本文已提交至 IEEE，等待其考虑发表。版权可能在不另行通知的情况下转让，届时此版本可能不再可供访问。

arXiv:2304.01915v2 [cs.DC] 3 Mar 2024

雾设备即服务（FDaaS）：面向公共雾环境中服务部署的框架

Sudheer Kumar Battula，*IEEE* 会员，Saurabh Garg，*IEEE* 会员，James Montgomery，*IEEE* 会员，以及 Ranesh Naha，*IEEE* 会员

摘要——由于缺乏公开可用的雾节点及其特性，在雾计算环境中满足未来服务的时延敏感需求并应对服务的负载峰值，是一项艰巨的任务。研究人员认为，可以将传统的自动伸缩技术与轻量级虚拟化技术（容器）结合，用于在雾计算环境中提供自动伸缩功能；少数研究人员则利用容器编排工具或系统的默认自动伸缩技术搭建了平台。然而，在公开可用的雾基础设施中，仅采用这些技术并不能保证服务质量（QoS），因为雾设备具有异构性及其特有的属性，例如资源频繁变化和高移动性。为应对这一挑战，我们在本研究中开发了一个雾即服务（FaaS）框架，可创建、配置和管理运行于雾设备上的容器，以部署服务。本文介绍了用于应对服务负载峰值、满足应用程序服务质量要求的关键技术和算法。本研究通过在真实场景下与现有技术进行比较，评估了该框架。实验结果表明，在不同场景中，我们提出的方法使已满足的服务请求平均增加至原来的1.9倍。

索引词—雾计算、资源部署、自动伸缩和物联网。

1 引言

雾 OG 计算是最新的计算范式，它将计算能力带到更靠近终端设备的位置，从而为各种更智能的应用提供类似云的服务 [1], [2], [3], [4]。在公共场所，任何人都可以通过提供设备来处理用户的服务请求，从而参与雾计算环境。该基础设施并非专门用于提供服务，也没有集中式控制。当请求服务的用户数量不断增长并且预计还会持续增加时，雾系统需要通过部署服务来扩充资源，这会导致请求服务时间延长。

在这种志愿者环境中部署和管理服务，以便高效应对服务负载的突增，需要确定适当扩缩虚拟雾设备的确切设备数量。这样既能保证服务质量（QoS），又能最大限度地提高整体资源利用率。如果系统低估了需要扩容的设备数量，就会导致资源配置不足。设备负载将会增加，可能无法满足应用的服务质量（QoS）。如果系统估计需要扩容的设备数量过多，就会导致资源配置过剩。由于资源稀缺，服务请求的拒绝数量可能会增加，成本也会随之上升。因此，鉴于雾计算环境中雾设备行为的不确定性、资源变化的高度动态性以及雾设备的高度移动性，这一过程是一项复杂的任务。此外，由于大量终端用户订阅了这些服务，应用负载需求可能随时出现峰值。

目前已有一些雾计算框架、平台和服务，用于管理雾设备并编排应用程序工作负载。Santoro 等人 [5] 提出了一种名为 Foggy 的工作负载编排架构框架和平台。Powell 等人 [6] 提出了雾开发工具包（FDK），该工具包抽象了资源分配的复杂性，并为开发人员提供高级接口，以便快速开发系统。然而，作者将自动伸缩功能视为未来工作。

许多研究人员使用现有的容器编排和分析工具构建了雾系统；他们利用编排工具默认的自动伸缩方案、技术或功能来提供自动伸缩特性 [7], [8], [9], [10]。然而，在这些默认方案中，资源会根据当前可用资源进行伸缩。因此，这些方案忽略了资源并非全部专用于提供服务这一事实。因此，仅在志愿者雾基础设施中采用这些方案或技术，并不能保证服务的服务质量（QoS）。

因此，本研究旨在开发雾即服务，以便在

• *S. K. Battula*、*S. Garg*、*J. Montgomery* 和 *R. K. Naha* 均就职于澳大利亚塔斯马尼亚大学信息与通信技术学院（*TAS 7005*），电子邮件：{*sudheerkumar.battula*；*saurabh.garg*；*james.montgomery*；*raneshkumar.naha*}*@utas.edu.au* 通讯作者：*S.K. Battula* 收稿日期：*2021*年*3*月*XX*日；修订日期：*2021*年*XXXXX XX*。

雾环境中提供高效且可自动伸缩的物联网服务。其核心思路是开发一个框架，即使负载突然达到峰值，也能为物联网及其他服务高效提供按需虚拟雾计算资源。

本文的贡献如下：1）提出并实现一种 FaaS，以便在公共雾环境中高效部署服务。2）提出一种自动伸缩技术，根据对未来工作负载的预测，并基于当前状态动态估计雾设备的未来状态。

本文的组织结构如下。第2节讨论雾计算系统和平台中的自动伸缩相关工作。第3节介绍雾即服务的设计目标。第4节概述系统模型，并介绍一种利用设备最低可用性信息的动态资源扩缩算法。第5节详细介绍实时案例。第6节展示不同场景下的实验结果。最后，第7节总结全文。

2 相关工作

自动伸缩问题在云计算和物联网领域已得到广泛研究。然而，由于雾计算环境具有独特的特性，用于解决这一问题的技术并不适用。此外，迄今已有多种平台、仿真器和框架被开发出来，以解决雾计算环境中的自动伸缩问题。本节将总结与雾计算环境中自动伸缩问题密切相关的研究工作。

Mayer 等人 [11] 提出了一种名为 EmuFog 的可扩展雾计算仿真器框架。在这项工作中，作者介绍了该框架的关键组件，并阐述了骨干连接算法和节点放置算法。然而，作者未将雾节点的资源动态性以及实现移动性模型纳入计划。De Alfonso 等人 [12]展示了采用容器创建虚拟计算集群以满足科学应用需求的可行性。作者使用开源工具 Docker 弹性集群（EC4Docker），并集成了 Docker Swarm 以启用自动伸缩功能。然而，这项工作需要手动创建或生成图像和 Docker 文件。此外，集群部署在裸金属机器上。

目前已有许多框架被提出，用于支持雾计算环境中的自动伸缩，这些框架利用了现有容器编排平台的可扩展性特性。Bellavista 等人 [13]提出了一种采用 Kura 网关的雾计算框架。该框架采用基于 Docker 的容器化，适用于资源受限的雾设备上的物联网应用。Kura 框架集成了本地代理，以支持自动伸缩。通用 Kura 网关会先汇总收集到的所有数据，再将传感器数据发送至消息队列遥测传输协议（MQTT）代理。部署在云上的 MQTT 代理通过地理分布式全球云管理计算任务。所提出的 Kura 框架扩展全面支持基础设施管理，例如更新、下载以及管理虚拟化的 Docker 镜像。容器化支持可移植性和互操作性，有助于利用不同的雾节点构建基于容器的雾环境。类似地，还有其他框架利用 Kubernetes 的自动伸缩功能 [14] [15] [16]。这些框架在自动伸缩时并未考虑到设备并非完全专用于处理应用程序这一情况（即公共雾设备）。

关于动态资源配置框架和技术，我们也能看到一些关于自动伸缩的有趣研究成果。研究者提出了一种名为 MIST 的新型雾计算通信方案，用于支持基于消费者数据、任务分配和虚拟机放置的不同物联网感知应用工作负载 [17]。在这项研究中，作者将自动伸缩功能列为未来工作。类似地，Tseng 等人 [18] 提出了一种通过集成虚拟机监控程序和容器化技术构建的轻量级雾计算平台；他们提出了一种基于资源当前状态的基于模糊逻辑的自动伸缩机制，以提高工业应用的可扩展性。Yousefpour [19] 为雾服务提出了一种名为 FOGPLAN 的轻量级、服务质量感知的动态雾服务配置（QDFSP）框架。作者将雾服务的部署与释放问题表述为一个优化问题。为解决这一问题，作者提出了两种高效的贪心算法。在这项研究中，作者采用二元变量来决策是在雾计算还是云中部署物联网服务。此外，作者计划在未来工作中，根据负载变化，使用非二元变量研究水平扩展和垂直扩展的性能。类似地，还有其他一些服务放置框架 [20], [21], [22],[23]。然而，这些研究大多依据上下文、吞吐量或资源感知来处理服务部署或放置。此外，这些研究在决策时仅将延迟和当前资源利用率作为考量因素。由于雾节点是由志愿者提供的设备且资源有限，我们必须考虑其他参数，才能满足应用的服务质量要求。因此，我们开发了一个 FaaS 框架，以弥补相关文献中公共雾环境下服务部署自动伸缩框架的空白。

3 FaaS 目标

本节介绍根据相关研究的局限性和未来工作提出的设计目标，并详细讨论用于扩展节点、以满足用户/应用需求的自动伸缩算法。

3.1 FaaS 目标

为满足未来长期运行服务的需求，这些服务应具备以下特性：

1) **移动性管理**：雾计算中的大多数雾节点本身具有移动性。FaaS 应以尽可能少的雾设备间迁移，自动处理雾节点的移动性。2)**大规模部署**：相同的服务必须能够在多个位置运行，且每项服务可能

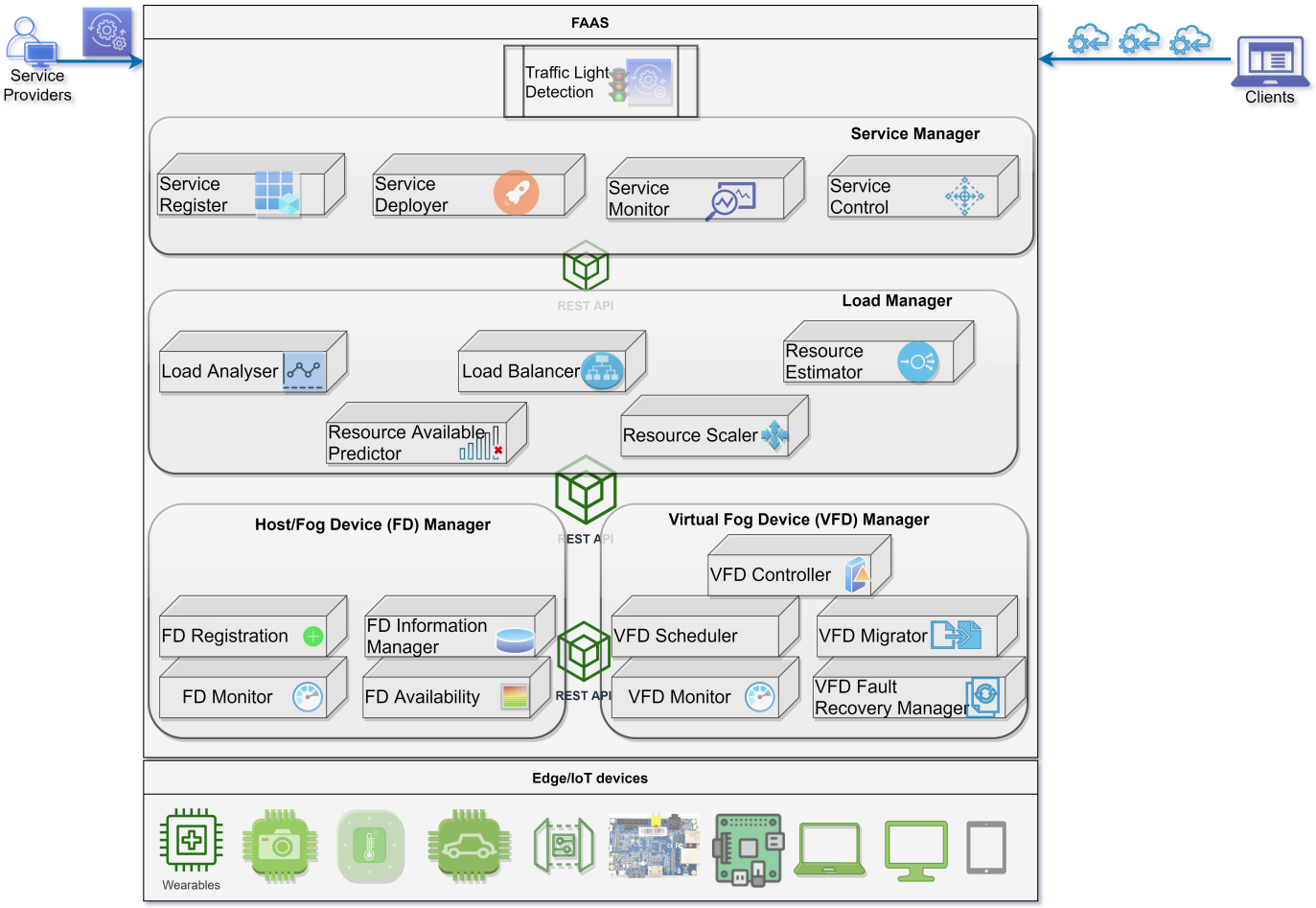


图 1. FaaS 框架。

需要多台设备才能满足用户的需求。FaaS 根据应用的需求自动配置资源，使其部署更加轻松。

3) **单击管理**：由于大多数设备都是分散部署且具有高度移动性，采用物理方式管理这些设备是一项复杂的任务。FaaS 能够通过单击配置并监控雾设备的当前状态。

4) **资源动态管理：** 雾计算环境中的雾设备并非完全专用于处理用户的请求。因此，设备的资源可能随时发生变化，从而导致应用程序不可用。FaaS 应以尽可能少的故障应对资源动态变化，避免资源短缺造成故障。5) **自动扩缩管理：** FaaS 支持自动伸缩功能，以满足用户的需求。6) **可扩展性：** FaaS 提供 REST API，可根据具体用例修改内置技术或重置默认参数。

4 系统模型

本节概述雾计算即服务（FaaS）及应用模型。我们假设应用程序或服务部署并运行在容器中，以便在雾设备上实现资源共享。我们假设在本研究中，服务持续运行于雾设备的容器中，而这些设备本身并非专用设备，且具有移动性。因此，要持续运行服务，需要同时使用多个设备。

4.1 FaaS 模型

雾计算在靠近最终用户的位置提供类似于云的服务。其中一项服务是以容器形式实现的虚拟化雾设备，称为雾即服务（FaaS）。FaaS 以容器形式提供存储、计算和网络。FaaS 模型向用户或应用提供虚拟雾设备，使其能够在雾计算环境中运行服务。根据应用负载，雾设备的数量可以扩增或缩减，以满足应用的需求。图 1 展示了所提出框架的概览。不同的管理器组件分别负责管理雾设备、虚拟雾设备和应用负载。在顶层，负载管理组件包含所提系统的多种功能，例如负载分析器、负载均衡器、资源估算器和扩缩器。所提出的 FaaS 的下一层包含两个组件：(i) 主机/雾设备（FD）管理器和 (ii) 虚拟雾设备管理器。这些管理器可以部署在不同节点上，并通过 RESTful API 进行通信。

有关这些管理器的更多详细信息将在以下各节中说明。

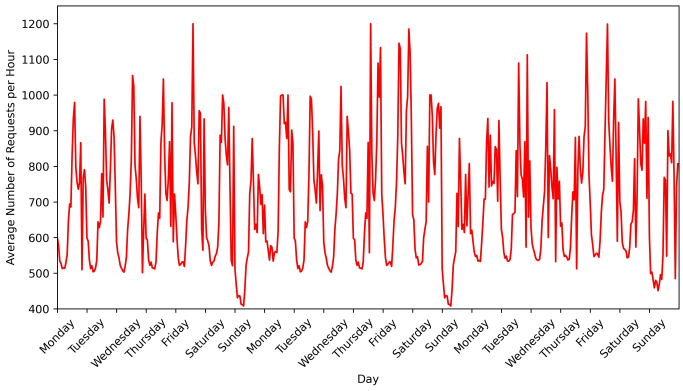


图2. 历史数据

负载管理组件包含五个子组件：(i) 应用负载分析器 (ii) 资源估算器/预测器 (iii) 可用资源预测器 (iv) 动态资源伸缩器和 (v) 负载均衡器。**应用负载分析器**：应用负载分析器负责预测特定时间可能到达的请求数量。此信息是根据应用代理收集的历史数据进行预测的。图5展示了基于工作负载请求率 [24] 的三周模拟期间每小时接收的平均请求数。应用程序的未来负载信息将发送至资源配置模块，以估算扩容或缩容所需的虚拟雾设备数量。时间序列分析 (TSA) 和时间序列建模 (TSM) 被广泛用于预测工作负载。由于工作负载存在不确定性，较早的工作负载数据对未来预测影响不大。因此，系统仅存储最近 N 天的数据 [25]。我们假设应用负载在一天中的高峰时段会遵循某种趋势。因此，我们采用季节性自回归积分滑动平均 (SARIMA) 模型。SARIMA 模型与 ARIMA 等价，但 SARIMA 会考虑数据的季节性。这类模型包含两部分：(i) 非季节性 ARIMA 模型，用于预测长期变化；(ii) 季节性 ARIMA 模型，用于预测季节周期。其形式如下：

$$ARIMA(p,d,q)\times(P,D,Q)_s$$

其中， p、 p 和 q 分别表示非季节性 AR 阶数、差分阶数和 MA 阶数。同样，P、D 和 Q 分别表示季节性 AR 阶数、差分阶数和 MA 阶数。s 表示季节性数据的长度。季节性 ARIMA 模型可表示如下：

$$\phi_p(L)\Phi_P(L)^s\Delta^d\Delta_s^D y_t = A(t) + \Theta_q(L)\widetilde{\Theta_q}(L^s)\varepsilon_t \quad (1)$$

其中， L是后移算子或滞后算子， s 是季节性趋势的时间跨度，因为我们预期季节性以每天为周期、按小时为单位出现，s=24。 ϕ_p 是 AR， Phi_P 是季节性自回归（AR）。 Δ^d 表示差分，Δ_s^D表示季节性差分。 y_t 是时间序列。 $A(t)$ $\widetilde{}$是常数趋势， Θ_q（L）是滑动平均（MA）。 Θ_q（L^s）是季节性 MA， ε_t 是误差。负载分析器将预测每小时的平均到达请求数。基于上述历史数据，SARIMA 模型的工作负载预测如图3所示。

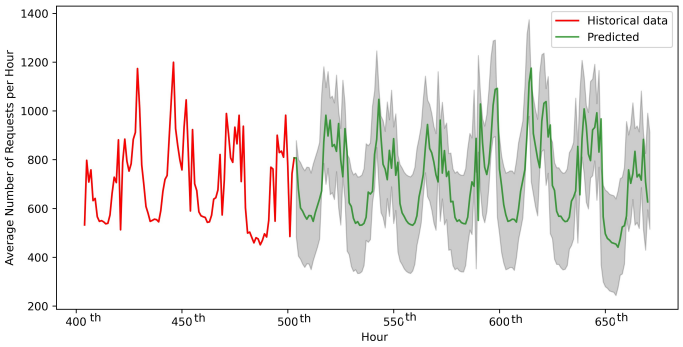


图 3. 预测请求

资源估算器/预测器：资源预测器负责估算满足应用程序服务质量所需的虚拟雾设备数量及其类型。所考虑的服务是 CPU 和内存密集型服务，因为需要上传图像以检测交通信号灯。每种虚拟雾设备类型都有预定义的规格。最优资源预测器需要根据未来的应用负载，估算虚拟雾设备的数量及其类型。我们将资源预测器定义为以下线性形式：

$$Q_{(t)} = \sum_{i=1}^{k} \alpha_i x(t_{(i)}, \mu) \quad (2)$$

其中， $Q_{(t)}$表示时间 t 的平均响应时间，该时间由某个服务质量参数指定。 k 表示虚拟雾设备类型的数量。 α 表示满足应用程序服务质量所需的特定类型资源数量。 $x(t_{(i)},\mu)$ 是一个查找表，提供动态工作负载下各类虚拟雾设备的平均响应时间信息。 μ 表示预测的应用程序传入请求数 IR 与系统在时刻 $nSRF$ 仍可容纳的请求数之间的差值。

$$\mu = IR - nSRF \quad (3)$$

该资源预测器的主要目标是最小化虚拟雾设备的数量，使其能够以最优方式满足虚拟雾设备中的请求，从而达到 QoS 目标。我们提取出可行的资源预测器，以最小化虚拟雾设备的数量及其规模，

通过求解以下优化问题：

$$\begin{aligned} & min \sum_{i=1}^{k} \alpha_i x(t_{(i)}, \mu) \\ & s.t. \\ & \sum_{i=1}^{k} \alpha_i x(t_{(i)}, \mu) \leq Q_{(t)} \\ & 0 \leq \alpha_i \leq A_i \\ & \forall i \in 1, 2, .., k \\ & \mu > 0 \end{aligned}$$

其中， A_i 是时间间隔 $t \to t + \delta t$ 内 i 类资源的可用数量。

可用资源预测器：可用资源预测器负责预测 A_i。时间间隔 $t \to t+\delta t$ 内可用的 i 类设备数量。该数量可通过以下方程计算，该方程来自我们的先前工作 [26]。

$$x^* = \min\{x : Pr(X_{(\ell)}(u) \in \mathcal{A}(\ell, k, x) \text{ for all } u \in [t, t + \Delta t]) \geq y\} \quad (4)$$

$$Pr(X_{(\ell)}(u) \in \mathcal{A}(\ell, k, x) \text{ for all } u \in [t, t + \Delta t]) = \sum_{i,j \in \mathcal{A}(\ell,k,x)} [\boldsymbol{\alpha} e^{\mathbf{Q}_{(\ell)} t}]_i [e^{\mathbf{Q}_{(\ell);\mathcal{A}(\ell,k,x)} \Delta t}]_{ij}, \quad (5)$$

其中$\mathbf{Q}_{(\ell);\mathcal{A}(\ell,k,x)} = [\ [\mathbf{Q}_{(\ell)}]_{ij}\ ]_{i,\ j \in \mathcal{A}(\ell,k,x)}$ 是一个非保守生成矩阵（部分行和为负），仅限于状态 $\mathcal{A}(\ell,\ k,\ x)$ 之间的转移。

动态资源伸缩器：一旦预测所需的资源可用于扩展，准入控制机制就会将此请求转发给动态扩缩器。动态资源伸缩器将选择所需的资源。算法4.1详细介绍了动态资源伸缩器服务。此服务会将预测的资源数量 $pres$ 与可用资源 $pavail$进行比较。如果 $pres$ 大于 $pavail$ ，则由于可用资源不足，请求将被拒绝；否则，服务会估算处理请求$etime$ 所需的时间，并将其与 $QoStime$ 进行比较。如果不匹配，请求将因处理时间超过服务质量规定的时限而被拒绝。否则，服务会获取已运行这些服务的主机列表 $cres$ 。检查当前运行应用服务的每台主机上的资源利用率，例如内存、CPU和存储，以决定是否执行垂直扩展。即使托管机器的配置足以扩展资源，动态资源伸缩器也不会启动扩展操作，因为这些机器并非完全专用于处理此服务。因此，服务将根据当前资源状态和可用的设备历史记录，预测下一小时的可用资源 $pnres$。如果预测的主机状态与预测的资源类型 $prestype$相匹配，则会增加VFD的规模。对所有已托管该应用程序的机器都会执行此操作。如果预测的资源数量仍有待扩展，则服务会预测并找到与预测资源 $Pdlist$ 相匹配的设备，并更新资源选择计划。资源选择计划包含 did ，即唯一的设备标识符，以及设备需要分配资源的时长。

负载均衡器：为了将请求分配给虚拟雾设备，以满足应用程序的时效性要求，我们采用了加权轮询负载均衡算法。具体而言，我们根据每台虚拟雾设备的规模和每秒平均处理请求数，为机器分配权重。权重较高的机器将收到更高比例的请求。

Algorithm 1 Dynamic resource scaler technique by exploiting device minimum availability information

Data: Registered Fog device information, App information,
Result: [did, duration]
preq ← predict the number of requests for the service next one hour pres ← estimate the required number of resources with type based on the *preq*
pavail ← predict the available resources **if** $pres \geq pavail$ **then**
  Reject the request, not enough resources available
else
  etime ← estimate the time taken to serve the requests.
  if $etime = QoStime$ **then**
    cres[] ← get the available resources of the hosts which are running the service.
    Sd←0;
    for *Resource r* ∈ *cres* **do**
      if *Presnum* ≥ *Sd* **then**
        pnres ← predict the available resources for the next hour based on the current resource.
        if *pnres* ≥ *prestype* **then**
          increase size of the virtual Fog device
          sd←sd+1
    Pdlist ← find the devices that match the predicted resources.
    while $PresnumSd$ **do**
      if $pnres \geq prestype$ **then**
        update the resource selection plan
        sd ← $sd + 1$
  else
    Reject the request if it takes a longer time than the QoS time.

虚拟雾设备管理器组件包含五个子组件：(i) VFD 控制器；(ii) VFD 调度器；(iii) VFD 迁移器；(iv) VFD 监控器；以及 (v) 虚拟雾设备故障恢复管理器。

VFD 控制器：虚拟雾设备控制器负责在主机设备中创建、启动、停止和删除虚拟雾设备。

VFD 调度器：虚拟雾设备调度器负责将主机设备分配给虚拟雾设备。

VFD 迁移器：虚拟雾设备迁移器负责将虚拟雾设备迁移到其他可用且正在运行的主机。

VFD 监控器：虚拟雾设备监控器负责

监控虚拟雾设备的 CPU、RAM 和网络使用情况。**虚拟雾设备故障恢复管理器**：虚拟雾设备故障恢复管理器负责在可用的雾设备中运行故障虚拟雾设备的类似服务。

雾设备管理器组件包含四个子组件：(i) FD 注册，(ii) FD 信息管理器，(iii) 雾设备监控器，以及 (iv) 雾设备可用性

FD 注册：FD 注册负责注册雾设备。设备注册时会记录其所有配置详情及可能的移动位置。FD 注册会创建并分配一个唯一标识符，用于识别该设备。

FD 信息管理器：FD 信息管理器负责存储主机设备的信息。

雾设备监控器：雾设备监控器负责监控雾设备的资源利用情况和位置信息。

雾设备可用性：雾设备可用性组件负责维护每台雾设备的可用性信息。

服务管理器负责注册、部署、监控和控制这些服务。**服务注册器：**服务注册器负责接收服务请求，并维护有关服务需求的信息。**服务部署器：**负责将部署新服务的决策传达给服务控制器。**服务控制器：**负责控制这些服务。**服务监控器：**服务监控器会定期更新客户端所请求服务的状态，同时更新相关信息。

本文所考虑的服务从物联网设备收集数据，在虚拟雾设备中进行处理，并将结果发送回服务器。本节将介绍我们认为可用于验证 FaaS 的用例示例。交通信号提醒服务就是一种应用服务。尽管云中已经存在这类应用程序或服务，我们仍在 FaaS 中将其部署到雾设备上，以降低应用程序的延迟。为了贴合现实场景，这类应用程序的处理时间必须在几秒之内，以便用户能够迅速响应。因此，我们在雾架构中实现这些应用程序，并将其部署到雾设备上，以改善其延迟。请注意，我们的贡献并不在于应用程序本身，而是利用这些应用程序来验证该平台。

交通信号提醒服务包含三个模块：(i) 交通信号输入采集器，(ii) 交通信号检测器，以及 (iii) 用户提醒系统。交通信号输入采集器模块负责按特定时间间隔采集图像，并将这些图像发送给交通信号检测器模块。交通信号检测器模块负责接收来自交通信号图像采集器的输入，通过执行图像分析（使用 Open CV 程序）识别图像中的信号颜色，并将信息发送给用户提醒系统。用户提醒系统负责根据以下内容提醒用户：

i交通信号检测器提供的信息 模块。

4.2 FaaS 工作流

如图 4 所示，雾设备提供商通过 FD 注册器注册雾设备，并由 FD 注册器指定监控间隔。随后，FD 监控服务会从各个雾设备收集资源信息，并将资源利用率指标推送至 FD 信息管理器。服务提供商通过服务注册器注册服务，同时提交相关需求。这些需求会发送给服务部署器，以便在虚拟雾设备上部署服务。服务部署器根据注册时指定的需求，请求服务控制器处理该请求。服务控制器将已分配的请求发送给资源伸缩器。资源伸缩器根据从 FD 信息管理器收集的信息，将 FD 列表及其类型传递给 VFD 调度器。VFD 调度器向 VFD 控制器发送请求，并告知 VFD 应在特定 FDS 中运行的时间和时长。VFD 控制器在 FD 上创建 VFD 并部署服务。此信息将更新至 VFD 监控器。VFD 监控器会持续跟踪所有 VFD 的资源信息。根据指定的系统负载和服务执行时间，FD 监控器和 VFD 监控器会触发服务伸缩事件。根据此事件，系统会通知资源伸缩器，根据收集到的信息分配或释放资源。VFD 监控器还会将故障 VFD 的详细信息发送给 VFD 控制器。VFD 控制器将信息转发给 VFD 迁移器，后者会通过资源伸缩器，将所有 VFD 及其服务迁移到其他 FD 上。

5 性能评估

本节介绍实验环境、用于评估我们所提方法的评估参数、实验结果及讨论。

5.1 实验环境

为模拟雾环境，我们使用了两台树莓派设备、五台台式机和一台笔记本电脑。这些设备的配置见表 1。为了模拟公共雾节点的行为，我们运行了不同的示例应用程序以增加系统负载。此外，我们通过公共雾动态行为模块自动断开和连接设备。我们使用有线和无线连接在设备之间进行通信。

FaaS 使用 Java、Spring Boot、JPA、Spring REST 和 Discovery 开发，各微服务分别部署在不同的雾设备上；这些微服务与其余 API 进行了交互。

在两个场景中，使用名为 Scale-X 的扩展方法评估所提出的算法。在 Scale-X 方法中，每当虚拟雾设备达到 CPU 和 RAM 的 X% 时，算法就会检查主机雾设备的资源利用率；如果主机雾设备的资源利用率百分比低于 X%，则会执行

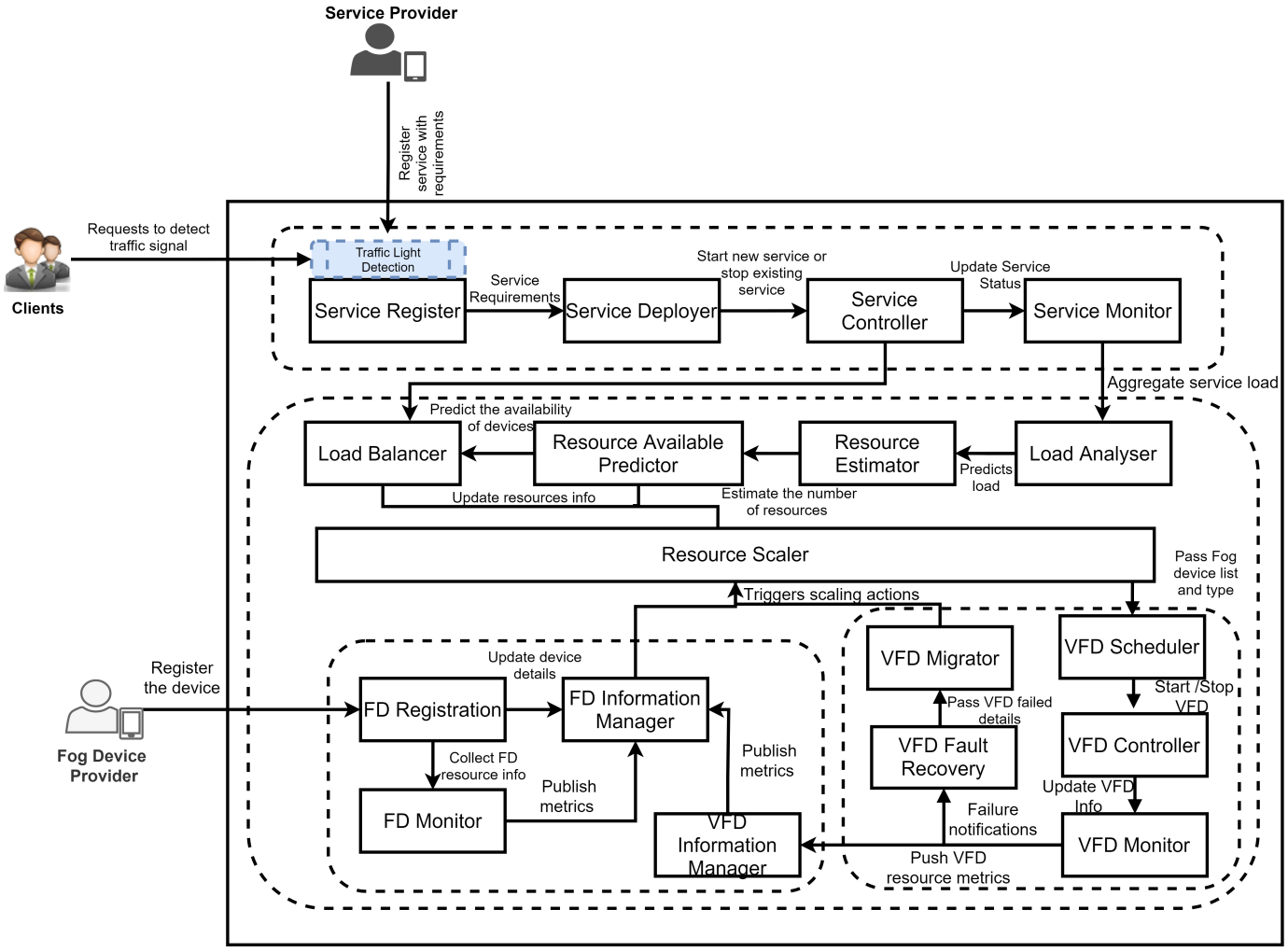


图 4. FaaS 工作流的全貌。

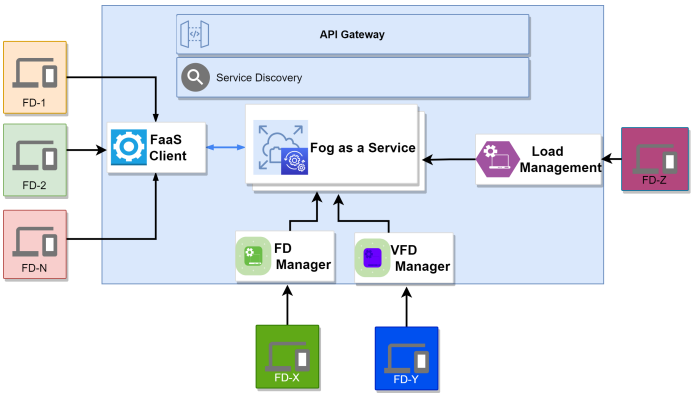


图 5. FaaS 部署

垂直扩展，即在同一雾设备上再创建一个虚拟雾设备。如果主机雾设备的资源利用率大于或等于 X%，则会执行水平扩展，在其他可用的主机雾设备上创建一个新的虚拟雾设备。在场景 1 中，水平扩展不需要迁移虚拟雾设备，因为新虚拟雾设备会创建在已有镜像的雾设备上。在场景 2 中，水平扩展需要迁移虚拟雾设备，以便在

表 1
雾设备配置详情

Fog 设备 Type	配置
台式机	英特尔酷睿 2 双核 E7500 处理器@2.93GHZX2，内存 4 GB，存储 150 GB，操作系统类型为 Ubuntu 64 位
树莓派 PI [1]	CPU：四核 1.2GHz 64 位；RAM：1GB；操作系统：Rasbian
Banana PI [2]	CPU：Realtek RTD1395 ARM Cortex-A53 四核——64 位；RAM：2 GB DDR4 内存，

另一个雾设备。我们将 Scale-X 百分比设为 80 和 90 作为扩容的最大阈值，并将低于 30 的值设为从 [27]– [28] 缩容的最小阈值。我们没有研究低于 Scale-80 的情况，因为大多数设备的资源利用率会偏低，对于规模更大的节点网络而言效率不高 [29]。

5.2 性能指标

我们考虑以下参数来评估所提出的框架 [30]:

1）未满足请求百分比：未能按时处理的请求数量占已处理请求总数的比例。

我们将应用截止期限要求设定为 100 毫秒至 500 毫秒。如果响应时间超过截止期限要求，则将其视为未满足请求。

2）未满足请求的平均响应时间：处理未满足请求所用的时间除以请求总数。3）平均响应时间：处理每个请求所用的时间除以请求总数。

6 结果与讨论

本节通过分析所提出技术在两种不同场景下相较于 Scale-X 技术的效率，讨论该技术的重要性。

图 6 和图 7 总结了不同 CPU负载类型下的性能。图 6 展示了在不同 CPU负载下部署相应数量虚拟雾设备所需的时间。在 Load-90 中，CPU负载在 CPU利用率的 60% 至 90% 之间变化；由于 CPU负载较高，部署时间从 5 个虚拟雾设备增加到 25 个虚拟雾设备。在 Load60 中，部署时间高于 Load-30。由于 CPU负载较高，这是预期之中的结果。

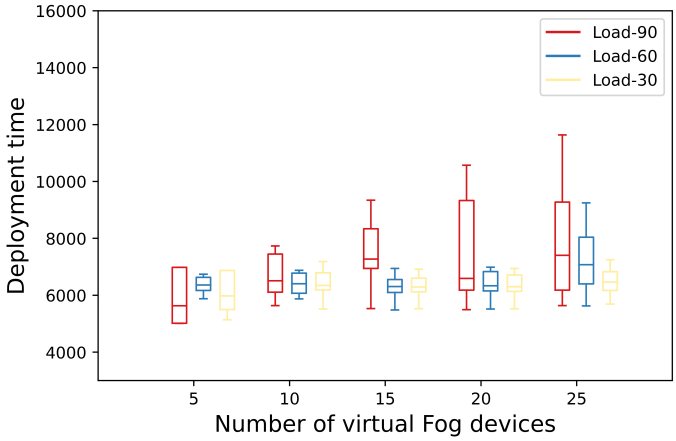


图 6. 虚拟雾设备的部署时间

图 6 展示了在不同 CPU负载下，将一定数量的虚拟雾设备迁移到不同雾设备所需的时间。当 CPU负载从 CPU利用率的 60% 变化到 90% 时，与 Load-60 和 Load-30 相比，将虚拟雾设备从 2 个增加到 10 个会使迁移时间延长。这是高 CPU负载所致。不同虚拟雾设备在 Load-30 下的部署时间都短于 Load-60。这是低 CPU负载所致。

6.1 执行水平扩展时无需迁移

图 8–10 展示了未满足请求百分比、未满足请求的平均响应时间，以及场景1的平均响应时间，如上所述。

图 8 展示了所提出算法在未满足请求百分比方面的效率，并将其与 Scale-80 和 Scale-90 方法进行了比较，比较条件是

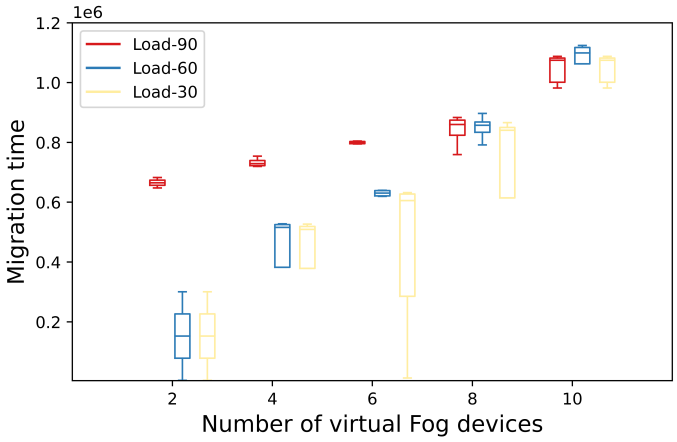


图 7. 虚拟雾设备的迁移时间

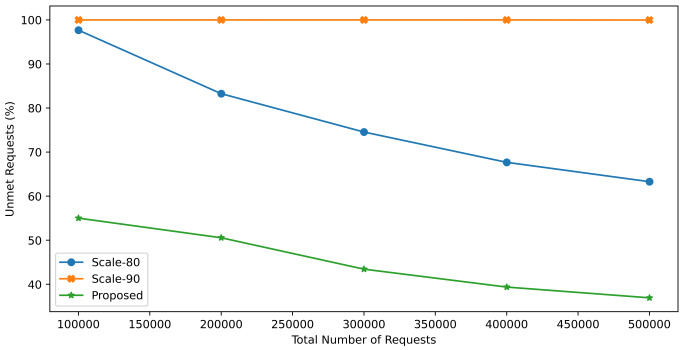


图 8. 不同应用请求的未满足请求百分比（不迁移虚拟雾设备）

请求数量增加时。与 Scale-80 和 Scale-90 方法相比，所提出的机制产生的未满足请求更少，因为该算法能够预测应用请求的数量，并根据需求持续时间预测主机利用率，从而创建并分配虚拟雾设备以部署服务。然而，Scale-80 和 Scale-90 的未满足请求百分比较高，这是因为每个虚拟雾设备承载的请求数量更多，导致响应时间增加，无法满足应用程序的时间要求。此外，所提出的技术和 Scale-80 均呈下降趋势，因为随着 FaaS实例数量增加，请求在各虚拟雾设备之间得到均衡分配。

所提出的算法具有最低的平均响应时间，未满足请求的平均响应时间如图9和图10所示。所提出方法中所有请求的平均响应时间介于543毫秒至710毫秒之间；Scale-80的平均响应时间介于2030毫秒至7454毫秒之间。Scale-90的平均响应时间介于15597毫秒至33148毫秒之间。这是因为我们的方法会预先预测所需虚拟雾设备的数量，并根据可用资源信息，在预测的雾设备上部署FaaS，并在这些设备之间分配请求负载。

然而，与其他方法相比，所提出的方法资源利用率较低。这是因为

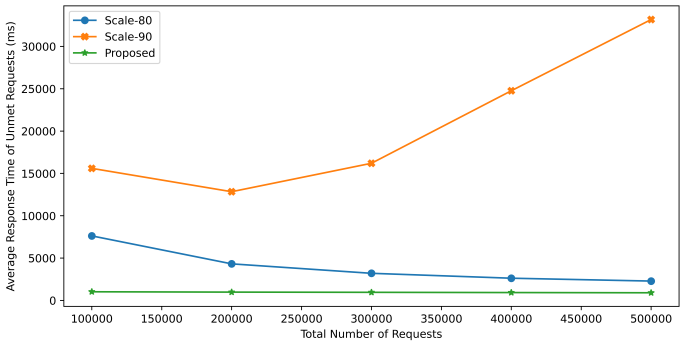


图 9. 不同应用请求的未满足请求平均响应时间（未迁移虚拟雾设备）

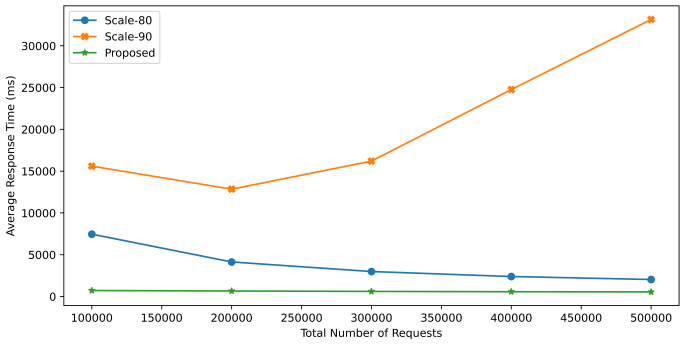


图 10. 不同应用请求的平均响应时间（未迁移虚拟雾设备）

高峰时段的空闲资源不足以及时处理请求。

6.2 执行水平扩展时所需的迁移

图 11–13 展示了场景 2 的实验结果。如图 11 所示，与其他方法相比，我们提出的方法的未满足请求百分比最低。此外，未满足请求的数量为

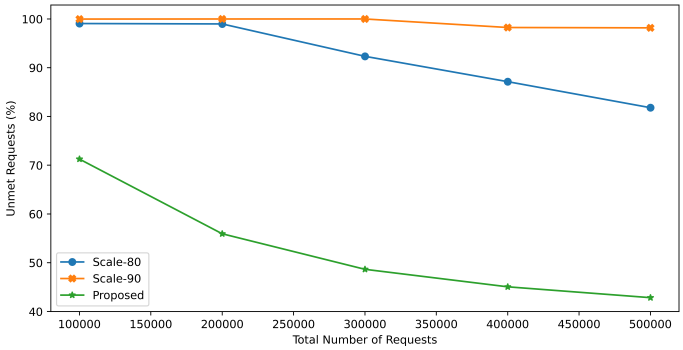


图 11. 不同应用请求的未满足请求百分比（迁移虚拟雾设备）

在 Scale-80 和 Scale-90 中，前 100,000 个请求期间的响应时间非常高，因为雾设备的资源更多地用于创建快照和迁移，导致请求的响应时间增加。

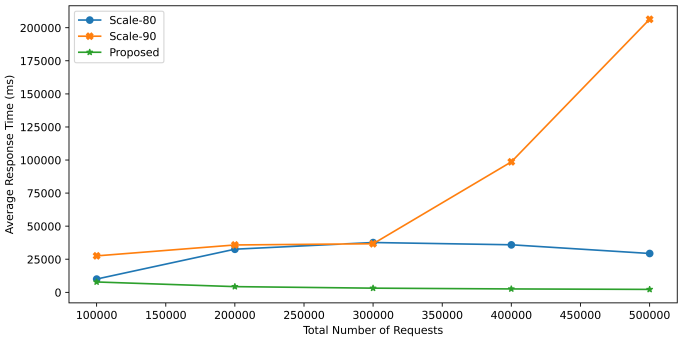


图 12. 不同应用请求的未满足请求平均响应时间（虚拟雾设备迁移）

图 13 显示，所提出的技术和 Scale-80 均呈下降趋势，这是因为随着 FaaS实例数量增加，请求在虚拟雾设备之间得到均衡分配。然而，Scale-90 的趋势基本相同，这是因为每个虚拟雾设备都存在过载请求。由于请求数量增加，且未达到目标资源利用率，系统会将更多请求分配给 FaaS。因此，如图 12 和图 13 所示，Scale-90 的平均响应时间有所增加。

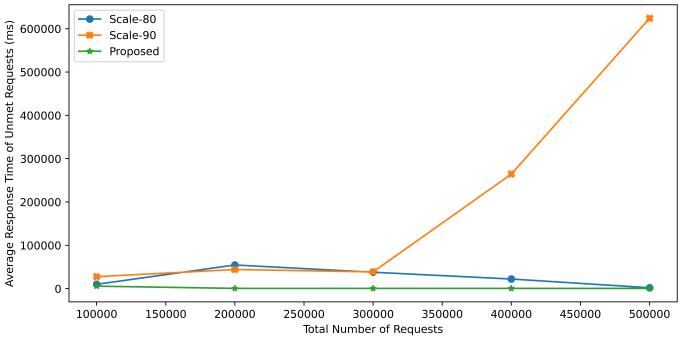


图 13. 不同应用请求的平均响应时间（虚拟雾设备迁移）

与另外两种方法相比，所提出的方法具有最低的平均响应时间。这是因为在我们提出的方法中，所部署的 FaaS 数量根据应用程序的需求（请求的截止期限）以及虚拟雾设备的数量确定。此外，雾设备的选择基于对资源未来可用性的预测，而该预测则基于当前的资源利用率。然而，其他方法只有在达到目标资源利用率时才会创建新的 FaaS 实例。

7 结论与未来工作

在本文中，我们评估了一个基于所提出的 FaaS 框架的系统，用于在雾计算环境中的虚拟雾设备上动态部署 FaaS 服务。我们提出了一种利用设备可用性信息的动态资源分配方法。我们在真实环境中实现并评估了所提出的方法。该方法采用的 SARIMA 模型有助于预测未来的应用请求。结果表明，

与其他方法相比，我们提出的方法往往能减少未满足的请求数量。在场景 1 中，其满足的服务请求数量平均提高了 1.9 倍；在场景 2 中，平均提高了 1.8 倍。最后，作为未来工作的组成部分，我们旨在改进设备注册流程，并计划创建动态集群，以支持大数据应用。我们还计划集成智能软件定义网络（SSDN），以自动控制集群之间的数据量，从而提高大数据流应用的性能。

参考文献

[1] K. Ujjwal、S. Garg、J. Hilton、J. Aryal 和 N. Forbes-Smith，《自然灾害建模系统中的云计算：当前研究趋势与未来方向》，*International Journal of Disaster Risk Reduction*，第 38 卷，文章编号 101188，2019 年。[2] K. Ujjwal、S. Garg、J. Hilton 和 J. Aryal，《用于自然灾害模型敏感性分析的云端框架》，*Environmental Modelling & Software*，第 134 卷，文章编号 104800，2020 年。[3] S. Tammishetty、T. Ragunathan、S. K. Battula、B. V. Rani、P. RaviBabu、R. Nagireddy、V. Jorika 和 V. M. Reddy，《用于协助应急车辆的基于物联网的交通信号控制技术》，载于第一届计算智能与信息学国际会议论文集。施普林格，2017 年，第 433–440 页。[4] S. Tuli、N. Basumatary、S. S. Gill、M. Kahani、R. C. Arya、G. S. Wander 和 R. Buyya，《Healthfog：在集成物联网与雾计算环境中，基于集成深度学习的智能医疗系统，用于心脏疾病自动诊断》，*Future Generation Computer Systems*，第 104 卷，第 187–200 页，2020 年。[5] D. Santoro、D. Zozin、D. Pizzolli、F. De Pellegrini 和 S. Cretti，《Foggy：雾计算环境中的工作负载编排平台》，载于*2017* 年 *IEEE* 云计算技术与科学国际会议（*CloudCom*）。IEEE，2017 年，第 231–234 页。[6] C. Powell、C. Desiniotis 和 B. Dezfouli，《雾计算开发工具包：雾系统的开发与管理平台》，*IEEE Internet of Things Journal*，第 7 卷，第 4 期，第 3198–3213 页，2020 年。[7] H.-J. Hong、P.-H. Tsai 和 C.-H. Hsu，《雾计算平台中的动态模块部署》，载于*2016* 年第 *18* 届亚太网络运营与管理研讨会（*APNOMS*）。IEEE，2016 年，第 1–6 页。[8] P.-H. Tsai、H.-J. Hong、A.-C. Cheng 和 C.-H. Hsu，《使用 TensorFlow 和 Kubernetes 在雾计算平台中进行分布式分析》，载于 *2017* 年第 *19* 届亚太网络运营与管理研讨会（*APNOMS*）。IEEE，2017 年，第 145–150 页。[9] W.-S. Zheng 和 L.-H. Yen，《基于 Kubernetes 的雾计算平台中的自动扩缩》，载于国际计算机研讨会。施普林格，2018 年，第 338–345 页。[10] P. Kayal，《Kubernetes：面向雾计算中分布式物联网应用的部署》，载于*ACM/SPEC* 国际性能工程会议伴随会议，2020 年，第 32–33 页。[11] R. Mayer、L. Graser、H. Gupta、E. Saurez 和 U. Ramachandran，《EmuFog：大规模雾计算基础设施的可扩展仿真》，载于*2017* 年 *IEEE* 雾计算世界大会（*FWC*）。IEEE，2017 年，第 1–6 页。[12] C. de Alfonso、A. Calatrava 和 G. Molt´o，《基于容器的虚拟弹性集群》，*Journal of Systems and Software*，第 127 卷，第 1–11 页，2017 年。[13] P. Bellavista 和 A. Zanni，《基于 Docker 容器化技术在 Raspberry Pi 上部署雾计算的可行性》，载于第十八届分布式计算与网络国际会议论文集，2017 年，第 1–10 页。[14] T. Goethals、B. Volckaert 和 F. De Turck，《面向 Kubernetes 集群实时拓扑变化的自适应雾服务部署》，载于*CLOSER2020*：第十届云计算与服务科学国际会议，2020 年，第 161–170 页。[15] F. Rossi、V. Cardellini、F. L. Presti 和 M. Nardelli，《使用 Kubernetes 高效部署地理分布式容器》，*ComputerCommunications*，第 159 卷，第 161–174 页，2020 年。

[16] R. Eidenbenz、Y. Pignolet 和 A. Ryser，《基于延迟感知的工业雾应用 Kubernetes 编排》，载于*2020* 年第五届雾计算与移动边缘计算国际会议（*FMEC*），2020 年，第 164–171 页。[17] H. R. Arkian、A. Diyanat 和 A. Pourkhalili，《MIST：面向物联网群智感知应用、采用经济高效资源供给的基于雾计算的数据分析方案》，《网络与计算机应用杂志》，第 82 卷，第 152–165 页，2017 年。[18] F. Tseng、M. Tsai、C. Tseng、Y. Yang、C. Liu 和 L. Chou，《面向工业应用的轻量级雾计算自动扩缩机制》，《*IEEE*工业信息学汇刊》，第 14 卷，第 10 期，第 4529–4537 页，2018 年。[19] A. Yousefpour、A. Patil、G. Ishigaki、I. Kim、X. Wang、H. C. Cankaya、Q. Zhang、W. Xie 和 J. P. Jue，《Fogplan：轻量级服务质量感知型动态雾服务供给框架》，*IEEE Internet of ThingsJournal*，第 6 卷，第 3 期，第 5080–5096 页，2019 年。[20] S. S. Nabavi、S. S. Gill、M. Xu、M. Masdari 和 P. Garraghan，《Tractor：在边缘云数据中心中使用人工蜂群优化实现流量感知与节能的虚拟机放置》，《国际通信系统杂志》，文章编号 e4747，2021 年。[21] G. Manogaran 和 B. S. Rawal，《物联网-雾-云架构中采用最优节点放置的高效资源分配方案》，《*IEEE*传感器杂志》，第 1–1 页，2021 年。[22] R. Mahmud、A. N. Toosi、K. Ramamohanarao 和 R. Buyya，《雾计算环境中工业4.0应用的上下文感知放置》，《*IEEE*工业信息学汇刊》，第 16 卷，第 11 期，第 7004–7013 页，2020 年。[23] F. Faticanti、F. De Pellegrini、D. Siracusa、D. Santoro 和 S. Cretti，《雾计算中应用的吞吐量感知型分区与放置》，*IEEE Transactions on Network and Service Management*，第 17 卷，第 4 期，第 2436–2450 页，2020 年。[24] R. N. Calheiros、R. Ranjan 和 R. Buyya，《云计算环境中基于解析性能与服务质量的虚拟机供给》，载于*2011*年并行处理国际会议。IEEE，2011 年，第 295–304 页。[25] J. Zhang、Q. Liu 和 J. Chen，《云环境中的高级负载均衡策略》，载于*2016* 年第 *17* 届并行与分布式计算、应用与技术国际会议（*PDCAT*），2016 年，第 240–243 页。[26] S. K. Battula、M. M. O'Reilly、S. Garg 和 J. Montgomery，《雾计算环境中资源可用性的通用随机模型》，*IEEE Transactions on Parallel and Distributed Systems*，第 32 卷，第 4 期，第 960–974 页，2021 年。[27] J. Bao、Z. Lu、J. Wu、S. Zhang 和 Y. Zhong，《为私有云资源管理平台实现一种新颖的负载感知型自动扩缩方案》，载于*2014* 年 *IEEE* 网络运营与管理研讨会（*NOMS*），2014 年，第 1–4 页。[28] E. Casalicchio，《CPU密集型容器化应用自动扩缩的性能度量指标研究》，*Cluster Computing*，第 22 卷，第 3 期，第 995–1006 页，2019 年。[29] C. Qu、R. N. Calheiros 和 R. Buyya，《云端自动扩缩 Web 应用：分类与综述》，*ACM Comput. Surv.*，第 51 卷，第 4 期，2018 年 7 月。 [在线]。链接：https://doi.org/10 1145 3148149 。/ M. S. Aslanpour、S. S. Gill 和 A. N. Toosi，《云、雾与边缘计算的性能评估指标：面向未来研究的综述、分类、基准测试与标准》，物联网，第 12 卷，文章编号 100273，2020 年。 [在线]。链接：https://www.sciencedirect.com/science/article/pii/S2542660520301062

[30]