用于收集和聚合分布式云指标的监控系统

Tamara Rankovi´c Mateja Rilak Janko Rakonjac Milo˘s Simi´c技术科学学院 技术科学学院 技术科学学院 技术科学学院诺维萨德大学 诺维萨德大学 诺维萨德大学 诺维萨德大学 诺维萨德，塞尔维亚 诺维萨德，塞尔维亚 诺维萨德，塞尔维亚 诺维萨德，塞尔维亚 tamara.rankovic@uns.ac.rs rilak.sr38.2021@uns.ac.rs rakonjac.sr33.2021@uns.ac.rs milos.simic@uns.ac.rs

arXiv:2603.05241v1 [cs.DC] 5 Mar 2026

摘要——**需要对海量数据进行实时处理的应用推动人们重新思考传统云，从而催生了新型云模型。分布式云（DC）是一种允许用户动态创建和销毁位于战略位置的临时云的模型，这些云包含最适合其需求的资源。要使该模型能够应用于现实场景，必须具备高度的可观测性。本文介绍了一种监控系统的设计与实现，该系统从数据中心收集指标，并向各类客户端提供访问。运行在节点上的代理负责收集机器、容器和应用程序级指标。在健康检查协议期间，数据会从节点传输到运行在云内部的数据中心控制平面。在那里，数据会被持久化，并通过多个 API 提供，包括流式 API。此外，还会针对每个数据中心聚合节点指标，以便更全面地了解系统状态。**

索引词—**指标、监控系统、分布式云、云计算、分布式系统**

I. 引言

过去十年间，云模型发生了显著转变。现代应用提出的需求是推动这一转变的主要动力。研究表明，若依赖远程集中式数据中心，实时传输、处理和检索海量数据的效率会很低，甚至根本无法实现 [1]。为解决这些问题，许多新型云模型应运而生，例如边缘计算和雾计算 [2, 3]。尽管提出的解决方案各不相同，但它们都采用了数据本地性的理念——在数据存储的位置处理数据，同时也将数据存储在靠近其生成位置的地方。现代制造设施、发电厂、自动驾驶车辆以及其他相关应用场景，都能从这种组织方式中受益匪浅；在这些场景中，数据源与使用者彼此接近，且实时数据处理和检索至关重要。除了提升效率外，数据隐私和安全性也得到了增强，因为数据可能始终不会越过可信位置或网络的边界。

分布式云（DC）模型是针对所提出问题 [4, 5] 的模型之一。它由云层和 DC 层组成。云层是传统的集中式云，而 DC 层旨在减轻云层的负载，并为对延迟和隐私要求严格的客户端提供服务。DC 层由用户创建和弃用的动态 DC 组成。每个 DC 都应根据用户的需求进行定制，包括其资源容量和位置[4, 6]。DC 的生命周期由云层中的控制平面协调，但 DC 本身负责确保正常运行，从而形成混合架构。用户需要为其 DC 指定配置和工作负载，而系统则负责使其收敛至预期状态。该模型使云与 DC 能够无缝协作。

要使数据中心模型成为适用于实际应用的可行方案，我们需要围绕它构建丰富的功能生态系统和可靠的工具。为此，人们引入了命名空间和数据空间，从而支持提供多租户能力的虚拟数据中心 [7, 8]。此外，还开发了用于动态系统和应用程序重新配置的机制 [9]，并原生支持错误配置预防和检测 [10]。数据中心中的任何资源均通过基于属性的分层模型进行管理 [11, 12]。

考虑到数据中心架构的复杂性及其组件之间的交互，确保高度可观测性十分必要。为了在任何环境中实现这一点，收集各种指标、日志和追踪信息[13]已成为标准做法。它们为监控系统的行为和性能以及检测瓶颈或故障奠定了基础。此外，工作负载调度器和基础设施自动伸缩器主要依靠指标作为决策输入。

现有监控解决方案已经变得十分成熟，功能也十分丰富；然而，它们都不是专为数据中心模型设计的。由于数据中心具有动态性，会定期组建或解散，且许多节点不断加入或离开，因此监控系统需要足够灵活，以适应这些变化。此外，由于构成数据中心的硬件通常资源受限，因此必须采用轻量级工具，以确保系统组件的资源消耗最低。

本文聚焦于设计并实现一种适用于数据中心的监控系统，更确切地说，是用于收集——

由欧盟资助（TaRDIS，101093006）。然而，文中表达的观点和意见仅代表作者本人，并不一定反映欧盟的观点。欧盟及其资助机构均不对此承担责任。

数据中心的指标并使客户端能够访问这些指标。我们将管理数据中心的终端用户视为主要客户端，因为他们必须了解数据中心基础设施及其上运行的应用程序的状态。不过，我们也考虑了更多潜在的客户端，包括工作负载调度器、基础设施和应用级自动伸缩器，它们会依据这些数据做出决策。此外，如果这些客户端集成了机器学习模型，还可以将收集到的指标用作训练集。

在第二节中，我们讨论现有的监控解决方案，尤其是云原生解决方案。第三节和第四节分别介绍我们的解决方案、设计与实现。在第五节中，我们阐述若干设计决策背后的理由，并讨论系统的局限性和可能的改进；第六节总结全文，并展望未来的研究方向。

II. 相关工作

Prometheus[1] 是一种应用广泛的监控系统，拥有丰富的工具和集成方案。它是一种集中式指标收集与分析解决方案，采用拉取式通信模型。Prometheus 实例经过配置后，会定期从指定目标抓取指标，并将其存储在时序数据库中。经过客户端库埋点后，目标可以直接暴露其指标。然而，通常在第三方系统中，这种方式并不可行，因此人们开发了大量导出器，用于从这类系统中提取指标并将其转换为 Prometheus 格式。为了促进不同监控工具之间的互操作性，人们推出了这种格式的标准化规范 OpenMetrics[2]，。

Kubernetes[3],是一种流行的容器编排器，具有资源指标管道和完整指标管道。前者通过 metrics-server 提供有限的一组指标；metrics-server 从节点收集指标，并仅在内存中短暂存储这些指标。该管道主要供自动扩缩容器使用。后者则提供丰富得多的节点指标，但要求用户实现能够暴露这些指标的适配器。所有指标均以 OpenMetrics 格式传输。

在集群联邦中编排工作负载时，调度器需要全面了解整个联邦的情况。Acala [14] 旨在聚合来自 Kubernetes 集群联邦的指标。Global View Cluster 包含 Prometheus 存储和一组 Acala 控制器，它们会定期从各个集群请求指标。每个集群中都有一个 Acala 成员，通过收集该集群内所有节点的指标来处理这些请求。返回响应前，它会应用数据缩减策略、聚合和去重。对于适用的指标，聚合会计算集群平均值并丢弃各个节点的值；去重则会将当前值与同一指标的先前值进行比较，如果两者相等，就丢弃当前值。

[1]Prometheus：https://prometheus.io/ [2] OpenMetrics：https://openmetrics.io/ [3] Kubernetes：https://kubernetes.io/

业界一直在努力将监控集成到类似数据中心的云扩展模型中，因此我们特别关注这类方案，因为针对一种模型开发的解决方案也可能适用于其他模型 [15]。EdgeCloud Mon [16] 提出了一种将云与边缘相结合的系统，可从 k3s[4] 边缘集群收集基础设施、Pod 和虚拟机指标，并将其传输到云端，以便长期存储和访问。每个节点都通过 Node exporter[5] 和一个提供额外机器指标的自定义代理来公开基础设施指标。此外，kubevirt-exporter 通过 KubeVirt API 收集运行在工作节点上的虚拟机指标。这项虚拟化技术适用于需要以 Windows 作为操作系统的工作负载。在每个集群内部，Prometheus 实例会定期从工作节点上的导出器以及运行在主节点上的 kubevirt-exporter 抓取指标，并将这些指标推送到云端。

与我们讨论过的所有集中式解决方案相比，FogMon [17]采用了促进容错的两层分布式架构。第一层由跟随节点组成，它们收集自身的指标，并将其发送给分配给自己的、位于第二层的领导节点。领导节点汇总来自跟随节点的指标，并通过流言协议，经由对等网络与其他领导节点共享这些指标。

III. 系统设计

为了全面了解数据中心的状态，我们需要收集多个层级的指标。第一个层级是机器级，涵盖每个节点的硬件和系统软件。除了收集各个节点的机器指标外，汇总每个数据中心的部分指标也很有价值。因此，我们区分节点级和数据中心级机器指标。第二个层级是容器级，因为所有用户定义的工作负载都在这一隔离环境中运行。当模型发展到支持虚拟机和单内核等其他环境时，这一层级也将随之扩展以纳入这些环境，因此我们也可以称其为隔离环境级。最后，应用级应允许客户端应用程序生成任意指标，并由系统进行处理和存储。

指标子系统由多个组件组成，负责收集、处理、存储和检索指标。这些组件可分为运行在节点上的组件和运行在控制平面上的组件。图 1 展示了扩展后的数据中心平台架构。

每个节点都包含以下组件：

- **机器级采集器** 提供用于访问节点级机器指标的 API。
- **容器级采集器** 提供用于访问容器级指标的 API。
- **指标代理** 负责收集各级节点的节点指标，并将其传输到控制平面。

[4]k3s：https://k3s.io/ [5]Node exporter：https://github.com/prometheus/node exporter _

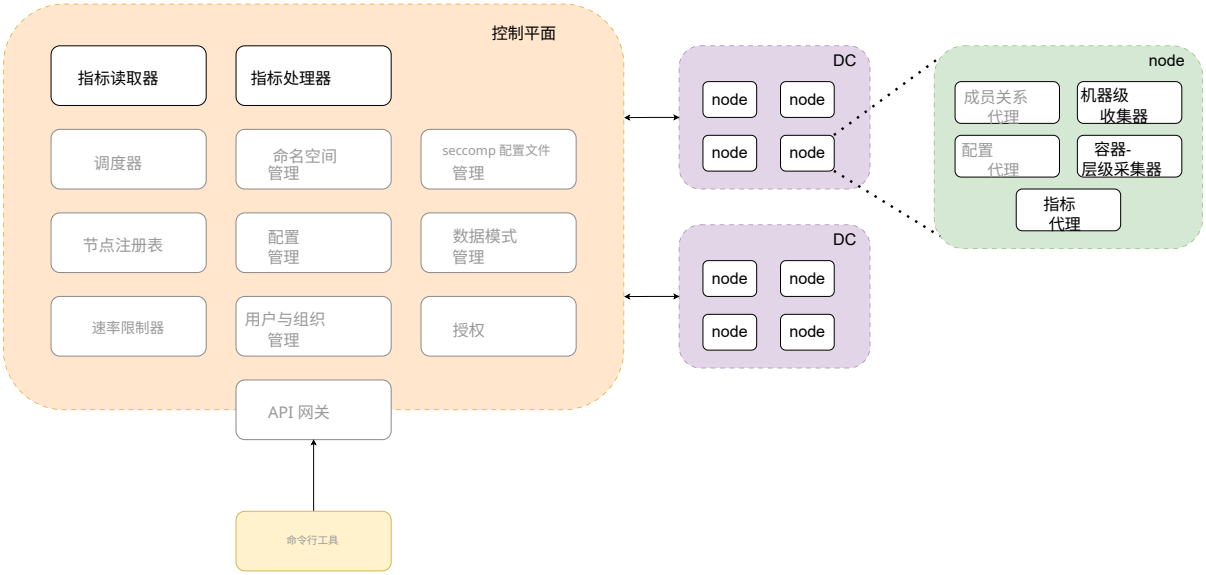

图 1：数据中心平台架构

在控制平面内部，我们还有另一组组件：

- **指标存储**，所有收集到的指标都存储于此。
- **处理器** 负责从节点收集指标、汇总并存储这些指标。

- **Reader**实现用于检索已收集指标的 API，供数据中心平台的客户端使用。

收集过程可以分为两个步骤：第一步是收集节点内部不同层级的指标，第二步是由控制平面从各节点收集指标。首先，我们讨论节点上的指标收集是如何设计的。

机器与容器指标由各自的采集器公开，这些采集器实现了约定的 API，因此可以无缝替换其实现。该 API 必须只有一个 HTTP GET /metrics端点，遵循 Prometheus 的示例。对该端点调用的响应应为自上次调用以来收集到的指标列表。在此以及所有其他指标传输环节中，我们遵循 OpenMetrics 标准，以简化与现有云原生工具的集成。指标代理定期向机器与容器采集器发送请求，并将结果暂时存储在文件系统中，直至将其传输至控制平面进行长期持久化。代理执行此操作的频率可以配置。

应用级指标的工作方式类似，主要区别在于采集目标是动态的，由数据中心用户定义。因此，指标代理必须维护一个目标列表。用户向控制平面提交新的应用时，应指定该应用是否实现了指标 API，其实现方式与机器和容器采集器所描述的相同。如果实现了，那么当调度器将该应用的一个实例分配到某个节点时，应通知指标代理该应用可访问的地址。当应用终止时，也应再次通知代理，以便其从目标列表中移除该地址。与其他层级的指标一样，代理会定期轮询目标、收集结果，并将其存储在文件系统中。需要指出的是，系统并不限制要采集哪些指标。指标完全可以自定义，用户可以从自己的应用中生成任何他们认为有价值的信息，只要正确实现 API 即可。

现在我们描述指标如何从节点传输到控制平面，并在那里进行处理和存储。处理器是一个组件，其主要目的是监控节点池的健康状况。它定期发送 ping 请求，如果某个节点在规定的时间间隔内没有回复，则被视为已失效。处理器依赖节点注册表获取所有节点的列表，并根据健康检查结果对其进行更新。我们将指标数据捎带在该协议上，目的是减少节点与控制平面之间的请求数量。当节点回复 ping 时，它还会在回复中附加自上次 ping 以来收集的所有指标。之后，它会从临时存储中清除已传输的数据。当处理器收到指标时，会将其写入指标存储。尽管如果处理器未能成功持久化数据，这种方法可能会导致数据丢失，但我们仍选择了它，因为节点的主要特性之一是它们是资源受限的。

正如我们之前提到的，某些机器指标，例如硬件资源总量和可用量，不仅应存在于节点级别，也应存在于数据中心级别。我们提出的一个简单策略是，让处理器定期检索每个数据中心的最新节点级指标，执行相应的聚合函数（通常是求和或平均值），并存储结果。

读取器组件实现了供客户端访问已存储指标的应用程序接口。它同时提供 REST 和流式传输

API。REST API 端点列于表 I 中。所有情况下，响应都是以 OpenMetrics 文本格式呈现的指标列表。

流式 API 允许客户端订阅最新的节点或数据中心指标。它可用于实时仪表板、调度器或自动扩缩容器。我们定义了主题层级*metrics.*{*nodes*|*dc*}.{*id*|*}，并允许使用通配符（*）一次订阅多个主题。如果客户端希望订阅 id 为 X 的数据中心的指标，应使用主题 *metrics.dc.X*。另一方面，如果希望接收所有节点的指标，主题应为*metrics.nodes.**

IV. 实现

我们提供的带有监控子系统的数据中心平台是一个名为 *constellations (c12s)*[6]的开源项目。Starometry[7] 是指标代理的一种实现，而处理器和 Reader 均为 protostar 项目[8]的组成部分。

所有组件，包括本文介绍的组件，均使用 Go 编程语言编写。控制平面与节点之间的通信以及流式 API 均使用 NATS[9] 作为消息代理来实现。Node exporter 和 Windows exporter[10] 用作节点上的机器级采集器，而 cAdvisor[11] 用于访问容器级指标。之所以选择 Prometheus 作为指标存储，是因为它原生支持 OpenMetrics 格式，并拥有丰富的查询语言。

该实现的正确性是在实验室条件下进行测试的，测试中模拟节点运行于虚拟机内。不过，在真实环境中使用异构硬件进行测试是我们未来的工作之一。

V. 讨论

本节将阐述系统设计决策背后的考量，并讨论我们解决方案的局限性，以及可能的改进方向。

在确定架构时，我们主要关注小型数据中心，在这类场景中，集中式采集和处理节点已足够。然而，在由数千个节点组成的大型数据中心中，该策略很容易使控制平面过载，并导致更频繁的故障或流量拥塞。针对这些场景，我们建议在每个数据中心内部组建对等网络。节点之间建立通信后，便可在内部执行健康检查，而控制平面只需与少量节点直接通信，即可采集整个数据中心的状态。此外，指标仍可搭载在健康检查协议上传输，从而使数据聚合负载能够在节点之间分担。实现方式有很多，从基于 gossip 的聚合 [18, 19], 到分层策略 [20, 21]。我们将这一问题留待未来工作。

[6]c12s 源代码：https://github.com/c12s/ [7]starometry 源代码：https://github.com/c12s/starometry [8]protostar 源代码：https://github.com/c12s/protostar [9]NATS：https://nats.io/ [10]Windows exporter：https://github.com/prometheus-community/windows exporter [11]cAdvisor：https://github.com/google/cadvisor

由于节点会持续产生大量指标数据，因此有必要采用策略，以减少传输中和静态存储时的数据量。在此，我们区分无损策略与有损策略。使用适用于时间序列数据的无损压缩算法，可以显著降低所需的网络和持久化存储量。不过，重点应放在轻量级算法上，因为压缩和解压缩会引入延迟，并需要更多处理能力。一种可大幅减小数据规模的有损策略是对指标进行动态采样，从而只记录相关数据点。如果数值在较长时间内保持稳定，仅偶尔出现显著变化，那么大部分数据可以在节点上丢弃。这样，只需将一小部分数据传输到控制平面并永久存储，其余数据则可以以令人满意的精度重建。

我们在第三节中提到的一项设计决策，在某些情况下可能会造成系统局限：如果节点发送指标后将其删除，但控制平面未能成功持久化这些数据，就可能发生数据丢失。我们选择这种做法，是为了降低节点的磁盘使用量，并基于一个假设：用户可以容忍偶尔丢失指标。不过，如果节点有充足的磁盘空间，且不允许丢失任何指标，则可以调整所提出的收集策略。控制平面可以扩展为在保存数据后发送确认消息，节点只有在收到确认后才会将数据从存储中删除。如果在规定的时间间隔内未收到确认，节点就会重新发送这些数据。

由于数据中心需要支持多样化的用例，且其结构可能各不相同，因此必须提供多种指标收集、聚合、压缩和存储策略。因此，我们认为平台应包含此处讨论的所有方法，并允许最终用户选择最适合其需求的配置。

VI. 结论

本文提出了一种在数据中心内部运行的指标收集、聚合和检索系统的可行设计，并讨论了其改进方案和替代方案。该系统涵盖从最低机器级到客户端应用程序生成的自定义指标。系统组件分为两类：一类运行于节点上，负责数据收集；另一类位于控制平面内，负责从节点收集数据、进行聚合并向用户提供数据。我们还将该设计与一个现有的开源数据中心平台集成，完成了相应实现。

由于我们的解决方案是集中式的，且严重依赖控制平面，因此我们计划采用一种更去中心化的策略作为补充：由数据中心收集并汇总数据，之后再将其共享给控制平面，以便更轻松地访问。此外，我们将重点研究轻量级算法，通过过滤和压缩指标，尽可能减少所需资源的数量。此外，完善的告警机制是任何监控系统不可或缺的组成部分，因此开发告警机制也是我们未来工作的内容之一。我们还计划利用该监控系统来开发基础设施

表 I：指标 REST API

方法	Path	参数	描述
GET	/api/metrics-api/节点/{节点标识符}/{时间戳}	节点标识符 - 节点标识符 时间戳 - 范围起始时间戳	该端点会检索指定节点的指标，这些指标从指定时间戳开始收集，直到现在。
	/api/metrics-api/节点/{节点标识符}/{开始时间}/{end}	nodeID - 节点标识符 start - 范围起始时间戳 stop - 范围结束时间戳	该端点会检索指定节点的指标，这些指标是在指定时间范围内收集的。
	/api/metrics-api/dc/{数据中心标识符}/{时间戳}	dcID - 数据中心标识符 timestamp - 范围起始时间戳	该端点获取指定数据中心的指标，这些指标从指定时间戳开始收集，直至当前时间。
	/api/metrics-api/dc/{数据中心标识符}/{开始时间}/{end}	dcID - 数据中心标识符 开始时间 - 范围起始时间戳 结束时间 - 范围结束时间戳	该端点会检索指定数据中心的指标，这些指标是在指定时间范围内收集的。
	/api/metrics-api/containers/{容器标识符}/{时间戳}	容器标识符 - 容器标识符 时间戳 - 范围起始时间戳	该端点检索指定容器的指标，从指定时间戳开始收集，直到 now.
	/api/metrics-api/containers/{容器标识符}/{开始时间}/{结束时间}	containerID - 容器标识符 start - 范围起始时间戳 stop - 范围结束时间戳	此端点检索指定容器在指定时间范围内收集的指标。
	/api/metrics-api/apps/{应用标识符}/{时间戳}	应用标识符 - 应用标识符 时间戳 - 范围起始时间戳	该端点检索指定应用的指标，这些指标从指定时间戳开始收集，直至 now.
	/api/metrics-api/apps/{应用标识符}/{开始时间}/{结束时间}	应用标识符 - 应用标识符 开始时间 - 范围起始时间戳 结束时间 - 范围结束时间戳	该端点检索指定应用的指标，这些指标是在指定时间范围内收集的。
	/api/metrics-api/{时间戳}	timestamp - 范围起始时间戳	该端点会检索从指定时间戳至今收集的所有指标。主要用于训练机器学习模型。
	/api/metrics-api/{开始时间}/{end}	start - 范围起始时间戳 stop - 范围结束时间戳	该端点会检索为指定时间范围收集的所有指标。也旨在用于训练机器学习模型。
	/api/metrics-api/节点/{节点标识符}/latest	nodeID - 节点标识符	该端点用于获取指定对象的最新指标节点。它仅包含关键指标的一个子集，用于在仪表板上显示。
	/api/metrics-api/dc/{dcID}/latest	dcID - 数据中心标识符	该端点用于获取指定对象的最新指标数据中心。它仅包含一部分关键指标，旨在显示在仪表板上。

和应用级自动伸缩器。为确保所有组件正确、高效地运行，我们将在真实环境中测试我们的系统。

参考文献

[1] A. J. Ferrer, J. M. Marqu`es, and J. Jorba, “迈向去中心化云：移动、ad hoc 和边缘计算的方法与挑战综述，” *ACM*计算调查 *(CSUR)*, vol. 51, no. 6, pp. 1–36, 2019. [2] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, “边缘计算：愿景与挑战，” *IEEE*物联网期刊, vol. 3, 646, 2016 5, 637no. pp. – .[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “雾计算及其在物联网中的作用，” 载于首届移动云计算*MCC*研讨会论文集, 2012, pp. 13–16. [4] M. Simi´c, I. Proki´c, J. Dedei´c, G. Sladi´c, and B. Milosavljevi´c, “迈向边缘计算即服务：微型数据中心的动态组建，” *IEEE Access*, vol. 9, pp. 114 468–114484, 2021.

[5] M. Simi´c、G. Sladi´c、M. Zari´c 和 B. Markoski，“边缘微型云中的基础设施即软件”，传感器，第 21 卷，7001，2021 年 21，第 p. .[6] A. Greenberg、J. Hamilton、D. A. Maltz 和 P. Patel，“云的成本：数据中心网络中的研究问题”，第 68–73 页，2008 年。[7] M. Simi´c、J. Dedei´c、M. Stojkov 和 I. Proki´c，“分布式云中支持多租户的分层命名空间方法”，*IEEE Access*，2024 年。[8] ——，“支持协作应用的分布式云数据覆盖网格”，*IEEEAccess*，2025 年。[9] T. Rankovi´c、I. Kova˘cevi´c、V. Maksimovi´c、G. Sladi´c 和 M. Simi´c，“分布式云中的配置管理”，载于信息技术及其应用会议。Springer，2024 年，第 224–235 页。[10] T. Rankovi´c、F. ˇSilji´c、J. Tomi´c、G. Sladi´c 和 M. Simi´c，“分布式云应用的错误配置预防与错误原因检测”，载于 *2024*年*IEEE*第*22*届智能系统与信息学国际纪念研讨会（*SISY*）。IEEE，2024 年，第 000297–000302 页。

[11] T. Rankovi´c、M. Simi´c、M. Stojkov 和 G. Sladi´c，“分布式微云环境中的访问控制”，载于信息技术及其应用会议。Springer，2024 年，第 435–447 页。[12] T. Rankovi´c、V. Maksimovi´c、M. Simi´c、B. Milosavljevi´c 和 G. Sladi´c，“在分布式云部署中实施零信任”，信息技术与教育发展 *ITRO 2023*，第 142 页。[13] S. Karumuri、F. Solleza、S. Zdonik 和 N. Tatbul，“迈向大规模可观测性数据管理”，*ACM Sigmod Record*，第 49 卷，第 4 期，第 18–23 页，2021 年。[14] C.-K. Huang 和 G. Pierre，“地理分布式 Kubernetes 集群联邦的聚合监控”，*IEEE* 云计算汇刊，2024 年。[15] B. Costa、J. Bachiega Jr、L. R. Carvalho、M. Rosa 和 A. Araujo，“雾计算监控：综述、分类与开放挑战”，*Computer Networks*，第 215 卷，第 109189 页，2022 年。[16] I. Korontanis、A. Makris 和 K. Tserpes，“Edgecloud Mon：面向 k3s 集群的轻量级监控技术栈”，*SoftwareX*，第 26 卷，第 101675 页，2024 年。[17] A. Brogi、S. Forti 和 M. Gaglianese，“温和地测量雾计算”，载于国际服务导向计算会议。Springer，2019 年，第 523–538 页。[18] D. Kempe、A. Dobra 和 J. Gehrke，“基于 gossip 的聚合信息计算”，载于第 *44* 届 *IEEE* 计算机科学基础年会研讨会，*2003* 年。论文集。IEEE，2003 年，第 482–491 页。[19] M. Jelasity、A. Montresor 和 O. Babaoglu，“大型动态网络中的基于 gossip 的聚合”，*ACM* 计算机系统汇刊（*TOCS*），第 23 卷，第 3 期，第 219–252 页，2005 年。[20] M. L. Massie、B. N. Chun 和 D. E. Culler，“Ganglia 分布式监控系统：设计、实现与实践经验”，*Parallel Computing*，第 30 卷，第 7 期，第 817–840 页，2004 年。[21] K. Graffi 和 A. Disterh¨oft，“Skyeye：一种基于树的点对点监控方法”，*Pervasive and Mobile Computing*，第 40 卷，610，2017 593pp. – 。