arXiv:2601.04123v1 [cs.DC] 7 Jan 2026

# 云边连续体上具有故障韧性和碳效率的微服务部署

Francisco Ponce[1], Simone Gazza[2], Andrea D'Iapico[3], Roberto Amadini[2], Antonio Brogi[1], Stefano Forti[1], Saverio Giallorenzo[2], Pierluigi Plebani[3], Davide Usai[1], Monica Vitali[3], Gianluigi Zavattaro[2], Jacopo Soldani[1*]

[1]比萨大学，比萨，意大利。[2]博洛尼亚大学，博洛尼亚，意大利。[3]米兰理工大学，米兰，意大利。

*通讯作者。电子邮箱：jacopo.soldani@unipi.it；

**摘要**

在异构且动态的云边基础设施上部署基于微服务的应用（MSA），需要平衡相互冲突的目标，例如故障韧性、性能和环境可持续性。本文介绍了FREEDA工具链，旨在实现云边连续体上MSA的故障韧性和碳高效部署。FREEDA工具链会根据不断变化的运行条件、资源可用性和可持续性约束，持续调整部署配置，力求在维持基于微服务的应用质量和服务连续性的同时减少碳排放。我们还介绍了一套实验套件，其中包含多种模拟和仿真场景，用于验证该工具链应对真实世界挑战的有效性，包括资源耗尽、节点故障和碳强度波动。结果表明，FREEDA能够自主重新配置部署，具体方式包括迁移服务、调整规格选择或重新平衡工作负载，从而成功实现韧性、效率和环境影响之间的最佳平衡。

**关键词：**云边连续体、微服务、故障韧性、环境可持续性

## 1 引言

智能互联物联网设备的能力迅速扩展，促使云计算向大型、普及化的分布式环境演进。这些环境必须尽量减少不必要的延迟，同时充分利用网络边缘 [1]可用的计算资源。支撑这一云边连续体的基础设施必然具有高度异构性和动态性。异构性源于所涉及设备的多样性，这些节点的计算和存储能力各不相同，采用不同的部署方式和软件技术，并通过多种协议进行通信。另一方面，动态性（例如，节点加入或离开基础设施、工作负载波动）和不确定性（例如，端到端连接不稳定或发生硬件故障）共同造成了这种多样性和变化性。二者相互叠加，加剧了防止服务质量（QoS）下降和处理故障的挑战 [2]。

与此同时，微服务在提供企业解决方案中的广泛应用，增加了对云边连续体中有效部署策略的需求。微服务应用由多个相互依赖的服务组成，且各自的部署要求不尽相同。这些要求可能包括云边资源租赁的成本约束、特定的硬件和软件依赖、安全性考量，以及严格的服务质量（QoS）要求，例如低延迟、充足的带宽和高可用性。鉴于微服务应用和云边基础设施本身的复杂性，以及边缘节点和服务的不稳定性，必须将故障视为首要关注的问题。工程师不仅必须考虑单个服务或节点故障，还必须考虑级联故障，即一个组件发生故障并传播至其他组件。为确保已部署的微服务应用持续达到其服务质量目标，DevOps 工程师必须主动将这些风险纳入设计考量，并将韧性融入部署和管理策略 [3]。

对弹性应用程序进行部署的需求，正日益与全球对可持续性和环境责任的关注相一致。诸如欧盟“建设气候中和、绿色、公平和社会性的欧洲”战略 [4]，（该战略推动包括信息技术在内的欧洲产业实现环境可持续增长 [5]，）等倡议，反映了这一更广泛的趋势。因此，在云边基础设施上部署应用程序时，环境可持续性正成为一项关键考量，例如寻求最大限度地减少已部署微服务应用的碳足迹。

然而，可持续发展目标可能与其他部署要求相冲突，例如故障韧性。例如，通过整合服务（即在彼此距离较近的节点上，甚至在同一节点上部署每项服务的单个实例），可以减少碳排放。然而，这种方法会削弱故障韧性。反过来，在多个地理分布式节点上复制服务能够增强韧性，但会增加能耗和碳排放。这种权衡凸显了在大型、异构且动态的云边环境中部署微服务应用的复杂性。因此，有效的部署支持必须平衡多项且往往相互冲突的需求，包括可持续性、韧性、成本、安全性和服务质量。此外，还必须考虑云边基础设施随时间变化的情况（例如，节点加入、离开、发生故障或变得过载）。对于已经部署的微服务应用而言，这一点尤为关键，因为它们经常更新，并通过 CI/CD 流水线持续交付。

已有多项工作探索了应用程序放置策略，以降低能耗或碳排放 [6–8]。然而，只有少数工作涉及自适应部署，即根据不断变化的上下文、目标或工作负载，动态调整应用组件及其放置方式。例如，Forti 和 Brogi[9] 提出根据操作员偏好和成本目标，将应用组件部署为不同的功能等效变体，并采用贪心策略来最小化运营开支。然而，他们的方法未纳入可持续性或碳感知。其他研究 [10，11]利用微服务架构中的变体来调整工作流并减轻环境影响，但未解决异构分布式基础设施中的部署挑战。

为此，FREEDA 研究项目[12] 旨在帮助 DevOps 团队管理高度异构且动态变化的环境。其主要目标是解决如何在云边基础设施上部署基于微服务的应用程序，同时实现故障韧性、能量感知和可解释性。FREEDA 可自动选择最优的组件变体，并在云边连续体中为其安排可行的部署位置，同时考虑依赖关系、拓扑、资源可用性、成本和可持续性约束。其首要目标是确定满足应用需求的部署配置，同时确保碳效率和故障韧性。

其基本理念是，在云端—边缘部署中支持组件变体，对于实现高水平的自动化和灵活性至关重要。变体表示组件的功能等效实现，它们在性能、资源需求或碳足迹方面存在差异。利用这些替代方案，运营人员可以在满足特定运行约束（如碳预算、成本或韧性要求）的同时，优化应用程序性能。

本文介绍了FREEDA工具链，该工具链能够自动完成组件变体的最优选择及其可行部署位置，并持续根据不断变化的运行条件、资源可用性和可持续性约束调整部署配置。FREEDA工具链还负责管理应对故障或提升整体碳效率所需的重新配置。事实上，当面对大量等效的调整方案时，需要变更的组件越多，部署调整所带来的开销就越高（服务停机时间、性能下降以及可能出现的不稳定性）。为此，我们在最新技术水平的基础上进一步改进 [13]现有的FREEDA模型，该模型是FREEDA工具链的基础；具体而言，我们引入了用于最小化部署变更的新约束（等价于最大化系统中保持不变的部分）。

我们还提出了一套实验套件，用于验证 FREEDA 工具链的有效性。该套件通过多种受控的仿真和仿真场景，测试 FREEDA 应对现实挑战的能力，包括资源耗尽、节点故障、碳强度波动，以及性能与可持续性之间的多目标权衡。结果表明，FREEDA 能够自主重新配置部署——通过迁移服务、调整规格选择或重新平衡工作负载——在韧性、效率和环境影响之间实现最佳平衡。

本文其余部分安排如下。第2节回顾并扩展基于约束的部署模型。第3 节概述 FREEDA 工具链。第4节介绍并讨论仿真结果，而第5节重点介绍仿真结果。最后，第6 节和第7 节分别回顾相关工作并给出结论。

本文扩展了我们之前的工作 [13, 14]，在之前的工作中，我们首次提出了约束模型 [13]以及 FREEDA 故障增强器背后的理念[14]。本文扩展了部署模型（第 2 节），并详细介绍了完整的FREEDA工具链（第3 节）。此外，本文还提供了全新的实验（第4 和 5 节），以评估其有效支持 FREEDA 能否支持在云边基础设施上对微服务应用进行可持续且具备故障韧性的部署。

## 2 部署模型

***FREEDA*模型**

微服务应用将应用程序分解为多个软件组件。然而，除了需要处理组件的部署之外，为适应不同的部署上下文，还必须提供组件的多个版本（即变体 [13, 15]），每个版本都具有特定的功能属性和非功能属性。因此，部署此类应用程序涉及两个相互依赖的决策：(i) 为每个组件选择合适的变体；(ii) 将这些组件映射到异构基础设施中的各个节点，这些基础设施涵盖从强大的云端到资源受限的边缘/物联网节点。

部署必须同时考虑多种需求和目标，包括延迟、带宽和可用性等服务质量约束；预算和成本效率等运营考量；以及日益紧迫的环境因素，例如能耗和碳排放。此外，还必须遵守组件之间的功能依赖关系以及基础设施限制。探索这一庞大且多维的解空间在组合层面复杂度极高，若没有系统化的自动支持，通常难以实现；尤其是考虑到竞争目标之间可能产生的冲突。

为应对这一挑战，我们在先前的工作中 [13], 提出了一种约束优化模型，为 FREEDA 框架奠定了数学基础。该模型旨在将高层部署规范——涉及应用组件、其变体、部署要求和基础设施特征——转化为严谨的优化问题，其解对应于可行部署。具体而言，该模型会同时：(i) 为每个组件选择相应的变体，并将其分配给一个节点；(ii) 确保符合 FREEDA YAML 规范 [16], 中规定的所有部署要求，包括能量和碳预算；以及 (iii) 在可能的情况下，优先部署

最强大的变体，以符合应用所有者的偏好。该模型的完整形式化规范 [13,第 4] 分为四个关键部分——参数、变量、约束和目标函数，下面的段落将对其进行非正式介绍。

参数为每个部署实例提供输入数据，其中包含应用组件及其对应的变体、组件之间的依赖关系、资源需求（包括中央处理器、随机存取存储器和存储等消耗性资源，以及可用性或安全性等非消耗性资源）、基础设施规格（例如节点容量、链路延迟和可用性，以及每种资源的货币成本和碳成本），以及支出和排放的预算阈值。每种变体都有一个重要性值：该值越高，变体的能力越强。

变量将部署决策编码为二进制指示变量 $D^c_{i,j} \in \{0,1\}$ 其中 $c$、$i$ 和 $j$ 分别表示基础设施中的组件、（属于$c$的）变体和节点。约束求解器当且仅当 $c$以变体 $i$ 部署在 $j$上时，才将 $D^c_{i,j} = 1$ 设为真。这种表示方法与求解器无关；事实上，多种求解技术都可以执行该模型，包括约束规划（CP）、混合整数线性规划（MIP）或布尔可满足性（SAT）。

约束对部署的有效性条件进行形式化定义。例如，它们确保每个组件最多部署一次（以某种变体部署在某个节点上），必需组件始终得到部署，所需依赖项由性能足够的变体满足，且非必需组件不会单独部署。资源约束确保总消耗不超过节点容量，而网络链路约束确保满足延迟和可用性需求。预算约束则确保符合财务和碳排放限制。

目标函数旨在最大化已部署变体的重要性，而非直接降低成本或排放，从而避免出现没有部署任何组件来减少开支的“空部署”。预算作为硬约束予以执行，同时优化会在可行的情况下优先考虑重要性更高的变体。用户可以手动定义某个变体的重要性，或从一组内置的重要性策略中进行选择。

最小化部署变更

在这项工作中，我们通过聚焦于减少连续部署之间的变更，推动了最新技术水平的发展 [13]。实际上，虽然我们需要根据不断变化的条件（如货币和能源预算，以及软件和基础设施故障）调整微服务应用，但尽量减少连续部署之间的变更数量，对于维持整体系统可用性（例如减少停机时间）、性能和稳定性至关重要。这一附加目标为求解器引入了新的最小化准则。具体而言，当有人请求新部署时，求解器会被指示尽量减少切换变体或迁移到其他节点的组件数量，并以当前部署为参照。我们通过跟踪当前配置中已部署的每个组件来体现这一特性（即满足以下条件的组件：$_cD^c_{i,j} = 1$对于某种变体 $i$ 和节点 $j$），并在可能的情况下，力求在后续配置中保留其部署。换言之，如果$D^c_{i,j} = 1$在当前部署中成立，求解器便会尝试在新部署中也强制满足$D^c_{i,j} = 1$ 。

形式上，这一要求对应于最大化在同一节点上仍以相同风味部署的组件数量，即：

$$\max \sum_{\bar{D}^c_{i,j}=1} D^c_{i,j} \tag{1}$$

其中，$D$ 表示从当前部署中已计算出的决策变量值矩阵，$\bar{D}$ 表示新部署的矩阵。风味变更和节点迁移被视为等价操作，因为这两种操作最终都需要重新部署该组件。

**待定：** 关于重新部署中的退化情况的评论（此处或在讨论部分）

# 3 FREEDA工具链

FREEDA 旨在满足 DevOps 对部署微服务架构的支持需求 A，将其部署在云边基础设施上I，并遵循一组部署要求 R，如图1所示。

基础设施描述 I 包含资源利用成本、节点的硬件和软件能力，以及节点当前负载和可用性等信息。相比之下，需求 R 规定了组成 A 的服务所需满足的条件，包括硬件和软件依赖项、网络服务质量、安全性和故障韧性。此外，R 还可以定义部署预算，包括货币预算（即可负担的最大成本）和可持续性相关预算（即允许的最大能耗和碳排放）。

FREEDA工具链会生成部署计划 D，用于 A 在I 上，通过全面识别 R中各项需求之间的权衡。如图 1所示，所提出的解决方案分为两个主要阶段（增强 和 权衡），每个阶段都由两个主要组件提供支持。第一阶段利用历史数据 H，提升应用程序的故障韧性和环境可持续性；这些历史数据包括 A 当前及过往部署的信息（例如日志），以及来自 I的监控数据（例如节点可用性或负载）。此步骤会生成一组增强后的需求 R+，其中纳入了碳感知和故障韧性方面的考量。这种自动化增强可确保应用程序部署满足这些非功能性方面的要求，无需直接干预，从而减少 DevOps 工程师所需投入的设计工作。

在第二阶段，权衡分析工具会处理 A 和 R+ ，生成部署计划 D 及其解释 E。该解释阐明了为何 D 能够在多项部署需求（这些需求可能彼此冲突）之间实现最佳权衡。重要的是，E 还记录了增强过程背后的推理，详细说明 R+ 如何以及为何被转换为 r+。这样可确保 DevOps 工程师不仅了解最终部署计划，也理解对应用规范及其需求所做任何调整的依据。

利用这些输入，FREEDA 将为 DevOps工程师提供一个有效的部署方案，以满足所有已表达的约束。该方法的核心是多准则求解器组件。在此背景下，解释 E 对应于约束模型本身；该模型在由多准则求解器执行后，将生成最优部署方案 D。

以下介绍图1所示 FREEDA 工具链的各个组件。

## 3.1 故障增强器

故障增强器负责在R 中生成软部署要求，以在部署前提升微服务的韧性。例如，这些要求有助于避免将服务部署到已知在特定负载条件下会发生故障的节点，或根据可用历史数据预测其即将发生故障的节点上。如[14], 所述，故障增强器会生成三类约束：亲和性、反亲和性和避免。 亲和性约束建议将组件C 和S 以当前风味部署在更近的位置，例如同一节点上。 反亲和性约束建议避免将组件C 和S 以当前风味放置在同一节点上。而避免约束则建议避免将组件C 以当前风味FC 放置到节点N上。图2展示了用于生成上述软约束的 Prolog 子句，接下来将对其进行讨论。

知识表示

当前的微服务架构部署信息通过 Prolog事实表示，例如 deployedTo(C,F,N)，表示组件C 以其风味F 部署到节点N。从微服务架构失败的角度来看，故障增强器假设组件C1与C2之间的超时事件通过带时间戳的事实表示，例如 timeoutEvent(C1,C2,Timestamp)。此外，组件C 的内部错误和不可达性分别通过类似 internal(C,Timestamp) 和 unreachable(C,Timestamp)的事实表示。

另一方面，考虑基础设施日志记录，谓词 congested(N,M,T)用于标识节点 N和M之间在时间 T 发生了链路拥塞。同样，如果节点 N在时间 T发生网络断连，则谓词disconnected(N,T)为真。谓词 overloaded(N,R,T)表示特定资源 R （例如，随机存取存储器、中央处理器、硬盘驱动器）在时间 T处于过载状态。最后，谓词 race(N,R,C,FC,S,FS,T)表示在时间T ，组件C （风味为 FC）和 S

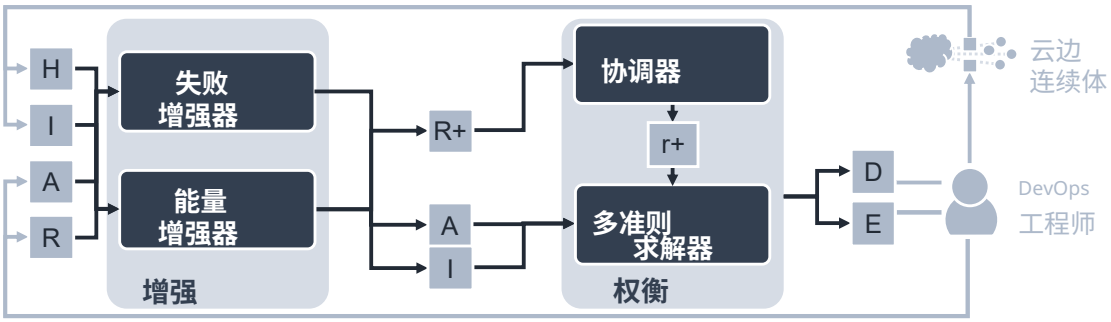


**图 1：** FREEDA 工具链概览

```
1  suggested(亲和性(d(C,FC),d(S,FS))) :-
2      deployedTo(C,FC,N),    deployedTo(S,FS,M),    dif(C,S),    dif(N,M),
3      timeoutEvent(C,S,T),
4      \+( 拥塞(N,M,T); 断开连接(N,T); 断开连接(M,T) ).

5  suggested(避免(d(C,FC),N)) :-
6      部署到(C,FC,N),    部署到(S,_,M),    dif(C,S),    dif(N,M),
7      timeoutEvent(C,S,T),
8      ( congested(N,M,T); 断开连接(N,T) ).

9  suggested(反亲和性(d(C,FC),d(S,FS))) :-
10     部署到(C,FC,N),
11     ( 不可达(C,T);          内部(C,T) ),
12     过载(N,R,T),            竞争(N,R,C,FC,S,FS,T).

13 suggested(避免(d(C,FC),N)) :-
14     部署到(C,FC,N),
15     ( 不可达(C,T);          内部(C,T) ),
16     ( (过载(N,_,T),           \+ 竞争(N,_,C,FC,_,_,T)) ;    断开连接(N,T) ).
```

**图 2：** [14] 中展示的故障增强器 Prolog 子句

（风味为 FS）在同一节点 N上竞争资源 R 。

### 增强故障韧性

基于以上内容，故障增强器通过嵌入用于改进部署决策的经验法则，生成一组建议的软约束，以提升当前部署的韧性。具体而言，它会检查所考虑知识库中哪些谓词子句suggested/1 会触发，从而找出所有不同的建议约束。这里我们讨论图2中展示的此类谓词的四个子句。不过，值得注意的是，这些子句可以轻松扩展或细化，以应对更多微服务架构故障和/或网络条件。

suggested/1 （第 1–4 行）的第一个子句指出，不同组件C（变体为FC）与S （变体为FS）之间发生了一次交互，它们分别部署在两个不同的节点N 和M上（第 2 行），发生了超时事件（第 3 行），尽管没有发生拥塞或断连事件（第 4 行）。为避免再次发生这种情况，故障增强器建议将C和S 部署得更近一些，即为这两个组件当前的变体添加亲和约束，具体为affinity(d(C,FC),d(S,SF))。

suggested/1 的第二个子句（第 5–8 行）指出，在时间T，部署在节点N上的FC 风味的C 组件发生超时事件，并涉及部署在另一个节点M 上的S 组件（第 6 行），可能是由N与M 之间的网络拥塞，或节点N断连所致（第 8 行）。故障增强器建议不要将C （风味为FC）部署到节点N上，方法是加入节点规避约束，即avoid(d(C,FC),N) （第 5 行）。事实上，该链路

节点N与M之间可能持续发生拥塞或故障。另有一个对称子句（未显示），用于处理对称情形（即节点M发生拥塞或断连），方法是建议加入 avoid(d(S,FS),M) 约束。

suggested/1 的第三个子句（第 9–12 行）指出，部署到节点 N （第 10 行）的组件 C（采用 FC 变体）在时间 T（第 11 行）要么无法访问，要么正经历内部错误。由于另一个组件 S （采用 FS 变体）与 C 竞争资源 R （第 12 行），该失败事件与节点N对资源 R 的过载重叠。故障增强器建议通过在采用当前变体的两个组件之间设置反亲和性 约束，避免将C 和S 放置在同一节点上，即antiaffinity(d(C,FC),d(S,SF))（第 9 行）。

suggested/1 的第四个也是最后一个子句（第 13–16 行）指出，部署到 N 、采用 FC 变体的组件C （第 14 行）经历了一次故障事件（第 15 行），这可能是由于节点过载（不存在竞争）或断连（第 16 行）造成的。故障增强器建议通过加入节点规避约束，避免将C （采用FC 变体）放置到节点N 上，即avoid(d(C,FC),N)（第 13 行）。

## 3.2 能耗增强器

能耗增强器与故障增强器协同工作，在 R 中生成旨在降低微服务应用执行环境影响的软部署要求。

能耗增强器是所提出解决方案的关键组件，负责支持在云端连续体上实现高能效、环保的应用程序部署。该组件会接收多种输入，包括应用描述 （即待部署的服务）、基础设施描述 （即可用的计算节点）、当前的电网碳强度，以及监控指标 （用于捕捉应用程序的运行时行为）。基于这些输入，该组件会生成一组约束，为多准则求解器生成合适的部署计划提供信息和指导。此外，能耗增强器会通过从以往部署中反复学习，持续改进其输出，使系统能够

适应应用行为和基础设施条件的变化。

该解决方案融合了多项协同工作的关键功能，用于生成和优化环境感知的部署约束。首先，系统持续收集并处理有关底层基础设施碳强度的信息。系统不依赖瞬时数值，而是考虑最近观测窗口内的聚合测量结果，从而为决策提供更加稳定且具有代表性的数据。

与此同时，系统通过分析历史监控数据，估算与应用服务及其服务间通信相关的碳足迹。这使系统能够利用能耗配置丰富初始部署描述，涵盖计算能耗和通信相关能耗。

利用这些丰富后的输入，该解决方案推导出能够反映当前应用行为和基础设施条件的部署约束。这些约束依据预先制定的模板和规则定义，并可根据需要扩展，以适应新类型的绿色感知策略。通过这种方式，系统能够保持灵活性和适应性，以应对不断变化的可持续发展目标。

为确保生成的约束建立在已有知识的基础上，系统会检索、完善并随时间逐步更新先前学习的信息。这一持续学习过程有助于在多个部署周期中保持一致性，避免丢失以往配置中的有用见解。

鉴于可能出现大量约束，该方案采用排序和过滤技术，仅保留预计对能源效率和碳排放影响最大的约束。这一步优先级排序可确保最终的约束集既相关又易于管理。

知识库

为了使部署决策更加有效，约束的生成不应仅依赖最新的监控数据，还应以长期积累的知识为基础。为此，该方案维护了一个专门的知识库，用于存储不同类型

的信息，这些信息涉及应用程序、它们之间的通信、底层基础设施以及先前生成的约束。

知识库保存了每个应用服务在能耗方面的历史记录。由于服务的能耗配置可能因部署位置和方式不同而有所差异，系统会维护其环境足迹典型范围的信息，包括根据过去部署期间收集的监控数据观测到的最小值、最大值和平均值。此外，系统还会存储服务间数据交换的碳影响信息。通过分析历史通信模式，系统会建立一份配置文件，反映不同部署中各项服务交互所产生的环境成本。知识库还包括基础设施节点碳强度的历史数据。由于节点的环境特征可能随时间变化，这些记录提供了典型排放水平，可用于在未来部署期间做出更明智的放置决策。

除了原始监控数据，知识库还会保留先前生成的约束。每项存储的约束都包含生成时对其环境影响的估算，以及反映其当前相关性的记忆权重。如果某项约束在多次迭代中未被重新生成，该权重就会降低，从而确保过时信息的影响逐渐减弱。

知识库通过持续纳入新采集的数据和近期生成的约束不断丰富，同时更新旧约束的相关性。在每次迭代中，系统通过观察当前可用数据生成新的部署约束，并检索有效的历史约束作为补充。随着时间推移，结合最新观察结果与积累的知识，系统能够生成更准确、更符合上下文且更可持续的部署决策。

### 增强环境可持续性

能量增强器会生成两种约束：亲和性 和避免。这些约束的含义与故障增强器生成的约束相同，但适用条件不同。

避免 约束与单个服务的能耗有关，旨在限制可能导致高能耗或高排放的部署。如果已知某个服务在特定配置下会在给定节点上消耗过多能量，系统就会生成建议，避免采用该特定部署。亲和性 约束针对服务交互造成的能量开销。如果两个服务交换大量数据，将它们部署在不同节点上可能会产生显著的通信能耗。在这种情况下，系统建议将这些服务共置，以减少此类开销。

这两类约束背后的逻辑可以用 Prolog 子句表示：

```
suggested(avoid(d(C,FC), N)) :-
highConsumptionService(C, FC, N).
suggested(affinity(d(C,FC), d(S,FS))) :-
dif(C, S),
highConsumptionConnection(C, FC,S, FS).
```

第一个子句会在监控数据表明某些部署能耗过高时，建议避免将特定服务风味组合部署到某些节点上。避免 约束的生成依赖于知识库中保存的历史信息。对于每种可能的服务、风味和节点组合，系统都会评估部署该特定配置是否会导致能耗或碳排放过高。系统将服务的能耗概况与节点的碳强度结合起来，并将结果与预定义阈值进行比较，以此作出判断。只要估算影响超过阈值，系统就会生成一项避免 约束，建议避免将该服务风味组合部署到相应节点上。这样可以确保识别出环境代价高昂的部署位置，并将其排除在部署规划流程之外。

第二条规则会针对通信模式可能导致高碳足迹的情况，生成建议，将两个交互服务部署在同一节点上。亲和性 约束的生成基于知识库中保存的历史信息。对于源服务、其风味和目标服务的每种潜在组合，系统会评估它们之间的交互是否相关

```
1 反亲和性(frontend,large,load_balancer,large). 2 #故障增强器生成的约束。 3 affinity(frontend,large,load_balancer,large). 4 #能耗增强器生成的约束。
```

**图 3：** 增强器生成的软约束冲突示例。

能耗较高。通过分析其通信的能耗概况，并将其与预定义阈值进行比较来完成评估。如果估算的通信成本超过该值，系统就会生成一个亲和性 约束。这些约束会引导调度器将相关服务放置在同一节点上。这两种约束共同构成了实现更绿色部署的基础。

## 3.3 协调器

协调器组件以应用程序需求的增强描述 R+ 作为输入。其主要功能是识别并解决软需求之间可能存在的冲突，并依据 DevOps 工程师设定的优先级进行处理，无论其侧重于韧性、可持续性，还是二者之间的平衡。处理完成后，协调器会生成 r+，即经过优化的需求子集（有时也可能是完整需求集），随后将其转交给多准则求解器，以供下一部署阶段考虑。

图3 展示了由故障增强器和能耗增强器生成的软约束相互冲突的情况。在此示例中，故障增强器建议将前端（*Large*规格）和负载均衡器（*Large*规格）服务部署在不同节点上，以通过反亲和性提高容错能力。相反，能耗增强器建议将这两项服务部署在同一节点上，即采用亲和性约束以尽量减少碳排放。发生此类冲突时，协调器会根据 DevOps 工程师设定的优先级进行处理：若优先考虑故障，则舍弃亲和性约束；若优先考虑能耗，则移除反亲和性约束。若未明确指定优先级，则后续部署会忽略这两项冲突的约束。

需要注意的是，协调器并不具备对微服务架构的完整视图。如前所述，它的职责仅限于验证两个增强器生成的软约束是否相互一致。如果生成的软约束与其他基础设施需求冲突，例如，对唯一能够承载关键组件的节点施加避免约束，协调器无法直接检测或纠正此类冲突。在这种情况下，如果多准则求解器判定采用协调器提供的所有约束进行部署无解，则会多次重新执行该问题。关于多准则求解器及其工作原理的详细说明，请参见第 3.4节。

## 3.4 多准则求解器

根据对 FREEDA 框 [13]架及其模型（参见第2节）的说明，应用程序A和基础设施I将转换为 MiniZinc [17]数据文件。MiniZinc 是一种高级、与求解器无关的建模语言，用于表达约束满足和优化问题。它使用户能够以标准化形式定义模型，从而无需修改即可在多个求解器上执行。通常，MiniZinc 模型在专用模型文件中指定，而问题专属数据则在单独的数据文件中提供。不过，也可以根据需要将模型和数据合并到同一个文件中。

如图4所述，配置文件A和I 根据增强器先前调用期间收集的数据进行更新（即从第二次调用多准则求解器开始，修订每个组件的能量值，从配置中移除故障节点，并相应更新资源使用情况和可用性）。协调后分析器生成的新约束与更新后的配置文件随后会重新转换，以将这些更改反映到新的 MiniZinc 模型中。该新模型采用公式（1）中所述的目标函数，最大限度地减少节点之间的组件迁移，从而提高求解器连续调用期间的部署稳定性。

此时，求解器已掌握微服务架构的完整视图，即每个组件的需求、每个节点的能力，以及（如有）已由协调器协调的约束。

如第 3.3 节所述，协调器提供的约束必须与 FREEDA 模型定义的约束相结合 [13]。如果没有其他约束，求解器会尝试根据 FREEDA 模型定义的标准计算最优部署 [13],，即最大化分配给每个已部署组件的变体的重要性。在初始仿真轮次中通常就是这种情况，因为此时尚无先前的部署数据。如果无法找到可行的部署方案，求解器将返回不可满足，工具链随即终止，并明确告知用户，给定的应用程序和基础设施不存在有效的部署方案。反之，如果找到可行的部署方案，则会将该解作为最佳部署计划传递至工具链的下一阶段 D。

然而，当协调器提供额外约束时，先验合并并不能保证部署一定可行，因为一般情况下，约束之间的交互难以预测。

因此，求解器会尝试求解包含所有约束的问题，以评估这些约束能否同时得到满足。如果找到可行的部署，求解器便会将其传递给工具链的下一阶段。否则，每次移除一个约束，再尝试寻找可行的部署。如果仍未找到解，则移除约束对，然后是约束三元组，依此类推，直到获得部署。一旦找到可行的部署，该过程便会停止，并将所得部署作为有效部署用于下一轮仿真。如果在所有组合中都无法找到有效部署，工具链便会终止，并告知用户未找到满意的部署。

# 4 仿真

本节概述整个仿真工具链，随后在下文各小节中详细介绍每个步骤。

如图 4所示，工具链始于 *YAML* [18] 基础设施和应用程序的描述，这些描述作为输入提供给多准则求解器。这些文件按照[13], 中所述的方法进行解析，转换为 MiniZinc 数据文件。多准则求解器生成部署计划，随后将其转换为 *ECLYPSE* [19] 配置。

*ECLYPSE* 仿真分多个轮次执行，每轮结合一个基础设施场景和一个应用场景。每个仿真轮次中的场景用于模拟故障或能量突增。应用场景通常会增加特定组件的工作负载或需求，而基础设施场景则会减少某些节点的可用资源。这种方法能够在广泛的运行条件下进行系统化、受控的实验。

每轮结束后，仿真日志由故障增强器和能耗增强器共同处理，它们会生成额外的约束并更新 YAML 文件（添加新的能量值，并更新每个基础设施节点的可用性），以改进部署。随后，这些约束会传递给协调器，由其根据用户偏好，优先考虑故障韧性或能源效率，从而解决眼前的不一致问题。之后，协调后的约束会输入多准则求解器，由其将这些约束纳入下一次部署，以适应场景所导致的基础设施或应用程序变化。该过程是迭代进行的，每个新的仿真轮次都在更新后的部署上执行。

## 4.1 仿真设置

应用程序和基础设施描述为了运行仿真，我们考虑一种参考架构，如图5所示。图中，节点代表应用程序组件，箭头表示各组件之间的依赖关系，每个节点上方标示了该组件可用的变体。所有组件都被视为可部署在不同机器上的服务。该应用程序由多个组件构成，代表了大型微服务应用中的一大类 [20]。具体而言，负载均衡器 会将传入的客户端请求均匀分配到多个前端服务实例上。前端 与用户交互，并通过 API 服务将请求转发给后端服务。接着，*API*服务充当业务逻辑中心，

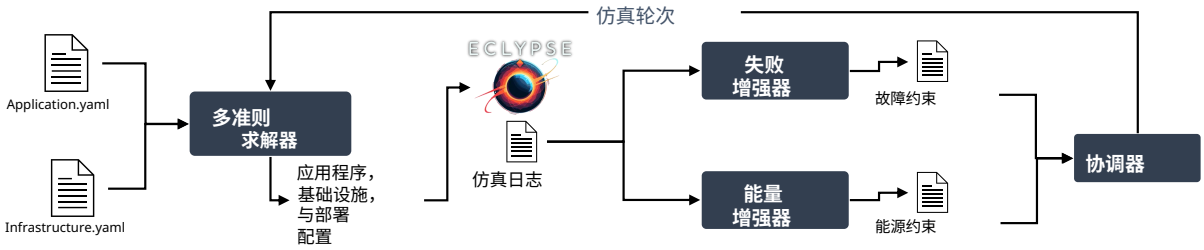


**图 4：** 工具链概览

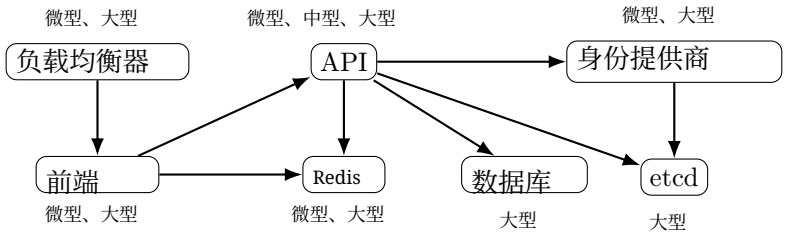


**图 5：** 展示各组件可用变体的应用程序概览。

处理来自前端的请求、实现核心应用功能，并协调对其他后端组件的调用。其中，*Redis* 充当缓存层，用于存储可快速访问的临时数据，有助于缩短响应时间并减少频繁的数据库查询；而数据库 则管理应用程序所需的持久化数据存储。最后两个组件是身份提供商，负责处理用户身份验证和授权；而*etcd* 负责管理分布式配置和服务发现。在变体方面，负载均衡器、前端、Redis 和身份提供商均可采用微型或大型规格部署；etcd 和数据库仅提供大型规格；API 组件则提供微型、中型和大型三种规格。应用程序规范通过 YAML [18]文件定义，格式遵循 [13]中所述。应用程序的完整描述可在配套代码仓库中找到 [21],，该规范使用 FREEDA YAML 规范定义 [16]。

与参考架构类似，我们为仿真定义了一种典型的基础设施。具体而言，我们考虑一种边缘云基础设施，其网状拓扑高度互联，节点之间具有多条冗余路径，可提供韧性和负载分配能力 [22]。该基础设施如图所示 6所示，其中节点代表可部署组件的物理节点。节点可以是公共节点或私有节点，因为某些服务只能部署在这两类节点中的一类上。箭头表示各节点之间的物理连接；两个相互依赖的组件需要部署在物理连接的两个节点上。该基础设施中包含节点*Public1* 和*Public2*，它们是彼此连接的云节点，支持云内直接通信。私有节点（共 5 个）构成一个完整的子网，其中包含彼此连接的本地边缘设备。请注意，公共节点要与部署在本地私有子组中的服务直接通信，唯一途径是通过节点 *Private1* 和*Private2*。

此外，每个节点都有一个子网属性，因此只有属性集匹配的服务才能部署到相应的节点上。例如，每个节点的子网属性可以是私有或公共（取决于节点类型），且每个组件也具有此属性，因此只能部署到特定节点上；应用程序的完整描述见 [21]。由于篇幅所限，这些属性的完整描述请参阅配套代码仓库 [21]中的补充材料。

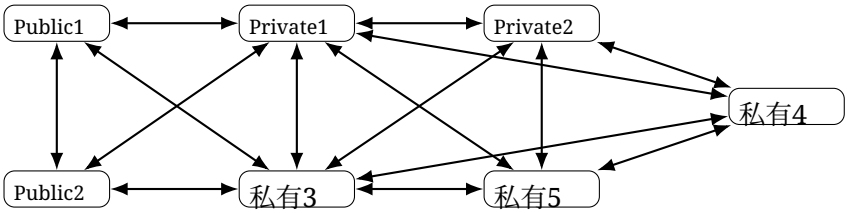


**图 6：** 带连接的基础设施节点 s

多准则求解器

如图4所示，多准则求解器接收应用程序和基础设施的 YAML 规范作为输入（可以是初始值，也可以是故障增强器和能耗增强器在每个仿真轮次后更新的值），以及协调器生成的约束。首先，YAML[18] 规范文件会经过解析阶段，生成 MiniZinc [17] （2.8.5 版）表示，并使用 [13]中提供的基础约束执行。从高级 YAML 规范解析为低级 MiniZinc 规范的详细信息见 [13]。

根据第 3.4 节的规定，求解器（从 MiniZinc 发行版中提供的求解器里选取，默认为我们实验中也使用的 Gecode 6.3.0）旨在根据仿真轮次所选定的标准，在 5 分钟的时间限制内找到可行时的最优部署（即在仿真的第一轮中最大化每种风味的重要性，或在后续轮次中尽可能减少需要迁移的组件）。计算完成后，结果部署会写入一个文本文件，随后解析为*ECLYPSE* 仿真对象。

故障增强器

此组件以从 *ECLYPSE* 生成的仿真日志作为输入，并按照第3.1节中的说明，生成亲和性、反亲和性或避免类型的约束。图4展示了仿真期间 FailureEnhancer 的工作流。该过程从 *Simulation.log* 文件开始，该文件由 *ECLYPSE* 模拟器生成，其样例如图 7所示。该日志由名为 *ECLYPSE* Parser 的组件处理，该组件提取相关信息并生成 Failure 所需的 Prolog事实

1 17:20:33 |ECLYPSE |仿真 - 事件 Start-0 已触发。 2 17:20:33 |ECLYPSE |仿真 - 事件 Tick-0 已触发。 3 17:20:33 | ECLYPSE |仿真 - 事件 Enact-0 已触发。 4 17:20:33 |ECLYPSE | 部署管理器 - 5 在 case_study 上部署 case_study 6 17:20:33 | ECLYPSE |部署管理器 - 7 {load_balancer_大型 ->公共1 |8 API_大型 ->私有1 |9 前端_大型 ->公共1 |10 Redis_大型 ->私有 3 |11 身份_提供商_大型 ->私有3 |12 数据库_大型 ->私有5 |13 etcd_大型 ->Private1} 14 17:20:33 |ECLYPSE |仿真 - 事件 Tick-1 已触发。 15 17:20:33 |ECLYPSE |仿真 - 事件 Enact-1 已触发。 16 17:20:33 |ECLYPSE |部署管理器 - 17 在 case_study 上部署 case_study 18 17:20:33 |ECLYPSE |部署管理器 - 19 {load_balancer_大型 ->公共1 |20 API_大型 ->私有1 |21 前端_ 大型 ->公共1 |22 Redis_大型 ->私有3 |23 身份_提供商_大型 -> 私有3 |24 数据库_大型 ->私有5 |25 etcd_大型 ->Private1} 26 ...

**图 7：** simulation.log 文件摘录。

增强器。这些事实描述了部署和故障事件，涵盖组件和节点，如图 8 所示。基于这些输入，故障增强器生成旨在提高当前微服务架构部署韧性的部署软约束，随后将其传递给协调器进行处理。

能耗增强器

能耗增强器以基础设施描述 I 为输入，其中存储了与节点相关的所有信息；以应用描述 A 为输入，其中包含服务相关信息；还以应用部署 D 为输入，详细说明了每项服务部署在哪个节点上。

```
部署到(API,          大型，Private1)。
部署到(数据库,          大型，private5)。
部署到(etcd,          大型，Private1)。
...

不可达(frontend,          31).
不可达(frontend,          32).
...

过载(public1, 中央处理器, 31, 98).
...
```

**图8：** deployment.pl 文件的摘录，由*ECLYPSE*解析器生成。

最后，它使用来自 *ECLYPSE* 的仿真日志，提供有关服务及服务连接能耗的数据。

基于*ECLYPSE*生成的报告，能耗增强器会读取报告并估算排放量。估算时仅考虑各服务或节点处于活动状态的时段，并采用监测期间的平均能耗。随后，能耗增强器会利用这些排放估算结果，生成与所测量的当前部署相关的约束。生成的约束类型可以是亲和性或避免。亲和性 约束表示两个服务之间有大量数据交换，因此应将它们部署在同一节点上。避免约束表示若将某服务部署在该约束所针对的节点上，其能耗会过高，因此最好将其部署到其他位置。生成约束后，我们会将其保存到知识库 中，以便将来检查生成的约束是否再次出现；如果反复出现，则可能表明这些约束具有参考价值，因此应再次提出。

能耗增强器完成执行后，将输出能源约束，并根据估算的排放量自动排序，同时为其分配一个权重，以表示内部排序。其输出示例见图9：约束类型之后是其涉及的元素，最后是与这些元素相关联的权重。

协调器
如图 4所示，协调器以双方生成的软约束作为输入

```
避免(d(identity_provider,大型),private3,0.883). 避免(d(数据库,大型),private1,1.0).
```

**图 9：** 与能量相关的约束示例。

故障增强器和能耗增强器。其作用是识别这些约束之间可能存在的冲突，并根据用户定义的优先级解决冲突；该优先级可以侧重于韧性、可持续性，或兼顾两者。处理完这些约束后，会将其发送给多准则求解器，以供下一次部署时考虑。

基础设施如图6所示，由七台以节点表示的机器组成。这些机器分为 2 个公共节点（用于模拟公开暴露的节点）和 5 个私有节点（用于模拟私有子网中的节点）。如代码仓库 [21],中所述，每个节点都包含一个名为 subnet 的资源列表属性，用于指定该节点是否暴露于互联网。具体而言，节点 Public1 和 Public2 的值设为 public，其余私有节点的值设为 private。节点之间的蓝色（双向）箭头表示机器之间的连接。应用程序的完整视图以 YAML 格式描述，可在 [21], 中按照 [13, 16]所述规范查看。

## 4.2 场景

本节介绍针对应用程序和底层基础设施评估的场景，并概述预期结果与观测结果。

场景定义为一组规则，用于修改仿真轮次期间各元素的行为。这些规则可能影响基础设施组件，例如节点上的中央处理器或随机存取存储器可用性；也可能影响应用程序层面的因素，例如服务的能耗。规则还可以选择性地应用于特定时间间隔，这些时间间隔称为仿真时钟周期，从而为可开展的实验类型提供灵活性。在实验阶段，采用了两种反复出现的模式：

•恒定修改，即在一定的时钟周期范围内，将资源调整一个固定值。

```
应用程序_策略 = [
    示例 1: [场景] * X          # 我们运行此操作
        在应用侧运行场景 X 次
    示例 2: [场景1] * i + [场景2]* j
        # 我们先运行场景1 i 次，然后我们运行
        应用侧执行场景2 j 次。
]
基础设施_策略 = [
    示例 1: [场景] * Y # 我们运行此场景
        基础设施侧执行场景 Y 次
    示例 2: [场景3] * n + [场景4]* m
        # 我们运行场景3 n 次，然后我们运行
        基础设施侧的场景4执行 m 次。
]
```

**图10：** 展示用于运行场景的配置模式的代码片段。

- 正弦波修改，其中资源按照正弦函数变化，以确保起始值和结束值保持一致。

第一种模式用于对阈值进行压力测试，检验部署是否会因缺乏足够的中央处理器或随机存取存储器而无法托管服务，进而导致系统内出现故障和停机时间。第二种模式旨在模拟异常但短暂的行为，例如在起始值和结束值处均未发现问题，以此展示 FREEDA 方法如何检测此类实例并进行相应管理。

该*ECLYPSE* 模拟器集成了一种称为更新策略的机制，通过以编程方式修改基础设施资源容量和应用程序级需求，使环境动态性得到受控调整，并贯穿各个仿真轮次。在实验阶段，该机制用于在仿真时间线的特定时间点引入预定义场景。图10 展示了用于此目的的配置模式示例。在该结构中，数组中的位置指定了应用特定策略的仿真轮次，以及该策略所针对的系统组件，即基础设施或应用程序。这种灵活的方法不仅支持执行多个场景，还能探索在单次仿真中组合不同场景的多种方式。在项目代码库中，您可以找到组合多个场景的若干仿真结果 [23]。

服务发生能源故障并不会直接导致该服务停机时间增加，而是会导致绿色性能变差，进而影响总排放量。因此，能源场景可以灵活应用于微服务架构中的任何服务，从而轻松探索由此产生的不同部署方案。

*ECLYPSE* 模拟器还提供了集成的部署策略，用于将应用程序放置到节点上。根据首次适配策略，服务会被放置到首个满足其资源需求的可用节点上。相比之下，最佳适配策略会选择能够最大限度提高资源利用率且不超出容量的节点。定义这些场景并进行仿真的目的，是比较采用集成的*ECLYPSE* 部署策略所获得的结果与采用 FREEDA 方法所取得的结果。

此场景的主要目标是通过对公共节点可用资源进行恒定修改来降低中央处理器和随机存取存储器的性能，从而影响负载均衡器和前端服务的运行与部署。当资源发生这种退化时，受影响的服务预计会迁移到其他节点，或以不同于原先分配的风味重新部署。通过这一设置，我们可以评估ECLYPSE 最佳适配放置策略和 FREEDA 方法如何应对关键节点上的资源约束，以及它们能否通过重新分配或重新配置服务来维持微服务架构的可用性。本场景的结果将在第 4.3 节中讨论和分析。

## 4.3 结果

如图10 所示，仿真阶段应用的场景旨在应对这样一种情况：某些节点通过持续修改降低其中央处理器和随机存取存储器资源，而部分服务则通过正弦波修改降低其能量性能。

在仿真过程中，我们评估了工具链的五种不同配置，以观察这些组件在相同场景下的行为。以下总结了这些工具链配置：

1.***ECLYPSE*** 最佳适配：仅使用模拟器提供的放置策略最佳适配 来部署应用程序。 2. **仅使用多准则求解器**：使用多准则求解器生成的部署配置来部署应用程序。 3.**多准则求解器和能耗增强器**：使用多准则求解器生成的部署配置来部署应用程序，并结合能耗增强器根据仿真日志生成的约束进行增强。

4.**多准则求解器与故障增强器**：应用程序采用多准则求解器的部署配置进行部署，并结合故障增强器根据仿真日志生成的约束。

5. **完整 FREEDA**：应用程序采用完整的 FREEDA 工具链进行部署，即多准则求解器的部署配置，以及能耗增强器和故障增强器共同生成的约束。

第一种工具链配置（*ECLYPSE* 最佳适配）依赖于*ECLYPSE* 模拟器的内部最佳适配 [19] 策略，将所有组件都以 Large 规格部署，即采用资源消耗最高的配置。此方法旨在模拟专家级 DevOps 工程师的决策：他们全面了解应用需求和基础设施能力，因此能够部署应用程序的完整版本。在此设置下，全部七个组件均在六个仿真轮次中成功部署。在第二次部署（对应第 1 轮）期间，*ECLYPSE*重新分配了某些组件以降低能耗，随后保持该配置，使能耗保持稳定。就故障而言，由于场景会重复出现，停机时间图显示各轮次的故障模式一致。

图 11 的第二行展示了第二种工具链配置（仅使用多准则求解器）。与前一种情况一样，多准则求解器尝试以最强风味部署所有组件。然而，由于多准则求解器不考虑能耗优化，其能耗高于第一种方法。不过，由于部署配置在所有仿真轮次中保持不变，因此能耗始终稳定。故障模式与之前观察到的相同，因为单独使用多准则求解器（不含故障增强器和能耗增强器）无法感知仿真轮次中发生的事件。

能耗增强器是负责整体部署能耗排放的模块。该模块在图11的第三行和第五行处于活动状态。仿真过程中，我们设置了一个对数据库产生负面影响的场景，导致该服务的能耗激增。下图12展示了能耗增强器在第一轮仿真后生成的约束。数据库被正确识别为能耗最大的服务，这也体现在分配的权重上：数据库的优先级最高。由于数据库必须使用加密存储作为安全措施，才能部署到节点上，而示例基础设施中只有两个节点具备此功能，因此能耗增强器会正确地仅针对这两个节点中的一个提出软约束，因为我们希望将数据库服务部署到某处。

在后续所有轮次中，能耗增强器生成的约束保持不变，使服务的能耗趋于稳定，如图11所示。第五行也呈现出相同的行为，其中同时考虑了能耗增强器和故障增强器。

多准则求解器和故障增强器的配置如图第三行所示11。在此设置中，故障增强器生成的约束使多准则求解器能够保持 100% 的正常运行时间，即使 Public1 和 Public2 节点上的资源出现退化也是如此。图13 展示了 ECLYPSE 解析器根据仿真日志为第一轮仿真生成的故障增强器输入。在这一轮中，前端和负载均衡器服务均采用 Large规格部署在 Public1 上。由于最佳拟合 场景会故意降低资源

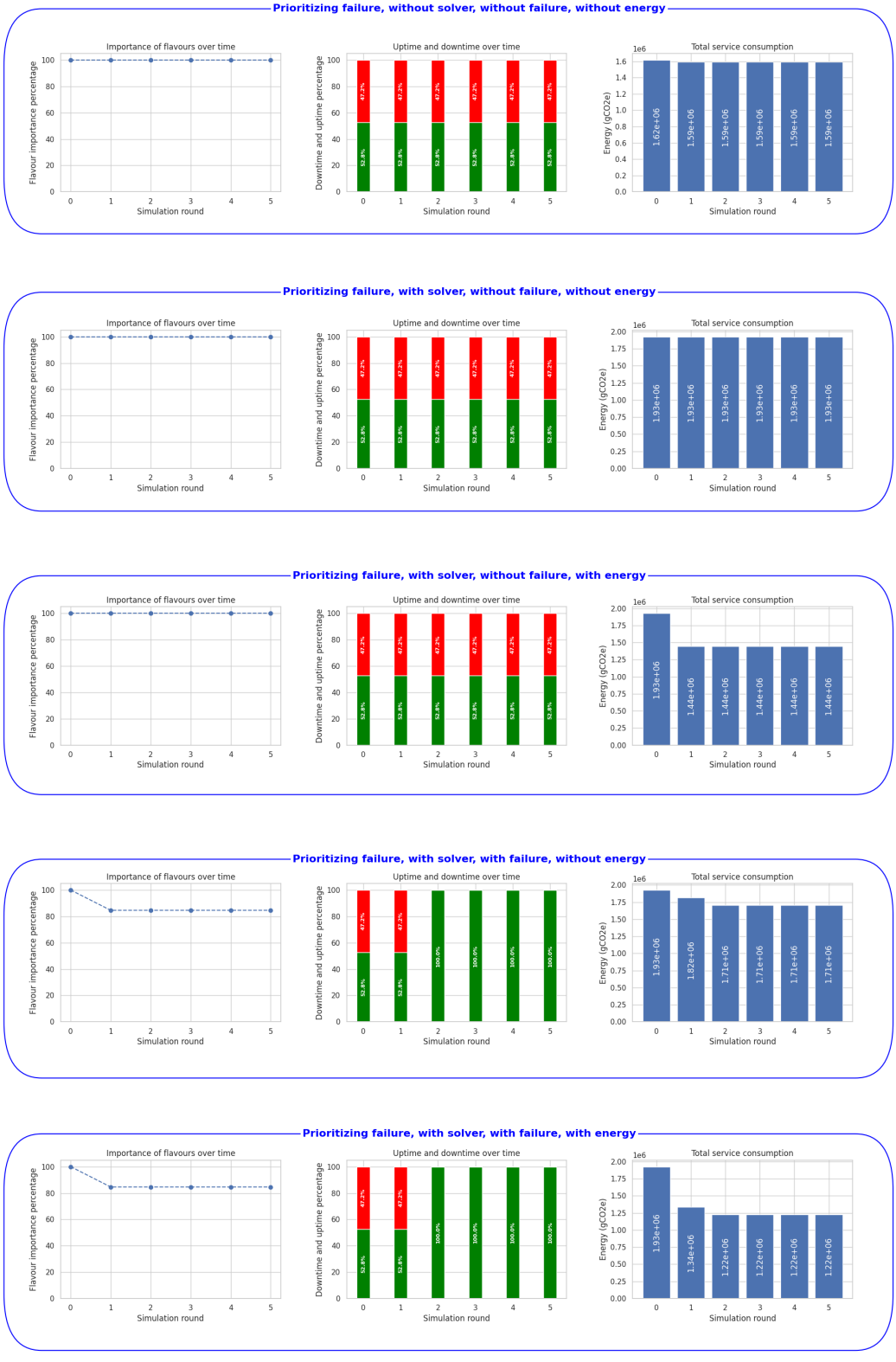


**图 11：** 针对每种工具链配置，在优先级故障下运行示例获得的结果。

```
1 避免(d(identity_provider,large),private1,0.493). 2 避免
(d(identity_provider,large),private3,0.883). 3 避免(d(identity_
provider,large),private5,0.413). 4 避免
(d(database,large),private1,1.0).
```

**图 12：** 能耗增强器在第一轮仿真期间生成的软约束。

```
1 部署到(api, large, private1). 2 部署到
(database, large, private5). 3 部署到(etcd, large,
private1). 4 部署到(frontend, large, public1).5 部署
到(identity_provider, large, private3).6 部署到(load_
balancer, large,public1).7 部署到(redis, large,
private3).8 不可达(frontend, 31)... 9 ...不可达
(frontend, 98). 10 不可达(load_balancer, 31)...11 ...不
可达(load_balancer, 98).12 过载(public1, cpu, 31,
98). 13 过载(public1, ram, 31, 98).
```

**图 13：** 故障增强器在第一轮仿真期间生成的 Deployment.pl 文件。

```
1 避免(d(frontend,large),public1). 2 避免
(d(load_balancer,large),public1).
```

**图 14：** 故障增强器在第一轮仿真期间生成的软约束。

如预期，Public1 的两个服务在时钟周期 31 至 98 之间均不可达。基于这些信息，故障增强器生成了图 14所示的软约束，随后将其传递给协调器，再由多准则求解器进行处理。

## 4.4 讨论

观察图11可知，完整的 FREEDA 方法能够利用各个单独模块提出的所有改进，并将其整合为一个同时具备故障韧性和能耗意识的解决方案。起初可以看到，原生的 *ECLYPSE* 部署策略可能比首个 FREEDA 工具链配置取得略好的能耗结果，但我们必须记住，*ECLYPSE* 最佳适配并未考虑各种部署配置这些配置正如第4.1节所述。它只是将能耗最高的服务放入能耗最低的节点，而不会恰当地检查该服务是否需要私有或公共节点、服务依赖关系或其他需求，例如数据库服务所需的加密存储。这使得初始的最佳适配部署比不含能耗增强器组件的 FREEDA 配置更加绿色。从故障角度来看，*ECLYPSE* 最佳适配没有解决可能出现问题的办法，因而导致服务持续中断。最后一行展示了完整的 FREEDA 方法。起初，能耗可能较高，原因如前所述，但随后会迅速下降至低于初始部署的水平，甚至低于 *ECLYPSE*部署策略。这可归因于多准则求解器遵循能源约束，并决定更改组件的风味，从而也降低了风味重要性。从第二个仿真轮次开始，情况趋于稳定，风味重要性得到成功最大化，同时彻底消除故障并将能耗降至最低。

# 5 仿真

本节概述仿真工具链，随后在后续小节中详细介绍各个步骤。仿真阶段的主要目标是收集有关 FREEDA 所提供的韧性和碳效率的数据。具体而言，我们测量服务可用性、应用质量以及集群产生的碳排放。

所有仿真实验均在使用 Minikube 部署的虚拟化 Kubernetes 集群上执行，从而能够在受控条件下复现测试场景。此设置允许有意引入性能下降、调整能耗参数以模拟碳强度波动，并对基础设施节点施加计算资源约束。

## 5.1 仿真设置

参考应用

为评估第 3 节中介绍的 FREEDA工具链，当时没有可用于管理多种服务变体的现有基准测试。因此，我们设计并实现了一款名为 BrewMonitor 的新应用程序。BrewMonitor 是一种开源的多风格微服务架构，可在 GitHub 上公开获取，地址为 [24]。它遵循云原生架构，由多个独立微服务组成，这些微服务可配置为两种变体（微型和大型），并能在逼真的环境中模拟数据的收集、聚合与分析。

图 15 展示了我们参考应用的架构。BrewMonitor 是一款面向酿造工厂的监控应用程序，以微服务架构（MSA）实现，旨在实现模块化、可扩展性和易于扩展。其主要用途是采集酿酒厂传感器的实时数据，对数据进行汇总和分析，并通过 REST接口向生产操作员和监控系统提供结果。该架构设计支持两种不同的变体：微型（图 15中的红色箭头）和大型（浅蓝色箭头），二者在功能、数据持久化和韧性特性方面有所不同。该微服务架构由四个主要微服务组成，每个微服务负责监控工作流中的一个特定阶段：

- **网关：**作为系统的唯一入口，提供 HTTP REST端点，统一访问底层服务。该服务使用 Python 和 Flask 实现，提供轻量级集成、集中式日志记录、身份验证和速率限制。

- **数据采集：**负责从模拟传感器采集过程数据（温度、湿度和 pH值）。在微型变体中，按需在内存中生成数值，从而提供轻量级的无状态服务。在大型变体中，读数会持久化存储在 MongoDB 中，并设置 24小时生存时间，以便访问历史数据进行分析和审计。

- **聚合器**：定期收集由一个或多个数据采集实例生成的数据。在微型风味下，它会查询 */data-gather/avg* 端点，并将结果直接转发给客户端。在大型风味下，后台线程会轮询每个数据采集实例

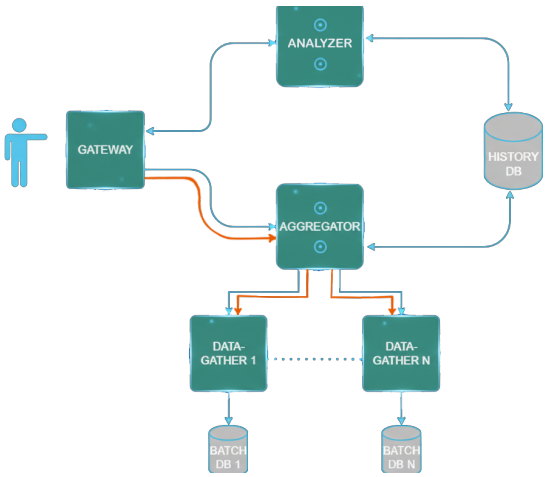


**图 15：** BrewMonitor 微服务架构的概览。

每分钟一次，将数据存储在 MongoDB 中，并通过 */aggregator/current* 端点公开最新的聚合记录，从而确保各采集节点之间的一致性。

- **分析器：** 仅在大型 风味中提供，此组件会对 MongoDB 历史数据进行统计分析。它会计算温度、湿度和 pH值的最大值、最小值、标准差和异常值等指标，并提供单一的 */analyzer/stats*端点，为每个受监控实例返回详细的 JSON 结果。

集群与节点配置

实验环境由一个 Minikube 集群组成，该集群包含七个虚拟节点，每个节点都具有不同的技术规格，体现了现代混合云环境中典型的异构性。

- *gatewaynodeefficient*：指定为集群主节点，采用均衡配置，配备 2 个 CPU核心和 4 GB 随机存取存储器，碳强度为 200 gCO2/kWh。此配置适合托管优先考虑稳定性而非原始性能的网关服务。

- *gatewaynodestrong*：提供更强的计算能力，配备 4 个 CPU核心和 8 GB 随机存取存储器，但碳强度略高，为 250 gCO2/kWh，体现了性能与可持续性之间的经典权衡。

• 数据库节点：作为数据管理的主要节点，配备 8 个 CPU核心和 8 GB 随机存取存储器，并针对较低的碳强度（100 gCO2/kWh）进行了优化。这种配置凸显了在保持较低环境足迹的同时，实现高性能数据持久化的重要性。

• 强应用节点：作为执行核心应用逻辑的主节点，配备 8 个 CPU核心、8 GB 随机存取存储器，并针对 100 gCO2/kWh 的低碳强度进行了优化。

• 高能效应用节点：提供均衡的折中方案，配备 6 个 CPU核心和 6 GB 随机存取存储器，同时也是集群中碳效率最高的节点，碳强度仅为 90 gCO2/kWh。该节点尤其适合用于测试 FREEDA工具链在有利条件下优先考虑碳效率的能力。

• 高成本节点：配备 8 个 CPU核心和 8 GB 随机存取存储器，但有意将碳强度设为较高的 1000 gCO2/kWh，以模拟在环境成本高昂的基础设施上进行紧急扩缩容的场景。

指标与监控
为监控参考 MSA，我们开发了一个自定义观测脚本。该脚本采用多层架构来收集和处理应用指标。通过集成原生 Kubernetes API、Prometheus 监控系统以及用于测量能耗的 Kepler，该脚本能够全面呈现集群在功能和能耗层面的性能。

仿真过程中监测了三项关键指标：应用质量、停机时间和二氧化碳排放。应用质量表示当前在微服务架构中启用的各个变体的重要性的综合衡量指标，以当前重要性评分与理论最大值的百分比之比表示。该指标以定量方式反映了 FREEDA 通过运行性能最高的变体来维持服务运行的能力。

停机百分比量化了微服务架构（MSA）无法以最低可接受质量水平提供所有必需服务的时间占比通过持续监控所有应用 Pod 的就绪状态来计算该指标，但不包括辅助 Pod（例如用于模拟系统压力的压载 Pod）。只有当所有预期 Pod 同时处于就绪状态时，该时段才计为正常运行时间，从而真实反映整体服务可用性。

二氧化碳总排放量是评估微服务架构环境影响的主要指标。其计算方式是对每个活动节点的瞬时功耗与碳强度之积随时间进行积分。电力数据通过 Prometheus 查询获取，而碳强度值则从应用于相应 Kubernetes 节点的动态标签中读取。

## 5.2 场景

为仿真定义的实验套件包含四个主题系列中的七种场景，每种场景均针对 FREEDA 工具链及其多目标优化机制的特定自适应特性。

**•系列 1.x - 资源耗尽：** 测试 FREEDA 处理资源短缺的能力。场景 1.1 和 1.2 人为增加关键服务的 CPU 需求，触发重新调度并激活故障管理逻辑。

**•系列 2.x - 节点压力与碳强度变化：** 评估 FREEDA 对局部竞争和环境变化的响应。场景 2.1 通过压载 Pod 模拟高资源使用情况，而场景 2.2 则动态提高节点碳强度，以测试碳感知工作负载再平衡。

**• 系列 3.x - 故障与策略困境：** 探讨复合故障与权衡。场景 3.1 将节点故障与资源过载结合起来，而场景 3.2 则引入多重压力和相互冲突的优化目标，以评估 FREEDA 在复杂条件下的决策能力。

**•系列 4.x - 碳感知自适应：** 考察系统对不断变化环境的持续自适应能力。子场景 4.1 至 4.5 逐步改变碳强度、工作负载和节点容量，以测试 FREEDA 对

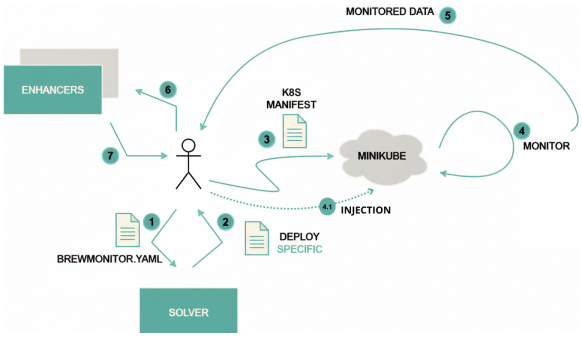


**图 16：** 每个仿真场景执行期间所遵循的工作流。

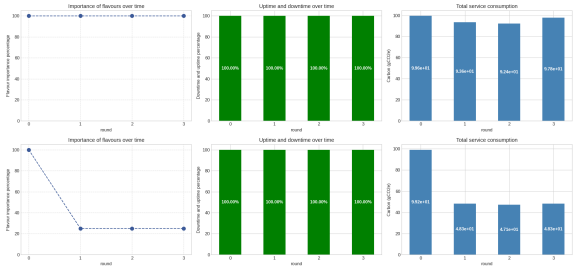

**图 17：** 场景 2.2 的仿真结果，第一行对应基线结果，第二行对应使用 FREEDA 获得的结果。

环境和基础设施参数逐渐变化及突然转变的响应能力。

该实验套件全面评估了 FREEDA 在多样且不断变化的运行条件下的韧性、效率和环境适应性。比较方法包括在两种不同配置下依次执行每个场景。基线阶段仅依赖原生 Kubernetes 编排，不受 FREEDA 干预。随后，自适应阶段启用 FREEDA 的优化机制，以减轻所引入劣化的影响。该设计能够对两种配置进行直接且定量的比较，隔离 FREEDA 算法决策的影响，并使观察到的改进能够可靠地归因于该框架的自适应策略。

与基准场景依赖 Kubernetes 原生编排不同，使用 FREEDA 优化的场景会实现一个手动反馈回路，以模拟智能控制系统，如图 16所示。每个场景的设计都旨在首先揭示传统编排的局限性，并记录缺乏智能优化时出现的故障模式、能源效率低下和性能瓶颈。随后，分析将聚焦于 FREEDA 实施的纠正措施，例如节点间的服务迁移、规格调整或整体资源重新分配，展示这些机制如何提升整个集群的韧性、效率和可持续性。

## 5.3 结果

本节介绍场景 2.2 和 3.1 的结果。所有其他场景的结果及其执行说明均可在 GitHub 上公开获取，网址为 [25]。

在场景 2.2 中，我们动态改变 *appn-odestrong* 节点的碳强度，将其从 100 gCO2/kWh 提高至 700 gCO2/kWh。该场景评估 FREEDA 的碳感知能力，验证该框架能否检测环境条件的变化，并相应地重新平衡工作负载放置，以最大限度地降低总体碳影响。图 17展示了基线执行与 FREEDA 优化部署之间的详细比较。如图所示，FREEDA 的干预从根本上改变了微服务架构的环境特征，展示了面向现代微服务应用的碳感知优化策略的潜力。

图17 的左下方面板显示，应用质量从第一轮优化开始便立即、有控制地降至约 25%，此后保持稳定。这种受控下降表明，FREEDA 实施了一种激进但经过校准的重新部署策略。该策略包括将服务从碳强度较高的节点大规模迁移至更具可持续性的替代节点，并选择性地降低部分变体的性能，以优化整体微服务架构的碳效率。这一策略兼顾了碳排放预算与成本之间的权衡。

在整个实验过程中，正常运行时间 （下方中央面板）始终保持在 100%，表明 FREEDA 的碳感知优化并未

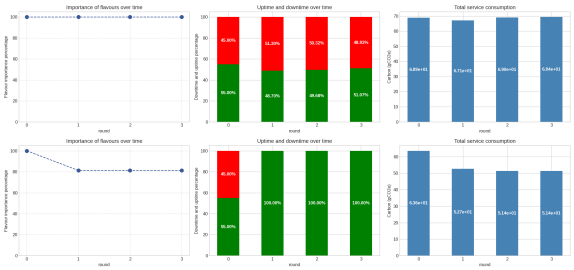

**图18：** 场景 3.1 的仿真结果，第一行对应基线结果，第二行对应使用 FREEDA 得出的结果。

牺牲运行可用性。总体而言，结果证实，FREEDA 能够在动态变化的环境条件下，有效平衡可持续性与服务连续性。

场景 3.1 引入了多重压力条件：首先，模拟 *gatewaynodestrong* 节点通过排空而完全故障，迫使所有已托管 Pod 迁移；随后，使其 聚合器 服务的 微型 风味遭受资源过载。目标是在极端压力条件下，测试 FREEDA 平衡容错、性能和环境可持续性的能力。图 18 展示了场景 3.1 中基线方法与 FREEDA 优化方案的各项指标详细对比。

该场景的基线执行结果（图 18，上排）凸显了节点故障对未配备主动韧性机制的微服务架构造成的严重影响。整个节点的突然丢失构成了系统性冲击，暴露出 Kubernetes 原生编排能力的局限性。如图 18 上方中央面板所示，基线微服务应用正常运行时间持续在 45% 至 55% 之间波动。这表明，微服务架构有将近一半的总运行时间处于完全不可用状态。这种不稳定并非暂时现象，而是在整个实验过程中持续存在的状况。这说明 Kubernetes 原生调度器未能在节点丢失后稳定配置，具体原因是没有可用节点能够容纳具有 网关 服务及其 大型 风味的节点。

因此，原生编排器陷入调度失败的无限循环，反复尝试重新分配孤立服务，却始终无法收敛到有效配置。这种行为揭示了标准编排在应对灾难性故障方面的根本局限。

相比之下，FREEDA 的干预展现出强大的应急管理能力，克服了这些结构性限制。扰动发生后，FREEDA 立即稳定了微服务架构，将正常运行时间恢复至 100%，并在整个实验过程中保持这一水平（见图 18 下方中央面板）。具体而言，FREEDA 自动修改了网关风味，并将其迁移至唯一合适的节点 *gatewaynodeefficient*，同时实现了功能恢复和碳效率提升。

在场景 3.1 中，无法将服务从强应用节点 节点迁移至高能效应用节点 节点，因为后者缺乏充分满足需求所需的资源。不过，FREEDA 仍在韧性与碳优化之间实现了最佳权衡。这一表现凸显了 FREEDA 的设计原则：在不损害基于微服务的应用质量或可用性的前提下，最大限度地提高可持续性和资源效率。

## 5.4 讨论

对七个模拟场景的分析证明了 FREEDA 多目标方法的有效性。在所有情况下，FREEDA 工具链都消除了停机时间，将不稳定的微服务架构转变为可用性达 100% 的配置。与基线版本相比，二氧化碳排放量减少了 21% 至 52%，所有场景的平均降幅为 35%。这种差异反映了针对各个场景的具体特征所采用的不同优化策略。结果凸显了 FREEDA 管理复杂场景的能力，这些场景具有多个压力因素同时存在以及逐步动态变化的特点。在最严峻的场景（场景 3.2）中，基线正常运行时间为 34%–43%，FREEDA 将可用性提升至 100%，同时将排放量减少了 21%–24%。

# 6 相关工作

约束推理最初被用于在云资源上优化部署多服务应用程序 [26–28]。其中， [28]致力于服务依赖建模，而[26] 和 [27] 则聚焦于服务的硬件、软件和可用性需求。近年来，约束推理也被用于生成容器化的微服务架构部署， [29]和[30]将 [26]的方法扩展至微服务架构，而 [31] 则介绍了基于服务质量需求的 Kubernetes 容器调度。相比之下， [32] 致力于在云虚拟机上部署微服务应用，并将硬件和软件需求编码为约束，以实现部署总成本最小化。总而言之，现有方法通常只处理微服务架构部署问题中相互独立的方面，例如硬件/软件需求、服务依赖或成本优化；即使上下文变化仅影响部署的一部分，也假设需要重新进行完整部署。我们的方案旨在弥补这些不足，通过在云边连续体上实现全面的微服务架构部署，并持续推理以支持云边连续体上的自适应微服务架构部署。

现有用于增强微服务应用故障韧性的方法，主要是提供设计与开发指南 [33, 34], ，或提供配置部署脚本的机制，使发生故障的服务能够自愈，通常是通过重启其承载容器来实现 [35]。然而，目前没有任何现有解决方案能够结合已部署的微服务架构与可用的云边基础设施进行分析，也无法在此类基础设施上自动实施具备故障韧性的部署。当前针对微服务应用故障分析的技术主要侧重于检测故障并识别其根本原因 [36–39]。尽管这些方法能够自动推断已观测故障的潜在根本原因，但通常止步于此，仍需由 DevOps 工程师手动检查日志或监控指标，以了解故障如何在整个系统中传播 [3]。朝这一方向推进的唯一一项工作是我们此前的成果 [40]，它为 DevOps 工程师提供了微服务应用中故障传播的解释，但仍要求他们手动指定每个服务的行为。总而言之，据我们所知，目前尚无现有技术能够自动分析云边基础设施上的微服务架构部署，从而在该部署中增强故障韧性。我们提出的方法旨在弥补这一空白，通过提供一种可解释的方式来提升微服务架构部署的故障韧性。

许多现有方法致力于提高云数据中心的能源效率 [41–43], ，而 [44, 45]则概述了提升数据中心可持续性的最佳实践。然而，计算基础设施日益分布于云边连续体之上，而该连续体本质上由异构节点构成。这种异构性会导致电能使用效率显著波动 [46], ，使云边资源的能耗感知分配成为一项持续的研究挑战 [47, 48]。此外，当应用程序分布于云边连续体之上时，其数据必须与应用程序本身紧密协同地存储和管理，以降低延迟并确保符合安全约束 [49]。尽管如此，当前提升部署效率的方法通常只关注目标基础设施，偶尔会将已部署应用程序的时间或地理分布纳入考量 [50–52]。另一些研究则侧重于应用程序层面 [53–55]，支持从头设计能源可持续应用程序，但对现有系统的适用性有限。与此同时，随着云计算的广泛普及，已部署应用程序的能量需求也不断增长，预计这一趋势将在云边基础设施中持续下去。不断增长的需求凸显了在云边连续体上进行能耗感知的 MSA 部署的必要性 [56]。我们的方案旨在弥补这一空白，以可解释的方式降低微服务架构部署对环境的影响。

# 7 结论

云边基础设施日益增强的异构性、规模和动态性，要求部署策略能够平衡多个往往相互冲突的目标。本文介绍了 FREEDA 工具链，这是一个全面的框架，可自动实现微服务应用的具备故障韧性和碳高效的部署

，覆盖整个云边连续体。FREEDA 将基于风味的优化与自适应编排策略相结合，以便根据基础设施可变性、资源约束和环境目标动态重新配置部署。

所提出的方法使 DevOps 团队能够在主动减少碳排放的同时，维持服务连续性和应用质量。通过一系列受控的模拟和仿真模拟实验，我们展示了 FREEDA 在缓解故障、重新平衡工作负载以及调整规格选择方面的能力，从而在韧性与可持续性之间实现最佳折中。实验结果表明，FREEDA工具链能够有效降低对环境的影响，同时不损害微服务应用可靠性或性能。

未来，我们计划通过开发一个完整的框架，扩展 FREEDA 提供的支持，以支持在云端-边缘连续体上设计、开发和部署微服务应用。该框架将包含用于监测已部署微服务应用碳足迹的技术和工具，并致力于提升可靠性以外的其他质量属性，例如安全性和性能效率。这样，未来版本的 FREEDA 将有助于提升现代信息通信技术系统的环境可持续性和整体质量。

## 致谢

本工作得到了*FREEDA* 项目（CUP: I53D23003550006）的支持，该项目由意大利大学与研究部（MUR）PRIN 框架和欧盟下一代计划资助。

## 参考文献

[1]Ferrer, A.J., Marquès, J.M., Jorba, J.：迈向去中心化云端：移动计算、自组织网络和边缘计算的方法与挑战综述。ACM 计算调查**51**（6）（2019年）https://doi.org/10.1145/3243929

[2] Gaglianese, M., Soldani, J., Forti, S., Brogi, A.：云边应用程序的绿色编排：最新技术水平与开放挑战。载于：2023年 IEEE 服务导向系统工程国际会议（SOSE），第 250–261 页（2023）。 h tt ps://doi.org/10.1109/SOSE58276.2023.00036[3] Soldani, J., Brogi, A.：(微)服务型云应用中的异常检测与故障根因分析：一项综述。ACM Comput. Surv.**55**（3）（2022） https://doi.org/10.1145/3501297[4] 欧盟：欧盟新战略议程（2019年）[5]欧盟：欧盟工业政策。（2022）[6] Abbasi-khazaei, T., Rezvani, M.H.：采用进化计算方法，在云数据中心进行能量感知和碳高效虚拟机部署优化。Soft Computing **26**（18），9287–9322（2022） https://doi.org/10.1007/s00500-022-07245-y[7] Ahvar, E., Ahvar, S., Mann, Z.A., Crespi, N., Glitho, R., Garcia-Alfaro, J.：Deca：一种适用于边缘云的动态能源成本与碳排放高效应用部署方法。IEEE Access **9**，70192–70213（2021） h ttps://doi.org/10.1109/ ACCESS.2021.3075973[8]Aldossary, M., Alharbi, H.A.：面向雾云架构中碳排放最小化的绿色方法。IEEE Access**9**，131720–131732（2021） h ttps://doi.org/10.1109/ ACCESS.2021.3114514[9]Forti, S., Brogi, A.：声明式渗透应用部署。载于：Polyvyanyy, A., Rinderle-Ma, S.（编）《高级信息系统工程研讨会》，第 177–190 页。施普林格，Cha m（2021）

[10]Vitali, M.：《迈向更绿色的应用：实现可持续性感知的云原生应用设计》。载于：Franch,X., Poels,G., Gailly, F., Snoeck, M.（编），《高级信息系统工程》，第 93–108 页。施普林格，Cham（2022）

[11]Vitali, M., Schmiedmayer, P., Bootz, V.: 《为云原生应用增添可持续性特性》。载于：2023 年 IEEE 云工程国际会议（IC2E），第 21–31 页（2023）。https://doi.org/10.1109/IC2E59103.2023.00011

[12] Soldani, J., Amadini, R., Brogi, A., Forti, S., Giallorenzo, S., Plebani, P.,Vitali,M., Zavat- taro, G.: 借助 FREEDA，迈向云端—物联网连续体上的微服务可持续部署。载于：第 4 届边缘灵活资源与应用管理研讨会论文集。FRAME '24，第 1–4 页。计算机协会，纽约，纽约州，美国（2024年）。https://doi.org/10.1145/3659994.3660311 。https://doi.org/10.1145/3659994.3660311

[13] Gazza, S., Amadini, R., Brogi, A., D' Iapico, A., Forti, S., Giallorenzo, S., Plebani, P., Ponce, F., Soldani, J., Vitali, M., Zavattaro, G.: 一种基于约束的方法，用于优化具有服务质量和能量感知能力的云边应用程序部署。ACM 互联网技术汇刊（2025年）https://doi.org/10.1145/3757061[14] Ponce, F., Forti, S., Soldani, J., Brogi, A.: 增强云边微服务的故障韧性：FREEDA 方法。载于：Pahl, C., Janes, A., Cerny, T., Lenarduzzi, V., Esposito, M.（编），面向服务与云计算，第 105–111页。施普林格，Cham（2025年）[15] Amadini, R., Gazza, S., Soldani, J., Vitali, M., Brogi, A., Forti, S., Giallorenzo, S., Plebani, P., Ponce, F., Zavattaro, G.: 选择一种风味：迈向云边应用的可持续部署。载于：Bowles, J., Søndergaard, H.（编），基于逻辑的程序综合与转换——第34届国际研讨会，LOPSTR 2024，意大利米兰，2024年9月9日至10日，论文集。计算机科学讲义，第14919卷，第117–127页。施普林格，???（2024年）。https://doi.org/10.1007/978-3-031-71294-4_7 . https://doi.org/10.1007/978-3-031-71294-4%5F7

[16] Simone Gazza, F.P., Soldani,J.: FREEDA YAML 模型规范（2024年）。https://github.com/FREEDA-Project/YAML-model/tree/v0.2

[17]Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: MiniZinc：迈向标准 CP 建模语言。载于：Bessière, C.（编）《约束编程原理与实践 – CP 2007》，第 529–543 页。施普林格，柏林，海德堡（2007年）

[18]YAML 语言开发团队：YAML 不是标记语言（YAMLTM）1.2.2 版。 https://yaml.org/spec/1.2.2/。访问日期：2025年1月（2021年）

[19] Massa, J., Caro, V.D., Forti, S., Dazzi, P., Bacciu, D., Brogi, A.: ECLYPSE：用于云边连续体仿真与仿真的 Python 框架（2025年）。https://arxiv.org/abs/2501.17126

[20]Francesco, P.D., Lago, P., Malavolta, I.: 微服务架构设计：一项系统性映射研究。《系统与软件杂志》**150**，77–97（2019年）https://doi.org/10.1016/J.JSS.2019.01.001

[21]Simone Gazza, F.P., D'Iapico, A.: 实验源代码（2025年）。https://github.com/FREEDA-Project/case%5Fstudy

[22]Sittón-Candanedo, I., Alonso, R.S., Corchado, J.M., Rodríguez-González, S., Casado-Vara, R.: 边缘计算参考架构综述及一种新的全局边缘方案。Future Gener. Com- put. Syst. **99**，278–294（2019年）https://doi.org/10.1016/J.FUTURE.2019.04.016

[23]Simone Gazza, F.P., D'Iapico, A.: 实验源代码（2025年）。https://github.com/FREEDA-Project/case%5Fstudy

[24]Usai, D.: 酿酒厂应用。https://github.com/FREEDA-Project/brew-monitor。访问日期：2025年10月（2025年）

[25]Usai, D.: 酿酒厂微服务测试平台。https://github.com/FREEDA-Project/

brew-monitor-experiments/tree/main/ 实验。访问日期：2025年10月（2025年）

[26]Ábrahám, E., Corzilius, F., Johnsen, E.B., Kremer, G., Mauro, J.：Zephyrus2：利用可满足性模理论和约束规划技术进行即时部署优化。载于：Fränzle, M., Kapur, D., Zhan, N.（编），《可信赖软件工程：理论、工具与应用》，第229–245页。施普林格，Cham（2016年）[27]Di Cosmo, R., Lienhardt, M., Treinen, R., Zacchiroli, S., Zwolakowski, J., Eiche, A., Agahi, A.：云应用的自动化合成与部署。载于：第29届ACM/IEEE自动化软件工程国际会议论文集。ASE '14，第211–222页。计算机协会，纽约，NY，美国（2014年）。https://doi.org/10.1145/2642937.2642980 。https://doi.org/10.1145/2642937.2642980[28] Fischer, J., Majumdar, R., Esmaeilsabzali, S.：Engage：一种部署管理系统。SIGPLAN 通讯 **47**(6), 263–274 (2012年) https://doi.org/10.1145/2345156.2254096[29] Bacchiani, L., Bravetti, M., Giallorenzo, S., Mauro, J., Talevi, I., Zavattaro,G.：微服务动态架构级部署编排。载于：Damiani, F., Dardha, O.（编），《协调模型与语言》，第257–275页。施普林格，Cham（2021年）[30]Bravetti, M., Giallorenzo, S., Mauro, J., Talevi, I., Zavattaro, G.：微服务的最优自动化部署。载于：Hähnle, R., Aalst, W.（编），《软件工程基础方法》，第351–368页。施普林格，Cham（2019年）[31] Lebesbye, T., Mauro, J., Turin, G., Yu, I.C.：Boreas——用于最优Kubernetes部署的服务调度器。载于：Hacid, H., Kao, O., Mecella, M., Moha, N., Paik, H.-y.（编），《面向服务的计算》，第221–237页。施普林格，Cham（2021年）[32] Eraşcu, M., Micota, F., Zaharie, D.：云端可扩展的最优部署

使用优化模块理论、数学规划和对称性破缺的基于组件的应用程序。《逻辑与代数编程方法期刊》**121**，100664（2021年）https://doi.org/10.1016/j.jlamp.2021.100664

[33] Giedrimas, V., Omanovic, S., Alic, D.：基于微服务的软件设计中的韧性方面。载于：Mazzara, M., Ober, I., Salaün, G.（编），《软件技术：应用与基础》，第589–595页。施普林格，Cham（2018年）

[34] Heorhiadi, V., Rajagopalan, S.,Jamjoom, H., Reiter, M.K., Sekar, V.：Gremlin：微服务的系统化韧性测试。载于：2016年 IEEE 第36届国际分布式计算系统会议（ICDCS），第57–66页（2016年）。https://doi.org/10.1109/ICDCS.2016.11

[35]Brogi, A., Carrasco, J., Durán, F., Pimentel, E., Soldani, J.：自愈式跨云应用。计算**104**(4), 809–833 (2022年) https://doi.org/10.1007/s00607-021-00977-z[36] Aggarwal, P., Gupta, A., Mohapatra, P., Nagar, S., Mandal, A., Wang, Q., Paradkar, A.：通过利用黄金信号挖掘日志中的因果依赖关系，定位云应用中的运行故障。载于：Hacid, H., Outay, F., Paik, H.-y., Alloum, A., Petrocchi, M., Bouadjenek, M.R., Beheshti, A., Liu, X., Maaradji, A.（编）《面向服务的计算——ICSOC 2020研讨会》，第137–149页。施普林格，Cham（2021年）[37] Liu, P., Xu, H., Ouyang, Q., Jiao, R., Chen, Z., Zhang, S., Yang, J., Mo, L., Zeng, J., Xue, W., Pei, D.：通过服务级深度贝叶斯网络无监督检测微服务跟踪异常。载于：2020年IEEE第31届国际软件可靠性工程研讨会（ISSRE），第48–58页（2020年）。https://doi.org/10.1109/ISSRE5003.2020.00014

[38]Meng, Y., Zhang, S., Sun, Y., Zhang, R., Hu, Z., Zhang, Y., Jia, C., Wang, Z., Pei, D.：在微服务中定位失败根本原因

通过因果推断。在：2020 IEEE/ACM 第28届国际服务质量研讨会（IWQoS），第1–10页（2020）。
h
ttps://doi.org/10.1109/I
WQoS49365.2020.9213058

[39] Wu, L., Tordsson, J., Elmroth, E., Kao, O.：Microrca：微服务性能问题的根因定位。载于：IEEE/IFIP 网络运营与管理研讨会（NOMS）（2020年）

[40] Soldani, J., Montesano, G., Brogi, A.：出了什么问题？解释基于微服务的应用中的级联故障。载于：Barzen, J.（编），面向服务的计算，第133–153 页。施普林格，Cham（2021年）

[41]Gholipour, N., Arianyan, E., Buyya, R.：一种新颖的能量感知资源管理技术，采用虚拟机与容器联合整合方法，用于云数据中心的绿色计算。仿真建模实践与理论**104**，102127（2020年）
https://doi.org/10.1016/j.simpat.2020.102127

[42]Vitali, M., Pernici, B., O'Reilly, U.-M.：为数据中心中的能源高效自适应应用程序学习目标导向模型。《信息科学》**319**，152–170（2015）
https://doi.org/10.1016/j.ins.2015.01.023 。大数据信息系统中的能源高效数据、服务和内存管理

[43] Wajid, U., Cappiello, C., Plebani, P., Pernici, B., Mehandjiev, N., Vitali, M., Gienger, M., Kavoussanakis, K., Margery, D., Perez, D.G., Sampaio, P.：论如何在云计算中实现能源效率并减少二氧化碳足迹。《IEEE 云计算汇刊》**4**（2），138–151（2016年）
https://doi.org/10.1109/TCC.2015.2453988

[44] Acton, M.、Bertoldi, P.、Booth, J.：2022 年欧盟数据中心能源效率行为准则最佳实践指南。欧盟委员会联合研究中心（JRC），比利时布鲁塞尔，技术报告 JRC128184（2022）

[45] Singh, H.、Reuters, T.、Azevedo, D.、Ibarra, D.、Newmark, R.、O'Donnell, S.、Ortiz, Z.、Pflueger, J.、Simpson, N.、S
mith, V.：数据中心成熟度模型。The Green Grid 技术报告（2011）[46]Jones, N.、等：如何阻止数据中心吞噬全世界的电力。《自然》**561**（7722），163–166（2018）

[47]Xiao, Y., Krunz, M.：雾计算网络中雾节点协作的服务质量体验与能效权衡。载于：IEEE INFO-COM 2017——IEEE计算机通信会议，第 1–9 页（2017 年）。
https://doi.org/10.1109/INFOCOM.2017.8057196

[48]Zhang, W., Zhang, Z., Zeadally, S., Chao, H.-C., Leung,V.C.M.：具有随机工作负载的分布式云端/边缘计算系统中的能效工作负载分配与计算资源配置。IEEE通信选刊 **38**(6), 1118–1132（2020 年）
https://doi.org/10.1109/JSAC.2020.2986614

[49] Plebani, P., Salnitri, M., Vitali, M.：雾计算与数据即服务：一种基于目标的建模方法，以实现有效的数据迁移。载于：Krogstie, J., Reijers, H.A.（编）《高级信息系统工程》，第203–219 页。施普林格，Cham（2018 年）

[50]Nylander, T., Klein, C., Årzén, K.-E., Maggio, M.：Brownoutcc：云应用响应时间界限的级联控制。载于：2018 年美国控制会议（ACC），第 3354–3361 页（2018 年）。
https://doi.org/10.23919/ACC.2018.8431282

[51]Papadopoulos, A.V., Krzywda, J., Elmroth, E., Maggio, M.：功率感知云端棕断：响应时间与功耗控制。载于：2017 年 IEEE 第 56 届决策与控制年会（CDC），第 2686–2691 页（2017 年）。
https://doi.org/10.1109/CDC.2017.8264049

[52]Xu, M., Toosi, A.N., Buyya, R.：一种用于管理应用程序的自适应方法

并利用可再生能源实现可持续云计算。IEEE 可持续计算汇刊 6(4), 544– 558（2021年） h ttps://doi.org/10.1109/TSUSC.2020.3014943

[53] Oliveira, F.G.,Ledoux, T.：面向服务的架构中能源足迹的自优化。载于：第一届绿色计算研讨会论文集。GCM '10，第 4–9 页。计算机协会，纽约，纽约州，美国（2010）。 https://doi.org/10.1145/1925013.1925014 。 https://doi.org/10.1145/1925013.1925014

[54]Cappiello, C., Fugini, M., Ferreira, A.M., Plebani, P., Vitali, M.：面向能源感知适应的业务流程协同设计。载于：2011 年 IEEE 第 7 届智能计算、通信与处理国际会议，第 463–470 页（2011）。 https://doi.org/10.1109/ICCP.2011.6047917

[55] Nowak, A., Breitenbücher, U., Ley mann, F.,等：自动化绿色模式，以补偿基于云端的业务流程产生的二氧化碳排放。载于：第八届先进工程计算与应用国际会议，第 132–139 页（2014年）

[56]Forti, S., Brogi, A.：云-IoT连续体中的绿色应用部署。载于：Cheney, J., Perri, S.（编）《声明式语言的实践方面》，第208–217页。施普林格，Cham（2022）