arXiv:2601.06344v1 [cs.RO] 9 Jan 2026

BLAZEAIOT：一种用于边缘、雾和云基础设施上实时分布式机器人的模块化多层平台

预印本

Cédric Melançon
电气工程系
高等技术学院
蒙特利尔，加拿大
cedric.melancon.1@ens.etsmtl.ca

Julien Gascon-Samson
软件与信息技术工程系
高等技术学院
蒙特利尔，加拿大
julien.gascon-samson@etsmtl.ca

Maarouf Saad
电气工程系
高等技术学院
蒙特利尔，加拿大
maarouf.saad@etsmtl.ca

Kuljeet Kaur
电气工程系
高等技术学院
蒙特利尔，加拿大
kuljeet.kaur@etsmtl.ca

Simon Savard电气工程系 高等技术学院 蒙特利尔，加拿大
si
mon.savard.2@ens.etsmtl.ca

2026年1月13日

摘要

分布式机器人技术日益复杂，推动了对能够无缝整合边缘、雾和云计算层，同时满足严格实时约束的平台的需求。本文介绍 BlazeAIoT，这是一种模块化多层平台，旨在统一异构基础设施中的分布式机器人系统。BlazeAIoT 提供动态数据传输、可配置服务和集成监控，同时确保韧性、安全性和编程语言灵活性。该架构采用基于 Kubernetes 的集群、代理互操作性（DDS、Kafka、Redis 和 ROS2）以及自适应数据分发机制，以优化不同环境中的通信与计算。所提出的解决方案包含多层配置服务、动态自适应数据桥接，以及用于处理大型消息的分层速率限制。该平台通过涉及导航和人工智能驱动的大规模消息处理的机器人技术场景进行了验证，结果表明其在实时约束下具有稳健的性能。结果凸显了 BlazeAIoT 在不完整拓扑中动态分配服务、维持系统健康状态并尽可能降低延迟的能力，使其成为面向机器人及更广泛物联网应用（如智慧城市和智能工厂）的成本意识型可扩展解决方案。

关键词 自适应通信协议 · 云机器人 · 动态数据分发 · 分布式机器人 · 边缘机器人 · 雾机器人 · 物联网平台。

1 引言

分布式机器人概念最初随着云机器人的出现而产生 [1] ，此后发展出边缘机器人 [2] 和雾机器人 [3]。从本质上说，机器人领域的分布式计算旨在应对机载资源的限制，以及电池供电系统所带来的能源自主性约束。将密集计算任务卸载到外部层可以缓解这些限制，但同时也会带来与网络连接 [4], 、延迟 [5], 以及分布式拓扑复杂性 [6]相关的挑战。

本研究基于我们此前开发的分布式数据流框架 BlazeFlow [7], 展开，该框架有助于在多层分布式配置中衔接数据。本文提出的 BlazeAIoT平台旨在通过单一解决方案涵盖分布式机器人技术的三个不同概念，即边缘、雾和云机器人。为展示其可行性，本文提出了一个具体的机器人应用场景，用以评估计算任务在这些层之间的分布，并考察其对机器人执行任务所产生的影响。鉴于大多数机器人服务都在严格的实时约束下运行，本文特别关注评估服务能否在每一层可靠执行，以及网络延迟和其他因素是否会导致某些部署不切实际。

1.1 动机

尽管文献中已提出多个分布式机器人平台（详见第 2 节），但没有一个能够同时支持全部三个层级（边缘、雾和云）。BlazeAIoT旨在作为一个平台提供更多功能，以应对此类分布式解决方案带来的挑战。开发新型分布式机器人平台的动机源自多个目标：

- **注重成本的解决方案**：利用开源组件，并与现有企业基础设施保持与供应商无关的兼容性，从而最大限度地降低平台成本。
- **编程语言灵活性**：支持多种语言（C++、Python、C#、NodeJS 等），以便与各类服务集成，尤其是基于机器人操作系统（ROS）构建的服务 [8]。
- **多层支持**：确保平台在边缘、雾和云层中均具备稳健性，并支持不完整拓扑（例如，缺少雾层）。
- **灵活的服务分布**：允许在任意层执行服务，但需要硬件交互的专用服务除外。
- **实时性能**：选择能够最大限度减少开销并满足严格实时约束的技术。
- **稳健性与安全性**：在确保数据传输中和静态存储时均安全的同时，提供抵御连接中断和服务故障的韧性。

该平台虽针对机器人学进行了优化，但同样适用于更广泛的物联网（IoT）和分布式系统场景，包括智慧城市、智能工厂等。

1.2 贡献

本研究的贡献如下：

- **面向分布式机器人的服务导向架构**：所有计算均通过服务完成，从而支持多层部署与配置。
- **动态数据传输**：服务可动态共享和使用数据，平台会自动连接跨异构代理的相关服务。
- **可配置服务**：提供标准化配置及可定制选项，以满足特定部署需求，确保服务能够适应执行环境。
- **监控方案**：集成性能指标采集机制，从而能够在层、节点和服务级别进行优化与系统健康监控。
- **ROS 集成**：将 ROS 软件包无缝集成到平台中，以涵盖大多数机器人学应用场景 [9], ，同时保持服务模块化。
- **多层部署**：在多集群／多租户环境中演示边缘、雾和云的集成，展示如何策略性地分配服务，以平衡延迟、资源限制和计算速度。

1.3 文章结构

本文的组织结构如下。第 2 节讨论了当前研究现状。第 3 节和第 4 节详细介绍了所提出的 BlazeAIoT 平台的架构，以及该平台如何契合第 1.1 节所述的目标。第 5 节介绍实验基础设施，包括机器人、各种 ROS 软件包和网络拓扑。第 6 节介绍评估场景，并分析平台在多个层上的性能。第 7 节讨论实验结果，并将平台与既定目标进行比较。第 8 节总结讨论结果，并展望未来的研究方向。

2 相关工作

分布式机器人领域的许多概念并非机器人学所独有。类似的原则也应用于物联网（Internet of

Things，IoT），它同样依赖边缘、雾和云拓扑结构 [10]。作者介绍了两种常见的物联网架构：三层架构（感知层、网络层和应用层）以及面向服务的架构（感知层、网络层、服务层和接口层）。在 [11], 中，作者讨论了在信息物理系统（CPS）中使用边缘计算和雾计算，以应对云计算的局限性，例如高延迟、带宽受限以及安全与隐私问题。机器人可被视为信息物理系统，因为它们是需要实时交互的可控系统。

过去十年间，云机器人得到了广泛研究。在 [12], 中，作者介绍了云机器人的概念，并详细阐述了其实现面临的挑战。他们指出了可能产生大量网络传输数据的计算挑战，以及网络延迟和潜在连接中断等通信挑战。XBot2D 项目 [13] 提出了一种面向机器人即服务（RaaS）场景的云机器人架构。为缓解网络故障，XBot2D 平台会在检测到网络性能下降时，将机器人进程切换到本地执行。为了适应资源可用性，任务会在两种环境（云端和本地）中以不同速率（200 赫兹和 1 KHz）执行。然而，这种方法会持续占用两个层的资源，这对电池供电机器人而言是个问题。

在 [3] 中，作者比较了面向人机交互场景的雾机器人和云机器人解决方案。他们提出了一种卸载方法，以优化任务执行。如果机器人无法执行任务，就会将任务卸载到雾计算；若仍无法执行，则最后将其卸载到云端。尽管云计算几乎能提供无限的资源，多层卸载也会引入延迟，可能妨碍实时执行。 [14] 的作者还将雾计算引入 E-poll 方案，以降低延迟并提高云计算的可靠性。他们还提出了一种节点配置方案，使节点能够在连接到网络时获取自身配置。收到的配置将影响数据包路由，从而优化通信速度和可靠性。雾计算也已应用于医疗保健等其他领域 [15]。 [16] 的作者使用资源管理模拟器，证明了与云计算相比，雾计算在实时且对延迟敏感的场景中的优势。

如 [17] 所述，边缘机器人与云机器人相结合，可从 5G 网络中获益。作者提出了一种基于控制质量的资源分配方案，以根据网络延迟、可靠性和任务周期性优化计算部署。尽管这一方案前景可期，但其部署策略是静态的，未考虑机器人环境的动态变化。虽然这并非当前工作的组成部分，但架构中已将这一方面纳入考虑，以便未来改进。类似地， [18] 探讨了 5G 赋能的边缘机器人在取放流程中的应用。文中提出了一种混合回退机制，将服务质量（QoS）感知的机器人适应与控制感知动态服务质量编排相结合，以应对网络不稳定并缓解机器人受到的限制，同时确保任务执行不会在不同层之间重复。该方案采用了神经网络，并强调需要一个稳健的模拟器，以便在真实环境中实现集成。

分布式机器人通常旨在将人工智能（AI）相关任务卸载出去，以减轻机器人的计算负担。在这方面，正如 [19], 所提出的那样，引入数字孪生是一种有价值的方法，可用于收集训练数据并利用仿真来改进人工智能模型。作者提出了一种分布式框架，使用 Kafka 作为数据代理，并使用 ROS 进行机器人通信。他们将应用划分为三个功能层：人工智能推理、车间（含物理设备）以及边缘云（负责数字孪生和非人工智能计算）。在分布式机器人解决方案中编排人工智能模型同样具有挑战性。 [20] 的作者提出了一种人工智能模型推理负载均衡方案，该方案在优先考虑能效的同时确保实时执行，尤其适用于电池供电的边缘节点。分布式机器人的另一个重要方面，是能够在各层之间高效、可靠地共享数据。 [21] 的作者提出了对比研究，将 Kafka 的基于消息队列的通信方式与传统的基于 HTTP 的通信方式进行比较。他们证明，随着节点数量增加，使用代理可以降低延迟。消息量会显著影响网络延迟。 [22] 的作者提出了一种基于 HTTP 的通信方式，可将消息优化分段，并通过多个并发线程发送。每个分段都会加密，服务器随后重新组装消息，从而以安全的方式发送此类消息并降低总体延迟。 [23] 提出的平台解决方案还引入了分布式架构，其中设备注册流程会改变解决方案的拓扑，并采用流共享方法来实现数据流。作者使用基于远程过程调用（RPC）的方式在云端实现数据通信，并使用消息队列遥测传输（MQTT）进行层间通信。正如 [7], 所述，MQTT 在高容量、高吞吐量场景下存在局限，这可能会给机器人应用带来问题。降低延迟的另一种方法是通过数据压缩来减小消息大小。 [24] 的作者分析了视觉里程计场景中的压缩效果。压缩会使里程计精度略有下降，但能显著节省带宽。根据任务的关键程度，可以在模型精度和性能提升之间取得平衡。

上述工作凸显了分布式机器人技术与物联网通信方法的多样性，涵盖从以云为中心的架构到雾和边缘解决方案，且通常依赖特定代理或静态放置策略。我们之前的工作 BlazeFlow [7], 提出了多层数据桥接的初步构想，并

展示了跨异构代理进行动态通信的潜力。虽然 BlazeFlow 奠定了基础，但尚未提供用于服务部署和编排的完整平台。与上述方法不同，BlazeAIoT 通过采用面向服务的架构推进了这一构想，将边缘、雾和云层之间的配置、监控及自适应数据分发整合起来。这使 BlazeAIoT 成为一个统一且具备韧性的实时分布式机器人技术平台。

3 提出的解决方案

本节概述了所提出的平台 BlazeAIoT 的架构，该平台支持自主机器人解决方案。我们将逐一介绍各个子系统，并说明其设计动机，以满足既定目标。

3.1 系统概述

图 1 展示了 BlazeAIoT 的高层架构。系统分为三个不同的子系统：边缘层、雾层和云层。所有层均通过数据分发服务（DDS）这一层间通信协议互连，从而避免始终依赖雾层转发消息，因为这会增加延迟。此外，这种设计支持更灵活的拓扑结构，各层可以按需省略或复制以实现冗余，从而确保高可用性。每一层都是由开源解决方案 Rancher[1] 管理的 Kubernetes 集群。边缘层集群采用轻量级开源 Kubernetes 发行版 K3S[2], , 主要因为它兼容 ARM64 架构，且适合边缘节点有限的资源。雾层和云集群则运行开源的 RKE2 Kubernetes[3]。

3.2 BlazeAIoT SDK：适用于每项服务的标准接口

BlazeAIoT SDK 是一个提供多种语言版本的库。目前支持的语言包括 C++、C# 和 Python，涵盖了自主机器人解决方案的大多数可能场景；同时也可以轻松集成对其他语言的支持。
SDK 是用于与 BlazeAIoT平台集成的实用工具。系统会为服务创建一个 ServiceManager 单例，作为各种管理工具的统一访问点：

- BaseConfiguration：解析配置文件，并提供对层、节点和服务配置的访问。
- 代理：提供对本地代理的通用访问，使其能够在主题上发布消息，并通过回调函数订阅主题，以支持事件驱动场景。
- 配置管理器：通过提供用于更改节点、层和服务配置的回调函数，使服务能够响应配置变更。
- 流管理器：使服务能够告知其他服务其生成（发布）和使用（请求）的主题。
- 监控管理器：作为 Prometheus[4] 的封装。Prometheus 是一款开源的指标与监控中间件。它会创建一个可供外部访问的指标服务器，并提供异步方法来存储指标，例如计数器和仪表盘。
- 缓存管理器：借助 Redis[5] 作为缓存，提供一些功能。它通过看门狗来共享服务可用性，并创建一个计时器，用于刷新数据库中的哈希值。该值会过期；如果在指定时间内未更新，最终将从数据库中删除。

使用 Redis 验证服务可用性的主要原因，是为了同步平台中相互依赖的服务（例如，BlazeConfig 需要 BlazeFlow，而 BlazeFlow 也需要 BlazeConfig）。否则，如果未遵循这一顺序，可能会出现未定义行为。

3.3 BlazeConfig：多层配置

BlazeConfig 服务是 BlazeAIoT 解决方案中的一项新增内容。它支持在该解决方案的所有层中管理并同步层、节点和服务配置。BlazeConfig 由三个组件组成：

- **MongoDB数据库**：该组件负责存储配置。在边缘层，每个节点都有自己的 MongoDB[6] 实例；在雾层和云层中，则可以将其部署为 Kubernetes 应用或部署在特定节点上。
- **BlazeConfig主服务**：该服务托管在最中心化的层（可用时为云层，否则为雾层）。它提供 REST API，用于管理系统各个方面的全局配置。所有数据均存储在 MongoDB数据库中。
- **BlazeConfig工作服务**：此服务管理节点/层的本地配置。在边缘层，它以 Kubernetes守护进程集的形式部署（每个节点运行一个实例，并连接到

[1]https://rancher manager.docs.rancher.com/ [2] https://docs.k3s.io/ [3]https://docs.rke2.io/

[4]HTTPS://prometheus.io/ [5] HTTPS://redis.io/community/ [6] HTTPS://www.mongodb.com/community/

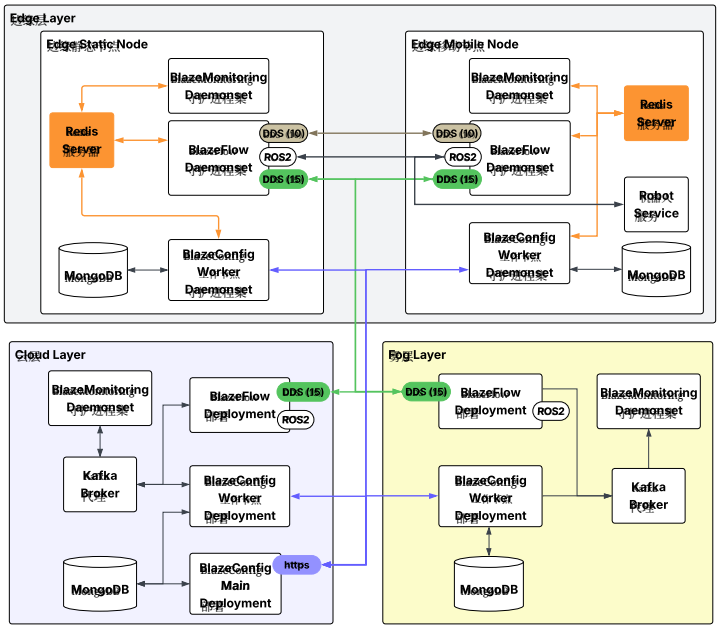

图 1：BlazeAIoT 高层架构。

本地 MongoDB数据库）；在雾层和云层，则以 Kubernetes 部署的形式部署。每个服务都通过本地代理从该服务获取配置（边缘层为节点内，雾层和云层为层内）。

工作器服务直接从主服务获取整个层的最新配置。收到的配置用于同步工作节点的 MongoDB 数据库。此后，主服务与各代理之间的所有交互均由本地代理处理。每个受变更影响的服务都会通过本地代理收到通知，并采取适当措施。措施可以是更新服务的本地参数，也可以是节点／层参数（例如数据库 IP 地址）。每个服务都负责针对配置变更实施相应措施。

3.4 BlazeFlow：多层通信

BlazeFlow 的初步构想最早发表于一篇研讨会论文 [7]。其目标是在分布式系统中实现高效、可扩展的多层通信。该中间件引入了三种不同的通信代理。节点内通信使用 Redis 在节点内部进行本地通信。不同层的层内通信方式各不相同。在边缘层内进行节点到节点通信时，使用 DDS 中间件；雾层和云层则使用 Kafka[7]。最后，用于在不同层之间传递消息的层间通信采用 DDS 中间件实现。

[7]https://kafka.apache.org/

BlazeFlow 还可通过启用支持其他协议（如 ROS2）的代理插件来实现互操作性。BlazeFlow 桥接器支持任意方向的协议互通（例如，ROS2 ↔ DDS ↔Kafka）。

在保留先前工作的功能名称的同时，BlazeFlow 进行了大幅扩展，以便与 BlazeAIoT 集成：新增了执行动态数据分发的能力，定义了 Kubernetes 中要使用的网络配置，实现了多个集群之间的通信，并引入了自适应数据分发的概念。因此，它是本研究的一项重要创新贡献。

3.5 BlazeMonitoring：系统监控

BlazeMonitoring 是一项服务，可提取关键平台指标，并使其可供后续优化使用。该服务分为两个独立的应用程序。

每一层的每个节点上都会启动 BlazeMonitoring 守护进程集。它负责收集节点指标，包括中央处理器、图形处理器、内存、温度和功耗（如果可用）。它还用于监控解决方案中各节点之间的延迟。ping 计时器任务会按可配置的时间间隔（60 秒）执行，并向解决方案中的所有节点发送 ping（包括该层之外的节点）。节点收到 ping 消息后，会回复 pong 消息；经过的时间除以二，即为两个节点之间的平均延迟。创建此任务而非使用常规 Linux ping 的主要原因是，它还会将 BlazeFlow 桥接器引入的延迟计算在内。

BlazeMonitoring 部署可用于收集特定节点的特定指标。在这项工作中，我们通过计算消息延迟来评估消息大小的影响。第一个服务会在发送前为每条消息添加时间戳。监控服务会监听预先配置的主题并计算经过的时间，该时间代表消息延迟（与守护进程集测得的网络延迟进行比较）。

所有指标均通过 BlazeAIoT SDK 存储在 Prometheus 服务器中，并可用于创建 Grafana 仪表板，或提取到笔记本中进行处理，正如项目实验部分所采用的方式。

4 BlazeFlow 作为动态数据桥接解决方案

本节讨论了集成到 BlazeFlow 中、用于实现 BlazeAIoT 目标的关键组件。

4.1 动态数据分发

BlazeFlow 提供了一种动态数据分发范式，支持自主机器人解决方案。该概念支持动态服务生成，确保服务能够接收预期的消息，并让其他服务对其进行处理。

BlazeFlow 充当动态桥接，支持多层通信。当服务在某个节点上生成时，它会将公布的主题和请求主题提供给本地 BlazeFlow 服务（在边缘节点上通过节点内通信，在雾/云节点上通过层内通信）。

BlazeFlow 根据这些信息执行以下三项操作：

- 存储公布的主题/请求主题，以便在另一项服务涉及同一主题时采取相应操作。
- 如果该层中存在一对公布主题/请求主题，则在两种协议之间创建桥接。
- 将公布主题/请求主题转发到其他层。执行源验证（节点和层），以防止消息循环。

4.1.1 桥接创建示例

图2展示了创建桥接以及桥接消息流转的示例序列。两个 BlazeFlow 方框包含其支持的各类代理。每个代理实例都是远程 BlazeFlow 代理的订阅者。每条收到的消息都会触发一个任务来执行 BlazeFlow逻辑。在此示例中，托管在边缘节点上的服务1必须向服务2发送数据；服务2托管在雾层或云层（这对逻辑没有影响）。首先，服务1要通告主题 'topic1'，因为服务1希望在该主题上发布。该消息被发送到节点内代理，并由 BlazeFlow服务接收。通告消息会转发到层内代理和层间代理，并存储在每个代理中。在下一层，BlazeFlow服务通过层间代理接收通告消息，并将其转发到层内代理。随后，它会将通告主题存储在每个代理中。

接下来，Service2 将发布请求的主题 'topic1'，因为 Service2 希望接收该主题的消息。该消息发送到层内代理，并由 BlazeFlow 服务接收。请求的消息随后转发到层间代理，并存储在各个代理中。在边缘层，层间代理接收请求的消息，并将其转发到层内代理和节点内代理，各节点会将其存储起来。

由于发现了一组发布/请求，BlazeFlow 服务需要创建桥接。在边缘节点上，会创建一个桥接，将数据从节点内代理传输到层间代理。在雾层/云层上，则会创建一个桥接，将数据从层间代理传输到层内代理。

最后一个操作示例展示了 Service1 在主题 'topic1' 上发送消息。BlazeFlow 桥接接收该消息，并自动将其发布到层间代理。在另一层上，BlazeFlow 桥接从层间代理获取消息，并将其转发到层内代理。Service2 接收该消息，并调用与该消息关联的操作来执行任务（即回调）。

桥接创建的逻辑取决于同一主题及其源是否存在通告/请求对。BlazeFlow 的目标是透明地建立从源服务到目标服务的连续数据流。如果目标位于同一层，则使用层内通信机制（简单情况），无需桥接；但如果目标位于该层之外，则使用层间通信。可能需要多个目标，因此 BlazeFlow 必须通过不同的代理路由消息。

4.1.2 一对多场景示例

图3展示了一个复杂场景示例，其中一个发布者（服务1）和多个订阅者（服务2 和 Service3）分布在不同的层上。此示例使用流程图来省略部分细节。服务1通过 ROS 2 消息代理提供数据。

Service2 部署在同一节点和云层中。Service2 使用默认的 BlazeFlow 代理。为了在边缘层接收数据，需要创建一个桥接，将 ROS2消息发布到节点内代理（Redis发布/订阅）。对于托管在云层中的 Service2，需要通过桥接将 ROS2 数据传输到层间代理（DDS）。在云层的 BlazeFlow 中，需要通过桥接将数据从层间通信

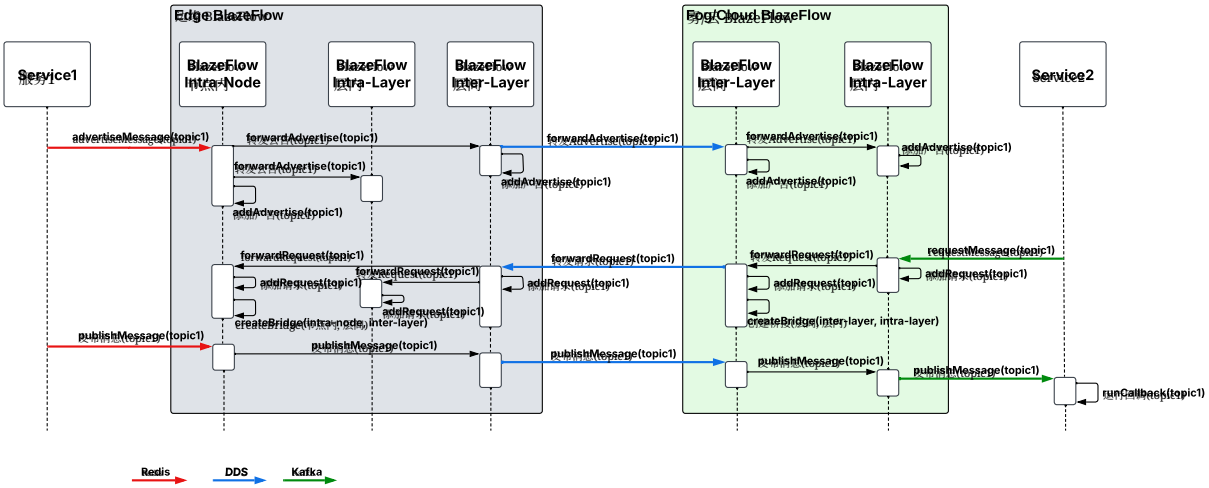

图 2：多层握手序列。

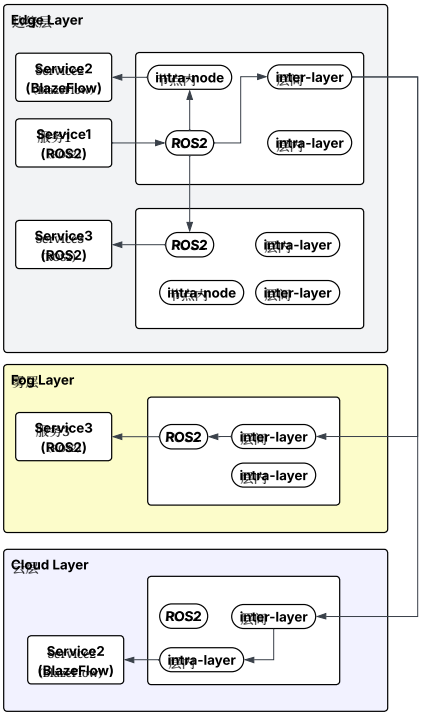

图 3：多桥接场景示例。

（DDS）传输到层内通信（Kafka），从而使 Service2 能够接收数据。

Service3 部署在一个独立的边缘节点上，也部署在雾层上。Service3 预期从 ROS 2 消息代理接收数据。正如前文所述，由于 ROS2 在同一层内被视为本地通信，部署在独立边缘节点上的 Service3 将直接从同一 ROS 2 网络接收数据，无需桥接。对于部署在雾层上的 Service3，它将利用边缘层上现有的桥接（ROS2 → 层间）。此外，还将在雾层的 BlazeFlow 上创建一个桥接，将数据从层间代理（DDS）传输到 ROS 2 层。

4.2 网络考量

平台未直接提供的代理（节点内、层内和层间）被视为可从层内访问，因为所有节点都属于同一个 Kubernetes 集群。集群上会创建一个 macvlan 接口，以便使用 Multus[8] 和 Whereabouts[9]共享数据。这可确保所有节点共享同一个虚拟网络。

层间通信通过 Submariner[10],实现，该组件在集群之间建立 IPsec 隧道。虽然还有其他效率更高的网络解决方案，但其配置更加复杂；因此，本研究采用了默认配置。IPsec 隧道传输对中央处理器的消耗非常大，这限制了网络性能；不过，它确保层间流量经过加密。由于资源有限，层间通信可用的带宽约为 160 Mbps。通过使用配备更快中央处理器的网关节点，或采用不同的电缆驱动技术，可以提升工业部署中的性能。

[8]https://github.com/k8snetworkplumbingwg/multus-cni [9] https://github.com/k8snetworkplumbingwg/whereabouts [10] https://submariner.io/

4.3 自适应数据分发

为优化数据流，所有桥接操作都在单独的任务中执行，并提交到一个根据节点可用资源确定大小的线程池中。服务回调也采用同样的方式。这种并行化的代价是资源消耗增加（内存和计算资源），但同时也能提升性能，以满足实时约束。若不采用这种并行化，对大型消息的支持会显著拖慢整体数据分发，使其无法用于分布式机器人场景。

为了支持摄像头图像等大型消息，BlazeFlow 实现了速率限制逻辑。小型消息优先处理，而大型消息则通过跳过帧来降低速率，使其符合带宽预算内允许的速率。例如，如果我们用未加密隧道替代 IPsec（代价是数据隐私降低），可用带宽预算便会增加，框架也会自动调整其速率限制逻辑。

BlazeAIoT 实现的速率限制逻辑是一种两级分层算法，结合了客户端级带宽分配与发布者级令牌桶调节。在客户端级别，可用带宽定义于公式 1 中。

$$B_{\text{avail}} = L_{\text{client}} \cdot \frac{1024^2}{8} bytes/sec \tag{1}$$

其中 L_{client} st ehmlit n ps Mb

带宽使用基于优先级的分配策略在各发布者之间进行分配：标准发布者首先获得其公布的速率。相比之下，大型消息发布者会分配剩余带宽，同时设置最低阈值 $r_{\min} = 2$赫兹，以防止发生饥饿。该算法通过因子 $\alpha = 1.02$ 考虑消息开销，并强制执行可配置的每条消息最大带宽百分比 $\beta = 0.95$ （默认为 95%），以防止带宽被独占。发布者按照带宽需求 $D_i = r_i^{\text{adv}} \cdot s_i \cdot \alpha$ 降序排序，其中r_i^{adv}为公布的速率， s_i 为以字节计的消息大小。对于每个发布者 i，其分配速率根据公式 2 计算。

$$r_i^{\text{alloc}} = \min\left(r_i^{\text{adv}}, \frac{B_{\text{remaining}}}{s_i \cdot \alpha}, \frac{\beta \cdot B_{\text{avail}}}{s_i \cdot \alpha}\right) \tag{2}$$

其中， $B_{\text{remaining}}$ 在每次分配后根据公式 3 更新。

$$B_{\text{remaining}} \leftarrow B_{\text{remaining}} - r_i^{\text{alloc}} \cdot s_i \cdot \alpha \tag{3}$$

在发布者层面，每个主题都采用容量为 $C = 2$ 个令牌的令牌桶过滤器，如公式 4 所定义。

$$T(t) = \min\left(C, T(t - \Delta t) + r_i^{\text{alloc}} \cdot \Delta t\right) \tag{4}$$

其中， $T(t)$ 是时间 t 时的令牌数量， Δt 是经过的秒数。只有当 $T(t) \geq 1$ 时才会传输消息，每次发布消耗一个令牌： $T(t) \leftarrow T(t) - 1$。这种速率限制机制允许短暂出现高于标称速率的追赶突发，同时维持长期平均合规性。它尽可能按照小载荷消息的发布速率交付这些消息，同时降低大载荷消息的速率，因为大载荷消息会消耗大部分带宽。

为减小消息大小，可以使用 LZ4 算法并通过可配置的压缩级别对有效载荷进行压缩。大型消息的默认压缩级别为 10，根据图像内容的不同，可减少 40%–60%。由于压缩图像的大小并不固定，限速算法始终考虑给定主题的最大大小，并会在接收到更大的消息时调整速率。

5 评估测试平台

本节介绍用于评估 BlazeAIoT 平台的实验环境，包括边缘层、雾层和云层的基础设施拓扑，并描述了机器人及评估所需的各类组件。

5.1 基础设施

为验证该解决方案，采用了图 4 所示的基础设施。由于网络限制，另行配置了一台服务器，作为本地云服务器。

所有层均提供一个 Kubernetes 集群。为便于配置，安装了开源解决方案 Rancher。选择该应用的主要原因是，它能够部署 RKE2 Kubernetes 集群（用于雾层和云层）以及 K3S Kubernetes 集群（用于边缘层）。

为实现多集群通信，配置了 Submariner 应用。每一层至少提供一个网关，网络通信通过该网关传输。每个集群的网关都共享一个由主部署（配置为运维人员）管理的虚拟网络。
Rancher 提供监控应用，其中包含一个 Prometheus 服务器，用于收集指标以监控集群和 Kubernetes 的性能。它还提供了一个 Grafana 部署，其中预先捆绑了多个仪表板。

该层的默认配置、节点和所有服务都存储在 Kubernetes 配置映射中，可将其作为卷挂载到各服务中。当 Blaze-Config 服务不可用时，这一功能非常有用，可用于配置基本功能，例如代理、数据库和缓存的地址与端口。

5.1.1 云集群

云集群中的所有节点都连接到同一个网络交换机。该交换机可通过路由器访问互联网和

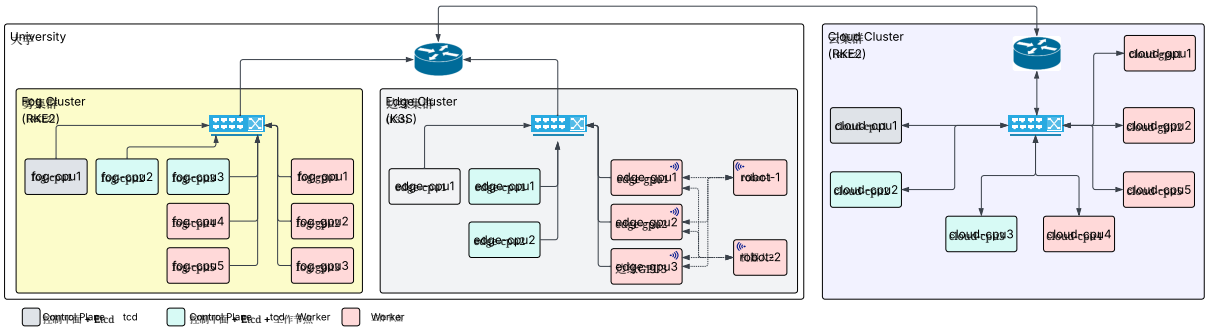

图 4：实验服务器基础设施。

其他集群。由于云网络位于其他集群附近，通过将路由器配置为在连接到承载雾集群和边缘集群的路由器端口上引入延迟，使该场景更加真实。增加了 50 毫秒的延迟和 10 毫秒的抖动，同时丢包率为 0.01%[25]。

云集群是一个由七个节点组成的 RKE2 集群。由于无法使用带虚拟化功能的真实服务器，因此每个节点都是一台配置各不相同的计算机。表 1 描述了每个节点的配置。

表 1：云集群节点

节点名称	CPU	RAM	GPU（显存）
cloud-cpu1	2	16 GB	0
cloud-cpu2	2	16 GB	0
cloud-cpu3	4	16 GB	0
cloud-cpu4	4	32 GB	0
cloud-cpu5	4	32 GB	0
cloud-gpu1	12	32 GB	1（8 GB）
cloud-gpu2	20	32 GB	1（24 吉字节）
总计	48	176 吉字节	2（32 GB）

部署了 Kafka 控制器/代理应用，用于层内通信。通过 Submariner 暴露的 Kubernetes 服务启用基于 DDS 的层间通信。集群上还部署了 Redis 应用。

cloud-cpu3 在 Kubernetes 外部安装了 MongoDB，并向其他节点公开其服务。

BlazeConfig主服务部署在云集群上，其 REST API 通过可经由 Submariner 访问的 Kubernetes 服务公开。BlazeMonitoring、BlazeFlow 和 BlazeConfig 工作应用也部署在该集群上。

5.1.2 雾集群

雾集群也是一个 RKE2 集群，其配置与云集群非常相似。不过，它不使用自己的网络，而是通过同一台路由器共享边缘集群的网络。所有节点都连接到同一台网络交换机，而该交换机连接到路由器；路由器提供互联网接入，并连通云集群网络。表 2 列出了雾网络中每个节点的配置。

表 2：雾集群节点

节点名称	CPU	RAM	GPU（显存）
fog-cpu1	8	8 Gb	N/A
fog-cpu2	8	16 GB	N/A
fog-cpu3	8	16 GB	N/A
fog-cpu4	2	4 Gb	N/A
fog-cpu5	8	16 GB	N/A
fog-gpu1	8	16 GB	1（8 GB）
fog-gpu2	8	16 GB	1（8 GB）
fog-gpu3	12	85 吉字节	1（48 吉字节）
总计	62	177 吉字节	3（64 吉字节）

与云集群类似，雾集群配置了 Redis 和 Kafka，用于层内通信；配置了一个通过 Submariner 暴露的服务，用于层间通信；并部署了 BlazeAIoT 应用（BlazeFlow、BlazeConfig 工作器和 BlazeMonitoring）。

5.1.3 边缘集群

由于节点类型的缘故，边缘集群采用 K3S 集群。本文使用的节点包括 Raspberry Pi 5 等小型计算机，以及各种 Jetson 节点（Nano、NX、AGX）。此外，机器人还配备了一个边缘节点，具体为 Jetson Orin AGX。中央处理器节点安装在网络交换机附近，以尽量降低延迟。图形处理器节点（即 Jetson 设备）分布在测试区域各处，并通过长以太网电缆连接到网络交换机。它们还充当无线网关（配置为 WiFi 接入点）。机器人是一个无线节点，会连接到可用的接入点，以尽量降低延迟。机器人节点还配备了图形处理器，使其能够执行关键的 AI 任务。

表 3 详细列出了边缘网络中每个节点的配置。

表 3：边缘集群节点

Node name	Node type	CPU	RAM	GPU（内存）
edge-cpu1	静态	4	8 Gb	N/A
edge-cpu2	静态	8	16 GB	N/A
edge-cpu3	静态	8	16 GB	N/A
edge-cpu4	静态	16	16 GB	N/A
edge-gpu1	网关	6	8 Gb	1 (8 GB*)
edge-gpu2	网关	8	16 GB	1 (16 GB*)
robot-1	移动设备	8	32 GB	1 (32 GB*)
总计	-	66	144 吉字节	3 (56 吉字节*)

* 共享内存

每个边缘节点都在本地安装了 MongoDB 和 Redis（位于 Kubernetes 之外）。本地 Redis 服务器用于节点内通信和缓存。通过 Multus/Whereabouts 网络附加定义实现使用 DDS 的层间通信，该定义创建了一个由集群中所有节点共享的 MACVLAN 网络。采用此配置的主要原因是它支持 UDP 多播；当多个节点共享相同主题时，这可以降低 DDS 的带宽需求。层间通信的设置方式与其他集群相同，即通过 Submariner 导出的 Kubernetes 服务。

BlazeFlow、BlazeMonitoring 和 BlazeConfigworker 应用以守护进程集的形式部署，这意味着每个节点都会运行一个实例。

5.2 机器人

机器人基于 Pioneer 3-AT 平台构建，如 [26]，所述，并将 Jetson 计算机升级为 Jetson Orin AGX，如图 5 所示。以下小节将介绍机器人导航所用的 ROS 软件包。

5.2.1 机器人定位服务

ROS2 使用一组标准坐标系在地图上定位机器人。这些坐标系包括地图坐标系，它表示机器人运行的全局区域；里程计坐标系，它位于地图坐标系中机器人的起始位置，用于跟踪机器人相对于起始位置的移动；最后是机器人的坐标系，它连接到机器人的基座连杆，表示机器人自身的位置。

这种标准的坐标系分离方式有助于在操作机器人平台时划分职责。地图坐标系可用于测量机器人在关注区域内的位置。然而，鉴于同步定位与建图（SLAM）和 UWB 等全局定位方法的更新速率相对较慢，且往往会产生离散跳变，因此不能直接用于机器人底层控制。另一方面，里程计坐标系能够保证连续，因此



图 5：改装版 Pioneer 3-AT。

适用于机器人控制，但会因里程计传感器误差而产生漂移。

机器人定位通过 ROS2 中使用 Robot Localization 软件包 [27] 实现的一对卡尔曼滤波器来完成。本地滤波器利用轮式里程计以及 IMU 的加速度计和陀螺仪，生成从里程计坐标系到基座连杆的变换。SLAM系统计算出的机器人位姿频率受多个因素制约，包括激光雷达扫描频率和本地卡尔曼滤波器的位置更新频率，因此频率仅为 1.5 赫兹，降低了导航可靠性。为提高这一频率，全局卡尔曼滤波器融合了与本地卡尔曼滤波器相同的里程计数据，以及 SLAM位姿；与里程计数据相比，SLAM位姿更离散，且频率更高。该卡尔曼滤波器对由里程计发布的各个 SLAM位姿之间的位置进行插值，使其能够以 20 赫兹的频率发布全局位置。为获得可比较的真实值，系统使用超宽带（UWB）模块在地图坐标系中生成绝对位置。为滤除 UWB 离散且噪声极大的数据，系统使用一组相同的本地和全局卡尔曼滤波器，以 UWB位姿替代 SLAM位姿，从而生成更可靠、频率为 20 赫兹的 UWB真实值信号。

5.2.2 纯跟踪服务

所使用的路径跟随算法是纯跟踪控制器，可视为一种胡萝卜加大棒法[28]。在各路径点之间绘制直线，以构建路径。机器人具有一个前视距离，并在其周围形成一个圆。控制器通过寻找该圆与由路径点形成的直线的交点来确定目标点，并优先选择距离下一个路径点更近的交点。随后，控制器利用机器人的定位数据和目标点的位置，计算当前时间步的指令。该过程以 25 赫兹的频率重复，直至

每次迭代，直到机器人到达目标点，即列表中的最后一个路径点。控制器根据定位数据和目标点计算指令，该控制器为比例-积分-微分（PID）控制器。

5.2.3 紧急停止服务

该服务是一个 ROS2 软件包，可检测停止手势（所有手指伸展），并触发紧急停止。

用于停止检测的人工智能模型架构详见图 6。手部关键点提取采用 MediaPipe 手部解决方案 [29], ，该方案由一个在大型手部图像数据集上预训练的轻量级卷积神经网络构成。关键点检测流程经过三个全连接层，然后最终判断是否检测到停止手势。该 ROS 软件包会发布紧急停止指示消息，机器人可据此停止所有运动。模型使用 HAGRID数据集 [30]进行训练。

6 实验结果

基于第 5 节详述的实验基础设施，本文在两个场景中对所提出的 BlazeAIoT平台进行评估，以考察计算分布对实时应用的影响。每个场景旨在验证平台的一个特定方面。第一个场景聚焦于速率较高的小型消息，以评估在任意层运行典型导航任务的可行性。第二个场景将大型消息与第一个场景中的消息相结合，以评估自适应数据分发及其对紧急停止等关键场景的影响。

6.1 场景 1：导航

第一个场景是一个不启用主动避障的简单导航场景。机器人使用纯跟踪算法在实验室内导航，从一间办公室移动到开放区域的尽头。

实验分为五个步骤。步骤 1 是直接在 ROS2 中的机器人边缘节点上运行所有计算，而不通过 BlazeAIoT框架执行。此步骤被视为实验的基准。实验旨在评估平台开销，以及在任意层运行服务时延迟对典型导航场景的影响，并将其与基准执行进行比较。

以下步骤均在 BlazeAIoT 框架内执行。步骤 2 在机器人的边缘节点上运行所有机器人服务。步骤 3 将 P3AT 服务保留在机器人的边缘节点上，同时在边缘层（远离机器人）运行所有其他服务。步骤 4 和步骤 5 与步骤 3 类似，但与机器人本身无关的服务分别在雾层和云层上执行。

为便于解读结果，图 7 展示了使用 BlazeMonitor ping 服务从其他各层 ping 时观测到的各层延迟分布。同一层内的延迟在边缘层约为 2.4 ms，在雾层约为 2.2 ms，在云层约为 3.8 ms。与云的通信受到延迟/抖动模拟的显著影响，平均延迟约为 30.1 ms。从边缘层与雾的通信平均延迟约为 3.7 ms。

表 4 显示了 ROS 话题的平均延迟及其发布速率。为测量该延迟，每一层都会创建一个订阅者探针，并由各订阅者将报头时间戳（由 ROS2 节点生成）与本地时间进行比较，以计算延迟。所有节点都与同一台网络时间协议（NTP）服务器同步，从而最大限度地减少时间漂移。

扫描（∼145 ms）和位姿（∼280 ms）主题的延迟相对较高，但这不一定是由框架造成的。激光雷达扫描提供的报头时间戳对应完整旋转的起始时刻，而一次完整旋转需要 100 ms。位姿主题源自同步定位与建图（SLAM），其运行依赖于扫描和里程计/全局主题，以及算法的处理需求；而这些需求又取决于节点可用的资源。

场景 1 的导航结果如图 8 所示，此外还提供了一段视频，展示机器人在每次执行中的行为[11]。基准运行是在机器人节点上直接执行、未使用 BlazeAIoT框架的运行。可以看到，在 BlazeAIoT框架中执行的运行大致沿着基准路径前进，但存在轻微偏差。旋转过程中误差更为明显，这可以用网络延迟导致 SLAM 位姿广播变慢来解释。

图 9 展示了相对于基准运行的欧几里得距离误差。边缘运行的误差更大（最高达 10.7 厘米），这与预期相反。不过，这是因为增加了两个订阅者（分别位于机器人节点和边缘节点）以监控延迟，从而带来了更多流量和更高的 CPU 负载。雾运行的误差低于边缘运行（最高为 8.5 厘米），这归功于雾节点较低的延迟和较强的计算能力。机器人运行的表现优于雾运行，误差最高为 7.3 厘米。云运行的误差最大（最高达 12.9 厘米），但遵循与其他运行相同的路径。误差增大的主要原因是网络延迟更高（+30 毫秒）。机器人收到指令后，会连续执行近两个控制步（频率为 50 赫兹）。机器人始终在努力弥补延迟。

毫不意外，在此场景下，最佳的资源分配方案是在机器人的边缘节点上运行所有内容。通信延迟可以忽略不计，而且由于整个通信过程直接通过 ROS2 进行，无需桥接，因此不会产生额外开销。前提是机器人的节点资源充足。向机器人添加更多服务可能会带来

[11] https://youtu.be/4Bp8EqYhM5c

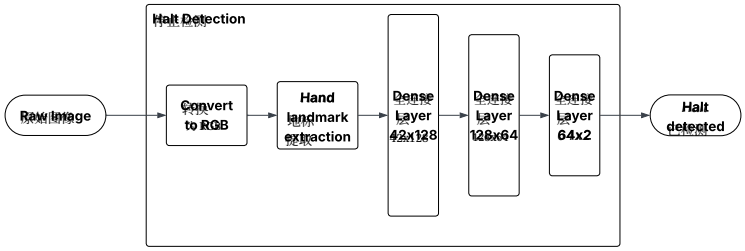

图 6：紧急停止检测模型。

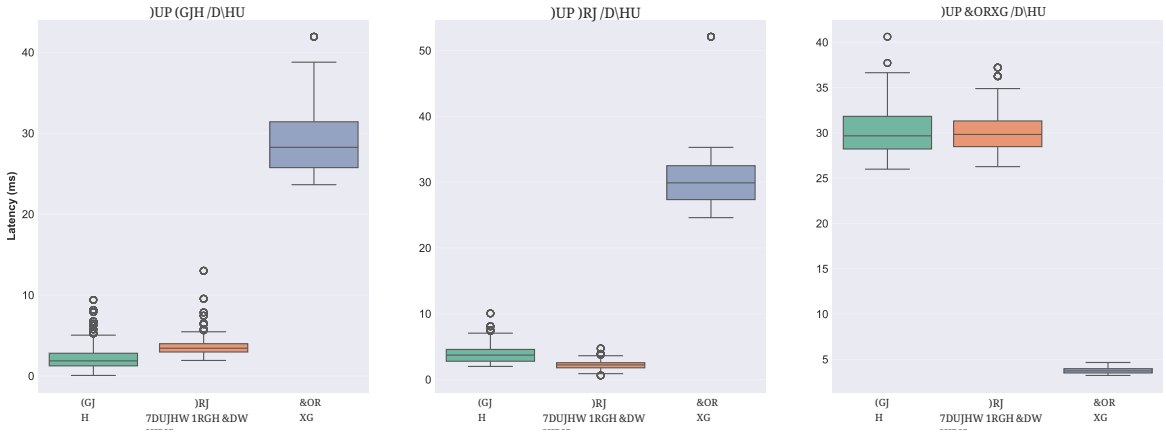

图 7：延迟分布。

表 4：ROS 主题延迟

主题	Rate (Hz)	机器人 延迟（毫秒）	Edge 延迟（毫秒）	Fog 延迟（毫秒）	云 延迟（毫秒）
编码器_odom	49.71	3.26	12.11	10.76	29.26
全局里程计	19.09	7.68	18.98	16.22	32.81
pose	1.09	218.68	285.18	219.71	216.00
scan	4.56	142.84	148.70	147.37	155.08
imu	49.17	3.30	11.26	9.38	28.14
/tf	24.65	21.62	16.67	17.98	34.43

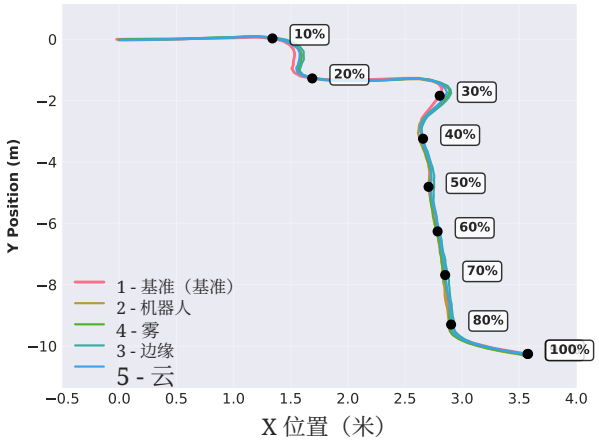

图8：空间地图。

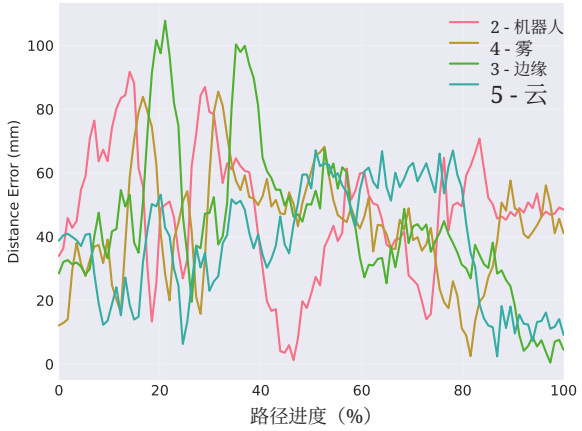

图9：总距离误差（相对于基准）。

不同的结论。这正是以下场景的目标，在该场景中，机器人的节点资源被推至极限。

6.2 场景2：紧急停止

第二个场景与 [13] 中的场景类似，并且提供了一个视频来展示场景的执行过程[12]。该场景涉及让机器人沿直线移动，并使用手势检测来识别完全张开的手，从而触发紧急停止。执行了场景 1 中描述的相同四个步骤。此场景旨在评估高数据量、高吞吐量情况下延迟的影响。摄像头传输并通过 BlazeAIoT框架共享的 RGB 图像会显著受到图像消息数据包大小的影响。此外，紧急停止服务使用图 6 所示的人工智能模型，与其他服务相比，该模型同样需要大量资源。此场景中的相关指标是手势出现与机器人作出响应之间的时间差。

[12]https://youtu.be/PYek5ER0_MM

为比较网络对机器人响应的影响，紧急停止 ROS节点同时在所有层上启动。监控服务订阅紧急停止话题，接收来自所有层的消息，并可比较各层的延迟。运行紧急停止 ROS节点的层节点包括 robot-1、edge-gpu2、fog-gpu3 和 cloud-gpu2。需要注意的是，MediaPipe 手势算法在中央处理器上运行，因为 Jetson 上原生不支持基于图形处理器的版本。尽管如此，结果仍揭示了资源受限节点的一个有趣之处。

表5展示了执行期间获得的指标。

图像帧率的变化是由于速率限制逻辑遵循带宽预算限制。相机图像是首先通过 BlazeFlow 进行桥接的主题。随后添加其余消息，包括 scan、tf、编码器_里程计和 imu 主题；除相机图像外，这些主题共同构成了最大的有效载荷。

机器人上运行的服务以 22 赫兹的速率接收图像，低于预期的 30 赫兹。造成这一差值的首要原因是节点的内存负载。由于 Jetson 与中央处理器和图形处理器共享内存，紧急停止进程、BlazeFlow 的运行以及所有机器人进程共同造成了巨大的内存需求，内存占用率达到 97% 至 100%。因此，紧急停止服务处理一张图像平均需要 725 毫秒。由于图像回调具有可重入性，多个帧会并行处理（约 16 个并发进程），从而增加了典型负载。速率差异的另一个原因与指标监测方法有关，与场景 1 的情况相同。为了计算主题延迟，需要在机器人节点和边缘节点上分别添加一个订阅者，这会增加网络流量以及机器人所需的中央处理器资源，因为默认 ROS 配置未利用多播；这意味着机器人节点会将主题消息发送四次，而不是两次，从而加重中央处理器的负担。

运行在边缘节点上的服务以与机器人相同的速率接收图像。不过，由于该节点未运行机器人和传感器服务，边缘节点的内存使用率约为 85%，处理时间为 190 ms。紧急停止服务的处理速度仍低于输入速率，导致处理线程不断累积（约有四个并发进程）。

为判断是否检测到手势，需要连续三次检测结果为阳性。如果服务在机器人本地执行，不计网络延迟，则生成紧急停止信号约需 861 ms。在边缘节点上，紧急停止会在 333 ms 后触发；而在雾层和云层上，则会在 380 ms 后触发。

从结果来看，紧急停止在雾层上执行得更快，与直接在机器人上运行相比，执行时间减少了 859%，尽管图像

表5：紧急停止指标

Node	图像 速率（赫兹）	图像 延迟（毫秒）	处理 时间（毫秒）	紧急停止 延迟（毫秒）	差值 (%)
robot-1	27	38.6	430	1287.3	-
edge-gpu2	21	40.9	190	581.6	-548
fog-gpu3	7.9	44.7	23	181.3	-859
cloud-gpu2	7.9	172.5	19	550.74	-572

传输到雾层的速率约为预期 30 赫兹的 26%。云端即使速率降低、延迟增高，执行时间仍减少了 572%。最后，在另一台边缘节点上远程执行紧急停止，执行时间减少了 548%。虽然在雾层和云端执行能获得更快的响应时间，但帧速率降低可能会带来问题，因为过滤后的帧可能导致信息丢失。

值得注意的是，紧急停止服务在所有场景中都会运行，由于中央处理器/内存使用率较高，这会造成最坏情况的时序问题。如果将该服务分配给远程节点，整体性能将会提升。

7 讨论

在这两种场景中，显而易见，对于处理小型消息的解决方案，延迟对机器人执行非关键任务时的行为影响不大。如果需要快速响应，网络延迟带来的额外时间可能会造成问题。对于处理大型消息和计算密集型任务的解决方案，将服务部署到其他层的能力会带来益处。如果本地节点正在运行多个服务，中央处理器资源有限可能会延长处理时间。尽管其他层的延迟更高，但高端中央处理器/图形处理器的速度可能远超常规边缘节点，从而弥补通信速度方面的不足。

BlazeAIoT框架通过压缩有效载荷和限制速率来促进通信，从而降低消息速率，使其符合允许的带宽。服务分配至关重要，但编排并不属于实际实现的一部分。借助 BlazeFlow 的动态能力，将服务从一个节点迁移到另一个节点十分顺畅，因为所有服务都会通告/请求主题。

7.1 BlazeAIoT 与目标的比较

介绍完 BlazeAIoT组件后，便可以评估该平台如何实现第 1.1 节中列出的目标。

- **注重成本的解决方案**：BlazeAIoT平台使用的所有技术都是开源技术（MongoDB、Kafka、RKE2、K3S、DDS、Redis 和 Prometheus）。使用 Rancher 作为 Kubernetes 管理工具，可便于创建基于 K3S 的轻量级 Kubernetes 集群，以及基于 RKE2 的全规模 Kubernetes 集群。该解决方案可以部署在任何云服务提供商的平台上，也可以替换为其他 Kubernetes 服务，例如 Azure Kubernetes Service、Amazon Elastic Kubernetes Service 和 Google Kubernetes Engine。最终形成的解决方案不依赖特定云平台，可以轻松部署在企业现有的基础设施中。
- **编程语言灵活性**：选择 BlazeFlow 数据技术时，便以此目标为依据。平台客户端不会直接访问 DDS，而是连接到 Kafka 或 Redis，因为它们提供多种语言的客户端库。ROS2 软件包仍可在同一层中直接通信。BlazeFlow 原生包含一个转换器，可在任何代理与 ROS2 之间进行转换。客户端服务通常使用软件开发工具包，该工具包支持多种语言；如有需要，也可以轻松添加对新语言的支持。
- **多层支持**：BlazeFlow 支持所有层之间的通信。在 Kubernetes 中，集群内的所有节点均可使用虚拟专用网络。要实现多层、多集群通信，可通过 Submariner 等 IP 隧道搭建加密隧道，打通层与层之间的连接。
- **灵活的服务分发**：借助 BlazeConfig，所有服务都可以获取层、节点和服务配置。如有需要，同一服务可以运行多个实例，每个实例均可拥有自己的配置。
- **实时性能**：BlazeFlow 代理的性能已在先前的研究中得到考察 [7]。为了监控平台内服务的指标，BlazeMonitor 会启用这些指标，以便后续进行改进，从而合理部署服务，确保实时执行。
- **稳健性与安全性**：本文未涵盖连接中断场景，但在资源充足、能够在等待连接恢复期间接管工作的情况下，当前的开发成果可轻松在本地启动所需服务。安全性本身是一个独立的议题，但所选技术均提供了安全选项。所有通信均可加密数据。MongoDB 还提供静态数据加密。使用 Kubernetes 可限制对其内部网络的访问，并提供授权和基于角色的访问控制，帮助保护整个解决方案。层与层之间的流量也通过 IPSec 隧道加密。此外，还可以在 BlazeMonitor 中添加监控服务，以收集用于此目标的指标并采取相应措施，但这不属于当前实现的范畴。

8 结论

本文介绍了 BlazeAIoT，这是一个模块化多层平台，旨在整合边缘、雾和云基础设施上的分布式机器人。通过扩展 BlazeFlow 框架，所提出的解决方案应对了分布式机器人领域的关键挑战，包括动态数据传输、自适应通信、实时性能，以及抵御连接中断的韧性。集成用于多层配置管理的 BlazeConfig、用于动态自适应数据分发的 BlazeFlow，以及用于系统健康指标的 BlazeMonitoring，为稳健且可扩展的部署提供了全面的基础。

通过导航和人工智能驱动的消息处理场景进行的实验验证表明，该平台能够满足严格的实时约束，同时在分布式拓扑中保持灵活性。结果证实，BlazeAIoT 能够动态分配服务、优化带宽使用，并确保在异构环境中的互操作性。

除机器人学外，该平台的设计原则同样适用于更广泛的物联网领域，包括智慧城市、智能工厂和信息物理系统。未来的工作将重点提升 BlazeFlow 中的负载均衡能力，探索更高效的集群间网络解决方案，并提出支持服务分配的编排方案。通过这些改进，BlazeAIoT 有望成为分布式机器人及物联网等其他分布式系统的基础。

参考文献

[1] João M. Quintas、Paulo J. Menezes 和 Jorge M. Dias。云机器人：迈向上下文感知机器人网络。第 *2* 届 *IASTED* 国际机器人会议论文集，*Robo 2011*，第 420–427 页，2011 年。doi:10.2316/P.2011.752-062.[2]K. Antevski、C. J. Bernardos、M. Groshev、A. Mourad、L. Cominardi 和 R. Gazda。利用上下文信息增强边缘机器人。*EM-5G 2018——2018*年*5G*实验与测量研讨会论文集，*CoNEXT 2018* 的组成部分，第 7–12 页，2018 年 12 月。doi:10.1145/3286680.3286682。URL https://dl.acm.org/doi/10.1145/3286680.3286682.[3]Siva Leela Krishna Chand Gudi、Suman Ojha、Benjamin Johnston、Jesse Clark 和 Mary Anne Williams。用于实现高效、流畅且稳健的人机交互的雾机器人。*NCA 2018——2018*年第*17*届 *IEEE* 网络计算与应用国际研讨会，2018 年 11 月。doi:10.1109/NCA.2018.8548077.

[4] Yuvraj Sahni、Jiannong Cao、Shigeng Zhang 和 Lei Yang。边缘网格：实现物联网分布式智能的新范式。*IEEEAccess*，5:16441–16458，2017 年 8 月。ISSN 21693536。doi:10.1109/ACCESS.2017.2739804。

[5] Christophe Bédard、Pierre Yves Lajoie、Giovanni Beltrame 和 Michel Dagenais。针对分布式 ROS 2 系统的复杂因果关联消息流分析。机器人与自主系统，161:104361，2023 年 3 月。ISSN 0921-8890。doi:10.1016/J.ROBOT.2022.104361.[6]Dominic Lindsay、Sukhpal Singh Gill、Daria Smirnova 和 Peter Garraghan。分布式计算系统的演进：从基础领域迈向新前沿。计算，103(8):1859–1878，2021 年 8 月。ISSN 14365057。doi:10.1007/S00607-020-00900-Y/FIGURES/2。URL https://link.springer.com/article/10.1007/s00607-020-00900-y.[7] Cédric Melançon、Guillaume Simard、Maarouf Saad、Kuljeet Kaur 和 Julien Gascon-Samson。BlazeFlow：用于实时分布式物联网应用的多层通信中间件。*Mid4CC 2023——2023* 年第*1*届计算连续体中间件国际研讨会论文集，属于 *Middleware 2023*，第 30–35 页，2023 年 12 月。doi:10.1145/3631309.3632837.[8] Steven Macenski、Tully Foote、Brian Gerkey、Chris Lalancette 和 William Woodall。机器人操作系统 2：设计、架构及其实际应用。*Science Robotics*，7(66)，2022 年 5 月。ISSN 24709476。doi:10.1126/SCIROBOTICS.ABM6074。URL https://www.science.org/doi/10.1126/scirobotics.abm6074.[9] David Portugal、Rui P. Rocha 和 João P. Castilho。向机器人操作系统社区了解 ROS 2 机器人中间件的采用情况。国际智能机器人与应用期刊，9(2):454–479，2024 年 6 月。ISSN 2366598X。doi:10.1007/S41315-024-00393- 4/TABLES/18。URL https://link.springer.com/article/10.1007/s41315-024-00393-4。

[10] Shams Forruque Ahmed、Shanjana Shuravi、Shaila Afrin、Sabiha Jannat Rafa、Mahfara Hoque 和 Amir H. Gandomi。《物联网（ IoT ）的力量：通过云、边缘和雾计算连接各个环节》。2023年9月。doi:10.48550/arXiv.2309.03420。网址：https://arxiv.org/abs/2309.03420v1。

[11]Priyanshi Gangrade、Virendra Singh Kushwah、Saroj S. Date、Archana D. Jagdale、Kamal Upreti 和 Vinai K. Singh。《信息物理系统中的边缘计算与雾计算》。*2025*年智能控制、计算与通信国际会议，*IC3 2025*，第172–176页，2025年。doi:10.1109/IC363308.2025.10956222。

[12] 胡国强、Wee Peng Tay 和文永刚。云机器人：架构、挑战与应用。*IEEENetwork*，26(3):21–28，2012。ISSN 08908044。doi:10.1109/MNET.2012.6201212。

[13] Luca Muratore和Nikos Tsagarakis。XBot2D：面向野外机器人技术的机器人混合云架构

《机器人与人工智能前沿》，10:1168694，2023年10月。ISSN 22969144。doi:10.3389/FROBT.2023.1168694/BIBTEX。

[14]Adnan Khalid、Muhammad Shahbaz 和 Im- ran Ahmed Khan。面向边缘云范式，智能利用雾设备辅助电子投票。《并发与计算：实践与经验》，31(8):e4957，2019 年 4 月。ISSN 1532-0634。doi:10.1002/CPE.4957。URL https://onlinelibrary.wiley.com/doi/full/1002/cpe.4957https: //onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4957https://onlinelibrary.wiley.com/doi/10.1002/cpe.4957。

[15] S. Aiswarya、K. Ramesh、B. Prabha、S. Sasikumar 和 K. Vijayakumar。基于物联网的医疗保健系统中使用雾计算的时间优化模型。*2021 年 IEEE 创新计算、智能通信与智能电气系统国际会议论文集，ICSES 2021*，2021 年。doi:10.1109/ICSES52305.2021.9633874。[16] Amel Ksentini、Maha Jebalia 和 Sami Tab- bane。雾计算能够在多大程度上提升智慧城市中异构延迟敏感型服务的性能？*2020 年第八届通信与网络国际会议，ComNet2020*——会议论文集，2020 年 10 月。doi:10.1109/COMNET47917.2020.9306096。[17] Neelabhro Roy、Mani H. Dhullipalla、Gourav Prateek Sharma、Dimos V. Dimarogonas 和 James Gross。基于控制质量的协作式边缘机器人技术资源配置。*IEEE 消费者通信与网络会议论文集，CCNC*，2025 年。ISSN 23319860。doi:10.1109/CCNC54725.2025.10976180。[18]Amit Baxi、Vincent Mageshkumar、Girish S. Murthy、Arvind Merwaday、Arvind Raju、Gowtham Hosamane、Sugnan Prabhu、Venkat Natarajan、Sus- ruth Sudhakaran、Dave Cavalcanti、Atul N. Hatalkar、Christian Maciocco、Ravishankar Iyer 和 Rath Van- nithamby。通过 5G 在边缘机器人系统中实现时间关键型控制。国际通信系统与网络会议，*COMSNETS*，（2025）：739–747，2025 年。ISSN 21552509。doi:10.1109/COMSNETS63942.2025.10885713。[19] Marc A. Riedlinger、Mehran Ghafarian Tamizi、Juilee Tikekar 和 Magnus Redeker。利用机器人和数字孪生的分布式拣选应用概念。*IEEE 新兴技术与工厂自动化国际会议，ETFA*，2022 年 9 月，2022 年。ISSN 19460759。doi:10.1109/ETFA52439.2022.9921659。[20] Sajja Suneel、Rakesh Kumar Yadav、K. Nirmala Devi、B. Teja sree、R. Aarthi 和 Shrikant Upadhyay。用于实时物联网应用的边缘智能混合框架：采用多阶段自适应处理方法。第六届移动计算与可持续信息学国际会议，*ICMCSI 2025*——会议论文集，第 380–386 页，2025 年。doi:10.1109/ICMCSI64620.2025.10883617。

[21]Kemal Cagri Serdaroglu、Baris Inanc 和 Seb- nem Baydere。消息模型对雾系统中可靠数据传输延迟的影响。《网络与系统管理杂志》，30(4):1–19，2022 年 10 月。ISSN15737705。doi:10.1007/S10922-022-09685-1/TABLES/4。URL https://link.springer.com/article/10.1007/s10922-022-09685-1。

[22] Raffaele Montella、Diana Di Luccio、Sokol Kosta、Giulio Giunta 和 Ian Foster。从雾向云传输数据时的性能、韧性与安全性：DYNAMO 传输框架方法。《计算机科学讲义》（包括《人工智能讲义》和《生物信息学讲义》子系列），11226 LNCS：197–208，2018年。ISSN 16113349。doi:10.1007/978-3-030-02738-4 17/FIGURES/5. URL https://link.springer.com/chapter/10.1007/978-3-030-02738-4_17.[23] Takafumi Sakai、Kenshin Hata、Takuya Wada 和 Katsuhiro Naito。利用信息流降低云端负载的物联网平台。《*2022*年*IEEE 第46*届年度计算机、软件与应用会议论文集》，*COMPSAC 2022*，第1211–1216页，2022年。doi:10.1109/COMPSAC54236.2022.00191.[24] L. Qingqing、J. Penia Queralta、T. N. Gia、H. Tenhunen、Z. Zou 和 T. Westerlund。在网络边缘进行压缩的车联网视觉里程计卸载。《*2019*年第*12*届移动计算与泛在网络国际会议（*ICMU 2019*）》，2019年11月。doi:10.23919/ICMU48249.2019.9006652.[25] 微软。Azure 网络往返延迟统计数据 | Microsoft Learn。URL https://learn.microsoft.com/en-us/azure/networking/azure-network-latency? tabs=Americas%2CCanada.[26]Cedric Melancon、Kuljeet Kaur、Julien Gascon-Samson 和 Maarouf Saad。利用数字孪生实现智能分布式机器人解决方案。《国际电力电子与驱动系统会议论文集》，2023年8月，2023年。ISSN 21645264。doi:10.1109/PEDS57185.2023.10246745.[27] Thomas Moore 和 Daniel Stouch。适用于机器人操作系统的广义扩展卡尔曼滤波器实现。《智能系统与计算进展》，302：335–348，2016年。ISSN 21945357。doi:10.1007/978-3-319-08338-4 25/FIGURES/9. URL https://link.springer.com/chapter/10.1007/978-3-319-08338-4_25。

[28]R Craig Coulter。纯追踪路径跟踪算法的实现。技术报告 CMU-RI-TR-92-01，卡内基梅隆大学，匹兹堡，宾夕法尼亚州，1992年1月。URL https://api.semanticscholar.org/CorpusID:62550799.[29] 谷歌有限责任公司。Hands - mediapipe，2020年。URL https://chuoling.github.io/mediapipe/solutions/hands.html.[30] KapitanovAlexander、Kvanchiani Karina、Nagaev Alexander、Kraynov Roman 和 Makhliarchuk Andrei。HaGRID——手部手势识别图像数据集。《*2024*年 *IEEE* 冬季计算机视觉应用会议论文集，*WACV 2024*》，第4560–4569页，2024年1月。doi:10.1109/WACV57701.2024.00451。