分布式计算环境

基于 KubeEdge 的边缘计算研究分析和实例演示

院（系）名称： 计算机学院

专 业 名 称 ： 计算机科学技术

学 生 姓 名 ： 万天娇、马朋辉、李宁、龚成

二〇二一年八月

目录

1. 边缘智能 5

1.1 在边缘上进行推断的深度学习模型 5
1.1.1. 深度学习模型的优化 5
1.1.2. 模型的拆分 5
1.1.3. 提前退出推断（EEol） 5

1.2 在边缘上训练的深度学习模型 6
1.2.1. 分布式训练 6
1.2.2. 联邦学习 6

1.3 用深度学习优化边缘计算 6
1.3.1. 用深度学习优化缓存策略 6
1.3.2. 用深度学习优化边缘任务卸载 7
1.3.3. 深度学习用于边缘节点的管理和维护 8

1.4 边缘计算对深度学习服务的支持 8

2. 优化与设计 9

2.1 物理架构 9

2.2 数据和服务放置 9

2.3 资源调度 10

3. 系统与工程 11

4. Kubernetes、KubeEdge 和 K3S 原理分析 13

4.1 Kubernetes 原理分析 13
4.1.1. Kubernetes 概述 13
4.1.2. Kubernetes 发展历程 13
4.1.3. Kubernetes 技术架构 14

4.2 KubeEdge 原理分析 15
4.2.1. KubeEdge 概述 15
4.2.2. KubeEdge 技术架构 16

4.3 K3S 原理分析 19
4.3.1. K3S 概述 19
4.3.2. K3S 技术架构 19

5. KubeEdge Data Analysis Demo 22

5.1 软件设计思路 22
5.1.1. 软件包版本 22

5.1.2. 业务逻辑思路.....22
5.2 运行效果截图.....22
6. KubeEdge Counter Demo.....25
6.1 软件设计思路.....25
6.1.1. 软件包版本.....25
6.1.2. 业务逻辑思路.....25
6.2 运行效果截图.....25
7. Temperature demo on Raspberry PI.....27
7.1 软件设计思路.....27
7.1.1. 软件包版本.....27
7.1.2. 硬件型号与配置.....27
7.1.3. 业务逻辑思路.....27
7.2 运行效果截图.....28
8. 分工与总结.....30
8.1 人员分工.....30
8.2 总结.....30
参考文献.....31

随着万物互联时代的到来，网络边缘设备产生的数据量快速增加，带来了更高的数据传输带宽需求，同时，新型应用也对数据处理的实时性提出了更高要求，传统云计算模型已经无法有效应对，因此，边缘计算应运而生。边缘计算的基本理念是将计算任务在接近数据源的计算资源上运行，可以有效减小计算系统的延迟，减少数据传输带宽，缓解云计算中心压力，提高可用性，并能够保护数据安全和隐私。得益于这些优势，边缘计算从 2014 年以来迅速发展，经历了技术储备期、快速增长期，从 2018 年起已进入稳定发展期。

我将边缘计算领域目前的主要研究内容分为三块进行分析，第一个是边缘智能，也是就边缘计算和人工智能的结合，是目前论文中的热点研究方向，研究内容主要围绕以下几类：在边缘上进行推断的深度学习模型，在边缘上训练的深度学习模型，用深度学习优化边缘计算，边缘计算对深度学习服务的支持。第二个是针对边缘计算的优化与设计，将研究内容自底向上地划分为物理架构、内容放置、资源调度三个板块进行分析，后两部分被研究较多；第三个是系统与工程，该部分的文章结合实际场景展开研究，SEC（ACM/IEEE Symposium on Edge Computing）会议里这部分论文出现的较多。

1. 边缘智能

当下人工智能（特别是深度学习）算法的计算需求增加，但受限于智能终端设备本身体积、能耗等因素，很多大型算法不能在终端上直接运行。为了解决这个问题，一方面我们可以探索模型压缩方法以减少运算量，另一方面我们可以运用边缘计算的方法将算力在终端、边缘与云之间进行合理的分配与卸载。边缘智能则是人工智能（深度学习）与边缘计算的一个交叉领域。边缘智能主要研究如何将人工智能模型，包括统计学习、深度学习及强化学习，放在网络边缘端执行。

对深度学习与边缘计算的研究主要总结为以下几类：在边缘上进行推断的深度学习模型，在边缘上训练的深度学习模型，用深度学习优化边缘计算，边缘计算对深度学习服务的支持。

1.1 在边缘上进行推断的深度学习模型

1.1.1. 深度学习模型的优化

深度学习任务通常是计算密集型的，需要占用大量内存。但是在边缘，没有足够的资源来支撑大规模的深度学习模型。所以需要优化深度学习模型，并量化它们的权重来降低资源成本。其中，最重要的挑战是：如何转换或重新设计深度学习模型使它们适合边缘设备，并且尽可能减少模型性能的损失。针对不同场景，先主要有两种优化方法，一是对资源相对充足的边缘节点，采用通用的优化方法；另一种是针对资源预算紧张的终端设备，采用细粒度的优化，针对模型输入、模型结构、模型选择、模型构架分别进行优化设计的研究。

1.1.2. 模型的拆分

对深度学习模型进行水平拆分，沿末端、边缘和云进行拆分是最常见的拆分方法，问题在于如何智能地选择拆分点。确定拆分点的过程一般可以分为三个步骤[1]：1）测量和建模不同 DNN 层的资源成本和层间中间数据的大小； 2）通过特定的层配置和网络带宽来预测总成本； 3）根据延迟需求等从候选拆分点中选择最佳选择。另一种模型拆分是垂直拆分，特别是对于 CNN [2]，与水平分割相反，以网格方式融合层并垂直拆分，将 CNN 层划分为独立计算任务。

1.1.3. 提前退出推断（EEol）

通过利用 EEoI，可以在边缘设备上使用深度学习模型的浅层部分进行快速

局部推理。通过这种方式，边缘设备上的浅层模型可以快速进行初始特征提取，如果有较高置信度，可以直接给出推理结果。否则，部署在云中的额外大型深度学习模型将执行进一步的处理和最终推理。与直接将深度学习计算卸载到云端相比，这种方法具有更低的通信成本，并且比边缘设备上的模型修剪或量化具有更高的推理精度[3]。

1.2 在边缘上训练的深度学习模型

1.2.1. 分布式训练

两种方案[4]：一种解决方案是，每个终端设备基于本地数据训练一个模型，然后在边缘节点聚合这些模型的更新。另一种是每个边缘节点训练自己的局部模型，并交换和细化它们的模型更新以构建全局模型。边缘的大规模分布式训练虽然避免了将大量原始数据集传输到云端，但不可避免地引入了边缘设备间的梯度交换通信成本。此外，在实际应用中，边缘设备可能会遭受更高的延迟、更低的传输速率，因此进一步阻碍了不同边缘设备间的深度学习模型梯度交换。

1.2.2. 联邦学习

联邦学习是一种新兴的但有前途的方法，主要步骤：1) 从中央服务器下载全局深度学习模型，2）使用自己的数据在下载的全局模型下训练其本地模型，3) 仅将更新后的模型上传到服务器进行模型平均。通过将训练数据限制在设备端，可以保护数据隐私和降低安全风险，从而避免将训练数据上传到云端所引起的隐私泄露问题。

联邦学习可以解决以下问题：1)训练数据可以不满足不是独立同分布的，2）有限的通信资源，3)各个节点训练数据和训练资源分布不均匀，4）数据隐私和安全问题。

目前对于联邦学习和边缘计算的研究主要围绕通信有效性、资源优化、安全提升等方向展开。

1.3 用深度学习优化边缘计算

1.3.1. 用深度学习优化缓存策略

深度学习的应用：传统的缓存方法通常计算复杂度高，因为它们需要大量的

在线优化迭代来确定用户和内容的特征，以及内容放置和交付的策略。使用 DNN 网络来处理从用户的移动设备收集的原始数据，从而提取用户和内容的特征形成基于特征的内容热度矩阵[5]。对热度矩阵用基于特征的协同过滤算法来估计核心网络的流行内容。当使用 DNN 网络优化边缘缓存策略时，可以通过离线训练避免在线大量计算迭代。一个 DNN 网络由一个用于数据正则化的 encoder 层和隐藏层构成，可以用最优启发式算法训练来确定缓存策略，从而避免在线优化迭代[6]。

强化学习的应用：与基于 DNN 的边缘缓存不同，强化学习可以利用用户和网络的上下文，并采用自适应策略来最大化长期缓存的性能[7]。传统的强化学习算法受限于难以手工提取特征，以及难以处理高维度数据和动作的缺陷[8]。相比于与深度学习无关的传统强化学习，例如，将 Q-learning 和 Multi-Armed Bandit (MAB) learning[9]结合，深度强化学习的优势在于 DNN 网络可以从原始观察数据中学习关键特征。结合了强化学习和深度学习的智能体，可以直接从高维观测数据优化其在边缘计算网络中缓存管理的策略。

在[10]中，用 DDPG 训练 DRL 智能体以最大化长期缓存命中率，以做出适当的缓存替换决策。这项工作考虑了单个 BS 的场景，其中 DRL 智能体决定是否替换缓存的内容，在训练中，奖励被设计为缓存的命中率。

1.3.2. 用深度学习优化边缘任务卸载

边缘计算允许边缘设备在能量、延迟、计算能力等条件约束下将部分计算任务卸载到边缘节点，这些约束引发了一系列问题：1) 哪些边缘节点应该接收任务，2）边缘设备应该卸载的任务比例是多少，3）应该为这些任务分配多少资源。任务卸载是 NP 难的问题，因为要涉及通信和计算资源的组合优化以及边缘设备的竞争。特别是优化需要同时考虑随时间变化的无线环境（例如变化的信道质量）和任务卸载的请求，因此更倾向选择基于学习的优化方法。其中，基于深度学习的方法比其他方法具有更大优势，所以许多研究用深度学习或强化学习的方法优化边缘任务卸载。

深度学习的应用：在[11]中，计算卸载问题被表述为一个多标签分类问题。通过离线穷举求解，得到的最优解可用于训练 DNN 网络，其中输出为卸载决策。通过这种方式，可能最优解不需要在线求解，避免了延迟的卸载决策，并且计算

复杂度可以转移到训练上。

强化学习的应用：虽然将计算任务卸载到边缘节点可以提高计算任务的处理效率，但是卸载的可靠性会受到无线环境质量的影响。在[12]中，为了最大化卸载效用，作者首先量化了各种通信模式对任务卸载性能的影响，并提出用 DQL 在线选择最佳卸载边缘节点和传输模式。为了优化总卸载成本，修改了 Dueling-DQL 和 Double-DQL 网络，智能体可以为终端设备分配边缘计算和带宽资源。

1.3.3. 深度学习用于边缘节点的管理和维护

其他研究围绕深度学习对边缘通信的优化、边缘节点安全性的提高，以及联合节点优化等展开。

1.4 边缘计算对深度学习服务的支持

深度学习服务的广泛部署，尤其是移动深度学习，需要边缘计算的支持。这种支持不仅仅是在网络架构层面，边缘硬件和软件的设计、适配和优化同样重要。具体来说，1）定制的边缘硬件和相应优化的软件框架和库可以帮助深度学习执行更高效；2）边缘计算架构可以实现深度学习计算的卸载；3）设计良好的边缘计算框架可以更好地维护在边缘运行的深度学习服务；4) 用于评估边缘深度学习性能的平台有助于进一步改进上述实现。

这一部分的研究主要围绕硬件对深度学习的支撑，通信和计算模型的支撑，定制的深度学习边缘框架，以及边缘深度学习的性能评估等工作展开。

2. 优化与设计

这一方向的论文主要针对三部分进行优化设计，我把它自底向上地划分为：物理架构、内容放置、资源调度。物理架构这部分论文研究的是边缘计算的架构，这涉及边缘站点的放置及部署、无线网络规划和路由等。第二部分，内容放置主要围绕服务选择、服务部署等问题提出解决方案，它是建立在物理架构基础上的。既然搭建好了边缘网络，接下来要做的就是数据和服务（内容）的部署和放置。这会涉及到许多问题要解决，比如，服务架构的选择，以微服务架构为例，需要为各类服务选择合适的边缘站点部署实例，这就得面临服务器选择、服务放置，服务部署等问题。如果待部署的服务有复杂的组合结构，那么也会涉及到服务组合的问题等，这一块的文章就主要解决这些问题。第三部分的资源调度是建立在物理架构和内容放置的基础上的，这类论文研究的是如何合理调度资源以提供更加优质的服务质量和用户体验。包括计算卸载方案的设计、用户数据在不同边缘站点之间的同步与迁移、移动性管理等。

2.1 物理架构

以 IEEE 期刊（2021 年）上的文章 Design and Simulation of a Hybrid Architecture for Edge Computing in 5G and Beyond[13]为例，该文章针对 5G 及更高版本的边缘计算（如自动驾驶汽车、增强现实和远程手术）超低延迟的要求，提出了一种混合架构，该架构利用可持续的技术（例如，D2D 通信、MIMO、SDN 和 NFV），并具有可扩展性、可靠性和超低延迟支持等主要特性。

2.2 数据和服务放置

以 IEEE 期刊（2021 年）上的文章 An Online Framework for Joint Network Selection and Service Placement in Mobile Edge Computing[14]为例。以前的相关工作主要集中在通过动态服务放置来优化通信延迟问题，而忽略了接入网络选择对接入延迟的关键影响。在这篇文章中，研究了联合优化接入网络选择和服务放置的问题, 通过平衡接入延迟、通信延迟和服务切换的成本来提高 QoS。

随着 5G 无线技术的快速发展和部署，移动边缘计算（MEC）已成为一种新的技术来满足移动应用程序的低延迟要求，但由于分散的无线通信环境和容量受限

的移动边缘计算节点，这种新技术的一个问题就是：如何为移动用户保持令人满意的服务质量（QoS）。这种 QoS 通常在端到端延迟方面，极易受到接入网络瓶颈和通信延迟的影响。为了解决这个问题，首先，这篇文章同时考虑了 MEC 中的接入网络选择和服务放置问题，考虑访问延迟、切换成本和通信延迟来提高 MEC 应用程序的 QoS。具体来说，当用户访问间接连接的边缘云上的服务时，会产生通信延迟。接入延迟和切换成本分别由 AP 的排队延迟和业务的动态迁移引起。由于公式化的长期优化问题本质上是随机的，即包含未来的不确定性，如用户移动性，本篇文章设计了一种高效的在线框架，将长期时变优化问题分解为一系列一次性子问题。为了解决一次性问题的 NP 难度，本篇文章设计了一种基于匹配和博弈论的计算效率高的两阶段算法，近乎实现了最优的解决方案。

2.3 资源调度

以 2021 年发表在《IEEE Transitions on Mobile Computing》上的 Robust Task Offloading in Dynamic Edge Computing[15]为例。

移动边缘计算通过把任务从终端设备卸载到附近的边缘服务器，来实现更好的应用程序响应能力。然而由于移动性和功率限制，边缘服务器集会变得不确定，当某些服务器出现故障时，在它们上运行的任务也将失败。为了克服边缘服务器的不稳定性，本篇文章提出了一种鲁棒的任务卸载方案，可以抵抗 h（≥1）个边缘服务器故障。对于 h=1 的情况，本文设计了在线原始对偶算法，可以在任务到达时卸载它们，并从理论上推导出其竞争率。本文还将原始对偶算法扩展到 h=2 的情况，并进一步扩展到 $h \geq 3$ 的一般情况。通过仿真实验，证明本文的方案可以很好地处理边缘服务器的故障，并实现接近最佳的 DEC 吞吐量。

3. 系统与工程

这类文章研究的场景包括视频监控的实时处理、目标检测、嫌疑行为识别等，车联网和自动驾驶中的数据实时感知与处理，无人机的实时路径规划与农业灌溉等。即使此类文章研究的是资源调度和优化的工作，也与从优化角度入手的论文完全不同。这些论文的工作通常借助一些单片机作为边缘设备，借助 Docker 和 Kubernetes 搭建 benchmarks，并开发相应的中间件来解决一些实际问题。和优化类的文章相比，这类文章所需的研究周期更长，可以看出，它们需要涉及较到要解决的问题所在领域的专业知识。

例如，2019 年边缘计算顶会 SEC 上的文章:Managing Edge Resources for Fully Autonomous Aerial Systems[16]。完全自主的空中系统可以通过软件完全执行复杂的任务。如果用户可以很好选择软件、计算硬件和飞机，则与人类相比，FAAS 驾驶的无人机系统能更快，更安全地完成任务。另一方面，管理不善的边缘资源会降低任务的执行速度，浪费能源并增加成本。本文提出了一种模型驱动的 FAAS 管理方法。研究人员进行实际的 FAAS 任务飞行，配置文件计算和飞机资源使用，并为预期需求建模。通过创建 FAAS 验证了模型的有效性，并在许多系统设置下测量了任务吞吐量。通过开源 FAAS 套件 SoftwarePilot 发布的 FAAS 基准，执行现实的任务：自主摄影、搜索和救援以及农业侦察。该模型错误预测为 4%，对比方法产生 10%到 24%的错误。作者团队研究了：（1）GPU 加速、扩展和横向扩展，（2）板载、边缘和云计算，（3）能源和资金预算，（4）软件驱动的 GPU 管理。研究发现，模型驱动的管理可以使任务量提高 10 倍，并将成本降低 87%。

再例如，另一篇文章 I-SAFE:Instant Suspicious Activity identiFicationat the Edge using Fuzzy Decision Making[17]。城市图像通常用作取证分析，通过设计可用于减轻事故。收集更多图像后，人们很难从成千上万个视频剪辑中确定具体事件的某些帧。研究人员希望有一个实时，主动的监视系统，该系统可以立即检测可疑人员，识别可疑活动，或提高警惕。本文通过采用模糊决策在边缘（I-SAFE）上引入即时可疑活动识别来提出一种法务监视策略。针对安全人员的决策制定了模糊控制系统。根据轻量级深度机器学习（DML）模型提取的视频功能做出决策。根据一线执法人员的要求，选择并模糊化了一些功能，以应对人员决策过程

中存在的不确定性状态。在边缘层次结构中使用功能可以最大程度地减少即时警报的通信延迟。此外，利用微服务体系结构，I-SAFE 方案可在网络边缘增加复杂性，并具有良好的可扩展性。经测试，I-SAFE 方案识别可疑活动的平均时间为 0.002 秒。

4. Kubernetes、KubeEdge 和 K3S 原理分析

4.1 Kubernetes 原理分析

4.1.1. Kubernetes 概述

Kubernetes 是一个开源的，用于管理云平台中多个主机上的容器化的应用，Kubernetes 的目标是让部署容器化的应用简单并且高效,Kubernetes 提供了应用部署，规划，更新，维护的一种机制。Kubernetes 一个核心的特点就是能够自主的管理容器来保证云平台中的容器按照用户的期望状态运行着。

该项目最初是 Google 内部面向容器的集群管理系统，而现在由 Cloud Native Computing Foundation(CNCF，云原生计算基金会)托管的开源平台。Kubernetes 系统拥有一个庞大而活跃的开发人员社区，现在 GitHub Star 数已达 80141，是 Go 语言最大的开源项目之一，Kubernetes 也被称为 K8s，是通过将 8 个字母 ubernete 替换为 8 而形成的缩写。

4.1.2. Kubernetes 发展历程

2003-2004 年，Google 发布了 Borg 系统，一个内部集群管理系统。

2013 年左右，Google 继 Borg 系统之后发布了 Omega 集群管理系统，适用于大型计算集群的灵活、可扩展的调度程序。

2014 年左右，Google 发布了 Kubernetes 系统，其是作为 Borg 的开源版本发布的，同年 Microsoft、RedHat、IBM 等加入 Kubernetes 社区。

2015 年左右，Google 正式发布 Kubernetes 1.0，并于 Linux 基金会合作组建了云原生计算基金会（CNCF），来对抗以 Docker 公司为核心的容器商业生态。

2016 年左右，Kubernetes 成为主流。

2017 年左右，互联网巨头纷纷表示支持 Kubernetes。

2018 年左右，无人不知 Kubernetes。

2014 年至 2015 年左右，在如火如荼的容器化浪潮里，一度出现容器编排“三国鼎立”的局面，不过最终 Kubernetes 胜出，编排之争拉下帷幕。

4.1.3. Kubernetes 技术架构

Kubernetes 是利用共享网络将多个物理机或者虚拟机组成一个集群，在各个服务器之间进行通信,整体架构如图 1 所示。一个 Kubernetes 集群由 Master 节点和 Node 节点组成。Master 是集群的网关和中枢枢纽，主要作用：暴露 API 接口，跟踪其他服务器的健康状态、以最优方式调度负载，以及编排其他组件之间的通信。单个的 Master 节点可以完成所有的功能，但是考虑单点故障的痛点，生产环境中通常要部署多个 Master 节点，组成高可用集群。Node 节点是 Kubernetes 的工作节点，负责接收来自 Master 的工作指令，并根据指令相应地创建和销毁 Pod 对象，以及调整网络规则进行合理路由和流量转发。生产环境中，Node 节点可以有 N 个。

Kubernetes 从宏观上看分为 2 个角色 Master 节点和 Node 节点，但是在 Master 节点和 Node 节点上都存在着多个组件来支持内部的业务逻辑，其包括：运行应用、应用编排、服务暴露、应用恢复等等，在 Kubernetes 中这些概念被抽象为 Pod、Service、Controller、Deployment、Secret 等资源类型。

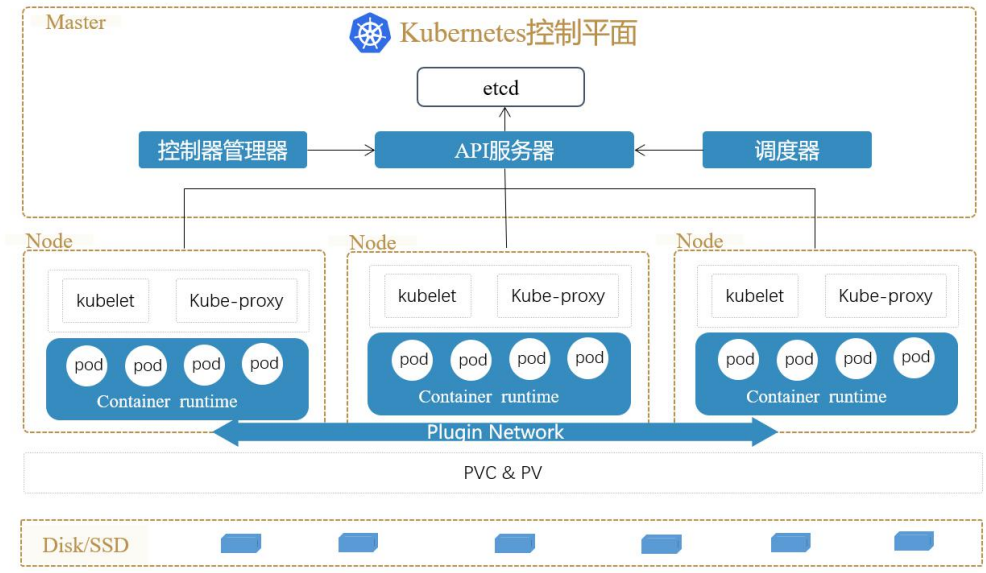

图 1 Kubenetes 技术架构

在图一架构图里，API 服务器:负责将 K8s“资源组/资源版本/资源”以 HTTP REST 风格的形式对外暴漏提供 API 接口及服务。Etcd:分布式键值存储 K8s 系统集群的状态和元数据（包含对象信息、节点信息等），仅与 API 服务器交互。控制器管理器：负责管理集群中的节点、Pod 副本、服务、命名空间、端点、资

源定额等，根据每种资源类型提供对应资源的控制器，确保资源实际状态收敛到所需状态。调度器：为集群中的 pod 资源找到合适的节点部署并在该节点上运行。Kubelet：管理节点，用来接收、处理、上报 API 服务器组件下发的任务，实现了 3 种开放接口 CNI（容器网络接口）、CRI（容器运行时接口）、CSI（容器存储接口）。Kube-proxy：节点上的网络代理，通过 iptables 为一组 pod 提供统一的 TCP/UDP 流量转发和负载均衡功能。

4.2 KubeEdge 原理分析

4.2.1. KubeEdge 概述

KubeEdge 是一个开源的系统，可将本机容器化应用编排和管理扩展到边缘端设备。它构建在 Kubernetes 之上，为网络和应用程序提供核心基础架构支持，并在云端和边缘端部署应用，同步元数据。100%兼容 K8s API，可以使用 K8s API 原语管理边缘节点和设备。KubeEdge 还支持 MQTT 协议，允许开发人员编写客户逻辑，并在边缘端启用设备通信的资源约束。

KubeEdge 的诞生，是为了弥补 Kubernetes 在边缘计算上的瓶颈，Kubernetes 太过繁重。在 Kubernetes 集群中，用户在 Master 节点上通过编写一个 Yaml 或者 Json 格式的配置文件，也可以通过命令等请求 Kubernetes API 创建应用，就直接将应用部署到集群上的各个节点上，该配置文件中还包含了用户想要应用程序保持的状态，从而生成用户想要的环境。

Kubernetes 作为容器编排的标准，人们自然会想把它应用到边缘计算上，即通过 Kubernetes 在边缘侧部署应用，但是 Kubernetes 在边缘侧部署应用时遇到了一些问题，例如

a) 边缘侧设备没有足够的资源运行一个完整的 Kubelet。
b) 一些边缘侧设备是 ARM 架构的，然而大部分的 Kubernetes 发行版并不支持 ARM 架构。
c) 边缘侧网络很不稳定，甚至可能完全不通，而 Kubernetes 需要实时通信，无法做到离线自治。
d) 很多边缘设备都不支持 TCP/IP 协议。

e） Kubernetes 客户端（集群中的各个 Node 节点）是通过 List-watch 去监听 Master 节点的 API Server 中资源的增删改查，List-watch 中的 Watch 是调用资源的 Watch API 监听资源变更事件，基于 HTTP 长连接实现，而维护一个 TCP 长连接开销较大。从而造成可扩展性受限。

为了解决包含但不限于以上 Kubernetes 在物联网边缘场景下的问题，华为于 2018 年 11 月 15 日上午宣布了 KubeEdge 开源项目，并在 2019 年 3 月捐给 CNCF 基金会。联合社区力量一起促进 Kubeedge 的开发。现在 GitHub Star 数已达 4100。

KubeEdge 的优点如下：

a） 边缘计算

通过在 Edge 上运行的业务逻辑，可以在生成数据的本地保护和处理大量数据。这减少了网络带宽需求以及边缘和云之间的消耗。提高响应速度，降低成本并保护客户的数据隐私。

b） 简化开发

开发人员可以编写基于常规 HTTP 或 MQTT 的应用程序，对其进行容器化，然后在云端或边端中的任何位置运行它们中的更合适的一个。

c） Kubernetes 原生支持

借助 KubeEdge，用户可以在 Edge 节点上编排应用，管理设备并监视应用和设备状态，就像云中的传统 Kubernetes 集群一样。

d） 大量的应用

可以轻松地将现有的复杂机器学习、图像识别、事件处理和其他高级应用程序部署到 Edge。

4.2.2. KubeEdge 技术架构

KubeEdge 架构图如图 2 所示，其架构分为了云端和边缘端。云端组件 CloudCore 部署于云端 Master 节点上，边端组件 EdgeCore 部署于边缘节点上。云端组件 CloudCore 主要包含模块 CloudHub、EdgeController、DeviceController。边端组件 EdgeCore 主要包含模块 EdgeHub、Edged、EventBus、ServiceBus、DeviceTwin、MetaManager。

在边缘侧，最大的问题就是网络波动、很不稳定，经常会离线，而 KubeEdge 解决了这一痛点，实现了边缘离线自治和云边协同。首先需要了解的就是云边通

信机制，是就衍生出了云端的 Cloud Hub 与边源端的 Edge Hub。这两个模块之间通过 Websocket 或者 Quic 通信，相当于建立了一条底层通信隧道，供 K8s 和其他应用通信。当然，使用什么协议通信不是重点，重点是当两者之间的链路都无法保障的时候使业务不受到影响，这就是 MetaManager 的要解决的问题了。

CloudHub 其实就是通信的 Server 端，是 Controller 和 Edge 端之间的中介。它负责下行分发消息(其内封装了 K8s 资源事件)，也负责接收并发送边缘节点上行消息至 Controllers。其中下行的消息在应用层增强了传输的可靠性，以应对云边的弱网络环境。

EdgeHub：一个 Web socket 客户端，负责云端与边缘端的信息交互，其中包括将云端的资源变更同步到边缘端及边缘端的状态变化同步到云端。

DeviceTwine：该模块利用 Kubernetes Custom Resource Definition(CRD)自定义资源机制，将自定义设备资源信息保存到本地数据库中，并处理基于云端的操作来修改边缘设备的某些属性；同时，将设备基于 EventBus 模块上报的状态信息同步到本地数据库和云端的中介。

Edged：是管理节点生命周期的边缘节点模块。它可以帮助用户在边缘节点上部署容器化的工作负载或应用程序。这些工作负载可以执行任何操作，从简单的遥测数据操作到分析或 ML 推理等。使用 Kubectl 云端的命令行界面，用户可以发出命令来启动工作负载。可理理解为 Edged 模块就是功能裁剪后的 Kubelet。该模块就是保障云端下发的容器组以及其对应的各种配置、存储能够在边端稳定运行，并在异常之后提供自动检测、故障恢复等能力。当然，由于 K8s 本身运行时的发展，该模块对应支持各种 CRI 应该也比较容易。

EventBus/ServiceBus/Mappper：消息传递接口。外部设备的接入当前支持 MQTT 和 HTTP 协议，这里分别对应 EventBus 和 ServiceBus。EventBus 就是一个 MQTT Broker 的客户端，主要功能是将边端各模块通信的 Message 与设备 Mapper 上报到 MQTT 的 Event 做转换的组件；而 ServiceBus 就是对应当外部是 Rest-API 接入时的转换组件。

MetaManager：该模块后端对应一个本地的数据库，所有其他模块需要与云端通信的内容都会被保存到本地数据库中一份，当需要查询数据时，如果本地数

据库中存在该数据，就会从本地获取，这样就避免了与云端之间频繁的网络交互；同时，在网络中断的情况下，本地的缓存的数据也能够保障其稳定运行，在通信恢复之后，重新同步数据。

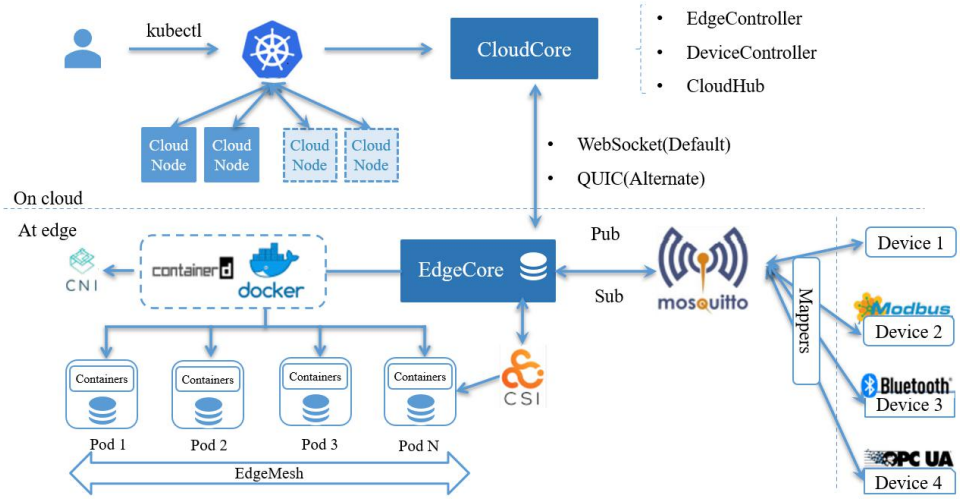

图 2：KubeEdge 技术架构

在图 2 KubeEdge 技术架构里，云端，CloudHub：一个 Web socket 服务器，负责监听云端的更新、缓存及向 EdgeHub 发送消息。EdgeController：一个扩展的 K8s 控制器，负责管理边缘节点和容器组元数据，同步边缘节点的数据，是 K8s API Server 与 EdgeCore 的通信桥梁。DeviceController：一个扩展的 K8s 控制器，负责管理节点设备，同步云端和边缘端的设备元数据和状态。

边缘端 EdgeHub：一个 Web Socket 客户端，负责云端与边缘端的信息交互，其中包括将云端的资源变更同步到边缘端及边缘端的状态变化同步到云端。Edged：运行在边缘节点，管理容器化应用的 Agent，负责容器组生命周期的管理，类似 Kubelet。EventBus：一个 MQTT 客户端，与 MQTT 服务端交互，提供发布/订阅的能力。ServiceBus：一个 HTTP 客户端，与 HTTP 服务端交互。为云组件提供 HTTP 客户端功能，以访问在边缘运行的 HTTP 服务器。DeviceTwin：负责存储设备状态并同步设备状态到云端，同时提供应用的接口查询。MetaManager：Edged 和 EdgeHub 之间的消息处理器，负责向轻量数据库（SQLite）存储或查询元数据。

4.3 K3S 原理分析

4.3.1. K3S 概述

k3s 是一个轻量级 Kubernetes，它易于安装，二进制文件包小于 40M，只需 512MB RAM 即可运行,非常适用于边缘计算-Edge、物联网-IoT、CI、ARM、Development 等场景。而且由于运行 K3s 所需的资源相对较少，所以 K3s 也适用于开发和测试场景。

K3s 最大的优点就在于轻量级，而且初衷是希望安装的 Kubernetes 在内存占用方面只是一半的大小，所以也形象的命名为 K3s，Kubernetes 是一个 10 个字母的单词，简写 K8s。所以，有 Kubernetes 一半大的东西就是一个 5 个字母的单词，简写为 K3s。目前 K3s GitHub Star 数已达 17596。

K3s 有以下增强功能：

a) 打包为单个二进制文件。
b) 使用基于 Sqlite3 的轻量级存储后端作为默认存储机制。同时支持使用 Etcd3、MySQL 和 PostgreSQL 作为存储机制。
c) 封装在简单的启动程序中，通过该启动程序处理很多复杂的 TLS 和选项。
d) 默认情况下是安全的，对轻量级环境有合理的默认值。
e) 添加了简单但功能强大的 Batteries-included 功能，例如：本地存储提供程序，服务负载均衡器，Helm controller 和 Traefik Ingress controller。
f) 所有 Kubernetes control-plane 组件的操作都封装在单个二进制文件和进程中，使 K3s 具有自动化和管理包括证书分发在内的复杂集群操作的能力。
g) 最大程度减轻了外部依赖性，K3s 仅需要 Kernel 和 Cgroup 挂载。

K3s 软件包需要的依赖项包括 Containerd、Flannel、CoreDNS、CNI、主机实用程序（Iptables、Socat 等）、Ingress controller（Traefik）、嵌入式服务负载均衡器（Service load balancer）、嵌入式网络策略控制器（Network policy controller）。

4.3.2. K3S 技术架构

K3s 的技术架构如图 3 所示，乍一看，貌似和 Kubernetes 架构图类似，控制节点都有 API 服务器、调度器、控制器管理器，工作节点有 kubelet、kube-proxy、Flannel 插件。但在 K3s 集群中，将运行控制平面组件与 Kubelet 的节点称为 Server，而只运行 Kubelet 的节点称为 Agent。server 和 agent 都有容器运行时和一个 Kube-proxy，管理整个集群的 Tunnel 和网络流量。之所以在 K3s 集群中不再命名为 Master 节点和 Worker 节点是因为 K3s 中的 Master 节点和 Worker 节点没有明显的区别。可以在任何节点上调度和管理容器组，而且一旦传递了 Server 的 URL，节点就会变成一个 Agent。

正如前文所述，K3s 的二进制文件包只有几十 M 大小，这款迷你 Kubernetes 发行版针对边缘进行了高度优化及对上游进行了极大删减，具体体现如下：

a) 旧的、Alpha 版本的、非默认功能都已经删除。

b) 删除了大多数内部云提供商和存储插件，可以用插件替换。

c) 新增 SQLite3 作为默认存储机制，Etcd3 仍然有效，但是不再是默认项。

d) 封装在简单的启动器中，可以处理大量 LTS 复杂性和选项。

e) 最小化到没有操作系统依赖，只需要一个内核和 Cgroup 挂载。

上游的 Kubernetes 发行版是臃肿的，且很多代码都可以删除，一些没必要的插件也没必要保留。Kubernetes 依靠分布式键值数据库 Etcd 来存储整个集群的状态，而 K3s 用名为 SQLite 的轻量级数据库取代了 Etcd，SQLite 是一个成熟的嵌入式场景数据库。很多移动应用都会捆绑 SQLite 来存储状态。虽然 SQLite 并不是分布式数据库，但 K3s Server 可以指向外部数据库端点。K3s 的另一优点就是“包含电池但可替换”的方式。例如，我们可以用 Docker CE 运行时替换 Containerd 运行时，用 Calico 替换 Flannel，用 Longhorn 替换本地存储等等。

K3s 发行版支持多种架构，包括 AMD64、ARM64。凭借一致的安装体验，K3s 可以在 Raspberry Pi Zero、NVIDIA Jetson Nano、Intel NUC 或 Amazon EC2 a1.4xlarge 实例上运行。

如果你需要一个在 AMD64 或 ARM64 架构上运行的高可用集群，安装一个 3 节点的 Etcd 集群，然后是 3 个 K3s Server 和一个或多个 Agent。这样就可以为你提供一个生产级的环境，并为控制平面提供 HA。

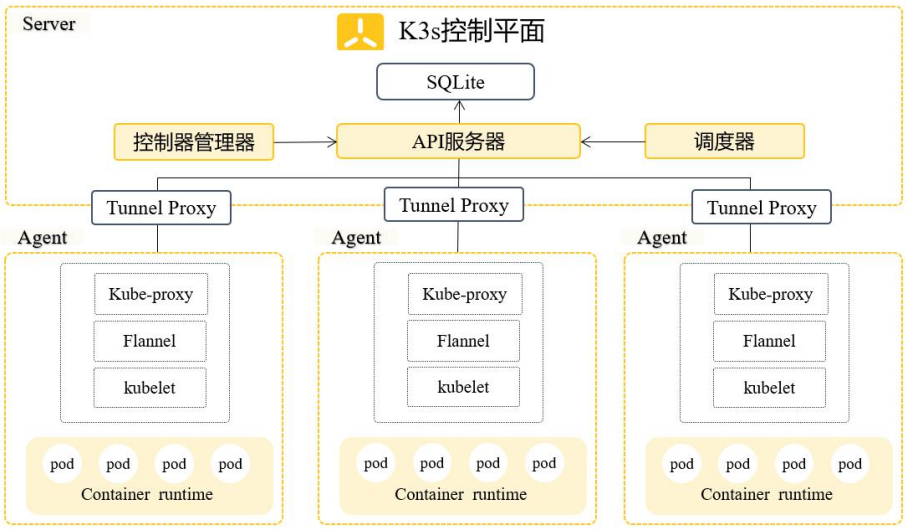
Server
K3s控制平面
SQLite
控制器管理器
API服务器
调度器
Tunnel Proxy
Tunnel Proxy
Tunnel Proxy
Agent
Agent
Agent
Kube-proxy
Flannel
kubelet
Kube-proxy
Flannel
kubelet
Kube-proxy
Flannel
kubelet
pod pod pod pod
Container runtime
pod pod pod pod
Container runtime
pod pod pod pod
Container runtime

图 3：K3s 技术架构

5. KubeEdge Data Analysis Demo

5.1 软件设计思路

5.1.1. 软件包版本

Kernel version: 4.15.0-121-generic

Kubelet version: v1.19.8

KubeEdge version： v1.6.1

Golang version: go1.16.2 linux/amd64 for Ubuntu 18.0.4

Docker version: v20.10.7

5.1.2. 业务逻辑思路

本实例环境在公有云上，通过 Kubekey 工具搭建集群，并添加两个不在同一局域网内的边缘节点，边缘节点主机名分别为 edgenode-mph、edgenode-pi。整套业务逻辑是使用 Apache Beam 来进行数据分析，从 mqtt 代理读取模拟设备的温度数据，然后对数据进行筛选，过滤数据发布温度过高告警提示或者过滤数据打印超过设定阈值温度的接收数据。业务部署流程为：首先在云端通过 deployment.yaml 文件部署应用，并利用节点 Selector 机制将容器组 pod 调度至边缘节点 edgenode-mph 上，边缘节点上跑应用部署容器。在云端通过查看应用日可获取数据分析结果。

5.2 运行效果截图

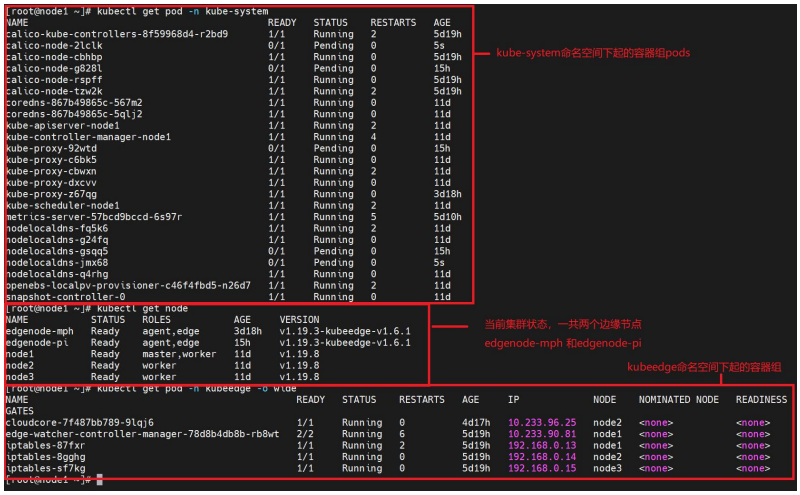

图 4：云端节点集群状态

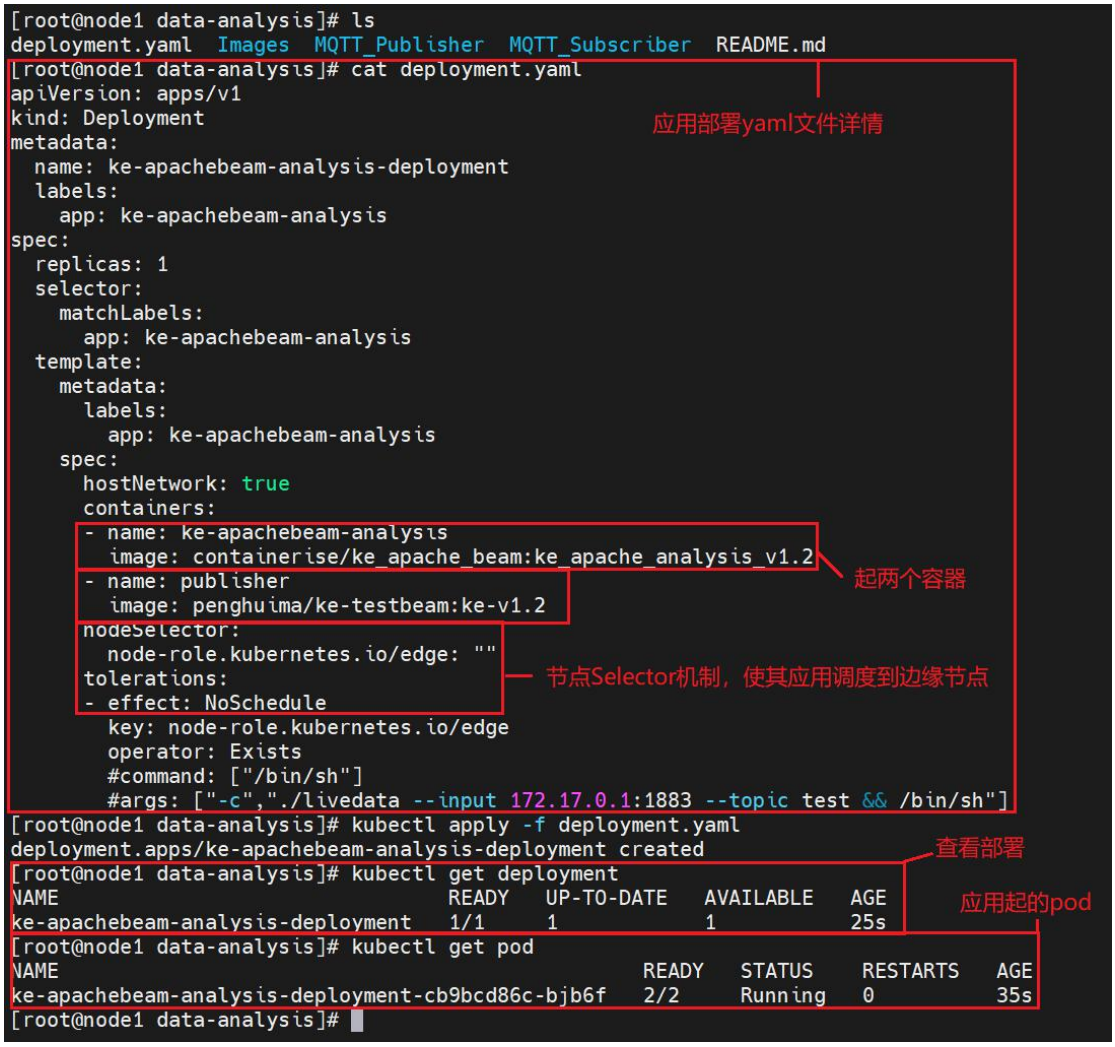

图 5：数据分析应用部署

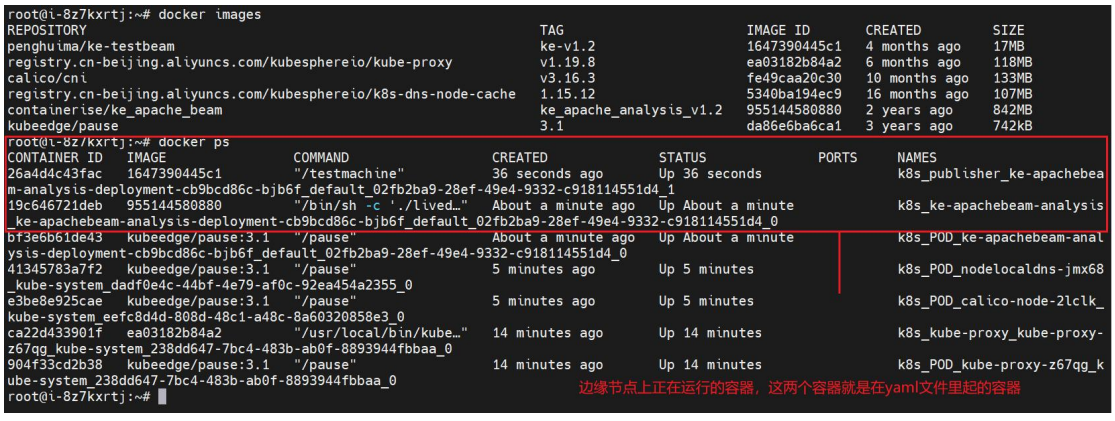

图 6：边缘节点容器运行情况

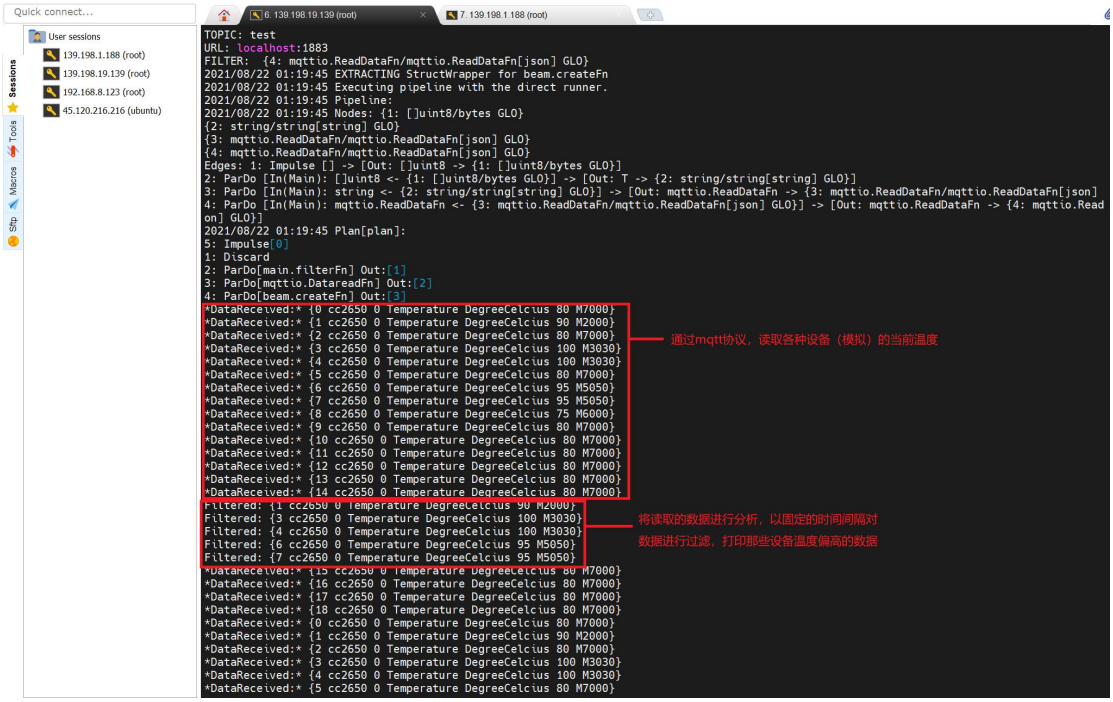

图 7：数据分析运行日志结果

6. KubeEdge Counter Demo

6.1 软件设计思路

6.1.1. 软件包版本

Kernel version: 5.11.0-25-generic

Kubelet version: v1.19.3

KubeEdge version： v1.7.1

Golang version: go1.14.4 linux/amd64 for Ubuntu 21.04

Docker version: v20.10.7

6.1.2. 业务逻辑思路

本实例使用 Vmware 虚拟机模拟各类型节点，通过 kubernetes 搭建集群并初始化主节点，集群规模为一主一从，在 k8s 集群基础上使用 keadm 工具在主节点 master 上部署 Kubeedge cloudcore 服务，使主节点充当云端控制节点，在从节点 node1 上使用 keadm 加入集群并部署 KubeEdge edgecore 服务，使从节点充当边缘节点，编写业务逻辑实现计数器功能，将应用打包成容器通过 KubeEdge 调度机制将计数器容器调度到边缘节点 node1 上运行，在云端控制节点运行 Webcontroler APP 并开放 8089 端口，实现云端 master 节点控制计数器开关，并实时获取边缘节点计数器状态。

6.2 运行效果截图

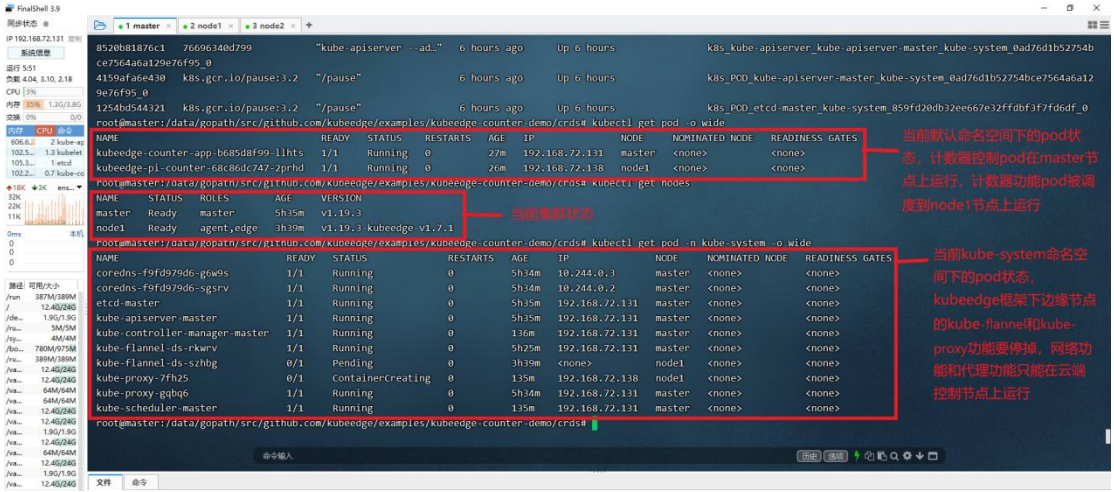

图 8：集群状态

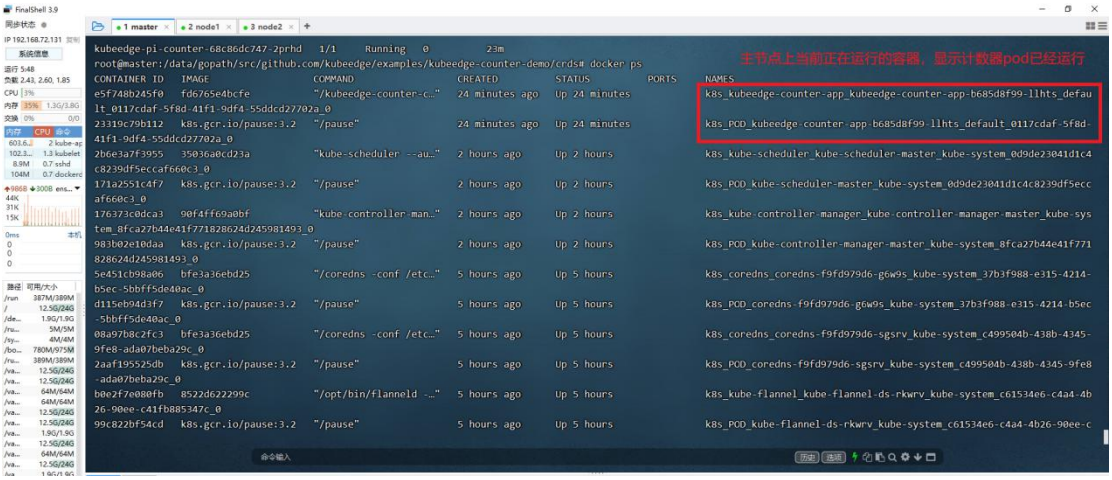

图 9：云端节点容器运行情况

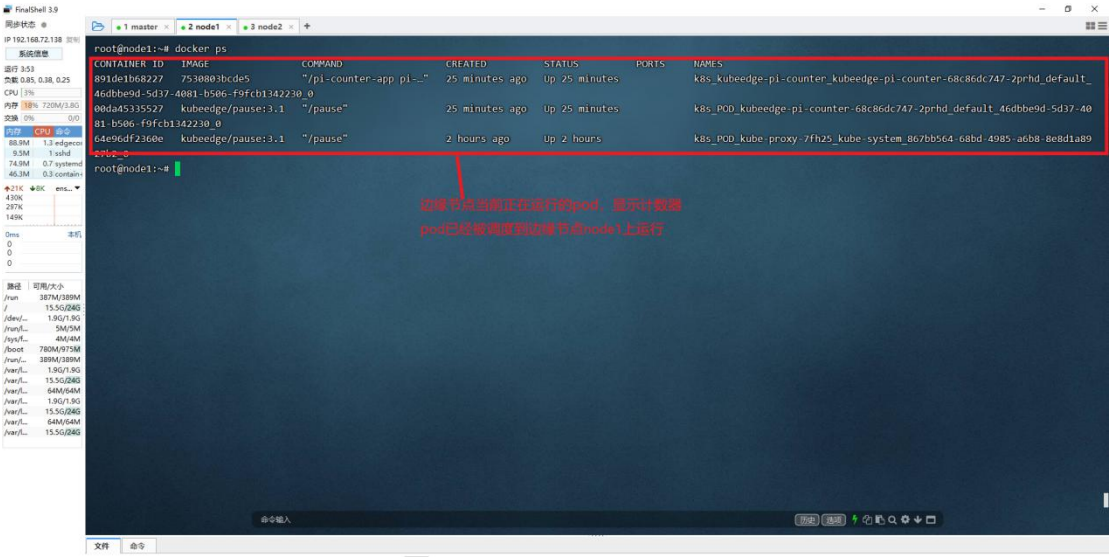

图 10：边缘节点容器运行情况

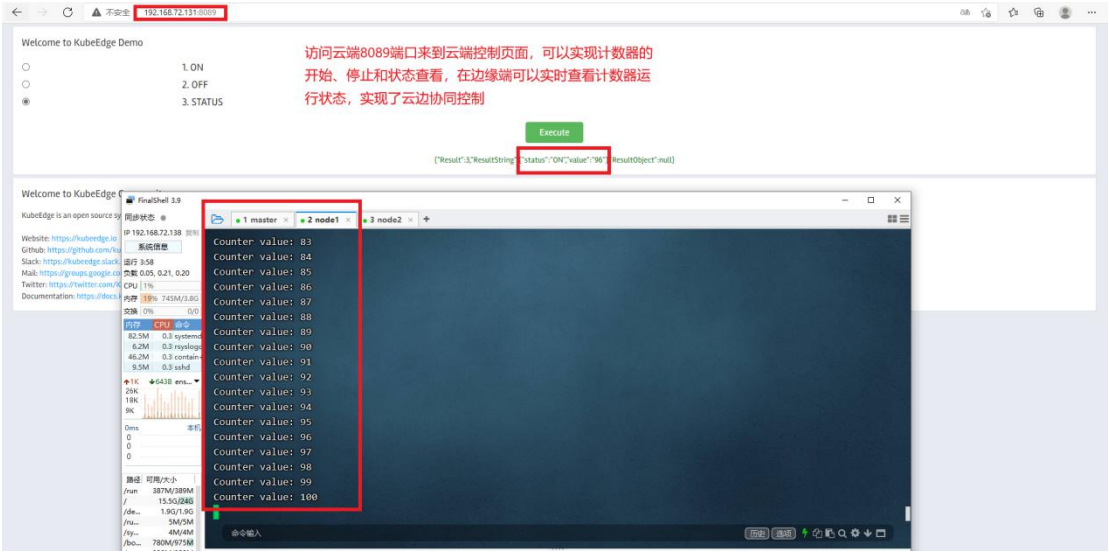

图 11：计数器状态控制与显示

运行效果录屏展示见附件 1。

7. Temperature demo on Raspberry PI

7.1 软件设计思路

7.1.1. 软件包版本

Kernel version: 5.11.0-25-generic

Kubelet version: v1.19.3

KubeEdge version： v1.7.2

Golang version: go1.14.4 linux/amd64 for Ubuntu 21.04，go1.14.4 linux/arm64 for Raspberry PI

Docker version: v20.10.7

7.1.2. 硬件型号与配置

Raspberry PI 4B

CPU(s): 4

Memory: 8G

Disk: 128G

Vendor ID: ARM

Model name: Cortex-A72

CPU max MHz: 1500.0000

CPU min MHz: 600.0000

温度传感器: DHT11

7.1.3. 业务逻辑思路

本实例在之前集群基础上加入树莓派物理节点，通过部署 edgecore 服务充当边缘节点，树莓派连接温度传感器获取实时环境温度数据。业务部署流程为：首先，在云端控制节点创建设备控制模型和实例；其次，在树莓派节点构建本地温度映射镜像；最后，在控制节点上创建温度映射 pod 并通过 nodeSelector 机制将 pod 调度至树莓派节点，利用第二步创建的镜像在本地将容器拉起。容器正常运行后通过 MQTT 将温度数据上传至 edgecore，通过 edgecore 与 cloudcore 的交

互，云端可以通过 kubelet 实时显示温度状态。

7.2 运行效果截图

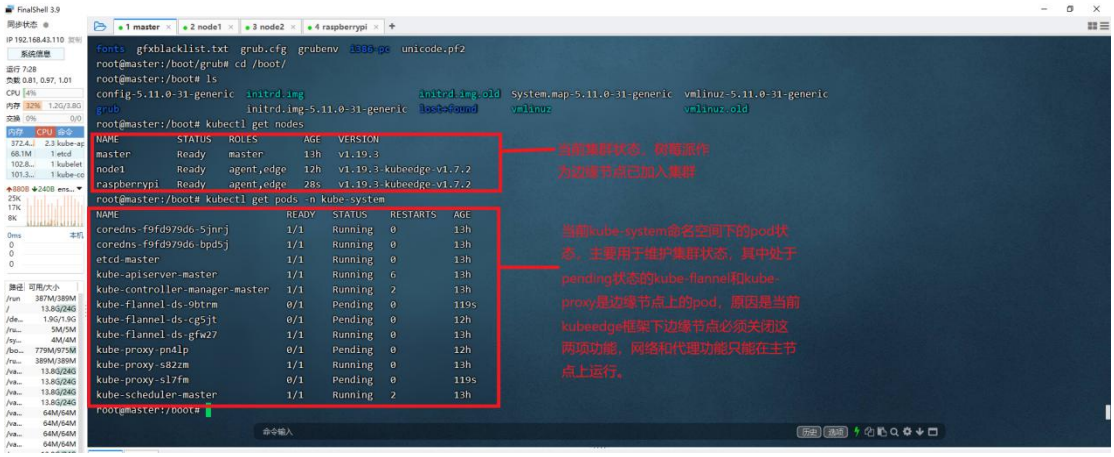

图 12：云端节点集群状态

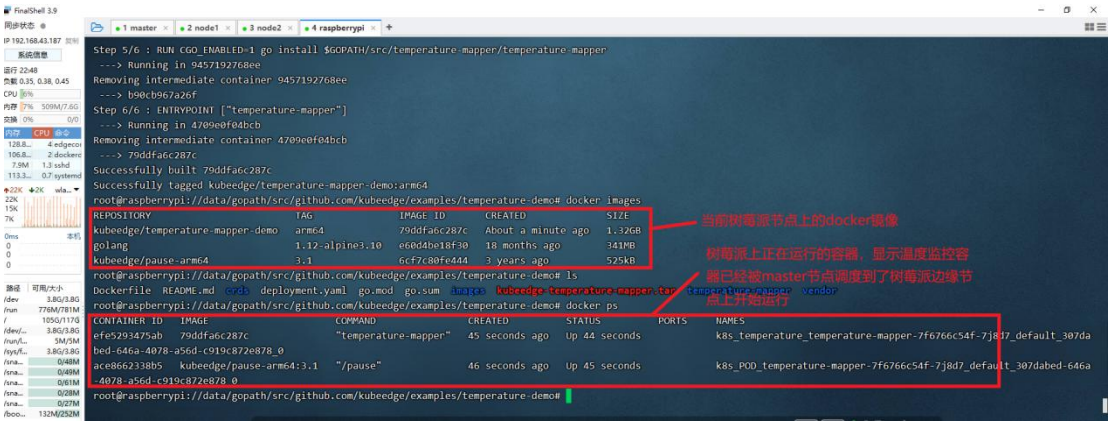

图 13：树莓派节点容器状态

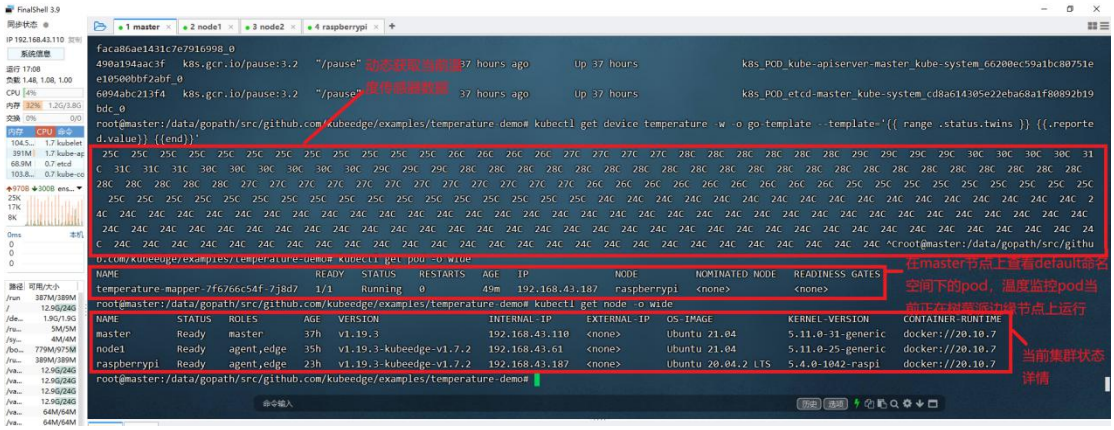

图 14：云端节点温度数据显示与 pod 详情

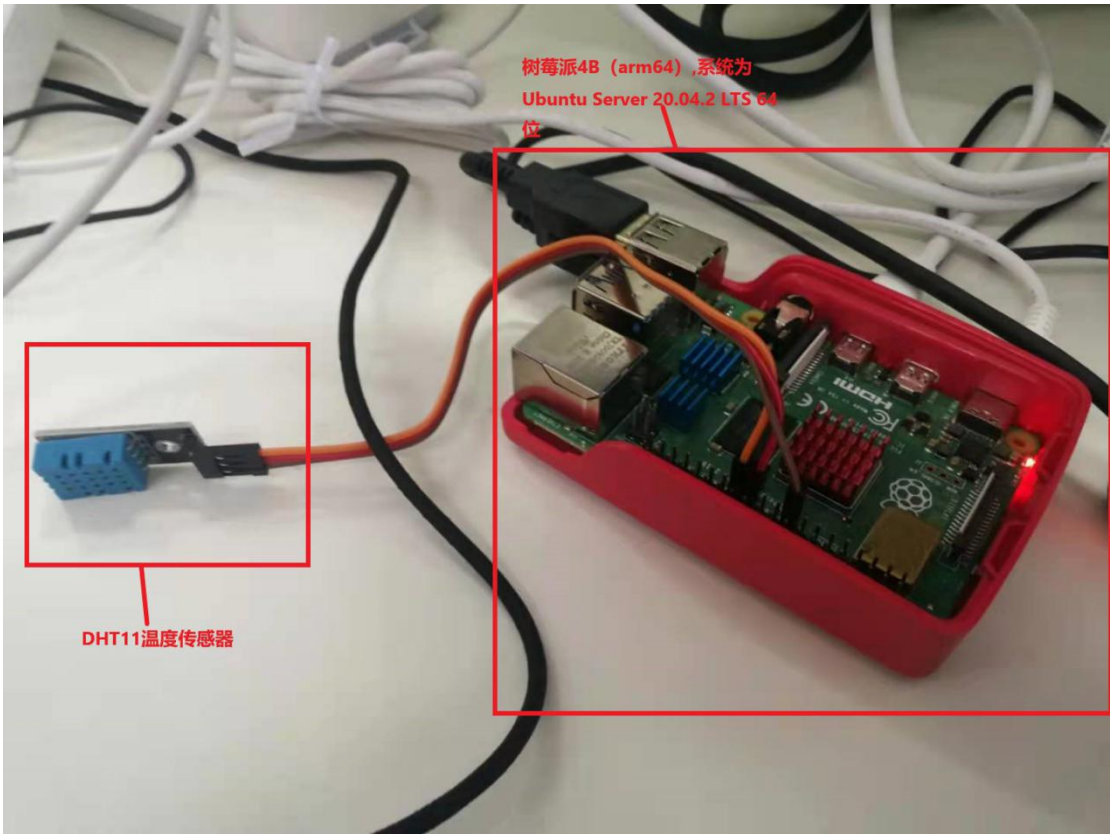

图 15：Raspberry PI 和 DHT11 温度传感器

运行效果录屏展示见附件 2。

本示例的详细部署过程已形成单独文档《Temperature demo on Raspberry PI 部署实例》一文，详见：

https://github.com/penghuima/kubeedge-demo/tree/main/doc

8. 分工与总结

8.1 人员分工

本次任务由万天娇、马朋辉、李宁、龚成共同完成，分工如下：

万天娇（20023124）：完成边缘计算研究现状分析，主要包括论文搜集、阅读并分析整理形成报告的第 1、2、3 部分。

马朋辉（20023108）：在 Github 上新建项目，对成员工作进行指派和维护，完成 Kubernetes、KubeEdge 和 K3S 原理分析，形成报告第 4 部分，并在青云公有云上搭建集群以及 KubeEdge 环境进行实验部署，形成报告第 5 部分。

李宁（20020045）：完成实验部分的工作，主要包括环境安装、硬件购置、集群搭建和应用部署，形成报告第 6、7 部分，并撰写《Temperature demo on Raspberry PI 部署实例》一文，梳理实验过程详细步骤、遇到的问题及解决方法。

龚成（20020085）：搜集成员工作内容，并负责最终实验报告和相关材料的汇总整理。

8.2 总结

通过本次课程的学习，我们对分布式计算领域技术的发展有了一个全面的认识，对相关技术有了初步的了解，为下步课题研究工作指明了方向并奠定了一定基础，尤其是通过最后的大作业使小组成员对边缘计算领域的发展有了更加深入的了解，对 Kubernetes、KubeEdge 和 K3S 等开源软件框架和原理有了直观的认识，通过论文阅读、分析、实验和报告撰写等环节锻炼了实践动手能力，并加深了对理论的理解。任务过程中也感受到自身能力的不足，相关工作并不是十分完善，还有很大的提升空间，这也是下步需要努力的方向。

本次任务的所有代码与相关文档已上传至 Github，地址为：

https://github.com/penghuima/kubeedge-demo

最后感谢丁博、余悦和史佩昌老师的辛勤付出！

附件 1：kubeedge-counter-app.mp4

附件 2：Raspberry PI temperature-demo.mp4

参考文献

[1] Z.Zhao, Z.Jiang, N.Ling et al.,“ECRT: An Edge Computing System for Real-Time Image-based Object Tracking,” in Proc.the 16th ACM Conference on Embedded Networked Sensor Systems (SenSys 2018), 2018, pp. 394–395.

[2] Z.Zhao, K.M.Barijough, and A.Gerstlauer,“DeepThings: Distributed Adaptive Deep Learning Inference on Resource-Constrained IoT Edge Clusters,” IEEE Trans. Comput. Aided Des. Integr. Circuits Syst., vol. 37, no. 11, pp. 2348–2359, Nov. 2018.

[3] S.Teerapittayanon, B. McDanel, and H.T.Kung,“Distributed Deep Neural Networks over the Cloud, the Edge and End Devices,” in IEEE 37th International Conference on Distributed Computing Systems (ICDCS 2017), 2017, pp. 328–339.

[4] L.Valerio, A. Passarella, and M. Conti, “A communication efficient distributed learning framework for smart environments,” Pervasive Mob. Comput., vol. 41, pp. 46–68, Oct. 2017.

[5] S.Rathore, J. H. Ryu, P. K. Sharma, and J. H. Park, “DeepCachNet: A Proactive Caching Framework Based on Deep Learning in Cellular Networks,” IEEE Netw., vol. 33, no. 3, pp. 130–138, May 2019.

[6] Z. Chang, L. Lei, Z. Zhou et al., “Learn to Cache: Machine Learning for Network Edge Caching in the Big Data Era,” IEEE Wireless Commun., vol. 25, no. 3, pp. 28–35, Jun. 2018.

[7] D. Adelman and A. J. Mersereau, “Relaxations of weakly coupled stochastic dynamic programs,” Operations Research, vol. 56, no. 3, pp. 712–727, 2008.

[8] H. Zhu, Y. Cao, W. Wang et al., “Deep Reinforcement Learning for Mobile Edge Caching: Review, New Features, and Open Issues,” IEEE Netw., vol. 32, no. 6, pp. 50–57, Nov. 2018.

[9] K. Guo, C. Yang, and T. Liu, “Caching in Base Station with Recommendation via Q-Learning,” in 2017 IEEE Wireless Communications and Networking Conference (WCNC 2017), 2017, pp. 1－6.

[10] C. Zhong, M. C. Gursoy et al., “A deep reinforcement learning-based framework for content caching,” in 52nd Annual Conference on Information Sciences and Systems (CISS 2018), 2018, pp.

[11] S. Yu, X. Wang, and R. Langar, “Computation offloading for mobile edge computing: A deep learning approach,” in IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC 2017), 2017, pp. 1－6.

[12] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, “Deep Learning Empowered Task Offloading for Mobile Edge Computing in Urban Informatics,” IEEE Internet Things J., vol. 6, no. 5, pp. 7635－ 7647, Oct. 2019.

[13] H. Rahimi ,Y. Picaud, K. Deep Singh, G. Madhusudan, S. Costanzo , and O. Boissier, “Design and Simulation of a Hybrid Architecture for Edge Computing in 5G and Beyond” IEEE Trans. Comput, VOL. 70, NO. 8, AUGUST. 2021.

[14] B. Gao, Z. Zhou, F. Liu, F. Xu, and B. Li, “An Online Framework for Joint Network Selection and Service Placement in Mobile Edge Computing” IEEE Trans. Comput. DOI 10.1109/TMC.2021.

[15] H. Wang, H. Xu, H. Huang, M. Chen, S. Chen, “Robust Task Offloading in Dynamic Edge Computing”in Transactions On Mobile Computing, VOL. NO. OCT. 2020.

[16] J. G, C. Stewart, J. Chumley, and S. Zhang, “Managing Edge Resources for Fully Autonomous Aerial Systems” in EC’19: Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, 2019, pp. 101－112.

[17] S.Y. Nikouei, Y. Chen, A. Aved, E. Blasch, and T.R. Faughnan, “I-SAFE: instant suspicious activity identification at the edge using fuzzy decision making”, in EC ’19: Proceedings of the 4th ACM/IEEE Symposium on Edge Computing, 2019, pp. 101–112.