GALOIS：混合式平台无关流处理架构

Tarek Stolz
tarek.stolz@rwth-aachen.de
亚琛工业大学 亚琛，德国

István Koren
koren@pads.rwth-aachen.de
流程与数据科学教席，亚琛工业大学，德国亚琛

Liam Tirpitz, Sandra Geisler
{tirpitz,geisler}@cs.rwth-aachen.de
数据流管理与分析，亚琛工业大学，亚琛，德国

arXiv:2305.02063v1 [cs.DC] 3 May 2023

摘要

随着物联网环境日益普及，处理海量分布式数据流的需求已成为一项关键挑战。边缘数据流处理（DSPoE）系统应运而生，旨在应对这一挑战，但它们往往难以适应硬件和平台的异构性。为解决这一问题，我们提出了一种名为 GALOIS 的新型混合 DSPoE 架构。该架构基于 WebAssembly（Wasm），不依赖特定硬件、平台或语言。GALOIS 采用多层方法，结合点对点和主从概念，实现组件间通信。我们展示的实验结果表明，与 Docker 中执行的算子相比，在 Wasm 中执行的算子能耗和 CPU 消耗更低，因此 Wasm 是 DSPoE 流处理算子的一个颇具潜力的选择。因此，我们预计基于 Wasm 的解决方案将显著提升 DSPoE 系统的性能和韧性。

CCS 概念

• **信息系统** →**流管理**。

关键词

边缘处理，数据流处理

ACM 参考文献格式：

Tarek Stolz、István Koren、Liam Tirpitz、Sandra Geisler。2023 年。GALOIS：混合式平台无关流处理架构。载于新兴分布式环境中的大数据国际研讨会（*BiDEDE '23*），*2023*年*6*月*18*日，美国华盛顿州西雅图。ACM，纽约州纽约，美国，6 页。https://doi.org/10.1145/3579142.3594287

1 引言

现代制造环境会以极高频率产生海量数据。例如，在精密落料（一种工业冲压工艺）中，单条生产线的数据生成速率最高可达 6 GBit/s [10]。在我们构想的由大型卓越研究集群体现的生产互联网 [2],中，跨组织处理和共享高频机器数据，对于实现大规模近实时分析以及支持数字影子以促进智能制造应用至关重要。然而，机器、可编程逻辑控制器和网络连接的高度异构性带来了严峻挑战数据处理环境。此外，生产线中不断增加的变体数量，以及最终在可持续性和韧性方面快速应对突发事件的需求，都要求采用快速且自适应的数据流管理解决方案。

成熟的分布式流处理（DSP）架构（例如 Apache Storm 或 Twitter Heron）问世已有多年，并已证明能够处理大量流数据。它们旨在利用现场数据中心或云中节点的丰富资源，将工作负载分配给配置同质的服务器。将多条生产线的数据传输到中央数据中心，会导致网络拥塞，并使近实时应用面临无法接受的延迟。边缘处理架构将部分数据处理（算子）迁移到边缘和雾，靠近数据源 [3, 21]。对于边缘环境而言，会出现许多与云环境中相反的挑战 [21]。例如，不安全、不稳定、受限的网络连接，以及设备故障或移动，都会形成高度动态的数据处理环境。此外，边缘设备在硬件或操作系统等方面具有高度异构性，而且资源和电池续航通常有限。边缘分布式流处理（DSPoE）系统旨在应对这些问题。然而，DSPoE 中的算子通常依赖特定语言或硬件实现，这使得系统缺乏灵活性，也难以维护和扩展。此外，DSPoE 通常依赖集中式管理组件，导致较高的通信开销。

为应对这些挑战，我们提出了 GALOIS，一种混合式平台无关流处理架构。该架构采用多层设计，以提高灵活性并高效分配工作负载，同时根据不同层级采用集中式或分布式管理。查询执行计划的重新优化结果会从较低层级向较高层级传播，从而尽可能减少与中央统一主节点之间的通信开销。此外，我们使用 WebAssembly（Wasm）这一新型字节码格式来实现算子，使其能够在多种硬件上运行，且不依赖特定硬件、平台或语言。这为 GALOIS 奠定了基础，我们将在下文对此进行更详细的讨论。

本文组织如下：第 2 节首先分析混合且硬件无关的 DSPoE 的需求。基于这些需求，我们在第 3 节介绍 GALOIS 架构的设计。第 4 节描述了实现这一概念的首个原型，并在第 5 节对其进行评估。最后，在第 6 节中，我们结合当前最新技术水平讨论所提出的架构，并得出结论。

BiDEDE '23，*2023*年*6*月*18*日，美国华盛顿州西雅图© 2023 版权所有归所有者/作者所有。出版权已授权给 ACM。本文为作者版本，仅供个人使用。请勿再分发。最终正式版本发表于新兴分布式环境中的大数据国际研讨会（*BiDEDE '23 18, 2023*,101145 35791423594287），*June* 西雅图，美国华盛顿州，https://doi.org/ 。/ 。

2 需求

下面概述 DSPoE系统 为了在高度动态的环境中有效运行所应满足的关键需求，同时适应海量数据流和边缘环境中的异构设备。这些需求源自对制造业边缘环境特征的深入分析，以及对现有 DSPoE 系统所需属性的考察和未解决挑战的梳理。

平台无关的算子执行 **(R1)**：为适应边缘环境的多样性，流算子的设计应不受硬件、语言和平台限制，能够在不同环境中执行。

原生边缘流处理与处理 **(R2)**：为实现快速响应，数据应在处理设备之间直接进行点对点流传输，不经过任何中间环节。此外，算子的执行应发生在边缘层级。

网络感知的管理与维护 **(R3)**：查询处理期间应考虑边缘网络的拓扑和属性（如链路质量和设备距离），以尽量减少网络引起的延迟和背压。

全局逻辑优化 **(R4)**：逻辑查询优化应基于启发式方法在全局层面进行，因为这会影响随后在算子放置过程中分发到边缘节点的算子数量和类型。

局部物理优化 **(R5)**：为利用最新且与上下文相关的信息进行查询优化，并降低通信开销，物理查询优化应在与算子执行相同的层级进行，从而增强各层的自主性和模块化。

集群单元中的空间组织 **(R6)**：系统在部署和管理边缘设备时应考虑空间约束。系统应优先将相邻算子部署在边缘设备密集的汇聚点（集群单元），以建立尽可能短的通信路径。

去中心化查询重新配置 **(R7)**：如果需要调整查询计划，应以去中心化且独立的方式进行，不得暂停整个系统。

管理升级模型 **(R8)**：只有在较低层级无法进行重新优化时，架构中的较高层级管理单元才应介入。

预防性容错 **(R9)**：弹性容错对于提升查询的稳健性至关重要。发生客户端故障时，查询应能够继续提供结果。这可以通过更具选择性的算子放置来提高查询的稳健性，并在算子之间提供多条路径来实现 [4, 18]。这些核心需求为我们的架构设计提供了依据。

由此得出的设计决策。 由于边缘环境可能高度动态，DSPoE 系统中用于算子放置的指标，在查询计划部署时可能已经过时 [16, 23]。因此，DSPoE 系统的开发越来越倾向于对查询和功能进行去中心化部署 [8, 23]。为此，Pinchao 等人 [16] 将边缘设备组织在点对点网络中，并在运行时而非部署时进行查询优化。由于查询是增量优化的，随着时间推移，查询执行不再最优的风险较低（**R7**）。然而，许多点对点网络会抽象掉底层网络拓扑，这可能导致算子放置不理想 [15, 19]。为了实现低延迟，算子应尽可能靠近数据源，相邻算子也应彼此靠近。因此，采用基于集群的方法 在最低层组织底层物理设备，似乎是解决这一问题的可行方案（**R6, R3**） [4, 5, 15, 17, 22]。

在此背景下，移动自组网（MANET）颇具吸引力，因为它们能够在边缘实现集群计算 [15]。MobiStream [22]为 MANET 中的集群定义了固定位置，但无法适应动态形成、消失和重组的集群。因此，集群应支持移动性，并能够自主注册 **(R6)**。对于 GALOIS，我们选择将空间上分隔的集群组织到区域中，以实现可扩展性和灵活性，这与 Mobile Storm 的理念类似 [17] **(R6)**。此外，还可以添加更多区域，从而实现类似 SpanEdge 的空间扩展 **(R4,R8)** [20]。集群间通信和区域间通信的开销应尽可能低，并采用点对点方式实现，以提供高度的灵活性和自主性。因此，查询算子执行应在集群和边缘设备级别完成 **(R2)** [15]。

在查询运行时，应在集群中执行连续查询优化。该集群应尽可能实现自治，仅在必要时（例如，由于无法解决的过载情况）才将问题上报给管理节点，以减少管理开销 **(R8)** [23]。查询的物理优化和逻辑优化应在本地层和全局层之间进行划分，确保全局实例无需掌握底层边缘设备和算子的完整信息。相反，物理优化高度依赖动态指标，因此应尽可能在靠近执行位置的地方执行 **(R6,R7)**。

为了最大限度地提高容错性，GALOIS 将采用优于检查点方案或基于副本的方案 [4] 的概率模型，并在算子放置**(R9)** [18] 中仅选择具有特别高故障安全性的节点，从而使架构更加稳健。

此外，该架构在不同查询语言方面具有可扩展性，其设计理念类似于 Calcite 或 Governor [1, 6]。最后，为了在异构系统环境中执行算子，算子应能够独立于硬件、平台和语言运行。迄今为止，我们所了解的所有 DSPoE 都依赖于特定语言（**R1**） [12, 16, 17], ，而 GALOIS 的架构正是为了解决这一问题。

3 概念

基于上一节阐述的需求和设计决策，我们构思了一种系统架构，以下将对此进行详细说明。

图1展示了 GA- LOIS 中各个组件的概览。该系统采用分层结构，包含四层通信 [23] **(R6)**。最低层是集群层，节点在此组织成集群，并在数据流中传输元组。集群表示具有物理邻近性的节点汇聚点，这种邻近性会得到利用 **(R2)**。在集群内部，数据流在节点之间交换。在图1中，集群以圆形表示，网络连接

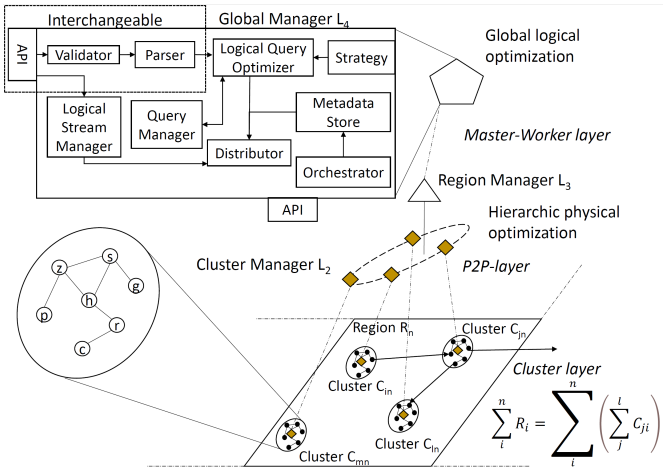

图1：GALOIS架构

节点之间的数据流处理以实线表示，节点在集群中以黑色圆点表示。集群之间的数据流处理以集群之间的实线表示，并通过网关实现。黄色方块表示集群管理器 (CM)，它们通过点对点网络（以虚线椭圆表示）相互连接，并在必要时与其所在区域的区域管理器 (RM)进行通信（以三角形表示）。为了提高可扩展性并构建本地独立的子系统，每个集群都分配给一个RM。可以根据应用程序的需求定义区域，例如，将地理位置相近的集群分组。RM 和 GM 组织成主（GM）-工作进程（RM）层。

全局管理器 （GM，以五边形表示）主要负责三项任务：(1) 注册逻辑数据源（模式）；(2) 接收逻辑查询计划；以及 (3) 逻辑查询优化 （**R4**）。模式的注册可以通过两种方式完成：(1) 由人工用户注册模式及相应的数据源；或 (2) 数据源通过相应的接口自主运行，注册自身及其模式。有多种形式化查询语言可用于注册和描述查询。与 Apache Calcite 或 NebulaStream 类似，GM 应能够支持多种语言 [1, 23]。GM 中的优化器可以利用优化策略对逻辑查询计划进行优化。与其他系统不同，GM 不执行物理优化，并严格分离逻辑优化和物理优化。成功部署的查询会存储在 GM 的查询管理器 中，以支持多查询优化。此外，查询管理器 还可以向用户提供当前正在运行的查询及其状态信息。优化策略应易于替换，以支持不同的优化策略，类似于 Governor 系统 [6]。查询优化完成后，会被划分为子查询，并由分发器发送至区域管理器（RMs）。除子查询外，还会向 RMs 提供区域间通信所需的信息，以支持不同区域中节点之间的区域间数据传输。子查询的划分使用存储在元数据存储库中的元数据完成。这些元数据包括区域特定元数据，例如某个区域中的源节点、区域性能统计数据，例如可用资源容量、正在运行的子查询，或平均元组吞吐量等关键指标。编排器 监控区域拓扑以及新区域的启动，而新区域可以动态添加或删除。

区域管理器（*RMs*）作为下一层抽象被引入，通过独立管理多个集群来实现可扩展性和局部性。区域可以按地理位置或延迟区划分，例如，为了确定延迟最优的区域，这与 SpanEdge [20] **(R3)**类似。我们假设，地理位置相距较远的点之间的延迟显著高于空间距离较近的节点之间的延迟。邻近性的概念可在实现层面定义，并取决于应用。因此，选择子查询时应确保跨区域通信仅限于必要程度。区域管理器（RM）的核心任务是根据汇总统计数据（例如特定集群的算子吞吐量或平均队列大小），将子查询分发给集群管理器。RM据此将逻辑子查询分发给集群管理器，并在本地查询管理器中管理分配，以便进行可能的重新配置**(R7)**。此外，RM还充当该区域中新出现的集群的引导节点，为新的集群管理器提供连接到P2P网络所需的所有信息。

集群管理器(CM) 以点对点网络的形式组织。集群管理器最重要的任务包括算子放置、收集元数据、监控 GALOIS 客户端，以及在点对点网络中与其他集群管理器通信，以便在运行时分配工作负载。基于收集到的指标，点对点网络中的集群管理器会自主进行子查询重新优化，从而减少元数据交换**(R8)**。此外，点对点网络应确保能够随时随地灵活（例如，在车辆或智能手机中）且动态地建立新集群。集群中的节点故障不应导致整个查询的处理停止。因此，必须避免像 MobiStream 中那样的延迟（发生故障时停止整个查询） [22]。在最低抽象层，GALOIS 客户端由安装了 DSPoE 客户端的边缘设备组成。通过节点发现，将每个节点分配到一个集群。如果附近没有集群管理器，孤立节点可以接管集群管理器的角色并建立新集群。新集群将保持非活动状态，直到集群中注册了足够数量的节点（集群大小可配置，以降低管理开销）。

除了担任集群管理器（h）外，节点还可以在集群中承担六种额外角色：休眠节点（s）、生产者（p）、消费者（c）、副本（r）、网关（g）和接收器（z）。这种角色方案支持概率模型，可根据可靠性为节点分配角色，以实现容错 [4]。非集群管理器节点初始化为休眠节点 。休眠节点尚未承担特定角色，可供集群管理器调用。查询的算子一经分发，相应节点便会被分配生产者（*p*）角色，成为数据流的来源。子查询中的所有后续算子都是消费者（*c*），负责增量处理数据流。为实现容错方案，节点会被分配副本 （replica）角色，这些节点在统计意义上具有特别高的故障安全性。它们会定期检查分配给自己的消费者 节点，并在发生故障时接管其功能。为在集群之间传输数据，需要集群间通信；此功能由网关 节点提供。它们充当集群之间的网关。*z*标记分配给接收器节点。

由集群管理器（CM）提供，这些 API 供用户或应用程序消费数据。这些节点类型及通用架构均通过原型实现得以实现。

4 原型

为了在原型中实现平台独立性的要求，需要采用某种抽象机制。实现平台独立性的常见解决方案是使用虚拟机或 Docker 容器。由于流算子应在边缘设备上执行，因此我们以资源需求和启动开销最小化的执行环境为目标。借助二进制指令格式 WebAssembly（Wasm）1，尤其适用于无服务器场景，[9]，算子可以作为函数在 Wasm 运行时环境中执行。在相关文献中，Wasm 被视为 Docker 的一种潜在替代方案，尤其是在边缘场景中，因为事实证明，与 Docker 相比，Wasm 具有更高的内存效率 [9, 13]。因此，我们在系统中以 Wasm 为目标实现流算子，使其执行不受平台、语言和硬件的限制 **(R1)**。我们还选择 Rust 作为客户端和各管理器的实现语言，因为 Rust 原生支持 Wasm 并能与其良好集成，同时能够生成体积较小的二进制文件。然而，算子也可以使用其他任何语言实现，且这一选择独立于客户端和管理器的具体实现。

第一个原型的实现采用自底向上的方法，重点开发 Wasm 算子，作为概念验证。为测试算子执行，仅创建了一个包含多个节点的集群。节点间的数据传输通过 Kafka 实现，但下一步将替换为高效的节点间通信。在该原型中，单个服务器组件统一承担 GM、RM 和 CM 的功能（GRC服务器）。为实现扩展，之后可以将这些组件分离。目前，GRC服务器包含查询、模式和节点管理模块，以及物理查询计划创建和简单的算子放置（**R5**）。此外，还集成了 Data Fusion，用于执行逻辑查询优化。未来将为这些组件补充更多功能，例如集群和资源管理，以及适用于边缘环境的容错协议，包括背压机制（**R9**）。GRC服务器通过服务器发送事件与节点通信

事件（SSE），包括节点*ID* 及其对应的命令。在 GRC 服务器完成算子放置后，也会使用命令将算子部署到节点上。GRC 服务器以 Docker 容器的形式部署在 Kubernetes 集群中，而工作进程则运行在 Raspberry Pi *Model B Rev 1.5*上。图 2 概述了最重要的组件。

5 评估

在研究的初步阶段，我们对原型系统进行了评估，重点比较了使用 Wasm 实现算子与采用 Docker 容器化方案的性能。此次评估旨在探究两种方法在延迟、能耗、吞吐量和 CPU 负载方面的差异。我们的假设是，由于 Wasm 能够近乎原生地利用硬件特性，且初始化开销较低，因此其效率将更高

[1]https://webassembly.org

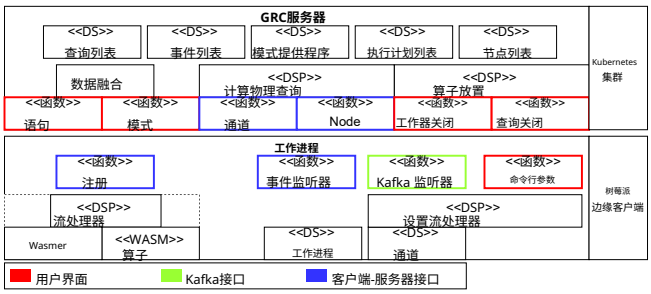

图 2：原型及其组件概览

在这些性能指标方面。评估采用了 ELEC 数据集 [14]，，其中包含来自澳大利亚新南威尔士州、按半小时记录的电价数据，共 45,312 个元组。该数据集涵盖 942 天，并包含以下属性：Date、Day、Period、NSW-price、NSWdemand、VICprice、VICdemand、transfer。评估设置如图 3 所示。

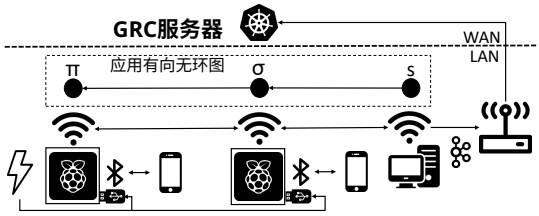

图 3：评估设置

该设置包含两台 Raspberry Pi 4 Model B Rev 1.5，运行 64 位 Linux raspberrypi 5.15.76-v8+。它们代表集群中的边缘设备，可将算子分配到这些设备上。除 Raspberry Pi 外，还使用了一台 Windows 10 计算机作为生产者，从上述数据集中生成数据流，并运行本地 Docker 化的 Kafka 实例。此外，还使用了两台 DollaTek UM25C USB 功率计来记录 Raspberry Pi 的功耗。由于监控软件一次只允许与一台功率计建立蓝牙连接，因此功率计分别连接到两部不同的智能手机。GRC服务器部署在生产互联网（IoP）的 Kubernetes 基础设施上，该基础设施位于另一个网络中。我们在 GRC服务器上注册了一个连续查询，其中包括对 date 和 period 属性的投影，以及 18 个选择条件（用于增加查询复杂度）。随后，系统将选择操作分配给一台 Raspberry Pi，将投影操作分配给第二台 Raspberry Pi，以分别测量每个算子的处理成本。我们通过让生产者以不同的数据速率生成元组来模拟系统负载变化，具体为 125 T/s（元组/秒）、500 T/s 和 1000 T/s。对于每种输入速率，我们都测量了边缘设备的功耗和 CPU负载，以及吞吐量（每个算子和完整查询的 T/s）和查询延迟（元组进入和离开系统之间的时间）。

5.1 结果

我们在下文中展示所述实验的结果，其中百分比表示平均值。算子处理

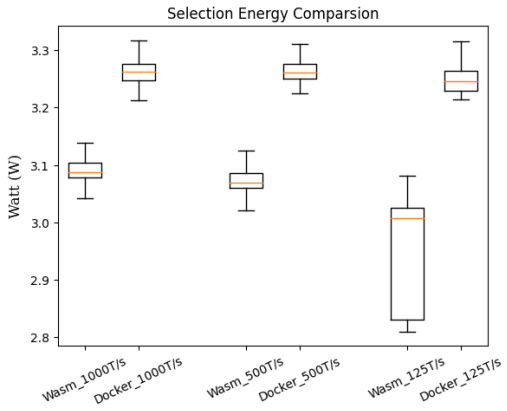

图4：能耗比较

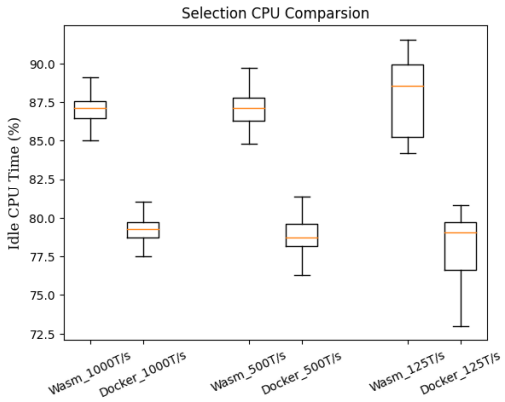

图 5：空闲 CPU 时间比较

时间 算子处理时间（OPT）是算子执行前的时间戳与执行后的时间戳之差。我们的实验表明，对于选择操作，在每种输入速率下，Wasm 的算子处理时间都比 Docker 低 9%。

功耗。算子执行时，功率计会以瓦特每秒为单位记录能耗。图 4 包含六个箱线图，展示了选择操作执行时的功耗。横轴上的箱线图以实验条件标注，每种条件对应执行环境（Wasm 或 Docker）与输入速率的一种组合。

Docker实验的能耗在1000 T/s时比Wasm高5.72%，在500 T/s时比Wasm高6.56%，在125 T/s输入速率时比Wasm高10.09%。*CPU*负载 平均CPU负载记录为仅在元组处理期间的CPU空闲时间（CIT）百分比，其中数值越高表示CPU负载越低，反之亦然。CPU负载被视为Docker虚拟化潜在额外工作负载的指标。图5展示了六个箱线图，x轴还标注了执行环境与输入速率的组合。y轴绘制了选择执行期间CIT的百分比。Wasm实验在1000 T/s时的CIT比Docker高11.36%，在500 T/s时比Docker高18.89%，在125 T/s输入速率时比Docker高23.05%。投影显示没有显著差异，只有所有测量值的标准差更小。

5.2 讨论

我们的结果支持这样一种假设：由于 CPU强度较低，Wasm 每个元组的处理时间比 Docker 更高效。这种效率源于几个因素：Docker 以独立进程的形式运行，需要管理其环境和虚拟网络；而嵌入 Rust 的 Wasm 只需执行算子。尽管我们预期选择方面的差异会更显著，但 OPT 仍相对稳定，这可能是由于运行和 CPU 的限制。值得注意的是，与 Docker 相比，Wasm 的两个算子对应的 OPT 波动都较小，这可能是因为 Docker 额外增加了 CPU 开销。

正如预期，Docker 因容器化开销而消耗更多功率。有趣的是，随着输入速率提高，Docker 与 Wasm 在 CPU 和能耗方面的差异会缩小，其中差距最大时的输入速率为 125 T/s。还需进一步实验来研究这些差异与输入速率之间的关系。我们的结果与 Hampau等人 [13], 一致：Wasm 的能耗低于 Docker。在 125 T/s 时差异最大，可能是因为 Wasm 的 OPT 更高，且元组等待时间更长；随着输入速率提高，由于背压，这种差异会消失。

受 CPU 限制的 OPT 可通过多线程加以改善，但会增加能耗，形成一种权衡 [23]。我们的实验旨在评估 Wasm 作为流处理算子库的可行性。结果表明，Wasm 有望成为 DSPoE 中 Docker 的轻量级、硬件无关的函数即服务替代方案。我们计划扩展算子库，纳入连接等标准算子，以及机器学习等高级功能。我们将通过深入实验评估复杂查询、算子放置、查询重新优化和通信成本，从而全面评估所提出的算子网络在数据流处理中的有效性与优势。

6 相关工作

NebulaStream、MobiStream、DART、Apache NiFi 和 MobileStorm 是相关的 DSPoE 系统，它们利用边缘计算的概念实现流处理 [7, 16, 17, 22, 23] ，我们根据提出的要求对它们进行了分析。这些系统中有许多，例如 MobileStorm 或 Apache NiFi，采用中央管理组件，例如负责监控、查询分发、客户端管理和故障处理，而客户端只执行算子。相比之下，DART 采用纯粹的去中心化方法，并将许多功能卸载到组成点对点网络的各个客户端上。在 GALOIS 中，我们采用混合模型，结合两种方法的优势，提供高可扩展性和灵活性。上述系统几乎都使用 Java 虚拟机（JVM）作为执行环境，以实现平台和硬件无关的算子执行。然而，JVM 并非语言无关。NebulaStream 使用查询编译，即将查询作为通用查询计划分发到边缘设备，在那里将其转换为 C++、编译并执行 [12]。上述方法均不

具备语言无关性。Groppe 和 Reimer 提出的在线 Wasm 编译器 [11]并不依赖 JVM，有助于在互联网浏览器网络中实现语言无关的查询处理，但其重点并非数据流处理。

基于云的数据流处理器（CDSP）过去被广泛采用，并作为 DSPoE 系统的基础。然而，该领域的大多数 DSPoE 系统都是用 Java 开发的，缺乏硬件或语言无关的算子执行能力。CDSP 也缺少适应边缘环境动态变化的故障处理机制，而且通常不支持查询的运行时优化，因为人们认为云端的性能是稳定的。鉴于云和边缘之间存在相互矛盾的需求，开发一种不依赖现有 CDSP 的 DSPoE框架是合理的。

7 结论与展望

本文提出了一种新型混合多层DSPoE架构 GALOIS。该架构采用 P2P网络概念，促进边缘集群间的通信，使本地层面的重新优化成为可能，从而降低通信开销。集群中的节点还可以承担不同角色，以此指导算子放置。同时，GALOIS 通过引入区域管理层实现可扩展性，该层包含多个集群和一个以主从模式实现的全局管理器。与现有 DSPoEs 不同，GALOIS 使用 WebAssembly 作为分布至边缘的流处理算子的执行环境，因此不受硬件、平台和语言限制。我们通过对该架构原型实现进行实验表明，与 Docker 相比，使用 Wasm 执行算子在算子处理时间、CPU 和能耗方面具有更高的资源效率。实验中观察到的较低吞吐量可归因于对 CPU 的保守使用，但未来可以通过利用多线程实现更高的并行度来解决这一问题。总而言之，Wasm 是在 DSPoE 环境中执行算子的 Docker 可行替代方案。不过，由于 Wasm 仍在开发中，它存在一些需要考虑的固有限制。例如，目前无法在不使用额外胶水代码的情况下传递Wasm复杂数据对象。我们计划通过以下方式扩展 GALOIS：（1）扩展 Wasm流式算子库，以支持更复杂的查询；（2）集成灵活的服务质量和数据质量监控，以支持查询的动态重新优化；（3）基于设想中的通信基础设施实现包含所有层级的完整原型，同时考虑网内处理。

致谢

由德国研究基金会（DFG）通过德国卓越战略资助——卓越集群 EXC-2023“生产互联网”（390621612）。

参考文献

[1] Edmon Begoli、Jesús Camacho-Rodríguez、Julian Hyde、Michael J. Mior 和 Daniel Lemire。2018 年。Apache Calcite：面向异构数据源的优化查询处理基础框架。载于*2018 年国际数据管理会议论文集（SIGMOD '18）*。ACM，221–230。[2] Philipp Brauner 等。2022 年。生产数字化转型的计算机科学视角。*ACM 物联网汇刊* 3，2（2022），1–32。

[3] Keyan Cao、Yefan Liu、Gongjie Meng 和 Qimeng Sun。2020 年。边缘计算研究综述。IEEE Access 8（2020），85714–85728。[4] Mengyuan Chao 和 Radu Stoleru。2020 年。R-MStorm：面向动态边缘网络的弹性移动流处理系统。载于 2020 年 IEEE 雾计算国际会议（ICFC）。IEEE，澳大利亚新南威尔士州悉尼，64–72。[5] Mengyuan Chao、Chen Yang、Yukun Zeng 和 Radu Stoleru。2018 年。F-MStorm：基于反馈的在线分布式移动流处理。载于 2018 年 IEEE/ACM 边缘计算研讨会（SEC）。IEEE，美国华盛顿州西雅图，273–285。[6] Ankit Chaudhary、Steffen Zeuch 和 Volker Markl。2020 年。Governor：统一雾-云环境中的算子放置。载于第 23 届国际数据库技术扩展会议论文集。EDBT。[7] Rustem Dautov 等。2017 年。借助流处理架构将智能推向边缘。载于 2017 年 IEEE 物联网国际会议（iThings）、IEEE 绿色计算与通信会议（GreenCom）、IEEE 网络、物理与社会计算会议（CPSCom）和 IEEE 智能数据会议（SmartData）。792–799。[8] Schahram Dustdar 和 Ilir Murturi。2021 年。迈向边缘上的物联网流程。载于《下一代数字服务：未来服务计算的回顾与路线图》，Marco Aiello、Athman Bouguettaya、Damian Andrew Tamburri 和 Willem-Jan van den Heuvel（编）。Springer，Cham，167–178。[9] Philipp Gackstatter、Pantelis A. Frangoudis 和 Schahram Dustdar。2022 年。借助 WebAssembly 运行时将无服务器推向边缘。载于 2022 年第 22 届 IEEE 集群、云与互联网计算国际研讨会（CCGrid）。140–149。[10] René Glebke、Martin Henze、Klaus Wehrle、Philipp Niemietz、Daniel Trauth、Patrick Mattfeld 和 Thomas Bergs。2019 年。大型网络物理系统中集成数据处理的必要性。载于《2019 年夏威夷国际系统科学会议》（HICSS）。7252–7261。[11] Sven Groppe 和 Niklas Reimer。2019 年。使用 WebAssembly 在 Web 中进行大数据处理的代码生成。《开放云计算期刊》（OJCC）6，1（2019），1–15。[12] Philipp M Grulich、Sebastian Breß、Steffen Zeuch、Jonas Traub、Janis von Bleichert、Zongxiong Chen、Tilmann Rabl 和 Volker Markl。2020 年。Grizzly：通过自适应查询编译实现高效流处理。载于 ACM SIGMOD 数据管理国际会议论文集（SIGMOD 2020）。ACM。[13] Raluca Maria Hampau、Maurits Kaptein、Robin van Emden、Thomas Rost 和 Ivano Malavolta。2022 年。面向边缘计算机视觉任务的人工智能容器化策略在性能与能耗方面的实证研究。载于 2022 年软件工程评估与测量国际会议（EASE 2022）。ACM，美国纽约，50–59。[14] Michael Bonnell Harries。1999 年。Splice-2 比较评估：电力定价。新南威尔士大学计算机科学与工程学院。[15] Thomas Kunz、Silas Echegini 和 Babak Esfandiari。2020 年。分层 MANET 中路由的点对点方法。《通信与网络》12（2020），99–121。[16] Pinchao Liu、Dilma Da Silva 和 Liting Hu。2021 年。DART：可扩展且自适应的边缘流处理引擎。载于 2021 年 USENIX 年度技术会议（USENIX ATC 21）。USENIX 协会，239–252。[17] Qian Ning、Chien-An Chen、Radu Stoleru 和 Congcong Chen。2015 年。Mobile Storm：面向移动云的分布式实时流处理。载于 2015 年 IEEE 第 4 届云网络国际会议（CloudNet）。IEEE，加拿大安大略省尼亚加拉瀑布，139–145。[18] Dan O'Keeffe、Theodoros Salonidis 和 Peter Pietzuch。2018 年。Frontier：面向物联网的弹性边缘处理。《VLDB Endowment 论文集》11，10（2018），1178–1191。[19] Rafiza Ruslan、Ayu Shaqirra Mohd Zailani、Nurul Hidayah Mohd Zukri、Nur Khairani Kamarudin、Shamsul Jamel Elias 和 R. Badlishah Ahmad。2019 年。点对点分布式哈希表（DHT）中结构化覆盖网络的路由性能。《电气工程与信息学通报》8，2（2019），389–395。[20] Hooman Peiro Sajjad 等。2016 年。SpanEdge：迈向中心数据中心与近边缘数据中心上的统一流处理。载于 2016 年 IEEE/ACM 边缘计算研讨会（SEC）。168–178。[21] Blesson Varghese、Nan Wang、Sakil Barbhuiya、Peter Kilpatrick 和 Dimitrios S Nikolopoulos。2016 年。边缘计算的挑战与机遇。载于 2016 年 IEEE 智能云国际会议（SmartCloud）。IEEE，20–26。[22] Huayong Wang 和 Li-Shiuan Peh。2014 年。MobiStreams：面向移动设备的可靠分布式流处理系统。载于 2014 年 IEEE 第 28 届并行与分布式处理国际研讨会。IEEE，美国亚利桑那州菲尼克斯，51–60。[23] Steffen Zeuch 等。2020 年。NebulaStream 平台：面向物联网的数据与应用管理。载于第 10 届双年创新数据系统研究会议（CIDR '20）。CIDR。