迈向可扩展联邦容器编排：CODECO 方案

Rute C. Sofia 1, Josh Salomon[2], Ray Carrol[2], Luis Garc´es-Erice [3], Peter Urbanetz[3], 焦耳¨urgen Gesswein[4], Rizkallah Touma[5], Alejandro Espinosa[5], Luis M. Contreras[6],Vasileios Theodorou[7],George Papathanail[7], Georgios Koukis[8], Vassilis Tsaoussidis[8], Alberto del Rio[10],David Jimenez[10],Efterpi Paraskevoulakou[12], Panagiotis Karamolegkos[12], John Soldatos[15],Borja Dorado Nogales[16],和 Alejandro Tjaarda[16]

[1]fortiss GmbH，德国[2]红帽，以色列[3]IBM 欧洲研究院—苏黎世，瑞士[4]西门子股份公司，德国[5]i2CAT 基金会，西班牙[6]Telef´onica，西班牙[7]Intracom Telecom，希腊[8]色雷斯德谟克利特大学 / 雅典娜研究中心，希腊[10]Universidad Polit´ecnica de Madrid，西班牙[12]比雷埃夫斯大学研究中心，希腊[15]Netcompany，卢森堡[16]马德里卡洛斯三世大学，西班牙

arXiv:2601.13351v1 [cs.DC] 19 Jan 2026

2026年1月21日

摘要

本文介绍了 CODECO，这是一个面向 Kubernetes 的联邦式编排框架，旨在解决以云为中心的部署所面临的局限。CODECO 采用数据—计算—网络协同编排方法，以支持异构基础设施、移动性和多提供商运行。

CODECO通过语义应用模型、基于分区的联邦以及人工智能辅助决策支持扩展Kubernetes，从而能够在联邦环境中对应用及其微服务进行上下文感知放置和自适应管理。混合治理模型将集中式策略执行与去中心化执行和学习相结合，在保持全局一致性的同时支持远端边缘自治。本文介绍了CODECO的架构和核心组件，概述了具有代表性的编排工作流，并提出了一个基于软件的实验框架，用于在联邦边缘—云基础设施环境中进行可复现评估。

关键词。关键词：边缘-云；编排；Kubernetes；人工智能/机器学习；异构网络；数据可观测性。

1 引言

“云–边缘–物联网 *(CEI)*连续体”正在重塑计算和数据处理的部署方式，推动功能向用户和设备靠近，以满足对低延迟、韧性、可持续性和上下文适应的严格要求。智能出行、工业自动化和沉浸式媒体等新兴应用日益依赖于在异构且往往间歇性连接的环境中进行分布式执行。

尽管经历了这些演进，大多数容器编排技术仍然主要以云为中心。Kubernetes (K8s)[1]，虽然得到广泛采用，但它假设集群相对稳定、采用集中式控制，并且位于单一管理域内的同质基础设施之上。在远边缘和多提供商场景中，这些假设变得脆弱，因为连接性不稳定、资源受限，而且治理边界各不相同。

在 CEI 环境中，有三项局限性尤为突出。首先，现有工具的跨层上下文感知能力有限，其编排决策主要由以计算为中心的指标驱动，同时对网络状况和数据约束考虑不足。其次，现有编排器建立在集中式控制假设之上，而这些假设在移动性、间歇性连接以及远端边缘的自主运行场景下并不成立

[1]https://kubernetes.io/

。第三，它们缺乏用于多提供商协作和受控信息交换的联邦抽象。

CODECO（认知去中心化边缘–云编排）[2] 通过为 Kubernetes 增添联邦编排能力、语义应用抽象和人工智能辅助决策支持，弥补了这些不足，详情见第 2 节和第 4 节。CODECO 采用数据—计算—网络协同编排范式，以实现分布式 CEI 基础设施中微服务的上下文感知放置和自适应管理。

本文从系统和架构角度阐述了基于 CODECO 联邦集群（*FC*）框架的联邦 Kubernetes 编排。其主要贡献包括：(i) 为云–边缘–物联网连续体设计并构建有界、基于邻域的联邦编排框架；(ii) 详细介绍结合应用意图、跨层可观测性和人工智能辅助决策支持的运维工作流；以及 (iii) 提供一个开源实验框架，以支持联邦 CEI 编排的可复现部署和评估。

本文其余部分安排如下。第 2 节介绍 CODECO 的设计原则与范围，包括其数据—计算—网络协同编排模型，以及联邦与治理方法。第 3 节探讨 CODECO 如何将上下文感知和能耗感知（绿色化）目标融入 CEI 连续体中的编排决策。第 4 节详细介绍 CODECO 的联邦架构，阐述其核心组件、管理层与时间尺度、应用邻域以及语义应用抽象。第 5 节聚焦 CODECO 组件的运行职责与运行时交互，包括学习辅助决策支持、网络管理、调度和元数据处理。第 6 节介绍具有代表性的工作流，涵盖平台安装、联邦应用部署和运行时自适应。第 7 节介绍 CODECO 的实验策略，以及用于可复现评估的软件定义测试床 CODEF。第 8 节探讨 CEI 连续体联邦编排面临的开放研究挑战与未来方向。第 9 节综述相关工作，并与现有多集群和联邦编排框架进行比较分析。最后，第 10 节总结全文并概述主要贡献。

[2]https://gitlab.eclipse.org/eclipse-research-labs/codeco-project

2 CODECO 设计原则与范围

本节介绍指导 CODECO 架构与运行的基础设计原则，重点阐述该框架的概念范围，以及塑造其编排逻辑的系统假设。具体而言，CODECO 有别于云中心编排，体现在：（i）采用数据–计算–网络协同编排范式，将网络、数据可观测性和计算视为一等调度输入；（ii）定义联邦治理模型，在集中式策略控制与远端边缘的去中心化执行与学习之间取得平衡。这些原则构成第 4 节所述架构实现的概念基础。本节余下部分将进一步阐述 CODECO 所依据的协同编排模型，以及其分层联邦、自主性与治理方法。

2.1 跨数据、计算与网络的协同编排

CODECO 的核心原则是，在 CEI 环境中实现有效编排，需要全面考量基础设施和应用。因此，CODECO 从三个相辅相成的视角对系统进行建模：网络、数据可观测性和计算。编排决策会综合考虑这三个视角，从而实现符合端到端应用需求（而非孤立资源指标）的放置与适应策略。

图1展示了这种集成方法：通过共同观测和评估应用数据流、网络路径及计算资源，支持在整个 CEI 连续体中做出明智的编排决策。

首先，网络视角涵盖了在边缘—云环境中部署和互连容器化应用所需的底层网络和覆盖网络连接。这不仅包括延迟和带宽等经典服务质量指标，还包括与移动性和动态拓扑变化相关的属性。

其次，数据可观测性视角捕捉与微服务相关的数据依赖关系、局部性约束和数据流特征。这些属性对于初始部署以及重新部署和自适应都至关重要，尤其适用于数据密集型和对延迟敏感的应用。

第三，计算视角 描述可用节点的能力和约束，包括处理能力、内存和能耗

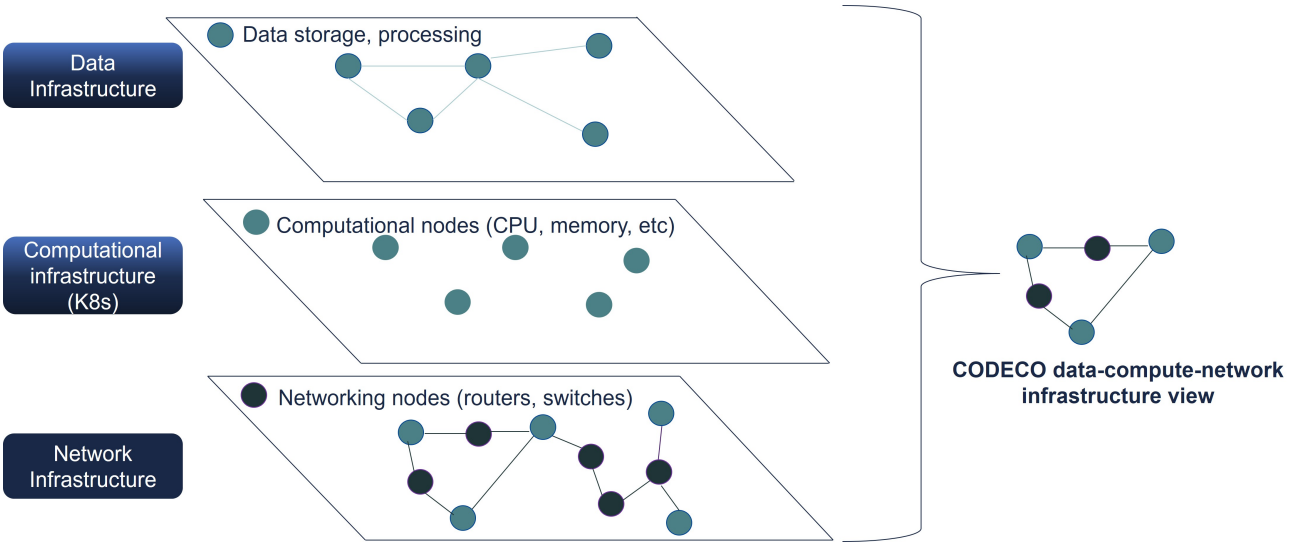

图1：CODECO 面向 CEI 上应用及其微服务部署的数据-计算-网络方法。通过观测数据工作流、网络和计算资源，CODECO 能够更细粒度地审视物联网-边缘-云基础设施，从而更好地支持跨边缘-云的应用编排。

相关属性，以便为应用组件确定合适的执行环境。

这种综合视角 [1]体现了 CEI 应用日益增强的移动性、异构性和数据密集性；在进行放置决策时，必须统筹考虑计算、网络和数据方面的约束，而非孤立地考量资源指标。

2.2 联邦、自治与治理模型

CODECO 采用了适用于联邦 CEI 环境的分层治理模型。在这些环境中，基础设施横跨多个管理域，且连接状况会随时间变化。CODECO 不依赖完全集中式或完全去中心化的控制，而是定义了一种混合管理模型，将治理、协调和执行职责分开。这种职责分离既能确保全局策略的一致性，又能保留远端边缘的本地自治和韧性。

治理层定义系统的权威期望状态。它负责表达应用意图、执行组织和监管策略，并维护整个联邦的可审计性。在 CODECO 中，治理集中于中心集群，并利用*Open Cluster Management*（*OCM*）[3], 提供的 Kubernetes原生联邦机制，在各集群间建立共享的控制与治理基础。OCM 提供了定义明确的机制，用于集群注册、身份管理、策略传播和应用生命周期协调，采用中心辐射式**架构**。这些能力对于在多提供商环境中执行组织策略、安全约束和合规要求至关重要。第 4 节所述的架构设计体现了间歇性连接和远端边缘自治所带来的影响。

协调层通过限制编排和信息交换的范围，确保可扩展性和效率。CODECO 将应用**邻域** 定义为边界明确的协调域，其中包含与特定应用相关的一部分托管集群。邻域规定了监控数据交换、学习组件协作以及调度决策优化的范围。通过将协调限制在特定应用的集群子集中，CODECO 避免了全局状态的传播，同时仍能在分布式环境中实现协作。

执行层位于受管理的集群中，负责利用本地可用的上下文来落实放置、迁移和网络配置决策。该层自主运行，即使与 Hub 的连接性能下降或暂时不可用，也能继续工作。该层的调度、自适应和学习组件会使用集中定义的意图和策略，同时依赖最新本地观测及时做出决策。这种设计确保远端边缘具备韧性和响应能力，并在连接恢复后继续朝着预期的全局状态收敛。

[3]https://open-cluster-management.io/

3 上下文感知与能耗管理

在现有的容器编排解决方案中，副本和扩缩容决策通常由一组有限的触发条件驱动，例如系统级指标（如 CPU 利用率或能耗阈值）、应用级约束或静态网络策略。虽然这些机制在相对稳定的云环境中行之有效，但对于高度动态的边缘-云场景而言却不够充分，因为资源可用性、连接性和执行情境都可能迅速变化。

为应对这些挑战，CODECO 将上下文感知融入卸载与调度流程。上下文感知是指系统利用有关其运行环境的知识来调整自身行为并采取适当行动的能力 [2, 3]。情境信息涵盖多个维度，包括计算情境、用户情境和环境情境。

上下文感知编排的先决条件是识别相关的情境类别、定义合适的模型来表示这些类别，并将这些模型整合到编排工作流中。CODECO 考虑以下几类上下文感知指标 [4]:

- **计算指标**，例如 CPU 和内存利用率，这些指标通过 Prometheus 从 Kubernetes 环境中收集，并通过 ACM 提供。

- **网络指标**，用于描述功能性服务质量属性（例如延迟、带宽），以及能耗等非功能性方面。

- **应用指标**，用于描述应用级服务质量和非功能性需求，包括时延界限、敏感度级别、认证和合规约束。

- **数据可观测性指标**，用于描述新鲜度、信息年龄、局部性和合规性等数据属性。

- **用户行为指标** 反映与远端边缘运行相关的用户上下文，包括用户位置、移动性、偏好以及社会或物理邻近性 [5, 6]。

这些上下文维度通过 CODECO 的可观测性流水线提供，并应用于应用生命周期的不同阶段，详见第 3.2 节。

3.1 能源感知与绿色度目标

CODECO 中的能源感知通过**绿色度**这一概念体现，它用于描述节点和集群层面的可持续性目标，例如降低能耗和减少 CO_2 足迹。CODECO 并不规定单一的定义，而是允许通过可定制指标来定义绿色度，并将其与其他性能目标结合起来，包括延迟、韧性和数据局部性。

绿色度指标被视为优化准则，而非硬约束，因此可以根据应用意图和运行情境动态进行权衡。

在当前实现中[4], ，绿色度通过以下能耗相关指标量化:

- **节点能耗 ($n_e(i)$)**：节点i 因所有活动进程而消耗的能耗。

- **链路能耗 ($l_e(i,\ j)$)**：节点 i 通过出口链路 j传输数据时消耗的能量。

- **网络能耗 ($L_e(i)$)**：节点 i所有出口链路的总能耗，定义为 $L_e(i) = \sum_j l_e(i,\ j)$。

- **流能耗 ($F_e(i)$)**：与源自节点 i 的应用数据流相关的能耗。

CODECO 支持定义和集成其他由用户自定义的绿色度指标，如第 4 节所述。

3.2 对编排决策的影响

具备上下文感知和能耗感知能力的指标会在应用生命周期的多个阶段直接影响 CODECO 的管理操作:

- **初始部署**，其中节点和集群的选择会考虑当前的资源可用性、网络状况和能耗特征。

- **运行时适应**，上下文发生变化（例如移动性、拥塞或能量耗尽）时，可能会触发工作负载迁移、重新配置或重新调度。

- **邻域重新评估**，当性能持续下降或运行不稳定时，可能需要重新评估与某个应用相关联的协调范围。

[4]请参阅 Eclipse CODECO GitLab:
https://gitlab.eclipse.org/eclipse-research-labs/codeco-project

•符合策略的优化，根据应用意图和治理要求，权衡延迟、韧性和绿色度，解决其中的取舍问题。

4 CODECO 联邦架构

CODECO 联邦架构旨在支持跨 CEI 连续体的以应用为中心的编排，其中基础设施资源涵盖异构的计算、网络和数据环境，且通常由不同的租户和管理域运营。CODECO 不会将集群视为彼此隔离的执行单元，而是将应用及其微服务视为一等实体，使其能够随时间推移在多个集群之间分布、迁移和调整。

4.1 核心架构组件

4.1.1 管理功能与组件映射

图 2 展示了 CODECO 组件及其高层交互：

• **ACM-FC（高级配置与管理 FC）**是面向用户的主要组件。它管理应用生命周期，通过语义抽象 *CODECO*应用模型（*CAM*）解析应用意图，协调邻域创建，并与底层联邦进行交互。

• **SWM-FC（无缝工作负载迁移FC）**负责跨集群的工作负载放置、迁移和重新调度。它根据应用约束、基础设施条件以及学习组件提供的推荐来执行调度决策。

•PDLC-FC（隐私去中心化学习与情境感知）提供人工智能辅助的情境感知推荐。它分析跨层指标（计算、网络、数据），并生成成本和稳定性指标，而不直接执行放置决策。

•NetMA-FC（网络管理与适配）提供联邦编排所需的网络可观测性、安全的集群间连接以及网络能力开放。

• **MDM-FC（元数据管理）**维护一个元数据图，用于记录数据局部性、新鲜度、可移植性和合规约束，从而支持数据感知编排决策。

每个组件负责编排的一个特定方面，并承担表 1 所列的具体管理职责。

表1：管理功能与组件映射。

管理功能	**CODECO 组件**
意图与策略管理	ACM-FC (CAM), OCM Hub
范围与联邦-化管理	邻域组合器（CNC），ACM-FC 邻域管理器
决策 and 感知	支持上下文 PDLC-FC (CA, 多智能体强化学习) 图神经网络,
调度 and 迁移	强制执行-ment SWM-FC（SWM-L1，SWM-L2）
网络配置 and 连接性	Con- NetMA-FC（NSM，Nemesys-FC、安全连接）
Data 与数据 ness	治理 感知型- MDM-FC（元数据图，连接器、API）
可观测性与反馈	ACM-FC, NetMA-FC, MDM-FC

4.2 管理层与决策时间尺度

CODECO 在三个时间层中组织管理决策：(i) 治理层以较长时间尺度运行，用于定义意图、策略和联邦成员资格；(ii) 协调层以中等时间尺度运行，通过应用邻域限定编排范围；(iii) 执行层以较短时间尺度运行，利用最新的本地上下文实施放置、迁移和网络配置。

如表1所总结，CODECO 组件根据其功能角色和时间要求分布在这些层中。ACM-FC 和 OCM 等中心辐射式组件主要运行于治理层，而 PDLC-FC 和 SWM-FC 横跨协调层与执行层，从而在不破坏全局策略的情况下实现自适应行为。

4.3 联邦与控制结构

CODECO 以 OCM 为联邦协调的基础。最小部署由一个*Hub* 集群和一个或多个受管集群 *(MCs)*组成，如图3所示。OCM 提供集群注册、身份管理、策略分发和清单分发，从而在整个联邦中建立通用的治理

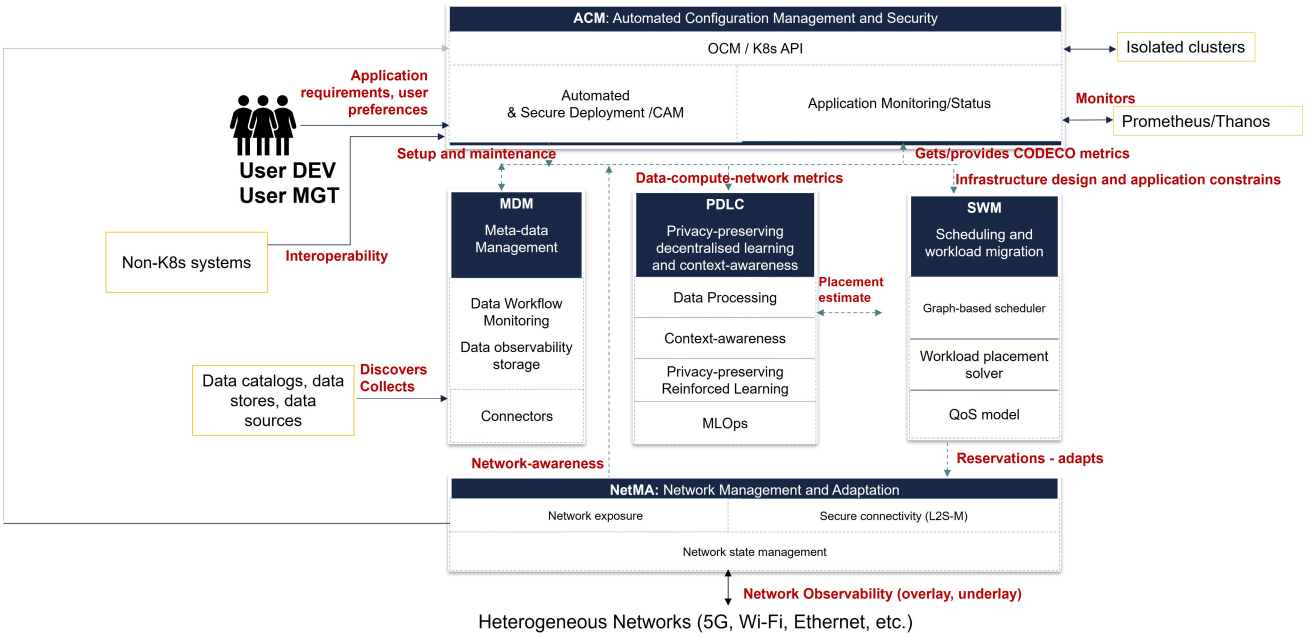

图 2：CODECO 组件及其高层交互。

层。

CODECO 不要求 Hub 与所有受管集群之间保持持续的低延迟连接。为支持具有移动性和间歇性连接特点的远端边缘环境，CODECO 采用混合控制平面模型。Hub 仍是期望状态和策略执行的权威来源，而受管集群则保留足够的自治能力，可在与 Hub 断连期间继续进行微服务放置、重新调度和集群间通信。这种设计既保持了治理的一致性，又能支持弹性运行。

4.4 Application 邻域：联邦分区

为控制复杂性并提升可扩展性，CODECO 引入了应用邻域这一概念。邻域是根据特定 Application 的需求、性能目标和上下文约束，为该 Application 选定的、边界明确且具有语义意义的受管集群分组。每个 Application 仅关联一个邻域，而一个托管集群可以属于多个邻域。邻域定义了调度、监控数据交换、协作学习以及集群间通信的协调范围。通过将协调限制在与 Application 相关的集群子集中，CODECO 避免了全局状态传播，降低了开销，并提升了可扩展性。邻域在部署时计算得出，正常运行期间预期保持稳定；不过，持续的性能下降、资源耗尽或环境不稳定可能触发重新评估。

为进一步降低编排开销，CODECO 将空间划分 与邻域选择解耦。可以定期（例如离线或周期性地）对受管集群进行基于相似性的聚类，以建立粗粒度分区；而特定于 Application 的邻域选择则仅限于部署时最相关的子集。这种分离显著降低了计算和通信成本，同时保留了对变化条件的适应性。

4.5 语义应用抽象与可观测性

用户仅通过 ACM-FC 与 CODECO 交互，提交使用 CAM 描述的应用。CAM 是一种基于 YAML 的语义抽象，用于指定微服务、服务依赖关系以及延迟、能源效率、韧性和合规性等性能意图。CAM 刻意不限定应用是部署在单个集群内还是跨多个集群部署，因此可以从本地执行无缝迁移到联邦执行。

CAM 的规范部分定义应用结构和要求，而状态部分则由 ACM-FC 持续更新，以提供运行时可观测性信息。

4.6 联邦组件放置

CODECO 组件根据各自的功能角色分布在联邦中。这种部署方式确保了集中式治理，同时让监控、学习和执行都靠近各组件所管理的资源：

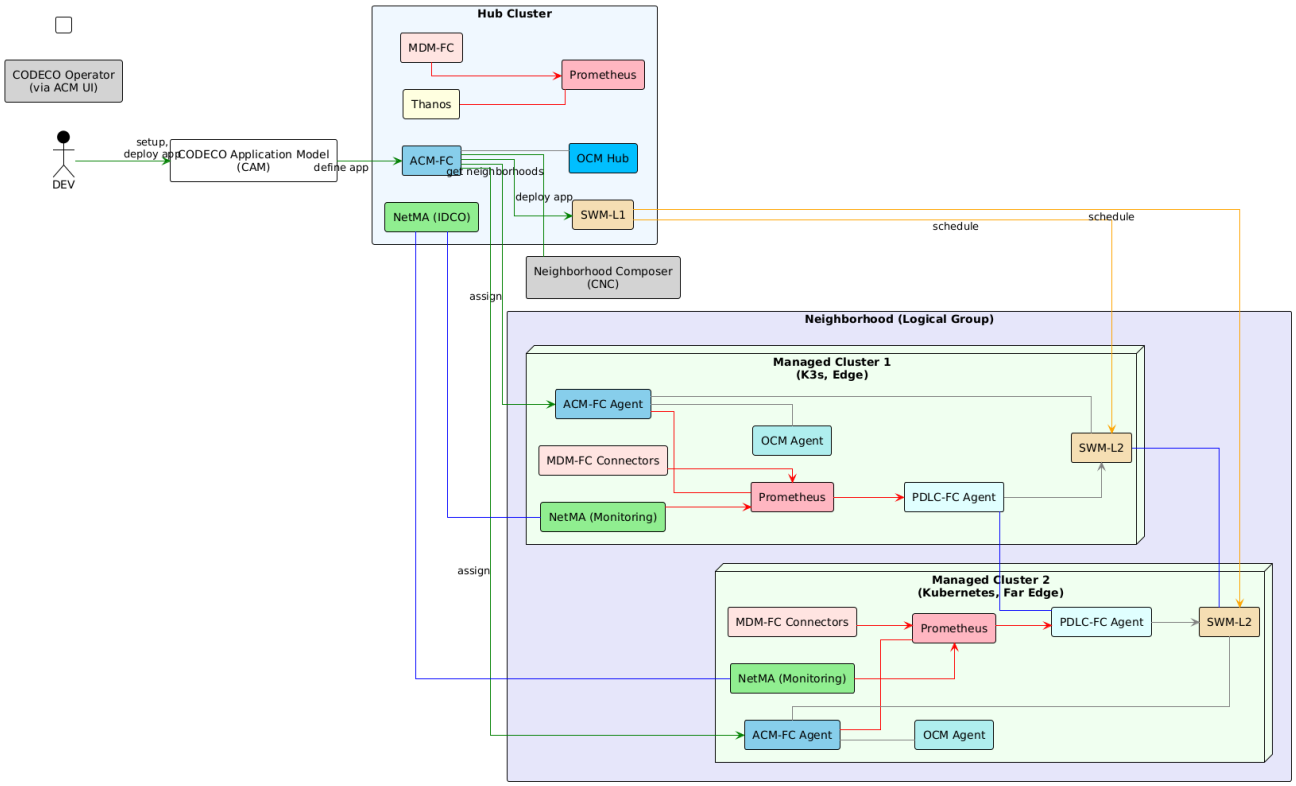

图3：基于 OCM 中心辐射模型的 CODECO 组件及其在联邦集群中的布局。绿色箭头表示用户与 CODECO 逻辑之间的交互；红色箭头表示 CODECO 可观测性接口。橙色箭头表示 CODECO 组件之间中心到辐射端的控制平面通信；蓝色箭头表示 CODECO 组件之间通过受管集群控制平面进行的通信。

- **ACM-FC** 运行于中心集群中，并在每个 MC 中部署轻量级代理。
- **PDLC-FC** 实例运行在属于某个邻域的受管集群的控制平面中。
- **SWM-FC** 分为托管在 Hub 中的全局协调层（SWM-L1）和部署在每个受管集群中的本地执行层（SWM-L2）。
- **NetMA-FC**覆盖整个联邦，其中监控代理靠近工作负载部署，域间连接则由 Hub 层级进行编排。
- **MDM-FC** 通常按邻域实例化，以管理与 Application 相关的元数据。

4.7 组件交互与人工智能驱动的决策流程

通过 CAM 描述并部署 Application 后，ACM-FC 会协调所有 CODECO 组件的激活。NetMA-FC、MDM-FC 和 ACM-FC 收集的监控数据会通过 Prometheus 对外提供。PDLC-FC 汇总这些跨层观测数据，并生成结合上下文的成本和稳定性估算，再将其传递——给 SWM-FC，作为节点和集群建议；这些建议基于特定的性能配置文件和现有资源。

SWM-FC 会结合 PDLC 的推荐、CAM 中描述的应用约束，以及 ACM-FC 和 NetMA-FC 提供的实时基础设施状况。SWM 依靠这些参数，最终制定最优方案，以执行放置、迁移和重新调度决策。网络配置指令会转发给 NetMA-FC，由其建立安全叠加网络并提供更新后的网络指标。MDM-FC 会持续更新数据相关约束和元数据，确保编排决策符合合规性、局部性和数据移动性要求。

5 组件的运行角色

为避免与第 4 节中的架构描述重复，本节仅聚焦于 CODECO 组件的运行时行为、输入、输出及组件间交互。

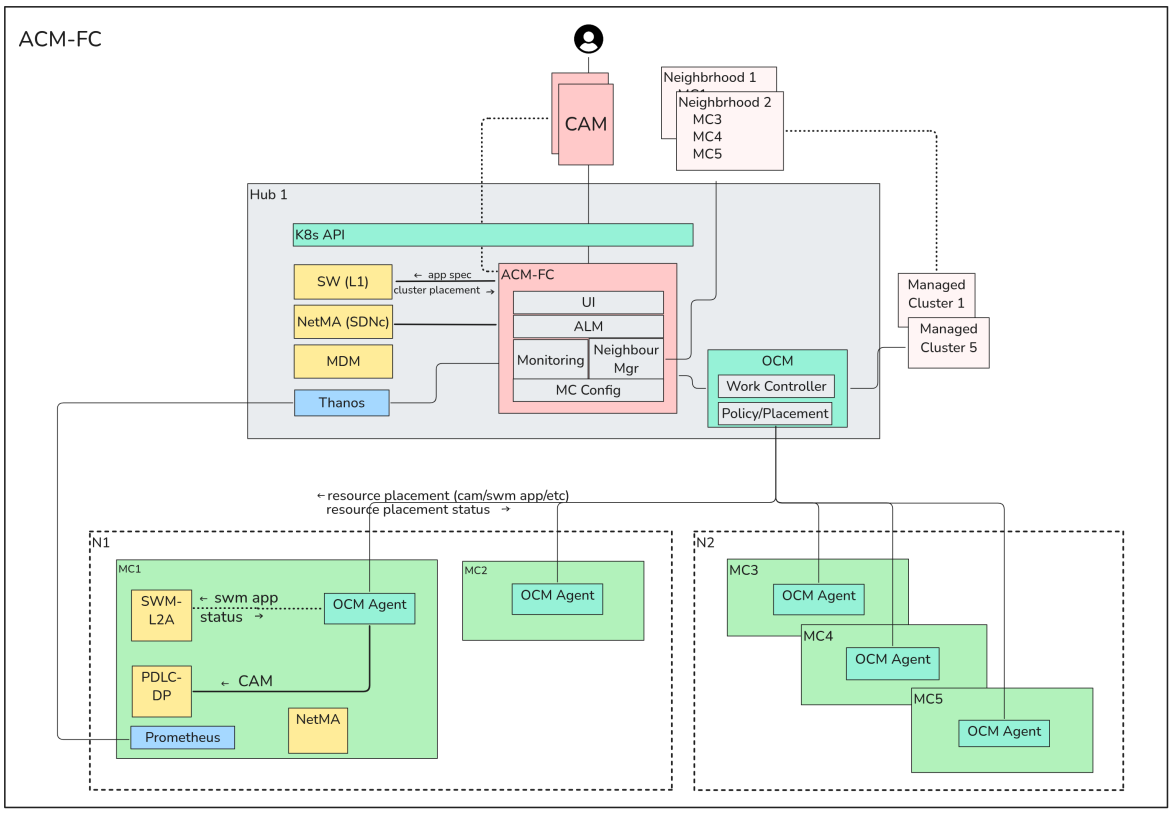

图4：ACM-FC 部署于一个 Hub 和两个托管集群中。

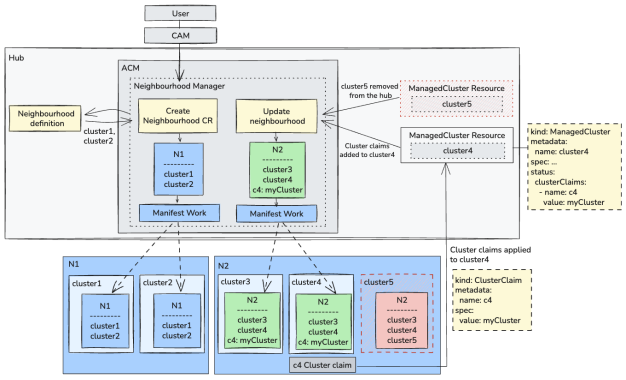

图5：ACM-FC 和邻域管理器。

5.1 CODECO 邻域组合器插件：划分问题空间

联邦 CEI 编排面临的一项关键可扩展性挑战，是在不牺牲适应性的前提下限制搜索与协调空间。CNC 旨在将两项任务分开处理，否则这两项任务在大规模场景下会变得代价高昂：（i）**空间划分**，以较慢的时间尺度将集群归入粗粒度的相似性类别；（ii）**邻域选择**，在部署时为特定应用选择数量有限的候选集群。这样可以减少重复的全局优化，并将监控与学习信息交换限制在与应用相关的集群范围内。

CNC 将粗粒度集群划分（不频繁执行）与部署时针对特定应用的邻域选择分离。给定 CAM 和目标性能配置，它会输出一个有界邻域规范，由 ACM-FC 使用并通过 OCM 传播。

5.2 ACM-FC

ACM-FC 是 CODECO 的治理入口，主要部署在 Hub 集群上。在联邦环境中，ACM-FC 承担三项主要管理职责：

- **联邦范围内的平台配置**，包括接入新集群，以及强制执行 CODECO 组件所需的基线配置。

- **意图与生命周期协调**，通过验证 CODECO CAM，并将应用意图转换为供调度、网络和学习组件使用的内部表示。

- **可观测性集成**，汇总来自联邦的监控信息，并将应用和基础设施状态反馈至 CAM。

在部署时，ACM-FC 会验证 CAM，触发邻域选择（通过 CNC 或基于策略的机制），并协调所选邻域中

CODECO 组件的激活。在运行时，ACM-FC 会维护 Application 的生命周期状态（更新、回滚、移除），并确保以一致的方式向用户公开从联邦收集的监控数据。

如图4所示，ACM-FC 内部由应用生命周期管理器（ALM）、多集群配置（MCC）和邻域管理器组成，它们协同协调 CAM、在各集群中部署 CODECO 服务，并维护邻域定义。

5.2.1 ACM-FC 邻域管理器

如图5所示，邻域管理器 是 ACM-FC 的内部子组件。它的主要职责是管理与应用相关的邻域定义的生命周期。

邻域生成后（由 CNC 或策略生成），邻域管理器会创建相应的 Kubernetes 资源，用于记录成员关系和约束，并通过 OCM *ManifestWork* 将其分发给相关受管集群。这样，其他 CODECO 组件（例如 PDLC-FC 和 SWM-FC）便可在本地发现邻域成员关系，而无需持续与中心集群交互。

邻域管理器通过协调来持续维护邻域一致性。集群生命周期事件（加入或离开）以及相关元数据变更（例如标签或声明）都可能触发邻域资源更新及其重新分发。通过将邻域处理隔离在 ACM-FC 内，CODECO 无需将联邦范围限定逻辑直接嵌入调度或学习组件[5]。

5.3 PDLC-FC

PDLC-FC 通过分析跨层可观测性数据，并生成放置和迁移建议，为联邦编排提供人工智能辅助决策支持。它以去中心化且隐私保护的方式运行，将原始遥测数据保留在各个托管集群本地，并仅在应用邻域内交换受保护的学习产物。PDLC-FC 不会强制执行决策，而是提供由 SWM-FC 使用的成本和稳定性指标。

图6给出了 PDLC-FC 的功能概览。CODECO 强制实现了明确的分离，即将推荐与执行分开：PDLC-FC 生成建议性输出，而微服务（工作负载）的调度和重新调度操作仍由 SWM-FC 负责。因此，去中心化学习并未用作主要的调度机制。这种解耦增强了与感知 Kubernetes 的调度器之间的互操作性，并避免在引入新的目标或约束时重新训练调度逻辑。

在联邦操作中，PDLC-FC 实例运行于受管集群中，且仅与属于同一应用邻域的对等实例交互。这种有界协作符合 CODECO 的隐私、安全和数据主权约束，并限制了通信开销。

PDLC-FC 运行持续的推荐循环，将本地计算、网络和数据指标汇总为成本和稳定性指标，并可选择结合短期预测进行增强；这些指标通过 Kubernetes 资源公开。

5.3.1 PDLC-DP：数据处理

PDLC-DP 负责从 Prometheus 获取跨层指标、对其进行归一化，并维护供 PDLC 子组件使用的本地遥测存储。每个 PDLC-FC 实例仅使用本地收集的数据运行，避免依赖共享存储并维护数据主权。PDLC-DP 还会获取邻域成员关系信息，以确定学习协作和建议传播的范围。

5.3.2 PDLC-CA：上下文评分

PDLC-CA 通过汇总归一化后的指标，生成简洁的节点评分和集群评分，以支持预定义和用户定义的性能配置文件。这些评分基于用户定义的目标配置文件，例如绿色度、韧性（参见第 3 节）。此外，开发人员可以将归一化的 CODECO 指标与新注册的自定义指标相结合，定义自定义配置文件。图 7 展示了此类自定义定义在 PDLC-CA-FC 中的结构及其引用方式。这些指标会提供给下游组件，包括 PDLC-DL 和 SWM-FC，以支持上下文感知放置和迁移决策，而无需修改核心调度逻辑，如图 8 所示。

5.3.3 PDLC-GNN：基础设施预测

PDLC-DL-GNNs 子组件预测短期基础设施利用率，以支持主动

[5]在实验早期阶段，可静态配置邻域成员关系以验证工作流。由 CNC 驱动的完整邻域选择功能将作为 CODECO 开源工具包的一部分逐步集成。

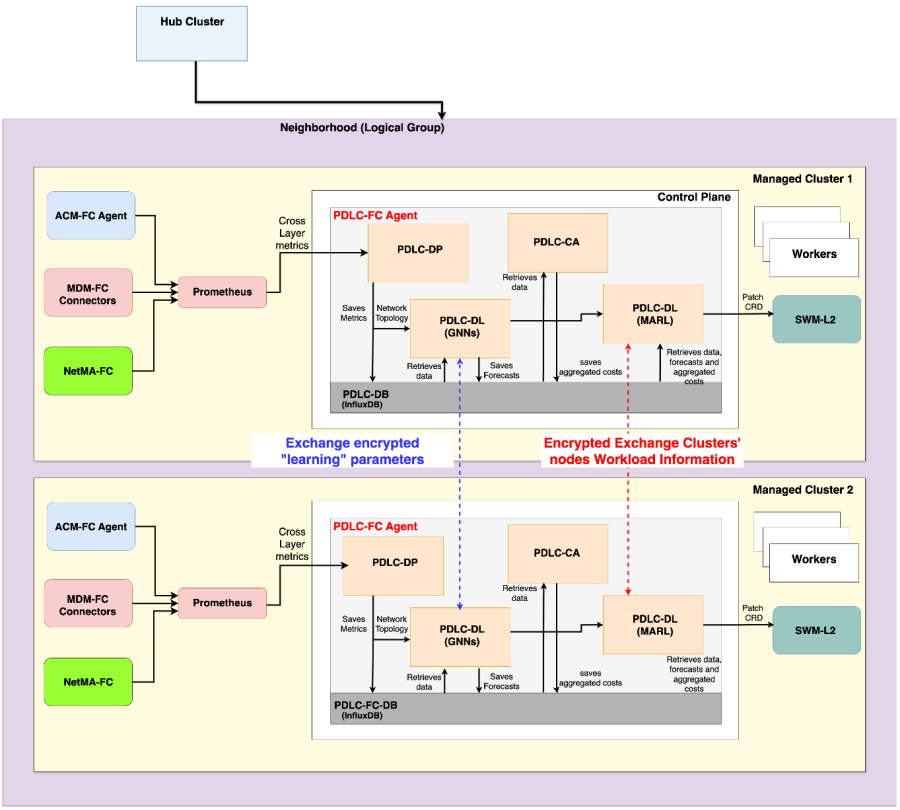

图6：联邦环境中 PDLC 子组件及其交互的功能性高层表示。

```
profiles.yaml: |
  profiles:
    - name: UserDefined
      equation: "cm_normalized_avgNodecpucustom * cm_normalized_avgNodefailurecustom"

custom_metrics.yaml: |
  custom_metrics:

    # CPU usage:
    - name: avgNodecpucustom
      norm_min: 0
      norm_max: 1
      promQL: 'avg by (instance) (1 - rate(node_cpu_seconds_total{mode="idle"}[1m]))'

    # Avarage node failure:

    - name: avgNodefailurecustom
      norm_min: 0
      norm_max: 20000
      promQL: 'sum(avg_over_time(kube_node_status_condition{condition="Ready", status!="true"}[1m]))
```

图7：用户定义的目标性能配置文件和权重示例。

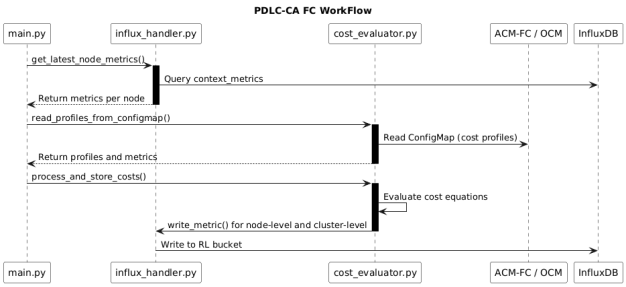

图8：PDLC-CA 时序图。

放置和迁移决策。每个 PDLC 实例都嵌入一个本地图神经网络模块，并在其集群控制平面内独立运行。该模型采用时空方法：它捕捉节点指标随时间的变化，同时考虑节点之间的结构关系，从而实现拓扑感知预测。

PDLC-DL-GNNs 围绕三个主要元素组织，如图9所示：

- **去中心化联邦训练：** 每个集群都使用 PDLC-DP 存储的指标在本地训练其图神经网络。在指定的时间间隔，各集群与邻近节点交换加密模型权重，并以安全方式聚合这些权重，从而在不共享原始遥测数据的情况下提升模型质量。

- **控制器：**通过从本地数据库检索近期指标、组装模型输入并跟踪节点成员关系来编排推理周期，以反映活跃拓扑。

- **推理 API：**加载适当的模型变体并生成短期预测（例如，5/15/30 分钟），这些预测会存储在本地，供 PDLC-DL-MARL 使用。

模型保存在本地注册表中。如果拓扑发生变化（节点加入或离开），PDLC 会选择或更新合适的模型，并可能在检测到不匹配或性能下降时触发重新训练。

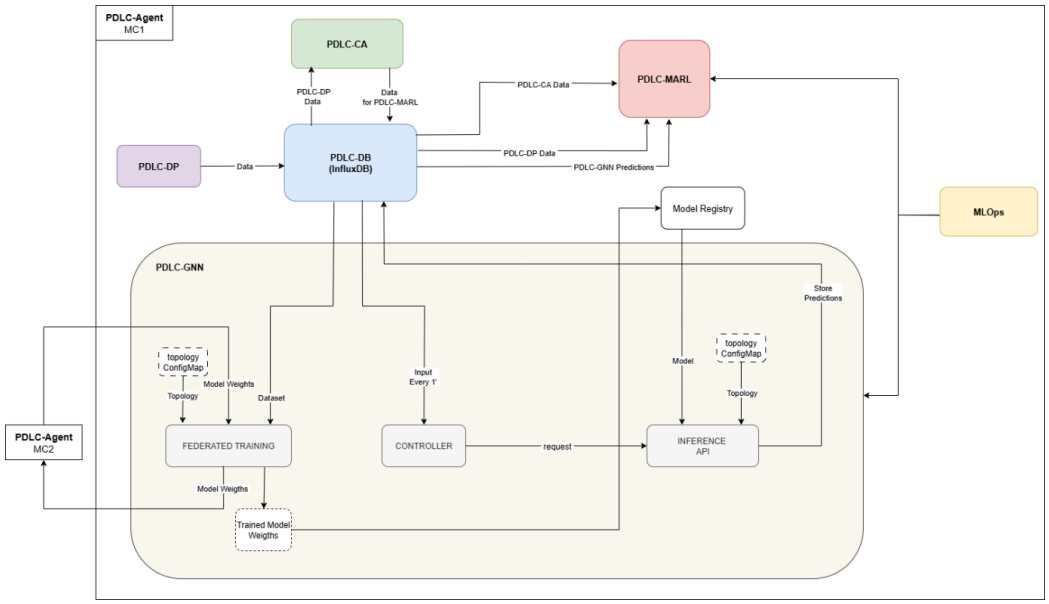

图 9：PDLC-GNN 的高层表示。

5.3.4 PDLC-DL-MARL

PDLC-DL-MARL 通过多智能体强化学习（MARL）设计实现邻域范围内的集群间推荐。图10展示了交互顺序。智能体在受管集群的控制平面中运行，无需中央协调器。智能体不交换原始状态或完整模型参数，而是共享高层决策值，从而支持符合 CODECO安全要求的联邦协作。

PDLC-DL-MARL 由以下部分组成：(i) 数据摄取层，从本地 InfluxDB 实例读取观测数据；(ii) 强化学习控制器，负责管理训练/推理模式以及通信/安全检查；(iii) 智能体间协调层，基于拍卖式机制，集群通过交换决策值，竞争承载工作负载。

训练控制器遵循强化学习循环，包括：(i) 通过填充实现**拓扑韧性**，以便适应不同规模的集群（最多 20 个节点）；(ii) 在运行时使用**动作掩码**过滤不可行的动作。观测数据从 InfluxDB 中获取，推荐结果则写入相应的自定义资源。

PDLC-DL-MARL 支持应用邻域内托管集群之间的去中心化协调。每个集群都会运行一个本地智能体，根据观测到的上下文和预测条件，评估自身托管工作负载的适宜性。智能体交换抽象的决策值，而非原始状态或模型参数，从而在保留隐私并降低通信开销的同时，实现协同推荐。

5.3.5 PDLC 在安全性、韧性和可扩展性方面的功能覆盖

PDLC-FC 凭借其去中心化设计，天然支持安全、韧性和可扩展性。遥测数据保留在各个集群本地，集群间的交换受到保护，协作范围仅限于应用邻域。因此，单个集群发生故障或不稳定不会扩散，而随着联邦规模扩大，与学习相关的通信量仍保持在有限范围内。

5.4 NetMA-FC

NetMA-FC 提供联邦 CEI 编排所需的网络可观测性、安全的集群间连接和能力暴露。它使 CODECO 能够将网络延迟、带宽、丢包和能耗指标作为放置和适应决策的一等输入。

图11展示了 NetMA-FC 的功能概览、其子组件以及与 CODECO 其余技术栈的接口。NetMA-FC 由三个主要子组件构成：

- **网络状态监控（NSM）**提供底层网络和覆盖网络的可观测性，捕获与应用服务质量和移动性相关的链路级及路径级状况。

- **网络暴露系统（Nemesys-FC）**充当 NetMA-FC 的北向接口，将监控信息转化为可用的成本映射和指标，并通过 Prometheus/Kubernetes 资源将其暴露给 ACM-FC、PDLC-FC 和 SWM-FC。

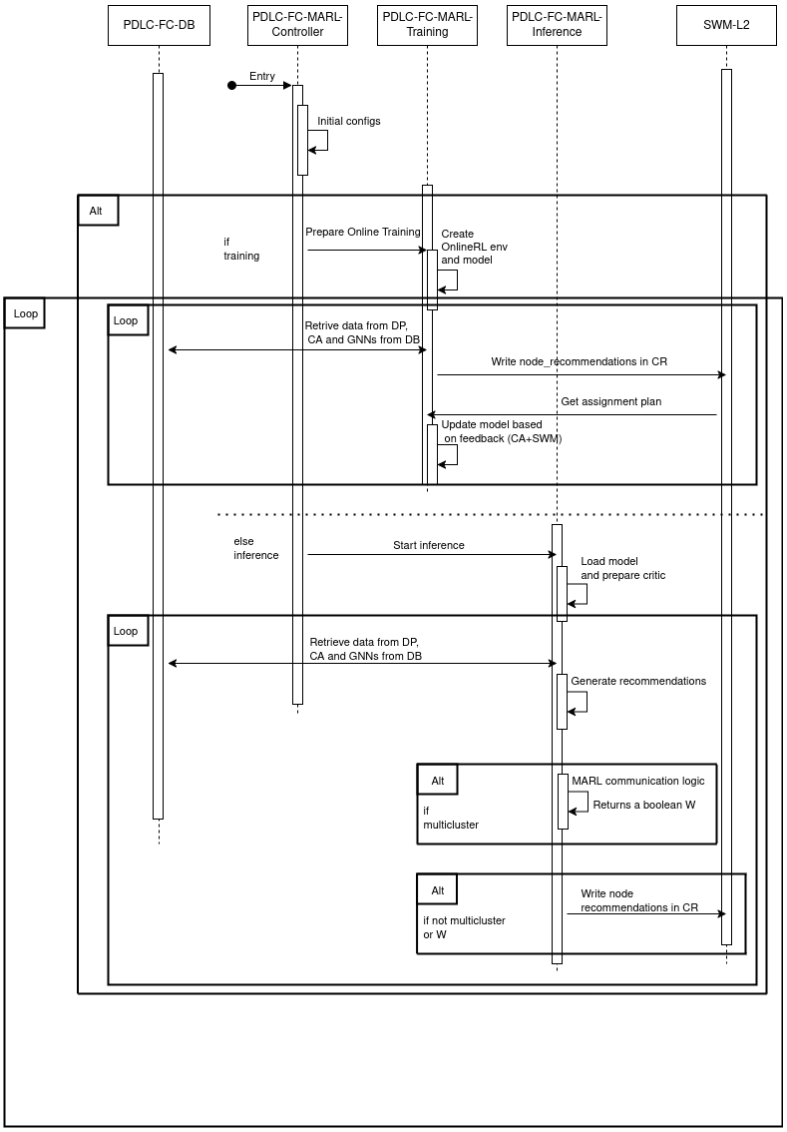

图10：PDLC-DL-MARL 通信序列。

•**安全连接（L2S-M）**为 CODECO 工作负载建立安全且隔离的集群间通信。在联邦阶段，这包括用于域间连接的分布式 SDN 控制平面，并支持预先建立和按需配置的覆盖网络拓扑。

5.4.1 NSM：网络状态监控

NSM 在 Kubernetes 工作节点级别提供底层网络和覆盖网络的可观测性，在此级别，应用程序执行上下文与数据包可见性并存。它收集延迟、带宽、丢包、链路故障以及链路/流能耗等指标，从而能够在联邦环境中进行应用感知和能耗感知的网络评估。

底层网络与覆盖网络可观测性NSM 使用基于扩展伯克利数据包过滤器的检测技术，在 Kubernetes 工作节点上监控底层网络和覆盖网络。指标包括延迟、带宽、丢包、链路故障和能耗，从而支持应用感知和能耗感知的网络评估。指标会导出到 Prometheus，供 PDLC-FC 和 SWM-FC 使用。

MON 模块（netma-nsm-mon）是 NetMA 的一个子组件，负责源节点被动网络探测以及底层指标的聚合。MON 基于开源 Kubernetes *k8s-netperf*[6] 插件构建，并扩展支持包括能耗相关指标在内的其他指标。MON 使用扩展伯克利数据包过滤器进行插桩，以监控表 2 所示的参数；对于能耗感知指标（链路级和流级），则采用 Feeney 等人提出的线性功耗模型 [7]。该模型将数据包传输功耗估算为以下变量的线性函数：

[6]https://github.com/leannetworking/k8s-netperf。

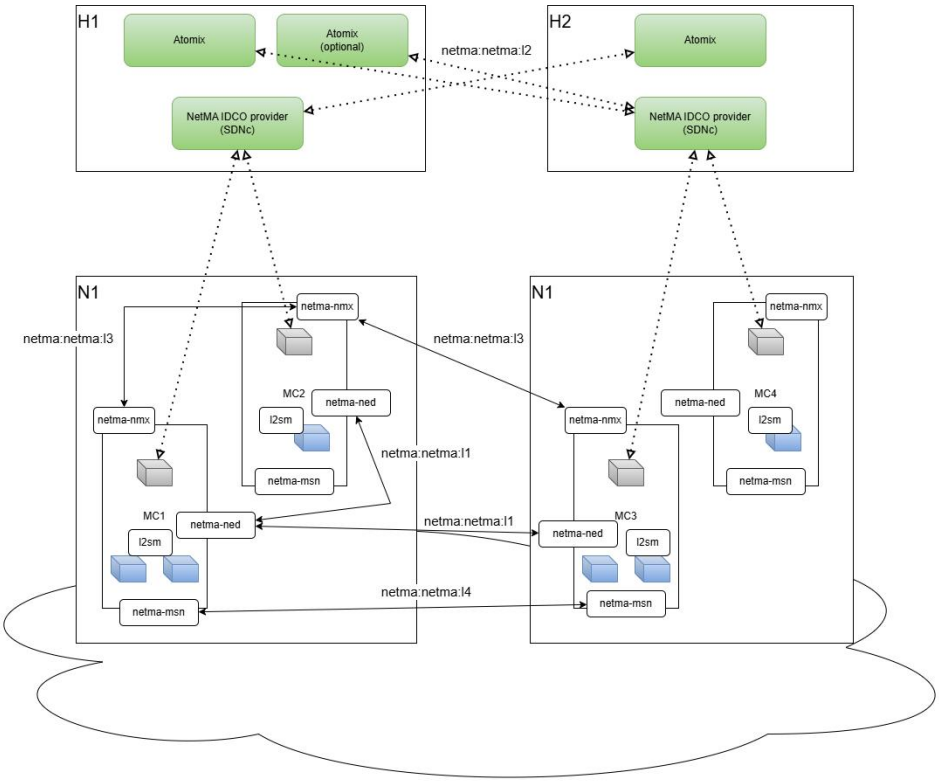

图1　　1：NetMA 子组件在 Hub 集群 H1、H2 和 MC 集群 N1、N2 中的放置。

数据包大小，将静态基线功耗与随大小变化的项结合起来。一个监测周期内的总链路能耗，通过累加各个数据包传输所消耗的能耗并计入各自的传输持续时间得出。也可以使用其他公式。

在覆盖层级，NSM 还支持覆盖网络特定的测量，用于量化隧道/路径行为（例如封装路径上的往返时延和丢包），这对于评估安全的集群间连接以及支持路径感知编排决策是必不可少的。

Nemesys-FC：联邦化暴露与统一网络视图

Nemesys-FC 是 NetMA-FC 的北向接口（*NBI*）。它收集由 NSM（底层网络和覆盖网络）生成的拓扑和性能数据，对其进行规范化和格式化，并通过 Prometheus 指标和 Kubernetes 资源将其提供给其他 CODECO 组件（尤其是 ACM-FC 和 PDLC-FC）。

Nemesys-FC 将底层网络和覆盖网络的测量数据汇总为统一的邻域范围网络视图，并通过 Kubernetes 资源和 Prometheus 指标公开。它支持基于 ALTO 的成本映射，用于表示延迟、带宽、丢包和能耗，从而支持 SWM-FC 和 PDLC-FC 进行多目标调度和自适应。Nemesys 与其他 NetMA 子组件的交互

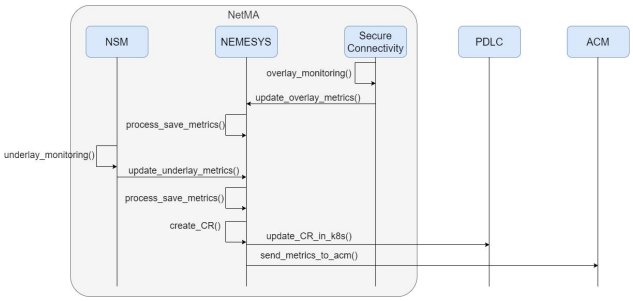

图 12：Nemesys 与 NetMA 子组件及 CODECO 组件的交互。

见图 12。

Nemesys-FC 采用模块化架构，包括：(i) 底层网络和覆盖网络数据接入模块（NSM）；(ii) 维护本地多指标图的 ALTO 核心；以及 (iii) 用于交换邻域范围摘要的联邦接口。ALTO 核心保留本地视图，并利用通过联邦接口收到的对等节点信息对其进行补充，然后将生成的成本映射转换为 Kubernetes 资源和 Prometheus 指标。这种设计既保留了本地自治能力，又支持协作式网络感知。

5.4.2 使用 L2S-M 实现安全连接

图 13 所示的安全连接子组件可为 CODECO 工作负载建立隔离且安全的跨

表 2：MON 导出的网络指标。

ID（属性）	描述	值	单位
节点名称	Kubernetes 中节点的标识符	字符串	n.a.
名称（链路）	节点链路的标识符	字符串	n.a.
链路故障数	链路故障次数	整数（字符串）	n.a.
数据包丢失	连接中的数据包丢失	浮点数（字符串）	n.a.
节点网络故障	链路和端点故障总数	整数（字符串）	n.a.
节点带宽	所有节点链路的总出口带宽	整数（字符串）	字节 (KB, MB)
uNodeDegree	连接到节点的活动链路数量	整数（字符串）	n.a.
uLatencyNanos	出口链路延迟（指数移动平均）	浮点数（字符串）	ns
uFlowEnergy	流能耗（指数移动平均）	浮点数（字符串）	J
uLinkEnergy	链路能耗（指数移动平均）	浮点数（字符串）	J

集群通信。它支持预先创建和按需构建的覆盖拓扑，并与联邦调度决策集成，确保网络路径与放置结果一致。持续监控集群间链路，可在条件变化时实现运行时适应。

分布式 SDN 控制平面 为在适应联邦操作的同时保持灵活性，安全连接引入了一个基于 ONOS 构建的分布式 SDN 控制器，并在每个中心集群中单独部署一个实例。每个实例都充当域间连接编排器（*IDCO*）提供方，并使用 RAFT 共识算法与对等节点协调 [8]。

虚拟链路、意图与流编程 安全连接将集群间安全连接表示为虚拟链路（虚拟链路）。虚拟链路会编译为控制器意图，再转换为具体的转发规则，并安装到网络边缘设备（NED）上（例如通过 OpenFlow）。路径选择可受 SWM-FC 决策的影响，使安全连接遵循编排结果，而非强制采用固定路径。

映射到邻域在联邦模型中，NED 由枢纽集群 SDN 控制器以多对一的方式管理（一个控制器管理多个 NED），而邻域与控制器之间采用一对一映射（一个控制器实例管理某个邻域内的所有 NED）。这样既能将协调范围限制在一定范围内，也能避免无关应用之间的全局控制器耦合。

故障切换与连续性为提高网络可靠性，故障切换机制允许 NED 在枢纽控制器不可用时重新关联到另一个活动的控制器实例，从而在发生局部故障时维持覆盖网络的安全运行。

集群间覆盖网络设置 集群间连接性依赖于一个互连各集群并连接到集群内安全覆盖网络的覆盖网络。支持两种替代方案：

- **预创建覆盖拓扑：** 集群间隧道（例如 VXLAN）在安装时根据规划的拓扑建立；链路监控立即开始。

- **按需覆盖拓扑：** 仅当 Workload 需要安全的跨集群通信时，才创建集群间隧道。在这种方法中，Workload 部署会触发对 Kubernetes 资源（例如 ConfigMap 或专用自定义资源）的更新，NED 侧监视器会据此配置或拆除隧道[7]。

预创建拓扑可缩短运行时连接建立时间，但需要更周密的规划和安装；按需拓扑提高了灵活性，但会引入额外的控制平面逻辑和 API 设计要求。

集群间链路性能监控（i-LPM）。链路性能监控（*LPM*）功能扩展至集群间场景（i-LPM），用于测量覆盖隧道性能（例如吞吐量、往返时延、丢包），并持续监控集群间链路的可靠性。

[7]在当前原型中，按需隧道控制仍在设计阶段；最终实现可能会使用专用自定义资源定义（CRD）取代 ConfigMap，以提供更强的生命周期保障。

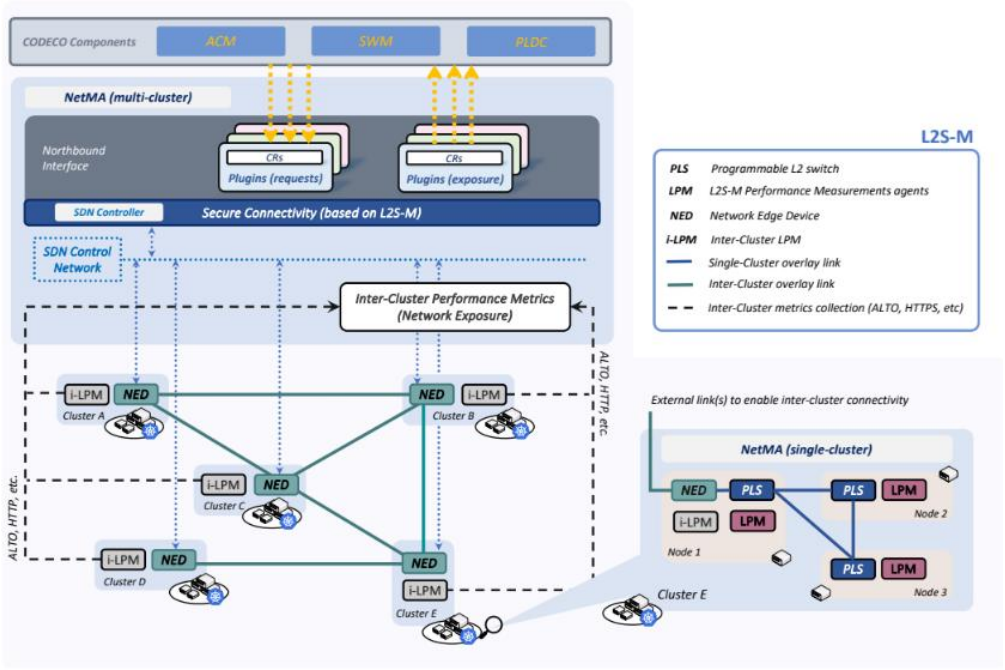

图 13：NetMA 安全连接设计。

i-LPM 与具体工具无关（初步支持通过*iperf*和*ping*实现），并将指标导出到 Prometheus。为支持动态（按需）覆盖网络，i-LPM 端点通过 Kubernetes 资源（例如 ConfigMap）进行配置，从而无需重新部署即可重新配置。i-LPM Pod 与 NED 部署在一起，以确保测量准确且延迟低。

集群间监控演进 NSM 最初是为使用轻量级基于套接字的守护进程集，在微服务之间进行集群内测量而设计的。它能够以极低的开销实时监控吞吐量、延迟和丢包，并可自然地与 Prometheus 集成，供 PDLC-FC 和 SWM-FC 使用。在联邦阶段，NetMA-FC 通过将 NSM遥测、Nemesys-FC 暴露的信息以及 i-LPM隧道测量结合起来，将这一监控模型推广到集群间覆盖网络，并在邻域范围的协调领域内运行。

5.5 SWM-FC：调度与工作负载迁移

SWM-FC 负责在联邦集群之间对应用工作负载进行初始放置、持续重新调度，并在需要时执行迁移，同时确保满足应用QoS约束。它通过将上下文感知可观测性输入与 PDLC- 生成的节点级和集群级建议相结合，支持 CEI 相关的优化目标，例如低延迟、降低能耗、数据敏感性和体验质量（QoE）。

FC.

SWM-FC 提供一组 Kubernetes 自定义资源定义（*CRD*），包括 ApplicationGroup、Application、Workload、Channel 和 AssignmentPlan。它不直接收集基础设施遥测数据，而是使用（i）来自 NetMA-FC 的拓扑和连接更新，以及（ii）来自 PDLC-FC 的放置建议。网络状况的变化（例如集群间带宽降低）可能使现有放置方案失效，并触发重新调度或迁移。Hub 与托管集群（MC）实例之间的协调依赖 OCM/ACM 机制，而所有调度逻辑和优化均保留在 SWM-FC 内部。

在联邦操作中，工作负载及其通信通道可能跨越应用邻域内的多个受管集群。SWM-FC 应对三个核心挑战：(i) 跨集群同步与调度相关的上下文；(ii) 仅公开抽象化集群视图，以保持可扩展性和隐私；以及 (iii) 持续更新集群间连接信息，以满足延迟和丢包约束。

在功能上，SWM-FC 将应用工作负载分配给执行节点，并将 *Channels* 映射到满足计算和通信要求（例如 CPU、内存、带宽、延迟和服务类别）的网络路径。与标准 Kubernetes 调度器不同，SWM-FC 支持跨集群放置和迁移，同时保持与 Kubernetes 的一致性：它

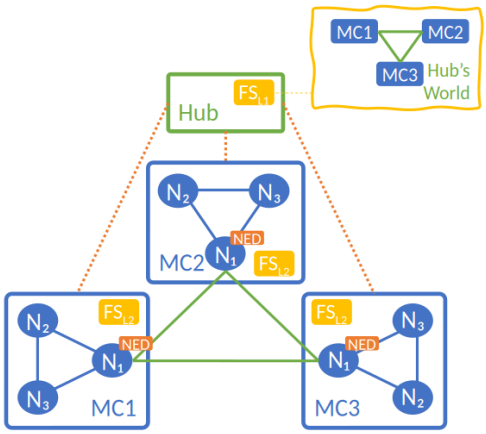

图14：SWM-FC 高层概览：Hub 协调邻域集群间的联邦调度，各集群配备求解器（FSx）并在本地执行。Nx 表示一个工作节点。

会迁移 Pod 及其关联的网络连接，但不管理应用状态；应用状态仍由原生应用和 Kubernetes 机制负责处理。

5.5.1 SWM-FC 实例的部署与角色

SWM-FC 在每个集群中运行一个实例：如图14所示，中心集群中运行一个实例，每个 MC 中也各运行一个实例。所有实例运行相同的软件，但 Hub 实例负责协调多集群放置。MC 实例 (i) 为其本地集群求解放置子问题；(ii) 执行已达成一致的放置方案中的本地部分。在 MC 中收到的多集群请求会转发给 Hub 协调器。

5.5.2 双层联邦调度模型

SWM-FC 假设，应用程序组的目标集群集合就是所提供的 CODECO 邻域。ACM-FC 会将该邻域提供给所有 CODECO 组件。应用程序组的调度一旦开始，邻域就视为固定不变，以避免放置过程中出现歧义和振荡。

从概念上讲，联邦调度可分解为一个第 1 层（L1）问题：利用汇总的集群信息（将集群视为不透明实体）在集群间分配工作负载；以及第 2 层（L2）问题：将工作负载放置到集群内的节点上。尽管考虑了两个可解层次，L1 并不会作为一个独立的优化阶段来求解。相反，SWM-FC 会直接使用邻域信息，随后对集群排序以进行顺序搜索，再求解一系列 L2 问题，每个已访问的集群对应一个。这避免了根据粗粒度汇总信息对 L1 进行“猜测”，并减少了将集群视为工作节点时可能产生的不必要的集群间通道。排序时优先考虑 (a) 可获取节点推荐的集群，以及 (b) 在搜索序列中较早出现的、集群间链路速度较快的集群。

SWM-FC 使用描述集群间链路的集群间拓扑资源（例如 NetMA 提供的 NetworkTopology）。集群间网络仅支持单跳连通性：只有当两个集群之间存在满足相关服务类别的直接链路时，才认为它们彼此连通。集群间通道有意不采用多跳路由（例如 B→A→C），以便保持模型的可处理性，并与现有连通性基础设施保持一致。

5.5.3 调度工作流

SWM-FC 假设：(i) CODECO 应用可以部署在多个集群中；(ii) 每个集群都具有用于集群间流量的网关/NED 角色；(iii) 与 NetMA（包括 L2S-M）协同工作，可提供保障型通道所需的服务类别。Hub-MC 通信的详细机制以及与 ACM/OCM 的最终集成点取决于具体实现，并作为 CODECO 控制平面集成工作的一部分处理。

SWM-FC 在 Hub 实例的协调下分三个阶段执行联邦式放置；只有最后一个阶段会分配资源，如图 15 所示：

1. **排列集群。** 根据推荐和集群间链路特征，计算要在邻域内访问的集群顺序。

2. **逐个集群求解 L2 放置。** 按顺序访问集群。每个集群使用本地详细信息，以及尚未访问集群的聚合表示，求解一个 L2 问题。第一个访问的集群会尽可能多地放置工作负载（由节点推荐引导），并将剩余工作负载暂定分配给其他集群（作为集群级节点）。Hub 整合每个部分结果，并将更新后的问题转发给下一个集群。

3. **执行放置。** 当所有工作负载都已分配且通道需求均可满足后，SWM-FC 会创建 Pod，并请求/安装所需的网络服务（集群内和集群间）。如果任何目标集群中的执行失败，SWM-FC 会回滚所有受影响集群中的分配。

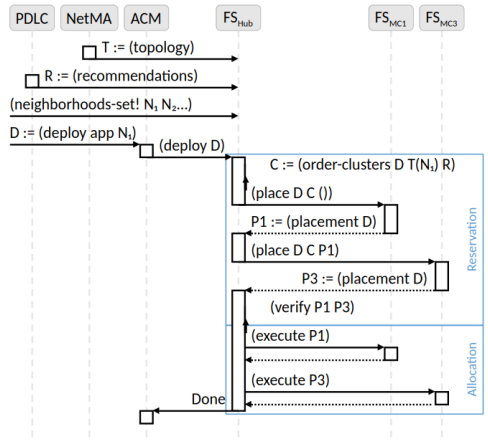

图 15：CODECO 联邦调度流程：对集群排序，依次求解 L2 问题，进行验证，然后执行或回滚。

两个集群中工作负载之间的通道会逐步验证。当源工作负载完成放置后（例如，在集群 A 中），SWM-FC 会尽量降低集群 A 内的延迟贡献（例如，到集群网关/NED 的延迟），以保留目标端的灵活性。最终可行性检查会在目标工作负载完成放置时进行（集群 B）：总延迟按 集群 A 内的延迟 + 集群间链路延迟 + 集群 B 内的延迟 计算，并根据所选服务类别和可用资源检查带宽可行性（以及其他任何请求的资源）。

5.5.4 鲁棒性：重新审议阶段

由于每个集群之外无法获知其内部情况，第一轮调度可能无法放置所有 Workload。为提高鲁棒性，SWM-FC 可选地执行一个重新审议阶段：通过重新遍历排序后的集群列表，再次尝试放置未放置的 Workload，并在策略允许的情况下放宽放置约束（例如扩大候选集，同时保留硬性服务质量约束）。由于剩余问题规模更小，且大部分拓扑/上下文输入已在参与实例处缓存，第二轮通常比第一轮成本更低。

5.5.5 工作负载迁移

SWM-FC 采用与初始放置相同的三阶段流程处理迁移（不同于 Kubernetes 的副本复制），当 Workload 推荐或基础设施条件发生变化时触发。推荐由 PDLC 提供，它可以推荐一个“更优”的目标节点或集群，以符合用户在 CAM 中设定的目标性能配置，例如绿色度、韧性等。在执行迁移时，SWM-FC 会在停止旧 Pod 实例之前，为新的放置预留资源，从而实现快速切换；如果无法立即创建新实例，也能降低服务中断的风险。

5.6 MDM-FC

元数据管理 组件（MDM-FC）通过为跨 CEI 环境的数据、应用和基础设施提供统一的语义视图，使 CODECO 能够实现数据感知编排。其主要目标是通过向控制平面公开与数据相关的约束和特性（例如局部性、新鲜度、可移植性和合规性），支持高效且符合策略的编排决策。

MDM-FC 集成了三种互补能力：(i) 语义元数据目录，用于捕获有关数据、系统和软件的跨领域信息；(ii) 元数据发现与丰富，将从原生系统收集的信息与从多集群运行中获得的见解相结合；(iii) 数据编排支持，使 CODECO 能够在放置和调整工作负载时考虑与数据相关的约束（例如，避免频繁访问远程数据集，或强制执行限制数据移动的合规要求）。

语义元数据模型 MDM 将元数据表示为类型化属性图，其中实体和关系可以携带任意定义的属性。该模型支持直接从图中推导应用级和基础设施级指标的丰富查询。典型属性包括数据源及其语义、模式和数据结构、特定于应用的特征（例如分析或整理阶段），以及合规性或观测到的性能等系统级属性。

目录通过推送和拉取机制持续填充。元数据可直接从 Kubernetes API、外部数据源获取，也可由用户输入并经 ACM-FC 和 Prometheus 指标传递。MDM 将这些信息维护为共享抽象层，使得能够以统一的方式处理源自多个集群和领域的元数据。

为将元数据摄取与存储和查询解耦，MDM 依赖于基于 Apache Kafka 的事件驱动架构。元数据变更以事件（插入、更新、删除）的形式表示，这些事件会影响实体或关系。这些事件会被记录下来，并有选择地物化到元数据图中，以满足其他 CODECO 组件（尤其是

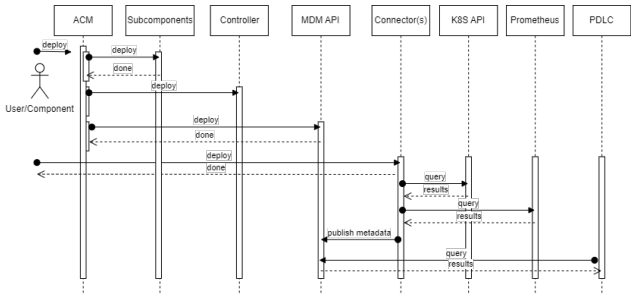

图16：CODECO 中的 MDM 组件及其交互。

PDLC-FC）的
需求。

实体、关系和可扩展性。 CODECO/Kubernetes 环境中的实体示例包括 *Pod*、命名空间、工作节点，以及与数据相关的对象，如 数据存储 或文件。关系具有类型和方向，并且恰好连接两个实体（例如，将命名空间与其 Pod 相连的 包含关系）。可以使用灵活的 JSON Schema领域特定语言 [9] 定义新的实体和关系类型。这种可扩展性使 MDM 能够集成来自任意原生系统的元数据，而无需更改核心数据模型。

5.6.1 组件

MDM-FC 由三个松散耦合的组件实现（图16）：

- **图数据库**。图数据库存储物化的元数据图，并提供分布式来源所收集的实体和关系的全局视图。在当前实现中使用 Neo4j，但也可以采用任何支持 Cypher 的图数据库。在开发和探索期间，可以直接执行图查询；但在生产环境中，访问通过 MDM API 进行中介，以保持抽象性和可移植性。

- **元数据管理API**。元数据管理API提供符合 OpenAPI 标准的 REST端点，允许其他 CODECO 组件摄取、查询和使用元数据。该 API 包含用于发布元数据事件、访问事件流、查询元数据图，以及获取 PDLC-FC 所需的特定领域视图的端点（例如，Pod 级别的可移植性、合规性或新鲜度指标）。系统内部使用 Kafka 将 API 数据摄取与图存储解耦，从而确保可扩展性和韧性。

- **连接器**。连接器负责从原生系统收集元数据，并将其作为结构化事件发布到 MDM。每个连接器都会管理其插入的实体的生命周期，确保彼此隔离，避免连接器之间相互干扰。实体会根据稳定属性（例如集群ID、命名空间、Pod 名称）分配确定性的全局标识符，从而确保在不同集群间能够一致地识别实体。

开箱即用，CODECO 提供了 PDLC-FC 常用的三个连接器：(i) *Kubernetes*连接器，可将 Kubernetes 对象及其关系反映到元数据图中，并通过 Pod 重启次数提供可移植性代理；(ii) *Prometheus*连接器，可使用新鲜度等应用级指标丰富实体；(iii) 合规性连接器，可集成集群合规性评分（例如通过 Kubescape 获取），以支持策略感知型放置决策。

5.6.2 联邦操作

MDM-FC 旨在联邦环境中高效运行，无需集群之间紧密耦合。虽然概念上可以实现带有全局元数据复制的完全对称联邦，但 CODECO 采用邻域范围的部署模型，以符合其有界协调原则。

每个集群对应一个邻域 在一个邻域内，由单个集群充当 MDM 中心，托管该邻域的图数据库和元数据管理API。部署在受管集群上的连接器会向 Hub 实例报告元数据，而 PDLC-FC 实例则查询该 Hub，以获取与部署在该邻域中的应用相关的数据相关指标。这样，在编排不需要时，就无需维护全局元数据视图。

如果托管集群离开某个邻域，其元数据便会停止更新。如果 MDM 中心不可用，ACM-FC 可以在另一个集群上重新部署 MDM 实例，并相应地重新配置连接器。尽管系统设计支持这种故障切换场景，但它不属于标准运维流程。

支持多个邻域当集群参与多个邻域时，
MDM-FC 必须避免在同一个中心集群上为每个邻域单独部署一个元数据栈。图 17 展示了数量有限的 MDM 实例

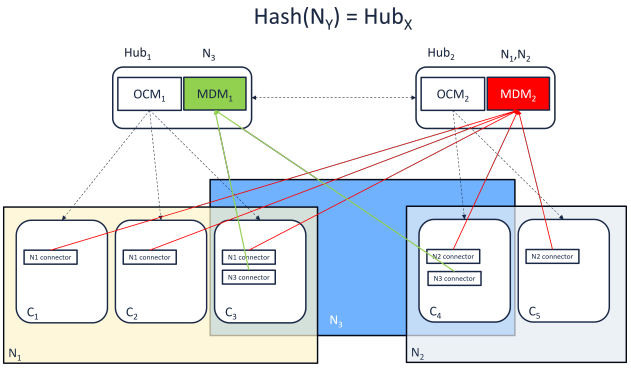

图 17：支持每个实例包含多个邻域的 MDM。

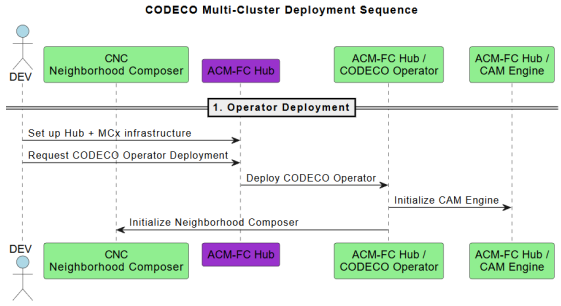

图 18：展示 CODECO 如何在包含一个 Hub 和多个托管集群的联邦集群中安装和部署的时序图。

托管在 Hub 集群上的实例如何通过将邻域标识符一致地映射到 MDM 实例，共同支持多个可能相互重叠的邻域。

连接器可配置为向多个 MDM 实例报告，使每个集群只需部署一个连接器即可服务于多个邻域，从而减少开销。对于大规模部署，Hub 集群可以维护分布式哈希表（DHT） [10]，以高效管理邻域到 MDM 的映射。

6 工作流示例

本节通过三个典型工作流说明 CODECO 的端到端运行过程：(i) 在联邦集群中安装和部署 CODECO 框架，(ii) 在云边基础设施（CEI）中部署应用程序，以及 (iii) 对应用程序进行运行时管理和自适应。这些工作流体现了前文所述的交互，并强调编排、监控和学习组件如何在不同生命周期阶段协同工作。

6.1 CODECO 安装与部署

要安装 *CODECO OSSFederated* 工具包，开发者（DEV）需从官方 CODECO 代码库获取软件并遵循安装指南。前提是拥有可正常运行的 OCM 环境，其中至少包含一个中心集群和一组受管集群（MCs）。整个过程分为三个阶段，如图 18–20 所示。

阶段 1：Operator 部署。 DEV 首先通过 OCM 将托管集群注册到 Hub，以准备好基础设施。DEV 使用 ACM-FC 接口，触发在 Hub 上部署 CODECO Operator。此步骤将初始化联邦编排所需的核心控制平面服务，包括用于解析和验证应用模型的 CAM 引擎，以及支持集群语义分组的 CODECO CNC。

阶段 2：组件部署。 Operator 激活后，ACM-FC 会协调 CODECO 组件在 Hub 和受管集群中的部署。在 Hub 上实例化以下组件：

- **SWM-FC（Hub 实例）**，用于协调联邦调度和 Workload 迁移，
- **NetMA IDCO**，用于管理集群间安全连接和覆盖网络控制，
- **MDM-FC**，用于维护应用、基础设施和合规性相关方面的元数据图。

在每个托管集群上，ACM-FC 部署：

- **SWM-FC（MC 实例）**，负责本地工作负载放置与执行，
- **NetMA-FC**，提供网络探测、覆盖网络支持和指标暴露，
- **PDLC-FC**，执行上下文感知的成本估算与去中心化学习，
- **MDM 连接器**，将本地元数据馈送到邻域 MDM 实例中。

部署后，将启动持续监控循环。NetMA-FC 开始收集网络级指标（例如延迟、抖动、丢包和能耗），MDM 连接器填充并更新元数据图，而 ACM 则观察用户偏好和应用意图。这些数据流构成后续编排和适应决策的基线。

6.2 跨联邦 CEI 环境部署 Application

部署工作流如下：

1. 在 ACM-FC 提交并验证 CAM。

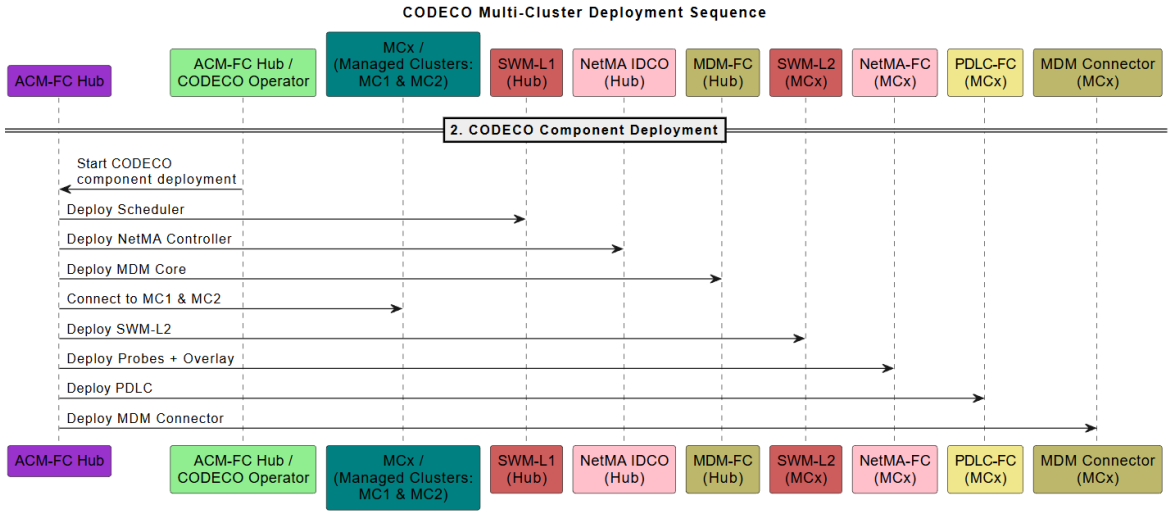

图19：展示 CODECO 如何在一个 Hub 和多个托管集群组成的联邦集群中进行安装和部署的时序图。

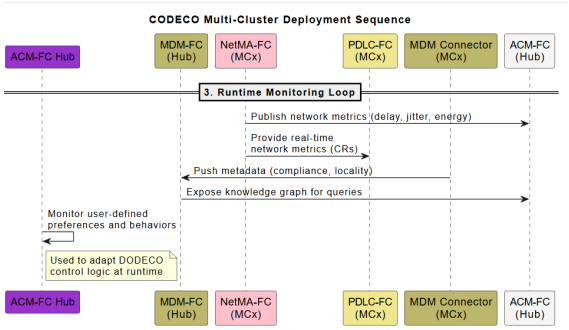

图20：展示 CODECO 如何在一个 Hub 和多个托管集群组成的联邦集群中进行安装和部署的时序图。

2. 通过 CNC 进行邻域选择。3. 在邻域内初始化 PDLC-FC 代理。4. 由 SWM-FC 执行联邦式放置和信道分配。5. 由 NetMA-FC 进行网络配置并设置安全连接。

6.3 运行期间的应用管理

部署完成后，CODECO 进入运行时管理阶段，在此阶段持续观察应用和基础设施的行为，并在必要时调整部署，如图20所示。

考虑一个由两个微服务 a_1 和 a_2组成的应用程序组，它们分别部署在同一 Hub 下的受管集群MC1和MC2上。在运行时，来自 NetMA-FC 和 ACM 的监控数据，以及来自 MDM 的元数据更新，会持续馈送到 PDLC-FC 中。如果出现触发条件（例如性能下降、资源争用、能源约束或应用意图发生变化），ACM-FC 会发出需要重新评估的信号。

PDLC-FC 会更新其成本估计和基于学习的稳定性评估，并通过自定义资源发布修订后的推荐。SWM-FC 会观察这些更新并启动新的放置计算；只要满足所有约束，就可能在集群内或跨集群进行工作负载迁移。

确定新的分配方案后，SWM-FC 会更新相关自定义资源（AssignmentPlan、ApplicationGroup），从而触发 NetMA-FC 在需要时重新配置网络路径。在整个过程中，组件之间完全通过 Kubernetes自定义资源进行协调，避免紧密耦合或对控制平面的直接依赖。

最后，ACM-FC 汇总当前系统和应用状态，并通过仪表板或 API 将其提供给 DEV，从而实现透明性和知情的人为监督，同时由 CODECO 自主管理应用生命周期。

7 CODEF：实验与软件定义测试床

评估联邦 CEI 环境中的去中心化编排机制，需要采用一种可复现、自动化且可扩展的实验策略，以捕捉计算、数据和网络资源之间的交互。与单集群实验相比，联邦场景还带来了与地

理分布、异构基础设施、多网络连接以及动态运行条件相关的额外挑战。

本节介绍 CODECO 采用的实验方法，该方法体现在*CODECO* 实验框架（*CODEF*） [11]软件定义的边缘-云仿真工具中。CODEF 提供端到端的软件定义实验环境，以一键自动化的方式简化基础设施配置、平台配置、实验执行和结果收集。

此外，我们还将介绍 CODEF 如何扩展以支持大规模、地理分布式的联邦测试床，从而能够对 CODECO 的去中心化编排机制进行真实场景验证。

7.1 CODEF 作为独立仿真器

CODEF 提供一套自动化的端到端工作流，用于在联邦 CEI 环境中设计、部署和评估实验（图 21）。它以基于微服务的框架实现，前置要求极少（例如 Docker 和标准命令行工具），并在整个实验生命周期中明确划分职责。

具体而言，CODEF 区分以下内容：

- **基础设施管理**，负责在边缘、云和虚拟化环境中配置计算资源；
- **资源管理**，用于安装操作系统、Kubernetes 发行版、容器网络接口插件（容器网络接口）以及所需的软件栈；
- **实验执行**，涵盖参数化工作负载部署、编排策略、迭代，以及故障、流量或负载注入；
- **结果处理**，包括指标收集、聚合、生成可复现的输出和报告。

所有 CODEF 组件都是容器化微服务，因此可以轻松地通过新技术、编排工具或评估能力扩展该框架。这种设计支持实验平台持续演进，同时保持可复现性。

CODEF 的一项关键能力是支持异构基础设施。**基础设施管理器**抽象了跨边缘—云和虚拟化环境（例如 XCP-）的节点配置过程。

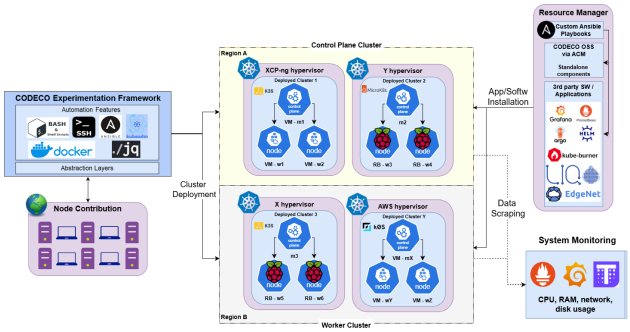

图 21：CODEF 端到端实验工作流。

ng[8], VirtualBox[9], 亚马逊云科技[10]），并支持与 CloudLab[11] 和 FABRIC[12] 等外部研究测试床集成。

资源管理器 将每个节点表示为容器化实体，并使用 Ansible 剧本部署操作系统、Kubernetes 发行版、CODECO 组件和实验工作负载。

在编排层面，声明式**实验控制器**将所有实验参数统一到单一的版本化规范中，包括基础设施拓扑、Kubernetes 配置、网络架构、工作负载定义、软件版本、指标和输出格式。这使得实验执行能够完全复现并一键完成。

对于联邦场景，CODEF 集成了成熟的多集群和多网络解决方案，包括 OCM、Submariner[13] 和 Liqo[14]。这支持在广泛的联邦模型中开展实验，从中心辐射式和基于邻域的部署，到点对点集群网状网络。因此，集群和各个节点可以在地理位置上分布，同时仍在单一联邦编排域内保持协调。

7.2 CODEF：协作式多租户联邦测试床

除了用于仿真，CODEF 还在 CODECO 中扩展为一个开放的联邦测试床，依靠互联网中不同实体贡献的节点，基于现已停运的 EdgeNet[15] 的原则，仿真真正的协作式 CEI 多租户联邦环境。该协作式运行环境允许用户、组织和外部测试床贡献计

[8] https://xcp-ng.org/ [9] https://www.virtualbox.org/ [10] https://aws.amazon.com/ [11] https://www.cloudlab.us/ [12] https://portal.fabric-testbed.net/ [13] https://submariner.io/ [14] https://liqo.io/ [15] https://www.edge-net.org/

算资源（虚拟机或裸金属节点），以构建共享的联邦环境[16]。

受大规模研究基础设施的启发，该能力支持众包式、地理分散的多集群部署，更准确地反映真实世界中的 CEI 条件。

7.3 原型设计与评估工作流

在联邦环境中评估去中心化编排，需要定义完善的原型设计与评估工作流，以适应异构硬件、多样化的网络架构以及动态的 CEI 规模条件。这些工作流必须保持平台无关性，确保实验能够在不同的 Kubernetes 发行版、虚拟化技术和边缘—云部署中一致地执行。

典型工作流始于对实验场景的声明式规范，其中包括集群组成、集群间连接和编排策略。在集群完成配置并实现联邦化后，部署应用工作负载，以评估可变延迟、间歇性连接、能源约束和资源争用等运营挑战。

在执行期间，跨层可观测性机制结合被动监控、主动探测和集成遥测，收集计算面、网络面和数据面的指标。这些测量结果支持对调度决策、应用行为以及由 PDLC、NetMA 和 SWM 等组件驱动的自适应或迁移策略进行系统评估。所提出的工作流与 ETSI 和 IETF 等机构正在开展的标准化工作保持一致；这些机构旨在为多集群编排、意图驱动管理和联邦遥测定义可互操作的机制。

在此背景下，CODEF 提供了将此类工作流付诸实践所需的自动化、可复现性和可扩展性，弥合了概念性编排模型与真实 CEI 实验之间的鸿沟。

7.4 可复现性与开源工具包

可复现性是实验验证的一项基本要求，尤其是在联邦环境中，基础设施多样性、地理分布和运行可变性都可能显著影响实验结果。开源工具包（例如在 CODECO 生态系统中开发的工具包）对于实现透明的实验、创新和可扩展性至关重要。

通过采用软件定义方法，CODEF 对平台异构性进行了抽象，并提供稳定的声明式接口，用于部署集群、配置联邦以及大规模开展编排实验。这使研究人员和从业者能够复现实验、验证结果，并通过新的编排策略、指标或基础设施技术扩展框架。因此，CODEF 为整个 CEI 连续体中的去中心化编排研究奠定了坚实基础，助力实现可复现、可扩展和协作式研究。

8 研究挑战与未来方向

尽管 CODECO 带来了诸多进展，但在 CEI 连续体中进行联邦编排仍面临许多尚待解决的研究挑战。这些挑战并非 CODECO 所特有，而是异构基础设施、管理和监管约束[12, 13, 14]下运行的更广泛一类去中心化、多提供商编排系统所共同面临的问题。

8.1 联邦容器编排中的调度挑战

一项根本性挑战在于如何在联邦调度中平衡最优性、可扩展性与隐私。在多集群环境中，全局优化通常需要访问细粒度的基础设施和工作负载状态，但出于隐私、所有权或可扩展性约束 [15, 16]，这些信息往往无法获取或不宜获取。CODECO 通过邻域范围调度和聚合的集群表示来应对这一矛盾；然而，如何制定既能保持调度质量、又能最大限度减少信息泄露的最优聚合策略，仍是一个尚待解决的问题。

另一个挑战涉及跨层耦合。尽管 CODECO 联合考虑了计算、网络和数据方面，但要准确捕捉它们之间的相互依赖关系，尤其是在移动性、间歇性连接和工作负载变化等动态条件下，仍然十分困难。先前的研究也指出了雾计算和边缘编排中的类似局限 [13, 12]。未来的工作可以探索更紧密地集成预测模型、不确定性感知调度和随机优化技术 [17, 18]。

8.2 稳定性、振荡与人工智能辅助控制

人工智能辅助编排引入了振荡行为的风险：当基于学习的组件对资源可用性或工作负载需求的瞬时波动反应过于激进时，可能会出现这种情况 [19，20]。

[16]2025年，当前的测试平台通过希腊和德国提供的节点连接了基于 CODEF 的设施。预计将于 2026 年向公众开放。联合 CODEF 测试平台的接口可通过以下网址访问：https://boss.swn.uom.gr/

CODECO 通过将建议与执行分离，并将学习组件限制在中等时间尺度上，来降低这一风险。然而，针对混合式人工智能与基于规则的编排系统的形式化稳定性保证仍基本未得到研究，这构成了一个重要的研究方向，尤其是在安全关键型 CEI 部署中 [21, 22]。

8.3 多枢纽与跨联邦运行

当前的 CODECO 设计假设每个联邦只有一个中心集群。支持多个枢纽、且每个枢纽治理各自的管理域，会带来命名空间冲突、信任管理、策略协调以及跨枢纽邻域构建等挑战 [15, 23]。将 CODECO 扩展为分层式或点对点枢纽联邦，符合去中心化云边缘基础设施治理的新兴愿景，也是自然而然的下一步。

9 相关工作与比较分析

Kubernetes 多集群与边缘编排已发展为丰富的项目和研究原型生态。介绍 CODECO 的设计与运行后，本节将阐述 CODECO 在以下方面的贡献：(i) Kubernetes 联邦与多集群控制平面；(ii) 以边缘为中心的 Kubernetes 发行版和框架；(iii) 集群间连接与服务网络；(iv) 面向联邦集群的学习辅助型调度器；以及 (v) 数据/元数据感知编排。

9.1 联邦与多集群控制平面

早期的联邦化工作（例如 Kubernetes 联邦（KubeFed[17]））主要侧重复制作资源，并支持跨集群部署原语 [15]。这些方法对跨层优化的支持有限，也没有将网络和数据约束明确建模为一等调度输入 [24]。

OCM 通过生产级中心辐射式架构，支持集群注册、策略传播和清单分发[25]，从而拓展了这一方向。OCM 提供了许多企业环境和多租户环境所需的治理基础层，但本身并不是工作负载优化器。CODECO 以 OCM 作为治理层，并通过应用邻域添加有界联邦，同时引入协同编排循环，将计算、网络和数据可观测性整合到放置和适应决策中 [1]。

Karmada 通过多集群调度、传播策略以及跨集群资源放置扩展 Kubernetes，为多云运作提供了实用的控制平面 [26]。其设计基本上将网络和数据约束视为外部信号。CODECO 通过明确区分推荐（PDLC-FC）与执行（SWM-FC），并引入通过 Kubernetes 资源和 Prometheus 暴露的网络感知与数据感知输入（NetMA-FC 和 MDM-FC），为此类系统提供补充。

9.2 联邦调度方法

PHARE 旨在实现跨联邦边缘集群的多组件应用放置，并明确应对网络变化下的有限本地资源和跨集群组合 [27]。尽管 PHARE 展示了联邦感知调度的重要性，但它没有在统一的 Kubernetes 原生控制接口下，为隐私保护学习、网络能力暴露和数据治理提供集成式技术栈。

CODECO 通过结合（i）邻域范围联邦、（ii）隐私保护去中心化学习（PDLC-FC）、（iii）可编程安全集群间覆盖网络和 ALTO 风格成本图（NetMA-FC），以及（iv）基于元数据图的可观测性（MDM-FC），扩展了这一研究方向，从而实现兼顾数据–计算–网络的调度决策。

9.3 以边缘为中心的 Kubernetes 框架

KubeEdge[18] 通过云/边缘分离控制平面 [28, 29]扩展了 Kubernetes，具备边缘自治、设备管理能力，并能容忍间歇性连接。它的优势在于边缘执行和设备集成，而跨提供商联邦和多集群优化并非其主要关注点。CODECO 与之互补：它假设边缘运行 Kubernetes，同时专注于跨管理领域的联邦编排。

OpenYurt[19] 同样强调边缘自治，并尽量减少对上游 Kubernetes[30]的偏离。相比之下，CODECO 将联邦提升为一等关注事项，结合基于 OCM 的治理、学习辅助推荐，以及明确的网络和数据约束，以支持跨集群的放置和迁移。

9.4 集群间连接与服务网络

Submariner 提供实用的跨集群连接和服务发现 [31]。虽然能够有效实现通信，但它并未向调度器提供统一的、

[17] https://github.com/kubernetes-retired/kubefed

[18] https://kubeedge.io/ [19] https://openyurt.io/

多指标网络视图。CODECO 基于这些原语构建，并通过 NetMA-FC 和 ALTO 风格的成本映射，使网络指标（延迟、带宽、丢包、能耗）可观测且可操作，从而对其进行扩展 [32, 33]。

Cilium Cluster Mesh 利用扩展伯克利数据包过滤器 [34]，提供可扩展的多集群网络和策略执行。CODECO 的不同之处在于，它将受监控的网络成本和服务类别与编排决策紧密结合，并将其纳入 PDLC-FC 推荐和 SWM-FC 执行中。

9.5 面向联邦 Kubernetes 的学习辅助调度

已有研究提出基于学习的调度器，以改进分布式和联邦环境中的放置与自适应 [35, 17, 36]。这些方法通常依赖集中式遥测或全局训练。
CODECO 引入了一种去中心化且保护隐私的学习流程：原始遥测数据保留在本地，仅交换受保护的学习产物，并将推荐与执行分离，以保持稳定性和可审计性 [1]。

9.6 数据/元数据感知编排与合规性

数据感知编排通常通过临时的亲和性规则或外部策略引擎来处理。关于企业元数据图和数据织构的研究表明，更丰富的元数据集成具有重要价值 [9, 37]。CODECO 提供了一个显式的元数据管理平面（MDM-FC），将数据局部性、新鲜度和合规性作为可查询的编排输入，从而能够在联邦集群中做出持续符合策略的放置决策。

9.7 与联邦 Kubernetes 方案的比较

表 3 对比了 CODECO 与具有代表性的联邦和多集群 Kubernetes 方案，即 Liqo、PHARE 和 Karmada。对比重点关注架构范围、联邦模型、调度策略、网络与数据意识，以及对人工智能辅助编排的支持。CODECO 的特色在于通过应用邻域实现有界联邦，并结合跨层可观测性和隐私保护的去中心化学习，从而实现 CEI 连续体上的认知协同编排。

10 结论

本文介绍 CODECO，这是一种认知型、去中心化的编排框架，它扩展了 Kubernetes，以支持跨 CEI 连续体的联邦操作。CODECO 通过引入数据—计算—网络协同编排模型、通过应用邻域实现有界联邦，以及与 Kubernetes 原生控制和治理机制保持兼容的人工智能辅助决策支持，解决了云中心编排的局限性。

CODECO 的核心贡献在于其架构层面的关注点分离。通过基于 OCM 的 Hub 控制和策略执行，保留集中式治理；与此同时，执行、监控和学习则以应用相关邻域为范围进行去中心化。这种混合模型无需采用全局状态共享或紧密耦合的控制回路，即可在连接间歇的情况下实现可扩展性、隐私保护和韧性。

CODECO 进一步展示了如何将跨层可观测性提升为一等编排输入。通过集成网络感知（NetMA-FC）、数据与合规性元数据（MDM-FC），以及隐私保护的去中心化学习（PDLC-FC），CODECO 能够作出明智的放置、迁移和适应决策，超越以计算为中心的调度。将基于学习的推荐与确定性执行明确分离，可降低人工智能驱动的编排中常见的不稳定和振荡风险。

为支持可复现评估，本文介绍了 CODEF，这是一种软件定义的实验框架，可自动完成联邦 CEI 场景的部署、执行和分析。CODEF 通过为跨异构基础设施及地理分布式测试床的大规模实验提供实用基础，为 CODECO 提供了补充。

尽管 CODECO 已展现出全面且可扩展的设计，但仍存在重要的研究挑战，包括隐私保护联邦调度的最优聚合策略、混合式人工智能辅助控制回路的形式化稳定性保证，以及对多 Hub 和跨联邦运行的支持。解决这些挑战对于在互联网规模部署认知编排框架至关重要。

致谢

本研究由欧盟委员会在地平线欧洲计划 CODECO 项目框架下资助（资助编号：101092696），并由瑞士教育、研究与创新国务秘书处（SERI）根据合同编号 23.00028 提供资助。

我们感谢 CODECO 联盟的所有成员为 CODECO 交付成果 D8、D9 和 D11 作出的宝贵贡献。此外，我们特别感谢

表3：CODECO 与代表性联邦 Kubernetes 方案的比较 .

维度	CODECO	Liqo	PHARE	Karmada
主要关注点	跨层协同编排	资源共享，泛洪 of-	联邦边缘部署- ment	多集群传播- tion
联邦模型	有界邻域	点对点 虚拟集群	联邦边缘集群	中心辐射式
治理	混合式（中心策略、本地执行）	去中心化对等连接	按部署集中式 ment	集中式枢纽
调度范围	邻域范围	本地 + 远程卸载- ing	跨集群	跨集群
调度策略	确定性 + 人工智能指导	调度器扩展	启发式 / 优化	策略驱动
人工智能 / 学习	去中心化，隐私保护	None	None	None
网络感知	一等（指标，cost 映射）	基本连接	有限 / 隐式	有限（策略提示）
集群间网络	安全 覆盖网络，SDN- 辅助的	基于网关	外部／假定	外部（例如，Sub- mariner）
数据/元数据感知	显式元数据图	None	有限	None
隐私与数据主权	受保护的本地遥测数据交换	基于信任的共享	未涉及	未涉及
边缘自治	专为应对间歇性而设计	部分	有限	有限
多提供商支持	明确针对	隐式	面向研究	企业／云
运行时适应	主动 + 被动	被动	被动	被动
Kubernetes 集成	原生（CRD、基于 OCM）	原生	原型	原生

以下同事：fortiss 的 Tina Samizadeh 为 CODECO 标准化和实验作出的贡献；Netcompany 的 Dorine Matzakou 为 CODECO 学习中心和 CODECO 成果利用计划作出的宝贵贡献；Almende 的 Andries Stam 为 CODECO 可持续性和资产利用计划提供的领导支持；Eviden 的 Ignacio Prusiel 为 CODECO 可观测性层初期开发作出的贡献；David Remon 和 Marco Jahn 为 CODECO 开源社区及 Eclipse KUDECO 项目开发作出的宝贵贡献。

参考文献

[1]R. C. Sofia, J. Salomon, S. Ferlin-Reiter, L. Garc´es-Erice, P. Urbanetz, H. Mueller, R. Touma, A. Espinosa, L. M. Contreras, V. Theodorou等，《认知型去中心化容器编排框架》，*IEEE Access*，第 12 卷，第 79 978–80 008 页，2024 年。[2]S. Bradner。（1997年3月）用于在 RFC 中表示需求级别的关键词。[在线]。可访问：https://www.rfc-editor.org/rfc/rfc2119[3] G. D. Abowd、A. K. Dey、P. J. Brown、N. Davies、M. Smith 和 P. Steggles，《迈向对上下文及上下文感知的更深入理解》，载于第 *1* 届手持式与普适计算国际研讨会论文集。Springer，1999年，第 304–307 页。

[4]N. Psaromanolakis（编），等，《CODECO D11：CODECO 基本操作与开放工具包 v1.0》，CODECO 交付成果系列，技术报告，2023年10月。 [在线]。可访问：https://zenodo.org/records/10148463

[5] W. Liu、X. Li 和 D.Huang，《上下文感知综述》，载于*2011*年国际计算机科学与服务系统会议（*CSSS*）。IEEE，2011年，第144–147页。

[6] R. C. Sofia，《社会感知网络在支持6G物联网中的作用》，载于面向可持续和以人为中心的未来的*6G*愿景。River Publishers，2023年，第55–78页。

[7] Feeney, L.M. 和 Nilsson, M.，《在自组织网络环境中研究无线网络接口的能耗》，载于*IEEE INFOCOM 2001*论文集：计算机通信大会——*IEEE*计算机与通信学会第二十届年度联合会议（目录编号：*01CH37213*），第3卷，2001年，第1548–1557页，第3卷。

[8] Ongaro, Diego 和 Ousterhout, John，《寻找易于理解的共识算法》，2014年。

[9] S. Rooney、L. Garc´es-Erice、D. Bauer 和 P. Ur- banetz，《Pathfinder：构建企业数据地图》，载于 *2021*年 *IEEE* 大数据国际会议（*Big Data*）。IEEE，2021年，第1909–1919页。

[10]Stoica, Ion、Morris, Robert、Liben-Nowell, David 和 Karger, David R. 以及

Kaashoek, M.Frans、Dabek, Frank 和 Balakrishnan, Hari，《Chord：一种适用于互联网应用的可扩展对等查找协议》，*IEEE/ACM* 网络学报，第 11 卷，第 1 期，第 17–32 页，2003年。

[11]G. Koukis, S. Skaperas, I. Angeliki Kapetanidou, V. Tsaoussidis, L. Mamatas，《边缘云连续体的开源实验框架》，载于*INFOCOM 2024 CNERT*：基于测试平台的计算机与网络实验研究研讨会（*CNERT*）。IEEE，2024年。 [在线]。可访问：https://doi.org/10.5281/zenodo.10840008[12] A. Ullah, T. Kiss, J. Kov´acs, F. Tusa, J. Deslauriers, H. Dagdeviren 和 H. Hamzeh，《云到物的计算连续体中的编排：分类法、综述与未来方向》，《云计算杂志》，第12卷，第1期，第135页，2023。[13] C.-H. Hong 和 B. Varghese，《雾计算/边缘计算中的资源管理：架构、基础设施与算法综述》，*ACM*计算调查，第52卷，第5期，第1–37页，2019年。[14] G. Baldoni, L. Cominardi, M. Groshev, A. Oliva 和 A. Corsaro，《管理远端边缘：当今的集中式解决方案是否合适？》《*IEEE*消费电子杂志》，第12卷，第3期，第51–61页，2021年。[15] Larsson, Lars、Gustafsson, Harald、Klein, Cristian 和 Elmroth, Erik，《去中心化 Kubernetes 联邦控制平面》，载于*2020 IEEE/ACM 第13*届国际实用与云计算会议（*UCC*。IEEE，2020年，第354–359页。[16]A. Cannarella，《Kubernetes 集群间资源代理的多租户联邦方法》，2022年。[17]T. Zheng, J. Wan, J. Zhang 和 C. Jiang，《基于深度强化学习的边缘计算工作负载调度》，《云计算杂志》，第11卷，第1期，2022年。[18] X. Xiong, K. Zheng, L. Lei 和 L. Hou，《物联网边缘计算中基于深度强化学习的资源分配》，《*IEEE*通信选题领域杂志》，第38卷，第6期，第1133–1146页，2020年。[19]L. Bus¸oniu, R. Babu˘ska 和 B. De Schutter，《多智能体

概述”，载于多智能体系统与应用创新。柏林和海德堡：Springer，2010年，第183–221页。[20] J. Browne等（2019年）闭环自动化：用于服务自愈与平台韧性的遥测感知调度器。英特尔。[在线]。可访问：https://bit.ly/3RgHmEP[21] Zhang, K.、Zhuoran, Y. 和 Basar, T.，“多智能体强化学习：理论与算法的选择性概述”，载于强化学习与控制手册，2021年，第321–384页。[22] Bellavista, Paolo、Foschini, Luca 和 Mora, Alessio，“联邦部署环境中的去中心化学习：系统级综述”，*ACM*计算机调查，第54卷，第1期，2021年2月。 [在线]。可访问：https://doi.org/ 10.1145/3429252[23] B. C. S。enel、M. Mouchet、J. Cappos、T. Friedman、O. Fourmaux 和 R. McGeer，“通过 Fed4FIRE 联邦化 EdgeNet+ 并将其节点部署在网络地址转换（NAT）之后”，载于*2022*年*IFIP* 网络会议（*IFIP Networking*）。IEEE，2022年，第1–5页。[24] Casalicchio, Emiliano，“容器编排：一项调查”，系统建模：方法与工具，第221–235页，2019年。[25]Open Cluster Management作者，“Open Cluster Management：Kubernetes多集群生命周期与策略管理（项目）”，https: // github.com/open-cluster-management-io，访问日期：2026年1月9日。[26] Karmada项目，“Karmada：Kubernetes多集群管理（项目）”，https://github. com/karmada-io/karmada，访问日期：2026年1月 09日。[27] Galantino, Stefano、Filaferro, Gabriele 和 Nicosia, Luca，“使用 PHARE 在联邦边缘集群间调度多组件应用”，*IEEE*通信学会开放期刊，第5卷，第1814–1826页，2024年。[28]Mueller, H.、Gogouvitis, S.V.、Seitz, A. 和 A., B.Bruegge，“跨越云与边缘的工业系统无缝计算”，载于*2017*年高性能计算与仿真国际会议（*HPCS*。IEEE，2017年，

g化学习 g: An p。209–216。

[29]Gogouvitis, S.V.、Mueller, H.、Prem- nadh, S.、Seitz, A. 和 Bruegge, B.，《利用容器编排实现工业系统中的无缝计算》，*FutureGeneration Com-puterSystems*，第 109 卷，第 678–688 页，2020 年。

[30]OpenYurt 项目，“OpenYurt：将 Kubernetes 扩展至边缘计算（项目）”，https://github.com/openyurtio/openyurt，访问日期：2026年1月9日。

[31]Submariner 项目，“Submariner：Kubernetes 集群的跨集群连接性（项目）”，https://github.com/submariner-io/submariner，访问日期：2026年1月9日。

[32] R. Alimi、R. Penno、Y. R. Yang、S. Kiesel、S. Previdi、W. Roome、S. Shalunov 和 R. Woundy。（2014-09）应用层流量优化（ALTO）协议。 [在线]。可访问：https://www.rfc-editor.org/rfc/rfc7285

[33] Camaz, G. 和 Caldas, P. 和 Sousa, P.，“应用层流量优化：ALTO 解决方案的概念验证、开发与实用性评估”，载于第 *15* 届伊比利亚信息系统与技术会议（*CISTI*），塞维利亚，西班牙，2020年，页码 1–6。

[34] S. Qi、S.G.Kulkarni、K.K. Ramakrishnan，“容器网络接口插件评估：功能、性能与可扩展性”，《*IEEE* 网络与服务管理汇刊》，第 18 卷，第 1 期，页码 656–671，2021年。

[35] Huang、Jiaming、Xiao、Chuming 和 Wu、Weigang，“RLSK：一种基于强化学习的联邦 Kubernetes 集群作业调度器”，发表于 *2020* 年 *IEEE* 云工程国际会议（*IC2E*），2020年，第116–123页。[36]Senjab、Khaldoun、Abbas、Sohail、Ahmed、Naveed 和 Khan、Atta ur Rehman，“Kubernetes 调度算法综述”，《云计算杂志》，第12卷，第1期，2023年，文章编号87。

[37]Ghiran, A.M. 和 Buchmann, R.A.，《模型驱动的企业数据织体：基于概念建模和知识图谱的提案》，载于 知识科学、工程与管理。*KSEM 2019*。*Springer*《计算机科学讲义》。Cham：Springer，2019年，第11775卷。