理层,实时监测边缘设备的资源利用情况,并根据资源的动态变化进行任务调度和资源分配。这种资源感知的任务调度方法能够有效地应对边缘计算环境中资源可变性带来的挑战,提高系统的任务执行效率和资源利用率,从而满足了大规模视频流处理应用场景对于高性能、低延迟的需求。

[0041] 在上述实施例的基础上,边缘计算节点可以通过轻量化容器编排工具部署和管理容器集群。轻量化容器编排工具采用轻量化数据库进行容器集群的数据存储,并通过二进制文件启动容器集群。

[0042] 作为示例,轻量化容器编排工具可以是K3s,其为边缘计算场景设计的轻量级Kubernetes发行版。图2示意性示出了一种可用于本发明实施例的视频流数据处理系统的K3s容器结构图。如图2所示,K3s主节点(Server)可以包括资源操作组件(API Server)、轻量化数据库(SQLite)、资源管理器(Controller Manager)以及调度模块(Scheduler),从节点(Agent)则包括管理容器(Pod)的代理组件(Kubelet)、组内通信组件(Flannel)以及管理服务访问入口的Kube Proxy。作为轻量化容器编排工具,K3s的优势可以体现在多个方面:首先,采用了轻量级的组件和容器运行时,例如SQLite替代传统的etcd,Flannel作为网络组件,大幅降低了资源消耗,适用于边缘设备和环境;其次,K3s保持与标准Kubernetes API和生态系统的兼容性,可以无缝迁移现有的应用程序和工作负载;最后,K3s提供了简化的部署流程,通过单个二进制文件快速启动整个集群,节省了部署时间,提高了边缘环境的响应速度和灵活性。在边缘场景中,K3s可用于各种应用场景,包括边缘计算、边缘数据中心和边缘AI等,为边缘环境提供高效、灵活的容器化平台。

[0043] 在上述实施例的基础上,流计算引擎层120可以包括流式计算组件和流式计算框架,其中,流式计算框架用于接收待处理任务,并通过与流式计算组件集成,调用容器集群以并行处理视频流数据。

[0044] 流式计算框架是系统中的核心组件,负责实现对实时数据的高效处理和分析。流式计算框架具有良好的容错性和可扩展性,能够适应不同规模和变化的数据负载。流式计算组件包括各种实时处理任务,例如视频解码、特征提取、目标检测等。这些组件根据实际应用需求定制,通过与流式计算框架的集成,可以实现对边缘视频流的实时处理和分析。每个流式计算组件都具有高度并行化和可伸缩性,能够充分利用系统的计算资源,实现高效的实时处理。

[0045] 在上述实施例的基础上,流式计算框架可以包括作业管理器和至少两个并行的任务管理器,作业管理器用于对待处理任务进行调度和分配,任务管理器用于接收和执行作业管理器分配的待处理任务,并在执行待处理任务时与其他至少一个任务管理器之间进行数据交换。

[0046] 作为示例,流式计算框架可以是基于Apache Flink的流式计算框架。图3示意性示出了一种可用于本发明实施例的视频流数据处理系统的Flink架构图。如图3所示,Flink架构可以由作业管理器和任务管理器组成。对于一个提交执行的作业,作业管理器负责管理调度,包括作业控制器、资源管理器和任务分发器;任务管理器负责执行任务处理数据,其可以有一个或多个,每个包含多个任务槽。可以注册槽位给资源管理器,执行作业控制器分配的任务,并与其他任务管理器交换数据。

[0047] 表1将Flink架构与Apache Storm、Spark、Samza等实时流数据处理框架进行了对比。

[0048] 表1

[0049]

功能/特性	Apache Storm	Apache Spark	Apache Samza	Apache Flink
数据处理模型	事件流处理	批处理、流处理	流处理	流处理、批处理
延迟	低延迟	较高延迟	低延迟	低延迟
并发性	高并发性	高并发性	适度并发性	高并发性,支持多种部署模式
吞吐量	吞吐量较低	吞吐量较高	一般	高吞吐量
可靠性	容易发生数据丢失,不支持精确一次处理	可靠性较高,支持精确一次处理	可靠性一般,只能保证至少一次处理语义	可靠性较高,支持精确一次处理

[0050] 其中Apache Storm是支持流计算的分布式数据处理框架,以事件流的形式进行数据处理分析。Storm框架在流数据处理上具有时延性和并发性两种显著优势,但是同时也包含吞吐量低、高延迟、响应缓慢、无法满足实时计算等缺点。Spark是基于MapReduce算法实

现的分布式框架,该框架在计算前按照时间间隔,以段的形式将数据切分为批处理作业进行计算。缺点是由于基于内存运算,数据量规模过大时会导致集群性能不稳定,实时性较差。Samza是基于发布订阅系统的流数据框架,消息队列系统更高层的抽象,在消息队列系统上以一种应用模式实现分布式流数据处理框架。但是只能作到一次处理保证,可能会导致数据出现重复问题。

[0051] 可见,Flink相较于其他三个框架,具有显著优势。无论是处理静态数据还是动态数据,Flink都能够进行有状态计算,并以流的形式处理任何数据。与此同时,Flink可以解决分布式框架在数据吞吐率方面的挑战,提供了可靠的数据保障,确保数据不会重复出现或丢失。它既能保持高吞吐率,又能实现低延迟处理,这在其他框架中难以同时实现。同时其与资源管理器(如YARN、Kubernates等)的紧密集成,可以同时满足数据规模大、实时性要求高、方便业务扩展、故障后恢复等具体要求。

[0052] 请继续参阅图1,在一些实施例中,任务协同处理层130可以包括资源感知组件,其中,资源感知组件用于采集和分析边缘计算节点在处理视频流数据时的指标数据,基于指标数据获得资源利用率。

[0053] 例如,可以通过资源感知组件监测边缘计算节点中CPU、内存、网络带宽等指标数据,然后对这些指标数据进行分析计算,得到边缘计算节点的资源利用率。资源感知组件可以通过定期采集和分析边缘节点的资源使用情况,提供实时的资源信息,帮助系统实现对资源的动态调配和优化。

[0054] 作为示例,在容器集群中,资源实时感知可以依赖于普罗米修斯(Prometheus)来抓取和存储指标数据。普罗米修斯作为一个开源的监控报警系统和时序列数据库(TSDB),适用于监控容器化环境,如上述K3s容器集群。以下是一个示例性的处理过程:

[0055] 首先,可以在每个容器集群节点上部署Node-Exporter作为守护进程集。每个节点n_i运行一个Node-Exporter实例。Node-Exporter采集的指标M_{n_i}为时间序列数据:

[0056] $$M_{n_i} = \{(t_j, v_j) | t_j \in T, v_j \in R\}$$

[0057] 其中t_j是时间戳,代表指标数据被采集的时间点。v_j是对应的指标数据,是实数类型,代表在t_j时间点的指标测量值。

[0058] Node-Exporter采集的指标包括但不限于:CPU,包括CPU使用率、各个核心的使用情况等;内存:包括内存使用量、可用量、交换空间使用量等;磁盘,包括磁盘使用量、/O读写速率等;网络,包括网络接口的接收和发送字节数、数据包数等;系统负载,包括1分钟、5分钟、15分钟等不同时间的平均负载等。这些指标可以为集群管理员提供丰富的系统性能数据。

[0059] 接着,通过HTTP服务将采集到的指标数据以文本格式暴露出来,系统按照配置定期从每个运行Node-Exporter的节点拉取监控数据D_{n_i},对于每个节点,普罗米修斯每隔Δt时间发送一个HTTP GET请求到节点n_i的/metrics路径:

[0060] $$D_{n_i}(t) = HTTP\_GET(n_i/metrics)$$

[0061] 然后,通过服务发现机制,普罗米修斯自动注册新加入的节点为监控目标,避免手动更新配置。在时间t集群中的节点集合为N(t),则普罗米修斯动态维护的目标列表为:

[0062] $$N(t) = \{n_1, n_2, n_3 \ldots, n_k\}$$

[0063] 请继续参阅图1,在一些实施例中,任务协同处理层130还可以包括任务调度组件,

其中,任务调度组件用于利用资源感知组件获取的资源利用率,基于预设的调度算法调整待处理任务在容器集群上的动态分配,以提高资源利用率。

[0064] 任务调度组件负责根据系统负载和资源感知信息,动态调度任务以优化系统性能。它通过预设的调度算法,实现对任务的动态分配和调度,确保系统能够充分利用可用资源,实现高吞吐率和低延迟的实时数据处理。任务调度组件与资源感知组件、流式计算框架等组件紧密协作,实现对系统的整体调控和优化。

[0065] 在一些实施例中,任务调度组件还用于调整容器集群的数量和容量,以实现容器集群的资源弹性伸缩。

[0066] 在一些实施例中,任务调度组件可以通过水平拓展和/或垂直拓展的方式调整容器集群的数量和容量。

[0067] 水平扩展可以通过增加或减少容器实例的数量来调整系统容量。例如,设当前容器实例数为C,目标实例数为C^*。控制器定期监控容器的CPU和内存利用率(CPU_util(t),Mem_util(t)),根据预定义规则计算目标副本数量:

[0068] $$C^* = f\left(\mathrm{CPU}_{\mathrm{util(t)}}, \mathrm{Mem}_{\mathrm{util(t)}}\right)$$

[0069] 然后,自动更新副本集对象:

[0070] $$\mathrm{C}(\mathrm{t}+\Delta\mathrm{t}) = C^*$$

[0071] 垂直扩展可以通过调整单个容器的资源配置来优化性能。例如,设当前容器资源配置为R,目标资源配置为R^*,目标资源配置计算公式为:

[0072] $$R^* = g\left(\mathrm{CPU}_{\mathrm{util(t)}}, \mathrm{Mem}_{\mathrm{util(t)}}\right)$$

[0073] 其中g是生成资源配置建议的函数。

[0074] 然后,调整容器资源配置使其符合目标:

[0075] $$\mathrm{R}(\mathrm{t}+\Delta\mathrm{t}) = R^*$$

[0076] 当资源配置更新后需要重启容器以应用新配置:

[0077] $$\mathrm{Restart}(\mathrm{C_i}) \text{ if } \mathrm{R_i} \neq \mathrm{R_i^*}$$

[0078] 通过上述操作,可以实现容器集群的资源弹性伸缩,确保系统在处理大量并发请求和优化单个容器性能的场景下都能保持最佳状态。

[0079] 在一些实施例中,预设的调度算法可以包括流拓扑协同调度方法,其中,流拓扑协同调度方法根据资源利用率调整流拓扑的结构,以调整待处理任务在容器集群上的分配。

[0080] 作为示例,流拓扑协同调度方法可以包括流拓扑构建和任务调度算法两部分:

[0081] (1)流拓扑构建

[0082] 首先,根据子任务需要的CPU、内存和K3s监控到的系统空闲CPU、内存对任务调度场景进行建模,可以建模为有向无环带权图G=(V,E)的数据结构。节点V代表子任务、边E代表依赖关系,权重w(e)表示数据传输量。其中节点属性(CPU,内存)表示为:

[0083] $$\mathrm{Attr}(\mathrm{v_i}) = (\mathrm{CPU_{v_i}},\ \mathrm{Mem_{v_i}})$$

[0084] 然后,使用改进的GraphSAGE框架进行任务特征分类,包括三个步骤:①采样邻居顶点:对每个节点v的邻居节点进行固定数量S的采样,少于S时用有放回抽样,多于S时用无放回抽样。②聚合邻居信息:选择LSTM作为聚合函数,先随机排序邻居,然后将其embedding作为LSTM输入;③生成顶点向量:通过k次聚合生成所有节点的特征向量,用于下游任务。公式可以表示为:

[0085] $h_v^{(k)} = LSTM(\{h_U^{(K-1)} | u \in Neigh(v)\})$

[0086] 其中$h_v^{(k)}$是第k次聚会后节点v的特征向量。

[0087] (2)任务调度算法

[0088] 首先,基于资源状态的流式数据处理任务分配算法,通过图注意力网络-优势演员评论家算法(GAT-A2C),实现给定一组任务和资源表示,调度方案给出子任务与计算节点之间的映射。

[0089] $\mathrm{GTA-A2C} \rightarrow \mathrm{Mapping(T,\ R)}$

[0090] 然后,将包含特征的任务和资源表示为图数据形式,便于图卷积网络对特征的提取,而后,使用A2C算法进行任务调度优化问题求解。为了实现A2C调度算法优化策略,要将任务调度的过程建立为由状态空间、动作空间和奖励组成的马尔可夫决策过程(Markov Decision Process, MDP):

[0091] $\mathrm{MDP} = (S, A, P, R, \gamma)$

[0092] 其中S表示问题的状态空间,A表示动作空间,P表示状态转移概率,被定义为$P(s_{t+1} | s_t, a_t)$,表示在已知做出动作a_t的情况下状态从s_t转换到s_{t+1}的概率。R表示奖励函数,γ表示折扣因子。

[0093] 通过上述关键步骤的设计,能够确保视频流数据处理系统在边缘计算环境中高效地进行任务调度和资源分配,应对资源的动态变化,提高系统的任务执行效率和资源利用率。同时,这些设计也为视频流数据处理系统的可扩展性和灵活性提供了坚实基础,使其能够适应不同场景和应用需求的变化。

[0094] 在上述实施例的基础上,视频流数据处理系统还可以进一步包括应用组件层。应用组件层可以包括具体的应用组件和业务逻辑,用于实现特定的边缘视频流处理应用。这些应用组件可以基于流计算引擎提供的功能,提供特定的应用服务,包括提供特定的业务功能和服务接口。以遥感数据流为例,应用服务可以包括基于遥感数据实时视频流的图像识别、目标检测等应用。这些应用服务通过与流式计算组件和任务调度服务的集成,实现对实时数据的分析和处理,为用户和其他外接系统提供高价值的应用功能和服务支持。

[0095] 基于上述多个实施例,本发明构建了一个视频流数据处理系统实例,以下结合图4对该视频流数据处理系统的处理流程进行说明。图4示意性示出了根据本发明实施例的视频流数据处理系统的处理流程图。

[0096] 如图4所示,在本实施例中,视频流数据处理系统包括容器云平台、集群管理终端、资源感知模块和任务调度模块等组成部分,容器云平台可以作为图1中的基础设施层110、集群管理终端可以作为流视频框架层120、资源感知模块和任务调度模块可以作为任务协同处理层130。整个系统的处理流程可以包括如下步骤:

[0097] 首先,在接收用户任务前,可以先对视频流数据处理系统进行初始化。例如可以对容器云平台中节点容器集群的初始化,检查边缘设备的可用资源,包括计算能力、存储空间和网络带宽等。同时,在任务协同处理层中加载作业调度算法插件和资源监控算法插件,以确保系统能够准确地监测和管理资源利用情况,为后续的任务调度和资源分配提供坚实基础。

[0098] 接着,接收用户提交的任务。用户通过界面或API上传应用并提交视频流数据处理任务到作业管理器。用户可以详细指定应用程序的具体需求,例如所需的计算资源、内存要

求等。作业管理器接收到任务后,将任务信息传递给后续流程,准备进行任务调度和资源分配。

[0099] 再接着,创建任务流拓扑。可以根据资源感知服务实时的资源感知结果,调用预先加载的调度算法,生成流拓扑。这个过程中,系统可以根据当前边缘设备的资源状况,动态调整流拓扑的结构,以最大程度地利用边缘设备的资源,并确保任务的顺利执行。在此过程中,系统还可以考虑任务之间的依赖关系和优先级,以确保整体系统的稳定性和性能。

[0100] 然后,对计算资源进行分配。基于生成的流拓扑,系统可以进行计算资源分配。系统可以综合考虑每个节点的角色和当前负载情况,为每个任务分配适当的计算资源、存储空间和网络带宽。在这个过程中,系统可以根据任务的性质和要求,灵活地调整资源分配策略,以最大化地提高系统的效率和性能。

[0101] 接着,启动应用。一旦资源分配完成,对应节点的任务管理器接收到应用的可执行程序,并启动应用。作业管理器负责管理应用的运行状态,监控任务的执行进度,并处理任何异常情况。应用启动后,任务会按照流拓扑的指定顺序开始执行,实时处理边缘设备上的视频流数据。

[0102] 最后,完成任务。当任务执行完成时,作业管理器向应用返回处理结果。同时,系统可以进行必要的清理工作,释放已使用的资源,并准备接收新的任务。这个阶段还可以涉及结果的汇总和存储,以便后续的数据分析和应用。

[0103] 实验数据对比

[0104] 以下,结合具体实施例来说明采用本发明的视频流数据处理系统。图5示意性示出了根据本发明实施例的视频流数据处理系统的资源利用率曲线图,图6示意性示出了根据本发明实施例的视频流数据处理系统的并行处理性能图。

[0105] 本具体实施例基于pytorch网络框架搭建仿真环境,对比了以K3s+Flink为基础的轻量化架构(本发明方案)和以K8s+Flink为基础的传统架构(对比方案)的最小系统配置要求,其中K3s的CPU、内存和存储需求均明显小于K8s,体现了本系统的轻量化优势。同时,通过设置CPU阈值,绘制了系统CPU使用率随时间变化曲线,验证了系统的弹性扩缩容与重调度能力,保障本系统能够进行高效的资源利用。最后,使用视频流数据测试本系统,通过记录不同并行度下的视频流处理时间与单位时间图像处理帧数,证明了本系统实现了基于Flink的高效并行流处理能力。以下为具体对比内容:

[0106] 仿真环境系统的底层硬件设施由4台x86_64的116c32G的CentOS7.8云主机,2台x86_64的80c250G的CentOS7.8物理机组成。这些物理设施上运行着轻量级的容器集群,作为系统的底层平台。在搭建系统的过程中,安装部署了应用容器引擎Docker、容器集群快速编排软件docker-compose、私有镜像仓库Harbor、共享存储 NFS、数据库MySQL组成的轻量容器集群。在集群上运行着系统的前后端业务容器、普罗米修斯Prometheus的监控相关的容器和目标检测应用的Flink任务容器和子任务容器以及调度算法容器。

[0107] 结合K3s和Flink在边缘计算场景下提供了显著优势。K3s的轻量级特性使得其适合于在边缘节点上部署Flink任务,能够实现数据的实时处理和分析。在资源受限的环境下,系统可以更好地利用资源,同时降低部署和维护成本。如下表所示,基于K3s实现的系统相比于K8s在CPU与内存上节约了50%,存储成本减少了约87.5%。

[0108] 表2

[0109]

	K3s+Flink 系统			K8s+Flink 系统		
	单节点	多节点		单节点	多节点	
		主节点	工作节点		主节点	工作节点
CPU(核)	1	1	1	2	2	1
内存(MB)	512	1024	512	2048	2048	1024
存储(GB)	1	1	1	8	8	4

[0110] 如图5所示,系统的CPU阈值设置为80%。当系统的CPU使用率达到这一阈值时,系统会自动触发弹性扩缩容与重调度机制。这意味着当前系统资源和任务的调度将被重新评估和分配,以确保系统资源的最佳利用和任务执行的高效性。实验结果显示,系统在达到CPU使用阈值后,能够迅速而有效地进行扩缩容和重调度,并且在多次扩缩容后系统CPU使用率稳定在60%左右。

[0111] 将图像处理任务与Flink结合,利用Flink强大的并行计算能力,可以实现对大规模图像数据的高效处理和分析。通过将数据加载和处理流程优化为批处理任务,显著提高了整体处理效率,特别是在处理大规模数据时。如图6所示,实验使用视频流数据集,并行度设置为1、2、4、8时,系统处理时间分别是337.5、163.8、80.0、41.7分,处理数量分别是0.75、1.70、3.04、5.51帧,接近线性扩展,拟合值约为0.9.表明随着并行度的增加,系统对视频流处理的时间显著减少、单位时间内处理图像数量明显增多,系统具有较好的并发数据处理性能。

[0112] 通过上述仿真实验,可以看出,系统在不同并行度下的性能表现,验证了系统具有高效的任务调度能力和轻量化部署的优势。该研究不仅展示了深度强化学习调度算法在复杂环境下的应用前景,还提供了在边缘计算场景下优化资源利用和提高处理效率的方法。

[0113] 综上,本发明利用轻量化容器技术,例如K3s轻量化容器,通过将应用程序及其依赖项打包成轻量级容器,实现了在边缘设备上的快速部署和灵活扩展,最大限度地利用了边缘设备的资源,提高了系统的灵活性和可扩展性。本发明提出了一种资源感知的任务调度机制,能够实时监测边缘设备的资源利用情况,并根据资源的动态变化进行任务调度和资源分配。这种方法可以在边缘环境中实现有效的分布式任务处理,应对资源的动态变化,提高了系统的任务执行效率和资源利用率。本发明将Flink与K3s融合使用,利用了Flink的强大流处理能力来处理海量实时数据,实现了边缘视频流的实时处理。Flink的并行处理机制可以将任务分解为多个子任务,并在不同的计算节点上并行执行,从而显著提高数据处理的效率和速度。

[0114] 基于上述视频流数据处理系统,本发明还提供一种视频流数据处理方法。图7示意性示出了根据本发明实施例的视频流数据处理方法的流程图。

[0115] 如图7所示,视频流数据处理方法可以包括步骤S710~步骤S730。

[0116] 在步骤S710,接收视频流数据和待处理任务。

[0117] 在步骤S720,根据待处理任务,调用边缘计算节点上的容器集群并行处理视频流数据。

[0118] 在步骤S730,根据边缘计算节点处理视频流数据时的资源利用率,调整待处理任务在容器集群上的分配。

[0119] 需要强调的是,上述视频流数据处理方法和上述视频流数据处理系统具有相同的技术特征和有益效果,此处不再赘述。

[0120] 本发明还提供了一种电子设备,图8示意性示出了根据本发明实施例的适于实现上文描述的方法的电子设备的框图。

[0121] 如图8所示,根据本发明实施例的电子设备800包括处理器801,其可以根据存储在只读存储器(ROM)802中的程序或者从存储部分808加载到随机访问存储器(RAM)803中的程序而执行各种适当的动作和处理。处理器801例如可以包括通用微处理器(例如CPU)、指令集处理器和/或相关芯片组和/或专用微处理器(例如,专用集成电路(ASIC)),等等。处理器801还可以包括被配置为缓存用途的板载存储器。处理器801可以包括被配置为执行根据本发明实施例的方法流程的不同动作的单一处理单元或者是多个处理单元。

[0122] 在RAM 803中,存储有电子设备800操作所需的各种程序和数据。处理器 801、ROM 802以及RAM 803通过总线804彼此相连。处理器801通过执行ROM 802和/或RAM 803中的程序来执行根据本发明实施例的方法流程的各种操作。需要注意,程序也可以存储在除ROM 802和RAM 803以外的一个或多个存储器中。处理器801也可以通过执行存储在一个或多个存储器中的程序来执行根据本发明实施例的方法流程的各种操作。

[0123] 根据本发明的实施例,电子设备800还可以包括输入/输出(I/O)接口805,输入/输出(I/O)接口805也连接至总线804。电子设备800还可以包括连接至输入/输出(I/O)接口805的以下部件中的一项或多项:包括键盘、鼠标等的输入部分806;包括诸如阴极射线管(CRT)、液晶显示器(LCD)等以及扬声器等的输出部分807;包括硬盘等的存储部分808;以及包括诸如LAN卡、调制解调器等的网络接口卡的通信部分809。通信部分809经由诸如因特网的网络执行通信处理。驱动器810也根据需要连接至输入/输出(I/O)接口805。可拆卸介质811,诸如磁盘、光盘、磁光盘、半导体存储器等等,根据需要安装在驱动器810上,以便于从其上读出的计算机程序根据需要被安装入存储部分808。

[0124] 根据本发明的实施例,根据本发明实施例的方法流程可以被实现为计算机软件程序。例如,本发明的实施例包括一种计算机程序产品,其包括承载在计算机可读存储介质上的计算机程序,该计算机程序包含被配置为执行流程图所示的方法的程序代码。在这样的实施例中,该计算机程序可以通过通信部分809从网络上被下载和安装,和/或从可拆卸介质811被安装。在该计算机程序被处理器801执行时,执行本发明实施例的系统中限定的上述功能。根据本发明的实施例,上文描述的系统、设备、装置、模块、单元等可以通过计算机程序模块来实现。

[0125] 本发明还提供了一种计算机可读存储介质,该计算机可读存储介质可以是上述实施例中描述的设备/装置/系统中所包含的;也可以是单独存在,而未装配入该设备/装置/系统中。上述计算机可读存储介质承载有一个或者多个程序,当上述一个或者多个程序被执行时,实现根据本发明实施例的方法。

[0126] 根据本发明的实施例,计算机可读存储介质可以是非易失性的计算机可读存储介质。例如可以包括但不限于:便携式计算机磁盘、硬盘、随机访问存储器(RAM)、只读存储器(ROM)、可擦式可编程只读存储器(EPROM或闪存)、便携式紧凑磁盘只读存储器(CD-ROM)、光

存储器件、磁存储器件,或者上述的任意合适的组合。在本发明中,计算机可读存储介质可以是任何包含或存储程序的有形介质,该程序可以被指令执行系统、装置或者器件使用或者与其结合使用。

[0127] 例如,根据本发明的实施例,计算机可读存储介质可以包括上文描述的ROM 802和/或RAM 803和/或ROM 802和RAM 803以外的一个或多个存储器。

[0128] 本发明的实施例还包括一种计算机程序产品,其包括计算机程序,该计算机程序包含被配置为执行本发明实施例所提供的方法的程序代码,当计算机程序产品在电子设备上运行时,该程序代码被配置为使电子设备实现本发明实施例所提供的训练方法以及检测方法。

[0129] 在该计算机程序被处理器801执行时,执行本发明实施例的系统/装置中限定的上述功能。根据本发明的实施例,上文描述的系统、装置、模块、单元等可以通过计算机程序模块来实现。

[0130] 在一种实施例中,该计算机程序可以依托于光存储器件、磁存储器件等有形存储介质。在另一种实施例中,该计算机程序也可以在网络介质上以信号的形式进行传输、分发,并通过通信部分809被下载和安装,和/或从可拆卸介质811被安装。该计算机程序包含的程序代码可以用任何适当的网络介质传输,包括但不限于:无线、有线等等,或者上述的任意合适的组合。

[0131] 根据本发明的实施例,可以以一种或多种程序设计语言的任意组合来编写被配置为执行本发明实施例提供的计算机程序的程序代码,具体地,可以利用高级过程和/或面向对象的编程语言、和/或汇编/机器语言来实施这些计算程序。程序设计语言包括但不限于诸如Java,C++,python,“C”语言或类似的程序设计语言。程序代码可以完全地在用户计算设备上执行、部分地在用户设备上执行、部分在远程计算设备上执行、或者完全在远程计算设备或服务器上执行。在涉及远程计算设备的情形中,远程计算设备可以通过任意种类的网络,包括局域网(LAN)或广域网(WAN),连接到用户计算设备,或者,可以连接到外部计算设备(例如利用因特网服务提供商来通过因特网连接)。

[0132] 附图中的流程图和框图,图示了按照本发明各种实施例的系统、方法和计算机程序产品的可能实现的体系架构、功能和操作。在这点上,流程图或框图中的每个方框可以代表一个模块、程序段、或代码的一部分,上述模块、程序段、或代码的一部分包含一个或多个被配置为实现规定的逻辑功能的可执行指令。也应当注意,在有些作为替换的实现中,方框中所标注的功能也可以以不同于附图中所标注的顺序发生。例如,两个接连地表示的方框实际上可以基本并行地执行,它们有时也可以按相反的顺序执行,这依所涉及的功能而定。也要注意的是,框图或流程图中的每个方框、以及框图或流程图中的方框的组合,可以用执行规定的功能或操作的专用的基于硬件的系统来实现,或者可以用专用硬件与计算机指令的组合来实现。本领域技术人员可以理解,本发明的各个实施例中记载的特征可以进行多种组合和/或结合,即使这样的组合或结合没有明确记载于本发明中。特别地,在不脱离本发明精神和教导的情况下,本发明的各个实施例中记载的特征可以进行多种组合和/或结合。所有这些组合和/或结合均落入本发明的范围。

[0133] 以上对本发明的实施例进行了描述。但是,这些实施例仅仅是为了说明的目的,而并非为了限制本发明的范围。尽管在以上分别描述了各实施例,但是这并不意味着各个实

施例中的措施不能有利地结合使用。不脱离本发明的范围,本领域技术人员可以做出多种替代和修改,这些替代和修改都应落在本发明的范围之内。

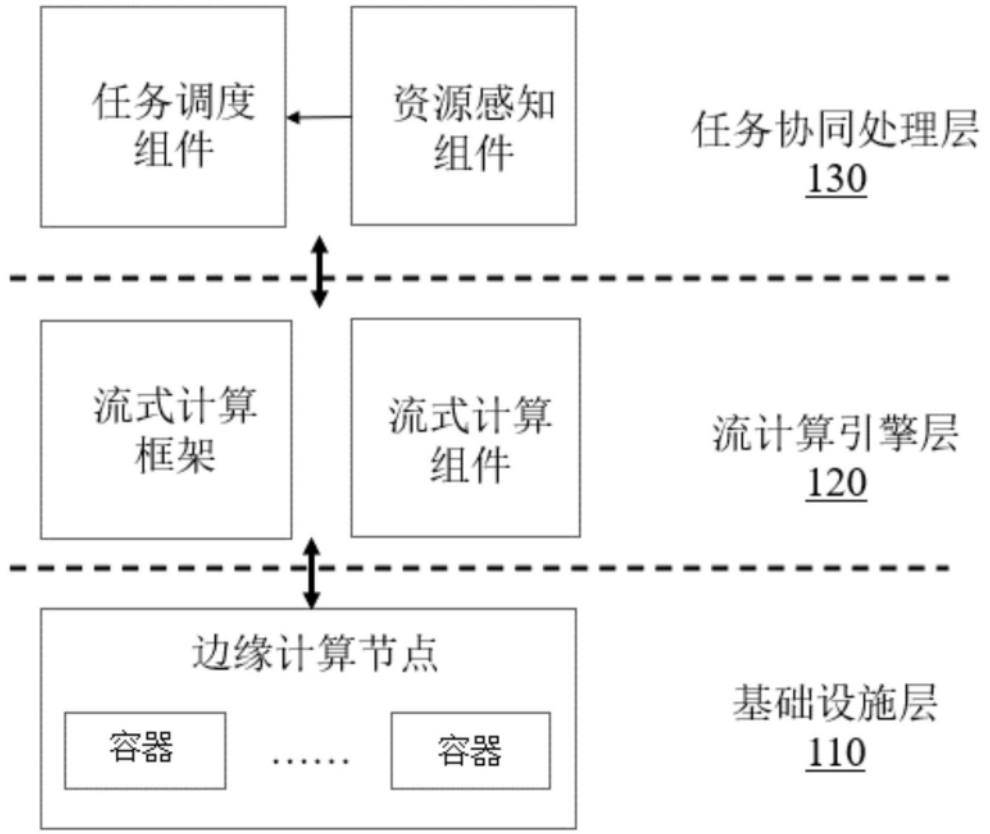

任务调度组件
资源感知组件
任务协同处理层
130
流式计算框架
流式计算组件
流计算引擎层
120
边缘计算节点
容器
……
容器
基础设施层
110

图1

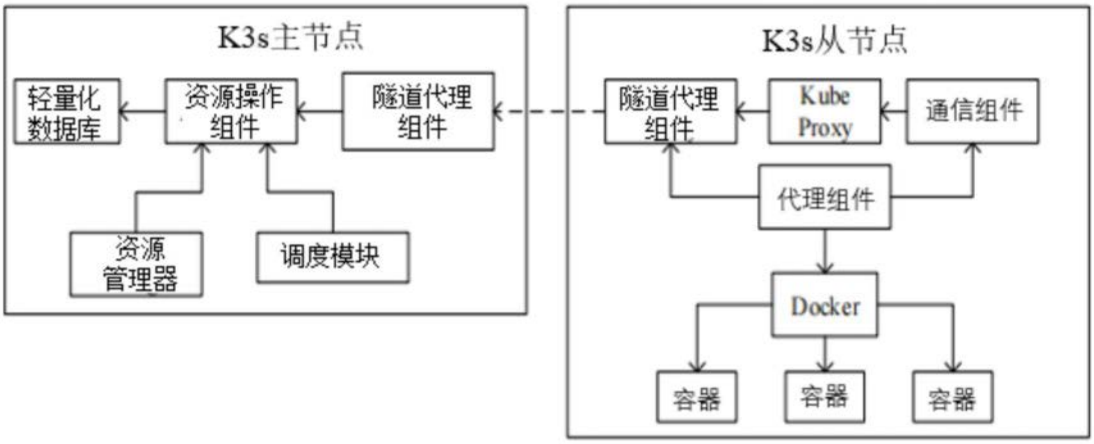

K3s主节点
轻量化数据库
资源操作组件
隧道代理组件
资源管理器
调度模块
K3s从节点
隧道代理组件
Kube Proxy
通信组件
代理组件
Docker
容器
容器
容器

图2

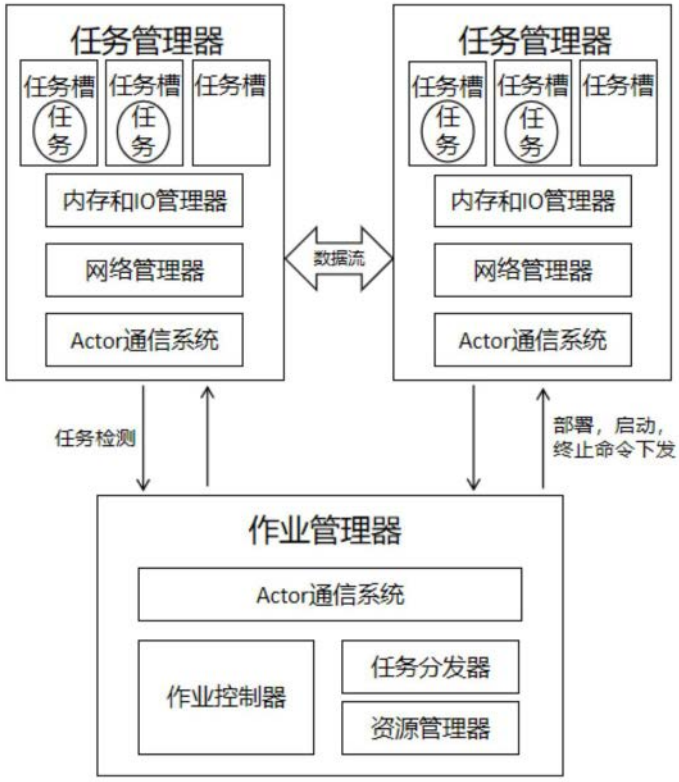

任务管理器
任务槽
任务
任务槽
任务
任务槽
内存和IO管理器
网络管理器
Actor通信系统
数据流
任务管理器
任务槽
任务
任务槽
任务
任务槽
内存和IO管理器
网络管理器
Actor通信系统
任务检测
部署，启动，
终止命令下发
作业管理器
Actor通信系统
作业控制器
任务分发器
资源管理器

图3

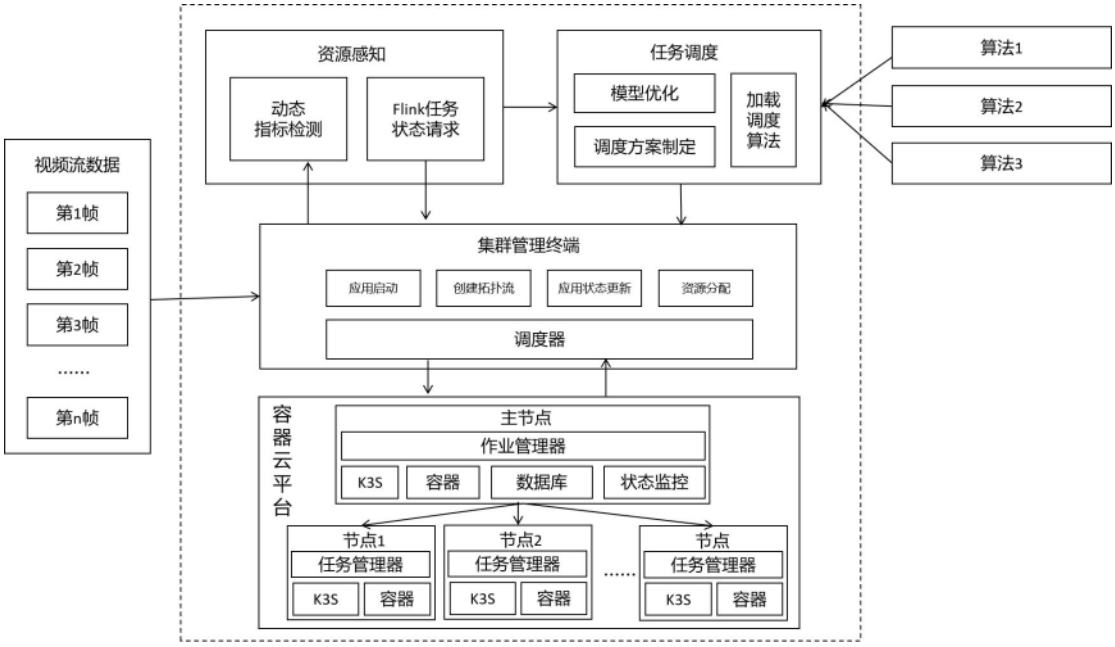

资源感知
动态
指标检测
Flink任务
状态请求
任务调度
模型优化
调度方案制定
加载
调度
算法
算法1
算法2
算法3
视频流数据
第1帧
第2帧
第3帧
.....
第n帧
集群管理终端
应用启动
创建拓扑流
应用状态更新
资源分配
调度器
容器云平台
主节点
作业管理器
K3S
容器
数据库
状态监控
节点1
任务管理器
K3S
容器
节点2
任务管理器
K3S
容器
.....
节点
任务管理器
K3S
容器

图4

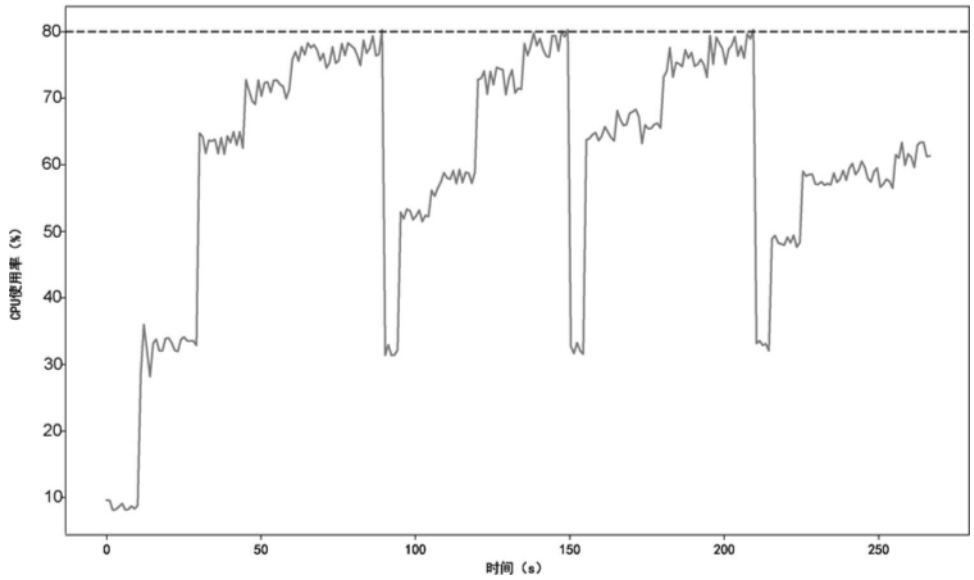

CPU使用率（%）
80
70
60
50
40
30
20
10
0
50
100
150
200
250
时间（s）

图5

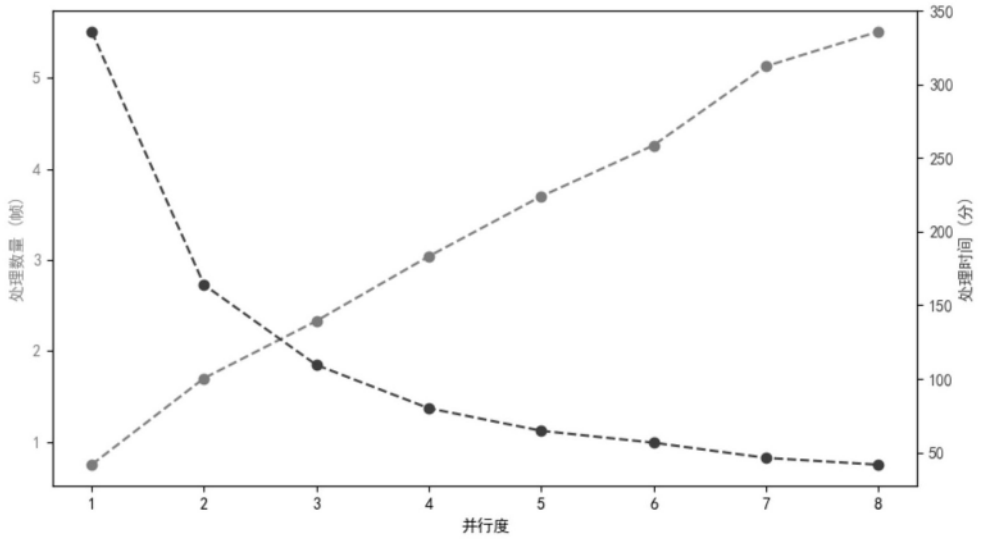

处理数量（帧）
5
4
3
2
1
处理时间（分）
350
300
250
200
150
100
50
1
2
3
4
5
6
7
8
并行度

图6

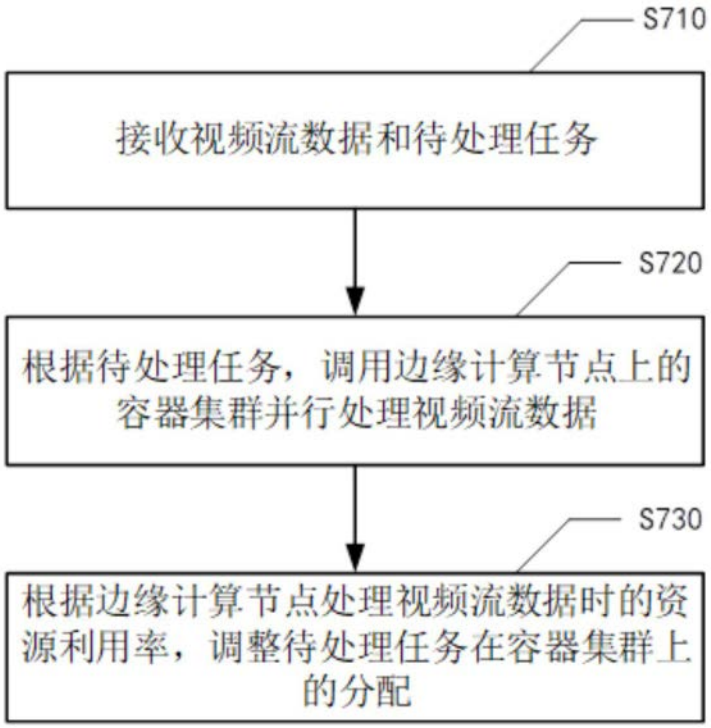

S710
接收视频流数据和待处理任务
S720
根据待处理任务，调用边缘计算节点上的容器集群并行处理视频流数据
S730
根据边缘计算节点处理视频流数据时的资源利用率，调整待处理任务在容器集群上的分配

图7

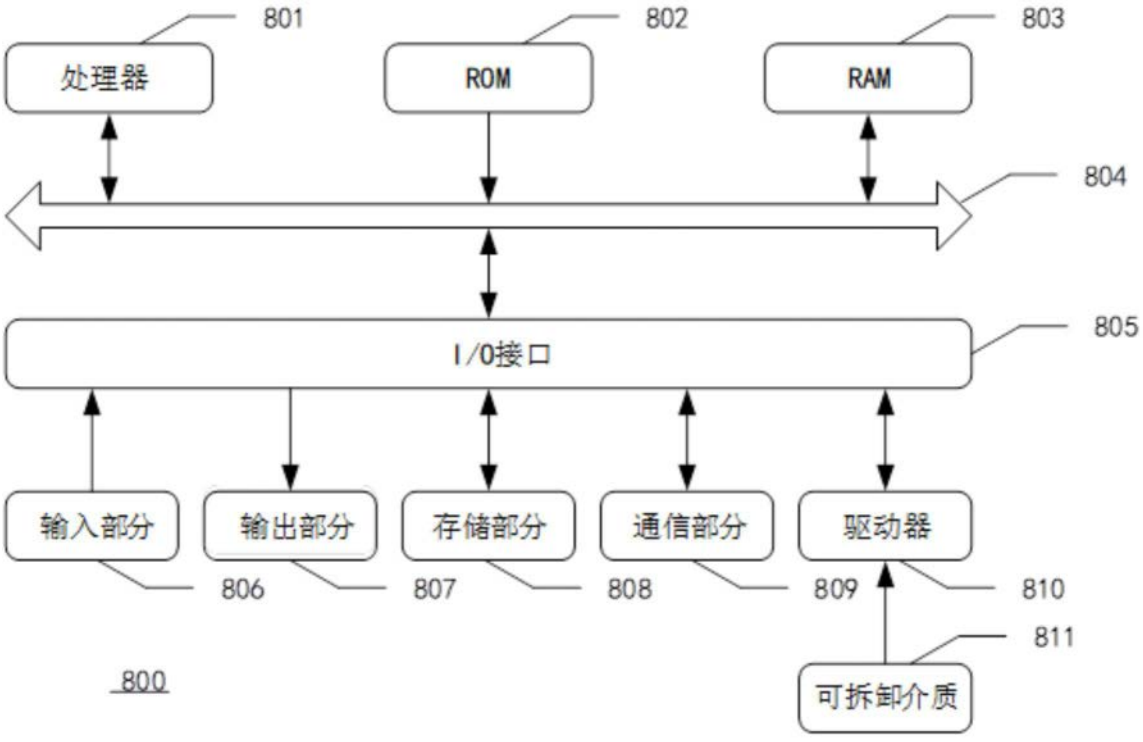

801
处理器
802
ROM
803
RAM
804
805
I/O接口
输入部分
输出部分
存储部分
通信部分
驱动器
806
807
808
809
810
811
可拆卸介质
800

图8