软件学报 ISSN 1000-9825, CODEN RUXUEW
[doi: 10.13328/j.cnki.jos.007645] [CSTR: 32375.14.jos.007645]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

KubeEdge 平台软件老化状态判定与抗衰策略生成*

刘 靖, 谭雪勇, 唐志伟, 牛超煜

(内蒙古大学 计算机学院 (软件学院)、人工智能学院, 内蒙古 呼和浩特 010021)

通信作者: 刘靖, E-mail: liujing@imu.edu.cn

摘 要: 边缘计算因其低延迟和高效的处理能力而被广泛应用于各个领域, 而 KubeEdge 平台系统作为边缘智能场景的核心基础软件, 其运行时可靠性至关重要. 然而, 边缘系统上的软件在长期运行后可能会出现软件老化问题, 导致系统响应延迟甚至服务中断, 进而影响用户体验乃至事故. 抗衰操作可以消除老化现象, 但目前针对边缘系统的老化研究相对较少, 且现有抗衰方法无法直接应用于边缘系统. 为解决上述问题, 针对 KubeEdge 边缘系统提出了一种称为 GIP-MI 的老化判定与抗衰综合方法, 该方法首先采用 GCN-Informer 方法对系统指标的空间关联和时序依赖进行建模, 相较于传统方法, 能够更精准、稳定地预测各指标的未来变化趋势; 进而将预测数据送入深度学习方法 ParNet, 通过多时间点切片与多分辨率特征融合, 实现对系统资源动态老化状态的更精准判定; 最后, 提出一种基于分解和信息反馈模型的多目标进化算法 (MOEA/D-IFM) 的任务卸载方法作为抗衰策略, 有效避免系统停机, 保证服务连续性. 实验结果表明, GIP-MI 在老化预测和状态识别精度上均优于基线方法, 并且与传统抗衰方法相比, 在停机时间等关键指标上均表现出显著优势, 能够有效恢复系统状态.

关键词: 软件老化; 边缘系统; 老化预测; 状态判定; 抗衰策略

中图法分类号: TP311

中文引用格式: 刘靖, 谭雪勇, 唐志伟, 牛超煜. KubeEdge平台软件老化状态判定与抗衰策略生成. 软件学报. http://www.jos.org.cn/1000-9825/7645.htm

英文引用格式: Liu J, Tan XY, Tang ZW, Niu CY. Software Aging State Determination and Rejuvenation Strategy Generation for KubeEdge Platform. Ruan Jian Xue Bao/Journal of Software (in Chinese). http://www.jos.org.cn/1000-9825/7645.htm

Software Aging State Determination and Rejuvenation Strategy Generation for KubeEdge Platform

LIU Jing, TAN Xue-Yong, TANG Zhi-Wei, NIU Chao-Yu

(College of Computer Science (College of Software) & College of Artificial Intelligence, Inner Mongolia University, Hohhot 010021, China)

Abstract: Edge computing has been widely adopted across various domains due to its low latency and high processing efficiency. As core foundational software in edge intelligence scenarios, the runtime reliability of the KubeEdge platform is critically important. However, software on edge systems may experience software aging after prolonged operation, leading to delayed system responses or even service interruptions, which may negatively impact user experience and potentially cause accidents. While rejuvenation operations can mitigate aging effects, current research on aging in edge systems remains relatively limited, and existing rejuvenation methods cannot be directly applied to edge systems. To address these challenges, this study proposes a comprehensive aging state determination and rejuvenation method named GIP-MI for the KubeEdge edge system. The method first employs the GCN-Informer approach to model spatial correlations and temporal dependencies among system metrics, enabling more accurate and stable predictions of future trends in these metrics compared to conventional methods. The predicted data is then fed into the deep learning method ParNet, which leverages multi-timepoint slicing and multi-resolution feature fusion to achieve more precise identification of dynamic aging states of system resources. Finally, a task offloading method based on the multi-objective evolutionary algorithm based on decomposition and information feedback model (MOEA/D-IFM) is

* 基金项目: 国家自然科学基金 (62462047); 内蒙古自然科学基金重点项目 (2023ZD18)
收稿时间: 2025-09-02; 修改时间: 2025-11-03, 2025-12-30; 采用时间: 2026-01-19; jos 在线出版时间: 2026-04-29

introduced as a rejuvenation mechanism, effectively avoiding system downtime and ensuring service continuity. Experimental results demonstrate that GIP-MI outperforms baseline methods in both aging prediction and state identification accuracy. Moreover, compared to traditional rejuvenation approaches, it shows significant advantages in key metrics such as downtime, enabling effective recovery of system states

Key words: software aging; edge system; aging prediction; state determination; rejuvenation strategy

随着工业互联网时代的到来, 边缘计算作为连接云端与终端设备的关键技术, 已被广泛应用于智慧城市、智慧医疗和自动驾驶等领域[1]. 在此背景下, 容器技术 (如 Docker) 因其轻量级、可移植和可扩展的特性, 在边缘计算场景的任务部署中得到广泛应用. 其中, KubeEdge 作为关键的容器编排与调度平台, 承担着云端与终端之间实时交互的重要功能, 其性能与稳定性直接影响系统的整体服务质量. 因此, 开展针对 KubeEdge 运行时可靠性的保障研究至关重要.

然而, 边缘系统在长期运行后可能会出现严重的软件老化问题[2]. 软件老化是指软件系统的运行状态随时间逐渐恶化, 主要原因包括系统资源 (计算、存储、网络等) 可用不足甚至耗尽, 以及数据损坏和数值错误累积等. 这些问题最终会导致系统性能急剧下降、可用性降低, 甚至直接崩溃. 软件抗衰通过偶尔或周期性地清除软件的老化状态 (例如对应用程序、虚拟机或操作系统等不同粒度进行重启), 可将软件的运行环境恢复到初始的正常状态[3]. 运行在边缘系统的任务往往对时延敏感, 需要实时处理. 软件老化导致的性能下降可能会引发较差的用户体验甚至事故. 因此, 边缘系统中老化问题的影响更加不容忽视.

目前, 针对边缘系统老化的研究较为有限, 且大多仅分析边缘系统中的老化现象, 缺乏对系统实际运行状态的判别. 此外, 鉴于边缘环境的实时性特点, 现有的抗衰方法 (如不同粒度的重启、检查点恢复等) 会导致系统服务中断, 无法直接应用于边缘系统. 为了解决上述问题, 本文深入研究 KubeEdge 边缘系统老化问题, 提出了 GIP-MI 综合方法来解决边缘系统中的老化现象. 该方法主要适用于 KubeEdge 边缘系统中边缘节点侧长期运行的、采用容器技术部署的云边协同计算任务场景 (如图像推理任务). 考虑到 KubeEdge 在云边协同、设备管理和边缘自治等机制上的长期运行影响主要体现为边缘端节点资源占用和任务响应性能的变化, 本文以 KubeEdge 系统的边缘端节点为粒度, 重点研究运行在该类节点上的边缘侧核心组件 EdgeCore 的运行情况 (含 Edged、DeviceTwin、EdgeHub 等核心模块运行及云端组件 CloudCore 的协同通信), 并选取该类边缘节点的 CPU 使用率、内存使用率以及任务平均响应时间作为节点运行的老化观测指标, 用于刻画节点 KubeEdge 边缘系统的软件老化过程. 当某个节点被判定为老化时, 通过任务卸载将部分任务卸载到其他边缘节点或云端, 减轻该节点负载并恢复其运行状态. 该方法充分考虑了数据在时间维度上的动态性及其在空间维度上的关联性, 核心内容包括老化预测、老化状态判定以及制定有效的软件抗衰策略. 本文贡献总结如下.

(1) 针对传统单步预测方法忽视边缘系统指标间关联性及长期趋势的缺陷, 提出一种基于 GCN-Informer 的长序列老化预测方法. 该方法考虑多个系统指标间的空间相关性, 通过引入图卷积网络 GCN (graph convolutional network) 来捕捉复杂的时空关系, 并结合 Informer 模型的自注意力机制进行时序预测, 为边缘系统的老化预测提供了一种新途径.

(2) 针对传统基于阈值或单一时间点分类易受数据波动干扰造成误判的问题, 提出基于 ParNet 模型的老化状态判定方法. 该方法通过结合数据在时间维度上的上下文信息, 以时序数据切片的方式保留更多系统状态信息, 并利用并行网络 ParNet 进行老化状态判定. 通过这种方法, 模型能够更加准确地识别边缘系统的老化状态.

(3) 针对传统软件抗衰策略在边缘系统中引发的服务中断问题, 提出一种基于 MOEA/D-IFM 算法的轻量级抗衰策略. 该方法将任务卸载建模为多目标优化问题, 同时优化时延、依赖度、资源适应度与释放收益这 4 个目标, 并采用信息反馈机制提升解集质量与收敛速度. 通过容器迁移实现任务卸载, 在不停机的情况下恢复系统状态, 使边缘系统更有效地应对老化问题.

1 相关工作

目前, 越来越多的老化研究正聚焦于软件老化预测、抗衰执行效果分析以及其他相关问题[4], 并涉及诸多场

景[5−7]. 同时, 云场景和边缘场景中的老化现象正受到越来越多的关注.

1.1 老化预测相关研究

Yakhchi 等人[8]通过比较几种经典算法, 例如线性回归、最小均方、高斯过程、树模型回归、决策树、序列最小化优化和多层感知器 (multilayer perceptron, MLP), 发现 MLP 算法具有最佳的预测准确性. Shi 等人[9]提出一种基于集成变分模态分解 (variational mode decomposition, VMD)、自回归差分移动平均 (autoregressive integrated moving average, ARIMA) 和双向长短期记忆 (bidirectional long short-term memory, BiLSTM) 网络的混合模型 (VMD-ARIMA-BiLSTM) 来预测软件老化问题. 该模型先通过变分模态分解将原始资源利用率分解为平稳时间序列和非平稳时间序列, 再利用移动平均自由回归和双向长短时记忆网络的优势分别预测平稳和非平稳序列, 最后重建预测结果以获得最终结果. Jia 等人[10]使用阈值自回归 (threshold autoregressive, TAR) 模型来预测软件系统的可用内存, 与其他自回归 (autoregressive, AR) 模型相比, 该模型能更准确地预测老化过程. 孟海宁等人[11]提出基于整合自回归差分移动平均和循环神经网络 (recurrent neural network, RNN) 组合模型 (ARIMA-RNN) 的软件老化预测方法, 该组合模型可提高预测精度和收敛速度. 随后, 孟海宁等人[12]又提出一种云服务器老化预测方法 SVs-GFF, 该方法利用统计特征指标提取数据, 采用支持向量回归进行处理, 然后通过高斯函数拟合模型预测老化曲线, 并使用优化算法选择最佳曲线.

1.2 抗衰策略相关研究

Ning 等人[13]提出一种基于双粒度检验的软件抗衰策略, 此策略减轻了两级软件老化的负面影响, 实验结果表明, 双粒度软件的抗衰比传统的单级软件抗衰策略更为有效. Zheng 等人[14]在系统可达流遵循马尔可夫可达过程 (Markovian arrival process, MAP) 的环境下, 对 6 种软件抗衰策略进行了数值比较, 结果表明基于等待时间的抗衰策略更优. Zheng 等人[15]提出通过采用相位扩展方法来对两个相似的软件抗衰模型的瞬态区间可靠性进行评估, 结果表明适度的相位扩展对分析抗衰模型的瞬态行为是有效的. Carnevali 等人[16]提出一种软件抗衰的非马尔可夫模型, 该模型能够制定结合时间触发和事件触发抗衰的混合抗衰策略. Torquato 等人[17]提出一种基于虚拟化系统可用性模型的安全评估方法论, 通过虚拟机迁移实现虚拟机管理器 (virtual machine manager, VMM) 抗衰, 确定合适的抗衰方案以达到所需的安全风险和可用性水平.

1.3 边缘系统中的软件老化相关研究

当前大多数老化研究针对以云平台为代表的分布式复杂软件系统. 云中的老化现象可能会对用户感知的服务质量产生负面影响. Liu 等人[18]提出一种结合自回归差分移动平均模型 (ARIMA) 和 LSTM (long short-term memory) 模型的混合老化预测方法, 通过对云服务中的时间序列数据进行分析, 能够准确预测云软件系统中的资源消耗, 并确定适当的抗衰时间. Jia 等人[19]提出一种基于分解的老化预测框架, 通过应用基于局部加权回归的季节趋势分解过程 (seasonal-trend decomposition procedure based on loess, STL) 方法和门控循环单元 (gated recurrent unit, GRU) 神经网络, 对云中的内存资源消耗进行预测. 此外, 边缘系统中的软件老化问题也会降低系统可用性. Andrade 等人[20]通过应用多种统计技术对系统在压力测试下的退化趋势进行分析, 并发现云端和边缘环境中的吞吐量和可用内存资源都存在退化趋势. Andrade 等人[21]指出, 由于边缘设备的资源稀缺问题, 软件在长时间运行后会出现老化现象且影响显著, 作者使用确定性随机 Petri 网 (deterministic and stochastic Petri net, DSPN) 来定量分析软件老化现象对使用边缘计算的系统的影响, 对一个工厂的传感器数据分析系统性能的下降趋势, 结果表明在边缘计算系统中十分有必要关注软件老化问题. Wang 等人[22]提出一种基于概念漂移的检查点重启方法 (CDCrest), 用于边缘服务的抗衰, 一旦检测到边缘服务的异常, CDCrest 便采用检查点重启机制来确保边缘服务迅速恢复到健康状态. Docker 是一个用于开发、交付和运行应用程序的开放平台, 便于容器的创建和使用, 常用于云边环境中的任务部署. Vinícius 等人[23]针对 Docker 平台中的软件老化问题, 提出一种系统性能评估和资源消耗预测方法. 他们使用压力-等待-抗衰 (stress-wait-rejuvenation, SWARE) 方法检测老化迹象, 并在单次实验中评估软件抗衰策略的有效性.

在以往的老化研究中, 关于边缘系统老化的研究较少. 大多研究只验证了边缘系统中的老化现象, 而缺乏对边

缘系统老化状态识别的研究. 鉴于边缘系统的高实时性, 现有的重启等抗衰方法无法直接应用于边缘系统. 这主要体现在以下 3 个方面: 一是周期性重启、检查点恢复和虚拟机迁移等典型抗衰手段往往会引入明显的服务中断, 而 KubeEdge 边缘节点上运行的大多是对时延敏感、需要频繁与终端交互的任务, 简单套用此类策略难以满足实时性要求; 二是虚拟机迁移、热备节点等机制依赖额外的计算与存储冗余, 与资源受限的边缘节点环境存在天然冲突, 不当使用甚至可能加剧资源紧张并加速老化; 三是许多抗衰方法隐含节点同质、任务可在任意节点间迁移的假设, 而 KubeEdge 场景下部分任务与本地网络和设备环境高度耦合, 直接迁移会面临设备访问路径变化等问题. 本研究基于边缘智能场景中 KubeEdge 平台系统软件老化问题. 首先, 预测边缘系统资源的未来趋势; 然后, 基于该数据判定老化状态; 若系统进入老化状态, 则采用任务卸载作为轻量级抗衰策略恢复系统. 本文提出的 GIP-MI 方法充分考虑了边缘系统的高实时性需求, 旨在以最小停机代价使系统恢复到健康状态.

2 GIP-MI 方法

本文提出的 GIP-MI 方法由 3 个相互关联的核心组件构成, 总体架构如图 1 所示. 系统首先在边缘节点上持续采集 CPU 使用率、内存使用率和任务平均响应时间等指标, 并将得到的时间序列送入 GCN-Informer, 对未来一段时间的指标变化进行预测; 然后将预测结果输入 ParNet, 判定节点是否已经进入或即将进入老化状态; 当某个节点被判定为老化时, 再基于 MOEA/D-IFM 生成任务卸载方案, 将部分任务迁移到其他边缘节点或云侧, 减轻该节点负载并恢复其运行状态. 上述 3 个组件串联运行, 在实际系统中依次完成监控、预测、判定和卸载, 并不断循环. 其中, 老化预测与判定模块部署在监控节点/云端, 负责接收各边缘节点上报的系统指标并统一生成卸载决策, 边缘节点仅执行指标上报和任务卸载操作.

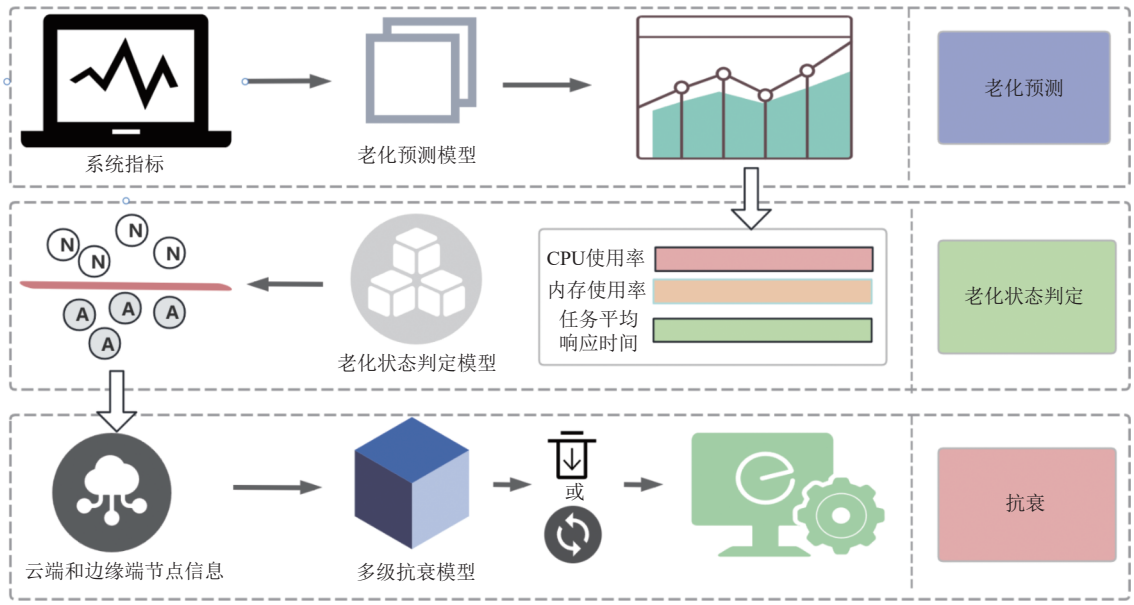

图 1 GIP-MI 方法总体框架

2.1 基于 GCN-Informer 模型的老化预测方法

在边缘系统从稳健状态到老化状态的过程中, 首要任务是准确预测各种系统指标的未来趋势. CPU 使用率、内存使用率以及任务平均响应时间被用作老化预测与判定的指标. 这些指标能够更准确地反映系统的当前状态.

传统的老化预测方法主要采用单步预测, 其仅关注单个时间点的预测结果, 忽略了系统在未来一段时间内的资源变化趋势. 此外, 这些方法通常依赖单一系统指标进行预测, 忽视了各指标之间的关联性. 本研究提出一种名为 GCN-Informer 的新方法, 通过集成图卷积网络 (graph convolutional network, GCN)[24]对现有 Informer 模型[25]进

行改进. 虽然 Informer 模型已在处理长序列预测问题上展现出优势, 尤其体现在其处理长期依赖关系的能力方面, 但通过引入 GCN 模块, 可以进一步提升模型捕捉多元时间序列中潜在空间关系的能力, 从而提高预测精度. 本文采用 GCN-Informer 来拟合系统未来的资源变化, 为系统老化状态的识别提供数据基础, 该方法架构如图 2 所示.

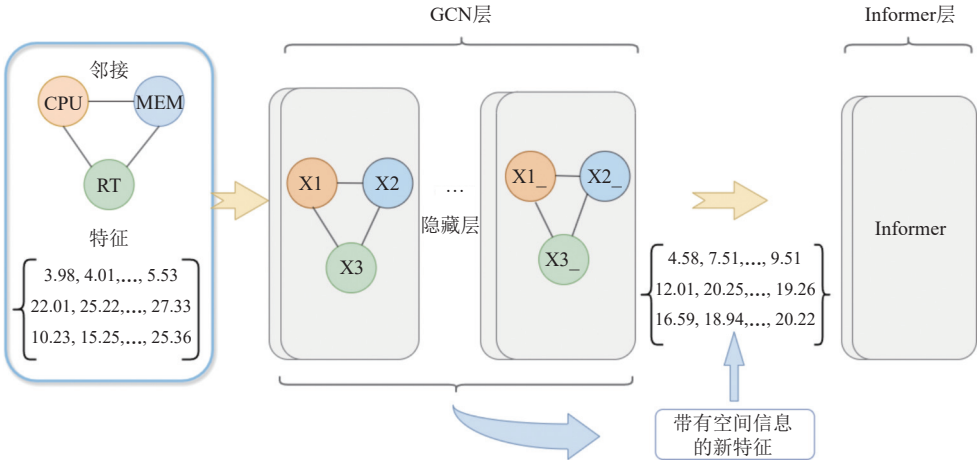

图 2　基于 GCN-Informer 模型的长序列老化预测方法

2.1.1　空间特征提取

为获取系统指标序列中隐含的空间关联特性, 本研究使用图卷积网络进行特征提取. GCN 是一类专用于处理图结构数据的神经网络模型, 其核心在于通过利用节点间的拓扑连接关系, 建模特征在空间上的相互依赖与动态交互机制. 该方法通过聚合每个节点的邻域信息, 对其特征表示进行重构与增强, 从而生成更适合后续任务的高层次特征. 通过对从边缘系统收集的系统指标数据进行分析, 发现各指标间存在较强的相关性, CPU 使用率、内存使用率、任务平均响应时间这 3 个系统指标之间的相关性热图如图 3 所示.

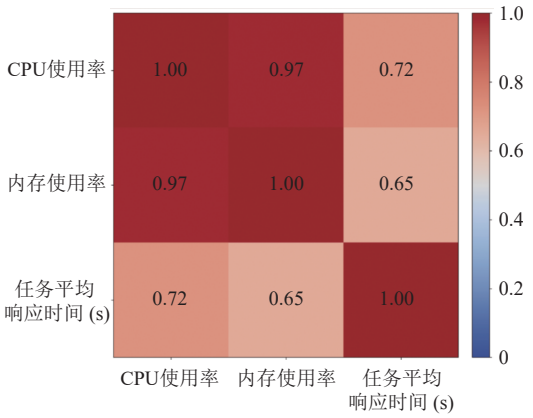

图 3　系统指标相关性热图

可以从图中观察到 CPU 使用率与内存使用率、CPU 使用率与任务平均响应时间、内存使用率与任务平均响应时间之间显著的正相关, 也就是说, 多指标间存在显著协同变化趋势, 通过提取各个指标之间的关联关系可以指导模型训练, 有助于提高预测准确度. 在本文的 GCN-Informer 模型中, GCN 用于整合数据中各指标间的相关性, 从而更好地理解数据的空间结构. GCN 的输入包括两部分: 邻接矩阵和特征矩阵. CPU 使用率、内存使用率和任务平均响应时间抽象为 3 个图节点, 基于指标间的相关性构建邻接矩阵. 若两个指标正相关或者负相关, 则对应两个节点之间存在一条边, 并将指标之间的相关性系数作为邻接矩阵中对应位置权重. 具体实现中, 我们在整个观测

序列上计算各指标两两之间的线性相关性系数, 并直接将其作为带权邻接矩阵的边权, 未额外设置人为剪枝阈值, 以保留 3 个核心指标之间的全部相关关系. 特征矩阵即为每个节点 (指标) 上的数值变化, 本研究使用了 3 个系统指标作为特征, 故特征矩阵的大小为 3×n, 其中 n 代表序列的长度.

2.1.2 时间序列拟合

在获得数据的空间特征之后, 为了对时间序列数据进行建模与预测, 本文采用 Informer 模型提取数据的时间特征并进行预测. 该模型的核心在于借助自注意力机制建模长程依赖, 结合全局与局部注意力, 有效改善了长序列中信息稀疏和计算负担大的问题. 在 GCN-Informer 模型中, 经 GCN 融合多变量空间信息后得到的特征将作为 Informer 的输入. 对于长度为 n 的输入序列, 首先在每个时间步上利用 GCN 对 3 个指标的节点特征进行聚合, 得到包含空间依赖信息的新特征向量, 再将各时间步的新特征按时间顺序拼接成 n×d 的特征序列, 作为 Informer 编码器的输入. Informer 模型采用编码器-解码器结构, 能够自适应地提取时间序列中的有效特征信息. 编码器负责提取时间特征, 解码器据此完成预测, 从而准确把握序列中的趋势与周期变化, 提升老化预测的精度与稳定性. 从计算复杂度角度来看, 设时间序列长度为 L、特征维度为 d, 系统指标数为 N (本文中 N=3, 对应 CPU 使用率、内存使用率和任务平均响应时间). GCN 部分在每个时间步仅在少量指标之间构图并进行图卷积, 节点数固定且很小, 其额外开销近似为随时间步数线性增长的常数因子; Informer 部分在长序列建模时采用 ProbSparse 自注意力机制, 相较于标准 Transformer 自注意力在序列长度 L 上的 O(L^2) 时间复杂度, 可将注意力计算的主要开销降低至接近 O(Llog$L$) 量级, 从而在保证预测精度的同时兼顾长序列预测的运算效率与资源消耗.

2.2 基于 ParNet 模型的老化状态判定方法

GCN-Informer 预测获得的 CPU 使用情况、内存使用情况以及任务平均响应时间这 3 个系统指标体现了系统未来一段时间的资源变化趋势, 对这些数据进行老化状态判定, 可以提前感知系统老化状态.

目前, 基于测量数据识别系统老化状态的研究较为有限, 主要侧重于对单一时间点的系统指标进行分类. 而实际在边缘系统长时间运行的过程中, 数据模式往往复杂多变. 如果只依赖某个单一时刻的系统指标进行状态判定, 容易忽视系统资源占用与性能退化的累积过程, 从而在某些系统波动较大的情况下出现误判. 本文充分考虑了数据在时间维度上的动态性, 即某个时间点前后系统的资源变化情况, 采用基于 ParNet[26]的深度学习方法来识别当前运行状态. 如图 4 所示, 将特定时间窗口大小的数据切片输入 ParNet 中判断当下系统处于正常还是老化状态.

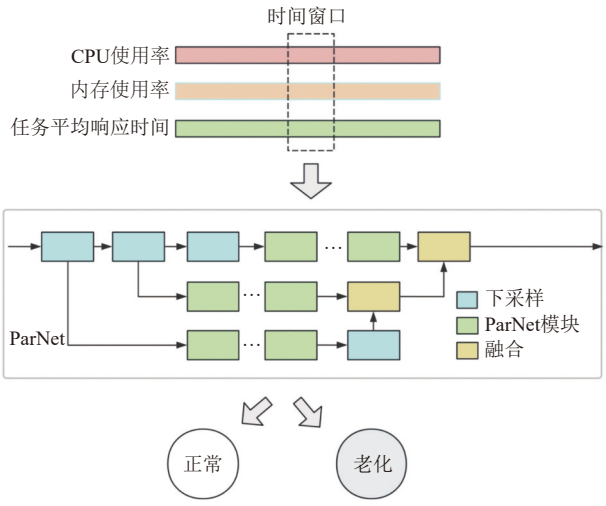

图 4 基于 ParNet 的老化状态判定方法

2.2.1 数据切片与处理

老化状态是一种连续状态, 单一时间点的信息无法充分反映系统的老化状态. 故本研究以数据切片的方式对

CPU 使用率、内存使用率和任务平均响应时间这 3 个指标数据进行标签化处理和分类, 如图 5 所示.

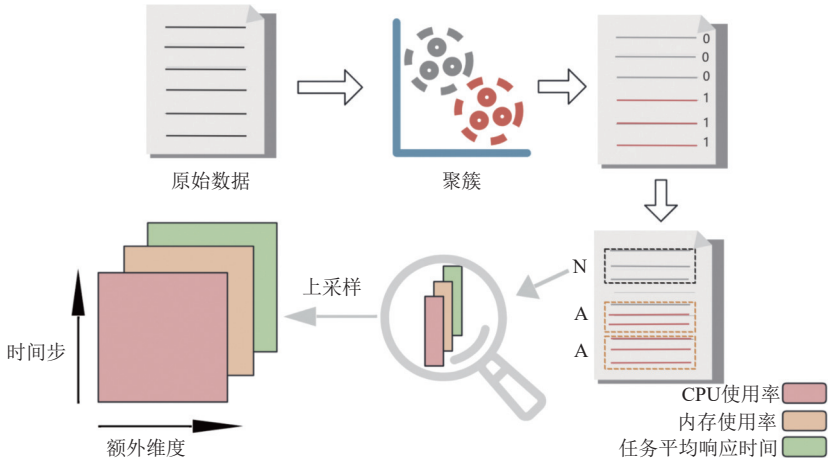

图 5 老化状态判定数据切片处理

首先, 结合系统实际运行状态, 采用聚类方法对数据进行标签化处理, 将其分为正常状态和老化状态, 生成单个时间点的标记数据. 随后, 采用滑动窗口技术对数据进行切片, 根据预定义的时间窗口将源数据划分为多个切片, 然后根据窗口内每种标记类型的数量对这些切片进行标记. 在此阶段, 数据转化为大小为 3×n 的带标签的切片, 其中 3 代表通道 (3 个指标), n 代表窗口大小. 这种切片包含了各指标在时间维度上的变化趋势, 具有更全面的特征信息, 能够更准确地描述系统当前状态. 为了提高模型分类精度, 将 CPU 使用率、内存使用率、任务平均响应时间这 3 个指标作为输入张量的 3 个通道, 并给数据切片增加额外的维度. 同时, 采用双线性插值进行上采样, 将其转化为 3×224×224 大小的数据切片, 用于老化状态判定.

2.2.2 基于 ParNet 的系统老化状态判定

ParNet 是一种并行化网络结构, 通过多条并行分支处理不同分辨率和尺度的特征, 后期融合以全面刻画输入数据. 对于需同时捕捉短期和中长期特征的老化检测任务, 该并行模式能以较小参数规模覆盖更多特征分布, 充分利用数据的时间与空间相关性. 其核心处理流程包含以下两个关键环节.

(1) 不同分辨率特征提取: 输入数据切片同时送入多个并行浅层网络分支, 各分支感受野不同, 即高分辨率分支关注局部细节和短期波动 (如 CPU 使用率瞬时尖峰); 低分辨率分支侧重全局趋势和长期模式 (如内存使用率在窗口时间内的持续缓慢上升), 与此同时捕捉系统状态的微观瞬时变化和宏观演变趋势.

(2) 系统状态特征融合: 各分支处理后的特征图在网络后端融合, 整合局部细节与全局趋势特征, 形成更全面、鲁棒的特征表示, 为最终二分类标签映射提供可靠依据. 得益于这种浅层并行结构, ParNet 的参数规模和计算量均明显低于常见的大型图像网络, 推理过程中只需较少的浮点运算, 适合部署在监控节点等算力相对充足的一端, 不会给资源受限的边缘节点带来额外显著负担.

处理后的数据切片包含连续时间点上的性能指标记录, 其特征信息更为丰富. 在完成切片与插值操作后, 每个切片均被标注为“正常态”或“老化态”, 并转化为对应的数据张量. ParNet 首先在该切片数据上进行训练, 将维度为 3×224×224 的数据输入网络, 以学习不同性能指标随时间变化所形成的特征模式. 在推理阶段, 我们对新采集的监测数据统一采用长度为 21 的时间窗口进行滑动切片, 并将切片后的序列输入已训练好的模型, 以预测系统当前状态. 其中 224×224 的分辨率参考了常见图像分类网络的输入尺寸, 便于复用成熟的卷积结构并控制模型的参数规模和计算开销, 该分辨率仅用于将窗口内的时间步和特征通过插值映射到统一的输入尺寸, 不改变原始指标的含义.

2.3 基于 MOEA/D-IFM 算法的抗衰策略生成方法

传统的软件抗衰研究大多采用不同粒度的重启或检查点恢复等技术, 这类操作往往会引起服务中断, 难以适用于承载实时任务的边缘计算系统. 针对该问题, 本文提出一种基于任务卸载的轻量级抗衰策略来恢复系统状态.

如图 6 所示, 若边缘系统进入老化状态, 将通过 MOEA/D-IFM 算法[27]计算最优任务卸载策略, 进行实时容器迁移, 再次判定系统状态, 并重复上述操作, 直到任务全部卸载完毕或系统恢复正常状态. 若任务全部卸载后系统仍处于老化状态, 则最终执行系统重启. 该卸载机制是重启操作的必要前提, 需确保系统中已无任务运行, 从而避免对终端用户体验造成负面影响. 与传统的重启类抗衰方法相比, 本文所采用的任务卸载策略属于轻量级恢复方式, 可以最大程度保证边缘系统的服务质量. 另外, 边缘系统通常运行多个任务, 确定卸载哪些任务以及卸载到云端或边缘的哪个节点是本研究的核心问题, 这需要考虑多种因素.

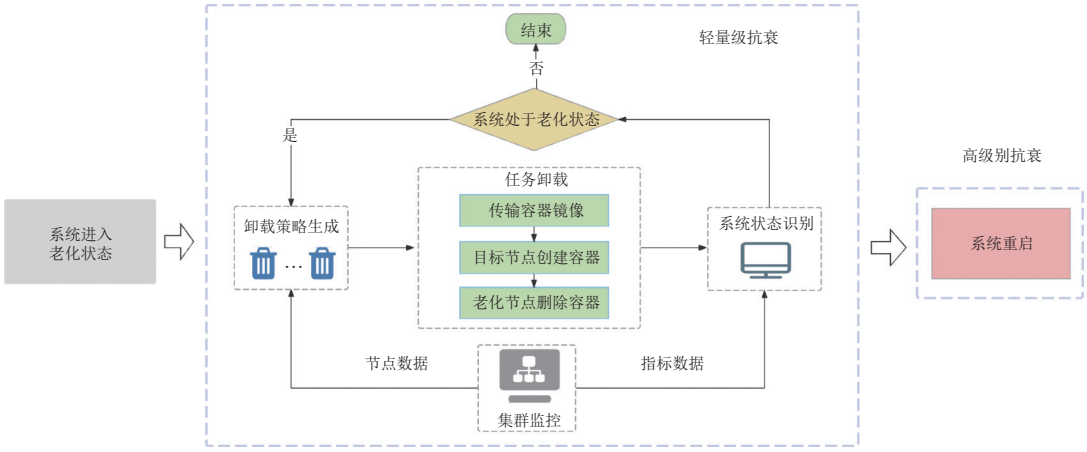

图 6 基于 MOEA/D-IFM 算法的抗衰流程

多目标问题广泛应用于现实生活中, 其难点主要在于各目标之间往往相互冲突, 因此很难得到最优的解决方案. 在任务卸载策略的制定中, 由于需要考虑节点资源、传输时间等多个方面因素, 难以找出一个最优的卸载方案, 因此该问题可抽象为一个多目标优化问题. 采用多目标优化的方法求解, 目的是找出尽可能多的 Pareto 解, 当有多个目标时, 由于存在目标之间的冲突和无法比较的现象, 某个解可能在一个目标上表现优越, 而在另一目标上表现较差. 若在不削弱任何其他目标的情况下无法进一步改进某一目标函数, 则称该解为非支配解或 Pareto 解.

MOEA/D (multi-objective evolutionary algorithm based on decomposition) 是一种多目标进化算法, 其核心思想是基于分解策略, 将原始多目标问题转化为一系列单目标子问题, 并并行地对这些子问题进行求解. MOEA/D-IFM 在 MOEA/D 基础上引入了信息反馈模型 (information feedback model, IFM)[28], 通过将历史迭代中的信息反馈至当前更新机制, 增强算法在演化过程中对解空间的引导能力. 该策略有助于促进优质解的生成, 加速解空间向 Pareto 前沿收敛, 从而提升算法的收敛速度与解集质量.

综上所述, 本研究将使用 MOEA/D-IFM 算法制定任务卸载策略, 共考虑 4 个目标函数, 包括最小化任务时延、最大化资源适应度、最大化资源释放收益以及最小化任务依赖度. 在多目标建模中, 将任务时延定义为数据传输时间与任务执行时间之和, 以衡量卸载后端到端响应效率; 资源适应度根据目标节点剩余 CPU/内存与任务资源需求的差值衡量卸载后是否仍处于合理负载区间, 以避免新的热点节点产生; 资源释放收益使用任务在老化节点上占用的 CPU 与内存比例进行度量, 优先卸载资源占用较大的任务以快速缓解老化节点的资源压力; 任务依赖度通过 3 个等级刻画不同任务对本地设备和时延的敏感程度, 并在优化时通过惩罚权重避免优先迁移高度依赖本地环境的任务. 任务时延包括数据传输时间与任务执行时间, 该目标值越小, 代表任务卸载到目标节点后的响应效率越高. 资源适应度指目标节点剩余资源与任务卸载所需资源之差, 本文希望任务卸载后尽可能保证目标节点的负载处于一个正常状态, 避免对目标节点造成影响, 故该值越大越好. 资源释放收益是指当前卸载的任务所占资源比例, 本文希望尽可能优先卸载资源占用高的任务, 这样可能更有助于让系统退出老化状态. 任务依赖度是指任务对当前节点的依赖程度, 部分任务可能需要实时与终端设备交互, 对时延敏感, 所以只能留在边缘端, 不宜卸载至远端云端. 在卸载策略生成中, 我们根据任务依赖度划分优先级, 优先卸载依赖度低的任务, 尽量避免迁移高度依赖

本地设备的任务. 因此本文将其依赖程度划分为 3 个等级, 等级越低则表示任务对时延容忍度越高, 对边缘端的依赖程度越低. 卸载策略优先选择依赖度低、对时延不敏感的任务, 以尽可能最小化服务中断风险. 系统重启仅作为最终恢复手段, 确保在极端老化情况下系统仍可恢复可用性. 该设计通过分层决策机制, 在抗衰效率与服务连续性之间实现平衡, 克服了传统单一策略的局限性.

本文所使用的 MOEA/D-IFM 算法把“选择某个任务卸载到云端某个节点或者其他边缘节点”这一卸载策略抽象为个体, 考虑优化 4 个目标, 并确保在一定约束条件下完成卸载策略的生成, 算法完整执行流程如算法 1 所示. 其中, *tasks* 包含任务优先级、CPU/内存需求、数据大小等属性; *Nodes* 包含边缘/云节点的剩余资源 (CPU、内存、带宽) 及类型; 交叉和变异概率由较大的初始值线性减小至较小的终值, 前期增强全局搜索、后期加强局部收敛, 便于在探索与收敛之间取得平衡.

算法 1. MOEA/D-IFM.

输入: 任务集合 *tasks*, 节点集合 *Nodes*, 种群规模 *pop_size*, 最大迭代次数 *gene_size*, 交叉最大概率 *pc_max*, 交叉最小概率 *pc_min*, 变异最大概率 *pm_max*, 变异最小概率 *pm_min*, 邻居数量 *T*;
输出: Pareto 最优解集 *EP*.

1. Loading *tasks* and *nodes*; #读取任务和节点, 构建满足资源约束的任务-节点分配组合
2. Generate initial population *pop* of size *pop_size* and calculate fitness; #随机生成可行解并计算原始适应度
3. Generate 4-dimensional weight vector set *lambda*; #生成均匀分布的权重向量集合
4. Build neighbor index set *B* with *T* nearest neighbors from *lambda*; #计算最近邻并形成邻居索引集
5. Initialize reference point *Z*; #取各目标函数的最小值
6. Function operator(*parent*1, *parent*2, *pc*, *pm*):
7. **if** random.random() < *pc* **then**:
8. *p* = crossover(*parent*1, *parent*2) #交叉操作, 用于在两个父代卸载方案之间重组任务-节点映射关系, 生成新的候选解
9. **if** random.random() < *pm* **then**:
10. *individual* = mutation(*p*) #变异操作, 在满足资源约束的前提下对候选解做小幅随机扰动, 以增加种群多样性并降低陷入局部最优的风险
11. Return *individual*;
12. Function Tchebycheff(*x*, *z*, *lambda*):
13. Calculate Tchebycheff value; #返回标量聚合值, 用于比较解的优劣
14. Function Tri_Dominate(*Pop*1, *Pop*2):
15. Return the dominance relationship between two solutions; #当且仅当 *Pop*1 在所有目标上小于等于 *Pop*2 且至少一个目标严格更优时
16. Procedure MOEA/D-IFM(*gene_size*, *pop_size*):
17. **for** *generation* = 1 to *gene_size* **do**
18. *pc* = *pc_max* – (*pc_max* – *pc_min*) × *generation* / *gene_size*;
19. *pm* = *pm_max* – (*pm_max* – *pm_min*) × *generation* / *gene_size*;
20. **for** *i* in *population* **do**
21. *j*, *k* = random(*B*[*i*].size); #在 *B*(*i*) 中选择两个邻居解
22. *pop* = operator(*B*[*i*][*j*], *B*[*i*][*k*], *pc*, *pm*); # 调用 operator 交叉变异生成子代
23. *pop* = IFM(*pop*); #使用 IFM 更新子代
24. Update reference point; #更新参考点

```
    for ind in neighbor do #更新邻居解 (若新解优于邻居解)
      if Tchebycheff(pop[ind], z, lambda[ind])>Tchebycheff(pop, z, lambda[ind]):
        pop[ind] = pop;
    if the pop is not dominated by EP then:
      Update EP with pop; #使用非支配解更新 EP
  return EP;
```

3 实验验证与结果分析

本节首先描述实验环境和数据集, 在分析确认边缘系统存在老化现象的基础上, 对 GIP-MI 总体方法中的 3 个核心组件分别开展对比实验分析, 最后完成 GIP-MI 方法总体抗衰效果的实验分析. 实验结果表明, 所提方法能够有效判定 KubeEdge 边缘系统中的老化现象, 并通过轻量级抗衰方法恢复边缘系统的性能.

3.1 实验环境设置与数据集

当前针对边缘系统老化现象的研究相对有限, 缺乏满足实验场景和需求的公共数据集, 因此本研究自主构建了一个边缘系统老化数据集. 具体地, 基于 KubeEdge 架构部署边缘计算实验环境, 通过配置任务负载并持续监测系统运行状态, 实现多维度指标的采集, 最终构建出一个具备时序特征的系统老化数据集. 在该平台上, 我们重点监控边缘节点的 CPU 使用率、内存使用率和任务平均响应时间, 用以量化 KubeEdge 集群在持续负载下的运行状态及老化趋势.

本研究使用 5 台 Linux 服务器作为节点, 包括 2 个云节点和 3 个边缘节点, 在云节点和边缘节点上分别部署 Kubernetes 和 KubeEdge, 并配置 Docker 平台用于创建容器. 该平台用于模拟并收集边缘系统老化数据, 同时也作为抗衰操作的实践平台, 各节点的硬件环境与软件配置如表 1 和表 2 所示.

表 1 边缘系统老化数据收集平台硬件配置

节点	处理器个数	内存 (GB)	硬盘 (GB)
cloud-master	8	4	40
cloud-worker	4	4	30
edge_1	4	2	20
edge_2	2	2	25
edge_3	1	2	20

表 2 边缘系统老化数据收集平台软件配置

节点	处理器
操作系统	CentOS 7.9
Kubernetes	v1.18.0
KubeEdge	v1.10.0
Docker	v18.06.1

关于老化数据集的收集, 本文在 3 个边缘节点中选取 1 个边缘节点作为采集对象, 在该边缘节点上创建容器, 容器内运行图像推理任务作为服务器. 本地终端模拟多个用户, 读取一批本地图像数据并持续传输至该服务器进行处理, 一旦服务器完成分类, 结果便返回给客户端. 本研究利用 Prometheus 监控工具和 Grafana 可视化插件, 对该边缘节点上的系统指标进行监控和分析, 在恒定负载和非恒定负载两种设置下跟踪并记录 CPU 使用率、内存使用率和任务平均响应时间等指标随时间的变化, 形成本文使用的边缘系统老化数据集.

3.2 基于容器的边缘系统中的老化现象

为研究边缘系统中的老化现象, 本研究利用 TensorFlow 中预训练的 MobileNetV2 模型进行推理. 该图像推理任务是边缘智能场景下典型的 AI 推理类负载, 具有明显的 CPU/内存消耗和时延敏感性, 有利于在可控环境中诱发并观察软件老化现象. 通过开发 Python 程序在本地模拟用户访问, 借助多线程与服务器建立连接. 该程序从本地读取图像, 发送至服务器进行分类, 并接收返回结果. 本研究采用两种数据收集方式: 恒定负载和非恒定负载. 在恒定负载方式中, 每 2 s 生成 2 个用户, 向边缘服务器发送 10 张图像进行分类. 该负载持续运行共 3 天, 数据每分钟收集一次. 在非恒定负载方式中, 每 1–5 s 随机生成 1–5 个用户, 每个用户随机选择 5–10 张图像发送至服务器. 该负载持续运行 40 h, 数据每 20 s 收集一次. 两种负载条件下, 选定边缘节点的 CPU 使用率、内存使用率和任务平均响应时间这 3 个关键性能的变化情况如图 7 和图 8 所示.

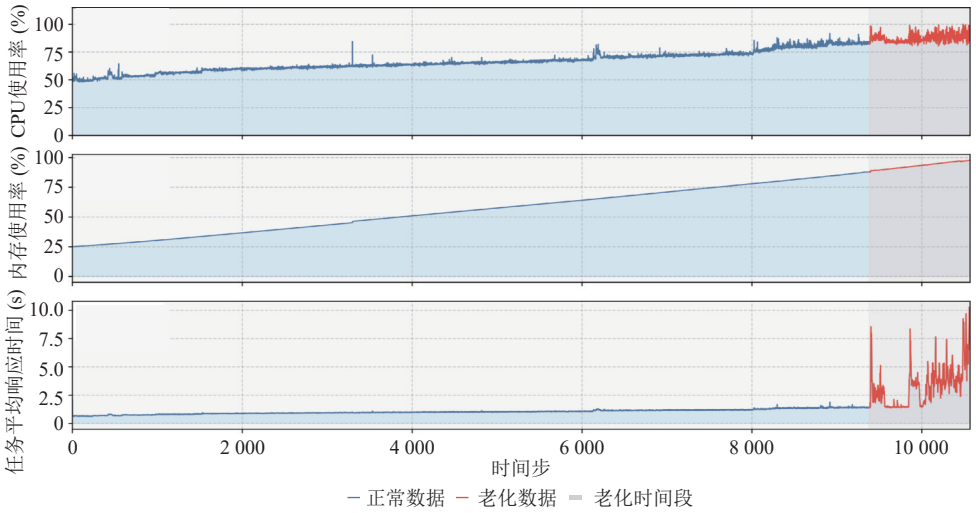

图 7　恒定负载下的边缘系统老化数据

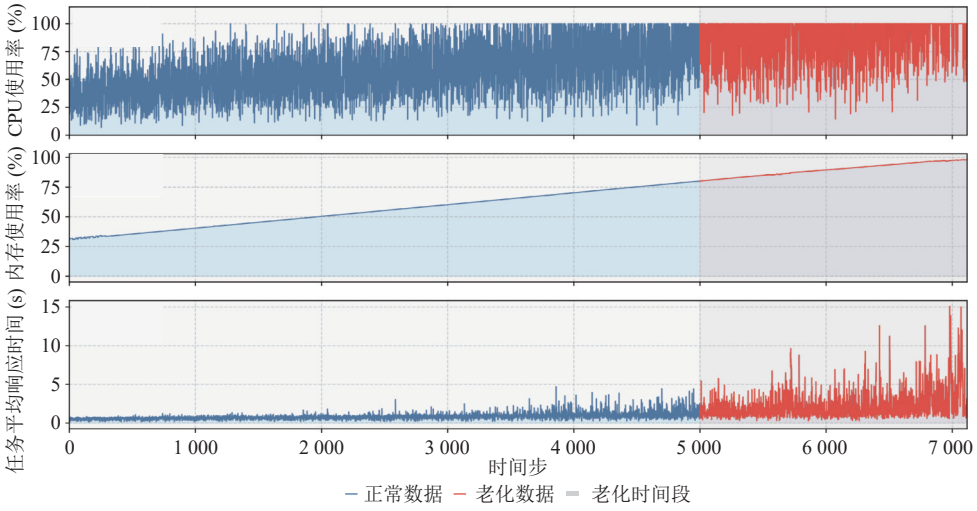

图 8　非恒定负载下的边缘系统老化数据

从图 7 和图 8 可以看出, 在长时间持续负载运行的情况下, 边缘系统的各项指标均明显表现出上升趋势. 具体而言, CPU 使用率表现出较大的波动性, 这很可能与系统运行过程中传输图像的大小和内容不断变化有关. 在非恒定负载条件下, CPU 的波动显著, 其原因可能在于生成随机数量的用户而导致边缘服务器在短时间内建立大量连接, 并且每个连接要处理的图片数量也不固定, 从而引起 CPU 使用率频繁起伏. 相比之下, 内存使用率虽然波动较小, 但依然表现出持续上升的趋势. 该现象可能是由系统中已创建的网络连接或程序执行过程中创建的各种对象资源未被及时释放造成的, 从而导致随着程序的持续运行, 内存消耗量不断累积. 无论在恒定或非恒定负载下, 内存的整体变化趋势基本一致, 波动均不明显.

除资源使用率的变化外, 任务平均响应时间也呈显著增长趋势. 任务平均响应时间从系统初始运行时约 0.6 s 逐渐增长至 1–2 s 左右, 并在系统运行的最后阶段急剧升高至 5–10 s 间的高延迟区间. 该现象表明系统处理和识别图像的速度越来越慢, 造成这一现象的原因很可能是系统的 CPU 和内存使用率持续升高, 导致可用资源逐步减少, 系统运行负荷逐步增大, 最终这种资源紧张的状态引发整体服务性能显著下降. 随着运行时间的延长, 系统整体负载持续累积, 最终 CPU 使用率和内存使用率都接近饱和状态, 并在此高位附近波动. 在该极限状态保持一段

时间后, 系统中的容器开始停止运行, 节点出现宕机现象, 整个系统进入失能状态.

综合上述观察和分析可以得出结论, 在持续高负载长时间运行后, 边缘系统的 CPU 与内存资源被持续消耗, 任务平均响应时间也逐步增加, 当任务平均响应时间持续超过 2 s 时, 即可判定其已经难以支持和终端设备实时交互的基本需求, 用户体验显著下降, 系统服务质量降至较低水平, 此时便可认定边缘系统已明显出现老化现象. 更严重的是, 在数据收集的最后阶段, 通过在云端 master 节点执行“kubectl get nodes”命令后发现, 边缘节点的状态已变为“NotReady”, 这表明边缘系统与云端系统之间的通信链路已完全中断, 边缘系统中负责与云端通信的核心组件 edgecore 已经停止运行, 无法与云端的 master 节点建立正常连接, 最终导致系统宕机. 此时边缘系统老化已极为严重, 完全丧失服务能力. 尽管集群规模有限, 上述结果已表明在真实 KubeEdge 边缘环境中存在可观测的软件老化现象, 可作为后续老化预测和状态判定的基础.

3.3 GCN-Informer 模型的老化预测方法结果分析

本节老化预测实验基于第 3.2 节图像推理任务产生的系统指标数据展开, 模型输入仅为平台层面的 CPU 使用率、内存使用率和任务平均响应时间, 与具体任务语义无关. 为评估所提出的 GCN-Informer 模型在老化预测任务中的长序列预测性能, 需选取合适的评价指标. 鉴于长序列的特点, 本文选择平均绝对误差 (*MAE*) 和均方误差 (*MSE*) 作为评估指标, 二者能够有效衡量预测值与真实值之间的偏差. 其计算方法如公式 (1) 和 (2) 所示, 其中 n 为数据点数量, Y_t 为时间序列中的真实值, y_t 为算法的预测结果.

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|Y_t - y_t| \tag{1}$$

$$MSE = \frac{1}{n}\sum_{t=1}^{n}(Y_t - y_t)^2 \tag{2}$$

为评估模型在不同长度序列上的预测性能, 实验将预测步长分别设置为 5、10、20、50 和 100. 由于本文的模型预测结果主要用于后续老化状态识别, 因此无需设置过大的预测步长, 100 步长的预测结果已能够涵盖足够的系统状态信息, 能够有效识别出系统状态. 为验证模型的有效性, 本文在 *MAE*、*MSE* 两项指标上, 与多种常用于长序列预测的模型 (包括 Autoformer[29]、Reformer[30]、Transformer[31]、DLinear[32]、NHITS[33]和 TimesNet[34]) 及常用于老化预测的 LSTM、ARIMA 模型进行对比. 在恒定负载数据集上的对比结果如表 3 所示, 最优结果加粗显示.

表 3 恒定负载下的长序列老化预测结果

Step	GCN-Informer		Autoformer		Reformer		Transformer		LSTM		ARIMA		Informer		DLinear		NHITS		TimesNet	
	MSE	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*
5	**0.095**	**0.175**	0.131	0.252	0.142	0.264	0.350	0.445	0.142	0.193	1.858	0.556	0.108	0.211	0.413	0.431	0.112	0.191	0.118	0.192
10	**0.127**	**0.231**	0.152	0.278	0.160	0.305	0.367	0.460	0.154	0.258	1.865	0.559	0.173	0.296	0.171	0.295	0.189	0.298	0.203	0.314
20	**0.109**	0.204	0.155	0.265	0.205	0.337	0.442	0.532	0.118	**0.152**	1.912	0.446	0.160	0.273	0.186	0.278	0.167	0.261	0.168	0.262
50	**0.118**	**0.226**	0.215	0.343	0.271	0.408	0.616	0.675	0.455	0.466	3.225	0.830	0.124	0.248	0.238	0.332	0.119	0.231	0.169	0.301
100	**0.231**	**0.334**	0.270	0.377	0.346	0.457	0.536	0.633	0.751	0.802	5.061	0.953	0.314	0.442	0.5991	0.571	0.265	0.398	0.376	0.495

在恒定负载下的实验中, GCN-Informer 模型在所有预测步长下均表现优异, 展现出全面的预测性能优势. 在短期预测方面, 该模型在 *MSE* 和 *MAE* 两项指标上均达到最优. 以预测步长 (step) 为 5 时为例, GCN-Informer 的 *MSE* 为 0.095, 相比模型 Autoformer 的 0.131 降低 27.5%; *MAE* 为 0.175, 相较于 Autoformer 的 0.252 下降 30.6%. 相较于 Transformer, 其 *MSE* 和 *MAE* 下降幅度分别为 73% 和 61%, 体现出其在捕捉局部时序依赖上的高效性. 随着预测步长增大至 100, GCN-Informer 仍保持稳定的性能, *MSE* 和 *MAE* 分别为 0.231 和 0.334, 误差增幅明显低于其他模型. 这种优势得益于模型融合了图卷积网络与改进的稀疏注意力机制, 前者显式建模变量间的空间关联, 后者通过降低计算复杂度增强了长序列全局特征的提取能力, 从而在短期精度与长期稳定性之间取得了平衡.

在参与对比的模型中, Autoformer 和 Reformer 在部分场景中预测性能表现尚可, 但 Reformer 的误差随步长增加上升较为明显. 例如, 当预测步长从 5 增加至 100 时, Reformer 的 *MSE* 从 0.142 上升至 0.346, 增长幅度达 144%. 在传统预测模型中, LSTM 在步长 20 时 *MAE* 达到了 0.152 的局部最优值, 但其 *MSE* 为 0.118 仍高于 GCN-Informer

在相同步长下的 0.109. 当步长增至 50 时, LSTM 的 *MSE* 上升至 0.455, 反映出其对长期依赖建模能力的不足. 而 Transformer 模型因自注意力机制的计算效率问题, 在步长 50 时 *MSE* 和 *MAE* 分别增至 0.616 和 0.675, 预测效果较差. ARIMA 作为线性模型表现最差, 当步长 5 时, 其 *MSE* 高达 1.858, 为 GCN-Informer 的 19.6 倍. 随着步长的增加, 其 *MSE* 和 *MAE* 增长速度较快, 当步长为 100 时, 其 *MSE* 增至 5.061, 这进一步验证说明了深度学习模型在复杂非线性任务中的必要性. Informer (在 GCN-Informer 中去除 GCN 模块) 在各预测步长下的 *MSE* 和 *MAE* 均高于 GCN-Informer, 说明在该场景中引入 GCN 建模多指标空间关联是有必要的. 对于 DLinear, 各预测步长下的 *MSE* 和 *MAE* 都明显高于 GCN-Informer. 例如步长为 5 时, DLinear 的 *MSE* 和 *MAE* 分别为 0.413 和 0.431, 远大于 GCN-Informer 的 0.095 和 0.175; 当步长增大到 100 时, DLinear 的 *MSE* 仍为 0.599, 而 GCN-Informer 仅为 0.231, 说明仅依赖线性趋势分解难以准确刻画该场景下老化过程的复杂变化. NHITS 的整体表现优于 DLinear, 在短期步长 5 时 *MSE* 和 *MAE* 分别为 0.112 和 0.191, 已经比较接近 GCN-Informer, 但在步长 10、20 和 100 时误差仍略高于 GCN-Informer; 在步长 50 时, 两者的 *MSE* 和 *MAE* 几乎相同, 表明 NHITS 在中等预测步长上具有一定竞争力. TimesNet 在各步长上的误差大多介于 DLinear 和 NHITS 之间, 例如步长 50 时的 *MSE* 和 *MAE* 分别为 0.169 和 0.301, 明显高于 GCN-Informer, 也略高于 NHITS. 综上所述, GCN-Informer 在恒定负载条件下具有优于对比模型的预测能力, 能够高效识别到系统的老化趋势.

非恒定负载下的长序列老化预测结果如表 4 所示. 可以观察到在非恒定负载下的实验中, GCN-Informer 在大多数预测场景中仍保持明显优势. 在短期预测中, 其 *MSE* 指标表现尤为突出. 以预测步长为 5 时为例, GCN-Informer 的 *MSE* 为 1.132, 相比 Autoformer 的 1.362 降低 16.9%; GCN-Informer 的 *MAE* 为 0.735, 虽然比 Autoformer 的 0.702 略高, 但综合误差指标仍具备竞争力. 与 Transformer 相比, GCN-Informer 的 *MSE* 和 *MAE* 分别降低了 30.5% 和 16.8%, 体现出更优的复杂时序模式建模能力. 随着预测步长增至 100, GCN-Informer 的 *MSE* 和 *MAE* 分别为 1.458 和 0.863, 误差增幅显著低于对比模型. 例如从步长 5 增至 100 时, Reformer 的 *MSE* 增幅达 30.7%, 而 GCN-Informer 仅增长 28.8%; 其 *MAE* 增幅为 17.4%, 远低于 LSTM 的 106.4%. 这种稳定性得益于模型融合了图卷积网络与改进的稀疏注意力机制, 前者通过动态空间关联建模捕捉负载波动特征, 后者以低计算复杂度机制强化长序列关键信息的提取, 从而在非恒定负载下同步实现预测精度与鲁棒性的协同优化.

表 4　非恒定负载下的长序列老化预测结果

Step	GCN-Informer		Autoformer		Reformer		Transformer		LSTM		ARIMA		Informer		DLinear		NHITS		TimesNet	
	MSE	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*	*MSE*	*MAE*
5	**1.132**	0.735	1.362	**0.702**	1.395	0.796	1.628	0.883	1.765	0.732	6.916	1.131	1.265	0.788	1.283	0.793	1.237	0.837	1.518	0.923
10	1.484	0.841	1.512	0.799	1.573	0.803	1.746	0.846	1.658	**0.754**	4.993	0.966	1.595	0.897	3.052	1.212	1.554	0.879	**1.440**	0.869
20	**1.434**	**0.800**	1.540	0.833	1.671	0.834	1.776	0.848	1.621	0.832	4.038	0.886	1.664	0.862	1.762	0.975	1.629	0.881	1.619	0.862
50	**1.236**	**0.750**	1.414	0.791	1.694	0.921	2.961	1.024	1.891	0.887	5.979	1.057	1.450	0.779	2.052	0.878	1.445	0.817	1.446	0.776
100	**1.458**	**0.863**	1.684	0.908	1.823	1.131	2.672	0.924	3.903	1.511	9.534	2.701	1.586	0.944	2.218	1.054	1.668	0.997	1.649	0.972

其他对比模型中, Autoformer 在步长 10 时的 *MAE* 为 0.799, 要优于 GCN-Informer 的 0.841, 但其 *MSE* 随步长增加快速上升, 例如在步长为 50 时, 其 *MSE* 相比于 GCN-Informer 高出 14.4%. 传统模型中, LSTM 在步长 20 时的 *MAE* 为 0.832, 明显高于 GCN-Informer 的 0.800, 且在步长增至 100 时, 其 *MSE* 急剧上升 140.8%, 显示出其在动态负载变化情况下的适应性不足. Transformer 模型的表现较差, 在步长为 50 时, *MSE* 达到 2.961, 高出 GCN-Informer 的 139.6%. ARIMA 作为线性模型, 表现最弱, 在步长为 100 时, 其 *MSE* 高达 9.534, 是 GCN-Informer 的 6.5 倍, 进一步说明深度学习模型在处理非恒定负载非线性问题中的必要性. Informer 在各预测步长下的 *MSE* 和 *MAE* 同样整体劣于 GCN-Informer, 进一步验证了在复杂负载波动条件下保留 GCN 模块能够带来稳定的预测精度提升. 对于 DLinear, 各预测步长下的 *MSE* 和 *MAE* 也普遍高于 GCN-Informer. 例如步长为 5 时, DLinear 的 *MSE* 和 *MAE* 分别为 1.283 和 0.793, 高于 GCN-Informer 的 1.132 和 0.735; 当步长增大到 10 时, DLinear 的 *MSE* 和 *MAE* 分别升至 3.052 和 1.212, 而 GCN-Informer 仅为 1.484 和 0.841, 误差差距进一步拉大; 在步长为 100 时, DLinear 的 *MSE* 和 *MAE* 仍为 2.218 和 1.054, 明显高于 GCN-Informer 的 1.458 和 0.863, 说明仅依赖线性分解难以稳定刻画该场景

下老化过程的复杂变化. NHITS 的整体表现优于 DLinear, 在步长 5 时, *MSE* 和 *MAE* 为 1.237 和 0.837, 已较为接近 GCN-Informer 的 1.132 和 0.735; 但在步长 10、20、50 和 100 时, 其 *MSE* 和 *MAE* 仍略高于 GCN-Informer, 例如步长 50 时, NHITS 的 *MSE* 和 *MAE* 为 1.445 和 0.817, 高于 GCN-Informer 的 1.236 和 0.750. TimesNet 在各步长上的误差大多介于 DLinear 和 NHITS 之间, 例如步长 5 时, *MSE* 和 *MAE* 分别为 1.518 和 0.923 明显高于 GCN-Informer, 而在步长 10 时, *MSE* 为 1.440, 略低于 GCN-Informer 的 1.484, *MAE* 为 0.869, 略高于 0.841. 综上所述, GCN-Informer 在大多数的预测步长下都具备高精度和强鲁棒性, 为非恒定负载条件下系统的动态老化趋势分析提供了有效解决方案.

3.4 基于 ParNet 模型的老化状态判定方法结果分析

本部分实验采用 ParNet 方法对预测数据进行老化状态判定. 为验证模型分类结果的准确性, 本研究选用准确率 (Accuracy)、精确率 (Precision)、召回率 (Recall) 和 *F*1 分数 (*F*1-score) 作为评价指标, 这些指标有助于全面评估模型在分类任务中的性能. 其中, 准确率用于衡量整体分类正确性, 精确率和召回率更关注分类器的错误类型, 而 *F*1 分数则综合两者, 对分类器性能进行全面评估.

本节通过上述 4 个指标与常用的分类模型进行对比, 具体包括 K 最邻近 (K-nearest neighbor, KNN)、随机森林 (random forest, RF)、支持向量机 (support vector machine, SVM)、高斯朴素贝叶斯 (Gaussian naive Bayes, GNB)、轻量级梯度提升机 (light gradient boosting machine, LightGBM)[35]以及残差网络 (residual network, ResNet)[36], 恒定负载下的老化状态判定结果如表 5 所示.

表 5　恒定负载下的老化状态判定结果

Model	Accuracy	Precision	Recall	*F*1-score
ParNet	**0.9885**	**0.9839**	**0.9912**	**0.9874**
ResNet	0.9770	0.9831	0.9667	0.9741
KNN	0.9572	0.9493	0.9868	0.9677
SVM	0.9090	0.9016	0.9649	0.9322
RF	0.9487	0.9487	0.9736	0.9610
GNB	0.9145	0.9230	0.9473	0.9350
LightGBM	0.9744	0.9740	0.9868	0.9804

从表 5 可以观察到, 本研究使用的 ParNet 模型在准确率、精确率、召回率和 *F*1 分数上均优于其他对比模型. 可以看到 ParNet 的准确率、精确率、召回率以及 *F*1 分数分别为 98.85%、98.39%、99.12% 以及 98.74%, 均优于其他对比模型. 结果表明, ParNet 在数据分类任务中具有出色的分类准确性, 能够有效识别老化样本并最大程度减少错误分类. 相比之下, ResNet 的准确率和精确率分别为 97.70% 和 98.31%, 与 ParNet 较为接近, 表现仅次于 ParNet. 而基于单一时间点数据进行分类的传统模型 (如 KNN、SVM 和 GNB) 在所有指标上均落后于 ParNet. 其中 SVM 准确率和精确率分别为 90.90%, 90.16%, 表现最差, 说明 SVM 在处理高维度、非线性特征丰富的数据时存在局限. 另外, LightGBM 的分类效果较好, 其召回率和 *F*1 分数分别为 98.68% 和 98.04%, 与 ParNet 比较接近, 表现仅次于 ParNet, 但在准确率和精确率上明显落后于 ParNet. 综上, ParNet 在准确率、精确率、召回率和 *F*1 分数上均表现出明显优势, 说明本文采取的通过数据切片进行分类的方式相比于传统基于单个时间点数据进行分类的方式可以保存更多系统状态信息, 从而提高模型分类准确度, 验证了本文方法的有效性.

表 6 为非恒定负载下的老化状态判定结果, 可以观察到 ParNet 在准确率、精确率、召回率和 *F*1 分数上均表现出明显优势. 具体来说, ParNet 的准确率、精确率、召回率和 *F*1 分数分别达到了 98.03%、99.14%、98.33% 和 98.72%, 都优于其他对比模型. 这表明 ParNet 在动态负载场景下仍能保持极高的分类准确性, 其数据切片策略有效提升了对复杂特征的识别能力, 相较于其他对比模型, 可显著减少误判.

对比模型中, LightGBM 的准确率、精确率、召回率和 *F*1 分数分别为 96.59%、96.55%、98.24% 和 97.39%, 表现仅次于 ParNet, 其中召回率与 ParNet 接近, 体现了该模型对于非恒定负载数据具有一定适应能力. ResNet 的准确率、精确率、召回率和 *F*1 分数分别为 95.40%、94.12%、96.49% 和 95.06%, 虽然对比传统模型表现更好,

但仍不如 LightGBM. 在传统模型中, SVM 在召回率上效果较好, 为 98.24%, 但精确率和 $F1$ 分数效果较差, 分别为 94.91% 和 96.55%, 分类稳定性不足. KNN、RF 和 GNB 的整体表现相对较差, 其中 GNB 的准确率和 $F1$ 分数仅为 90.59% 和 92.81%, 进一步表明传统模型在处理动态变化数据时存在明显局限. 综上所述, ParNet 在 4 个指标上都表现出显著优势, 验证了数据切片策略在非恒定负载下的有效性.

表 6　非恒定负载下的老化状态判定结果

Model	Accuracy	Precision	Recall	$F1$-score
ParNet	**0.9803**	**0.9914**	**0.9833**	**0.9872**
ResNet	0.9540	0.9412	0.9649	0.9506
KNN	0.9487	0.9487	0.9736	0.9610
SVM	0.9545	0.9491	0.9824	0.9655
RF	0.9401	0.9480	0.9605	0.9542
GNB	0.9059	0.9220	0.9342	0.9281
LightGBM	0.9659	0.9655	0.9824	0.9739

综上, 无论在恒定还是非恒定负载条件下, 基于 ParNet 模型的老化状态判定方法表现最好, 这表明采用数据切片的方式可以更充分地保留系统状态信息, 结合深度学习模型, 可以显著提高老化判定的精准性.

此外, 为分析窗口大小对判定性能的影响, 本文在恒定负载和非恒定负载两个数据集上将时间窗口长度分别设置为 5、9、13、17、21、25 和 29, 并计算对应的准确率、精确率、召回率和 $F1$ 分数, 如表 7 所示. 实验结果表明, 在上述窗口长度范围内 4 项指标整体均处于较高水平, 随窗口长度变化的幅度相对较小, 说明在这个范围内, ParNet 的判定性能对窗口大小不太敏感; 其中窗口长度为 21 时在两种负载下 4 项指标的综合表现相对更好.

表 7　在恒定和非恒定负载下不同窗口大小的老化状态判定结果

window_size	Accuracy		Precision		Recall		$F1$-score	
	constant	non-constant	constant	non-constant	constant	non-constant	constant	non-constant
5	0.9909	0.9568	0.9924	0.9570	0.9888	0.9569	0.9905	0.9568
9	0.9854	0.9706	0.9879	0.9711	0.9821	0.9706	0.9848	0.9706
13	0.9863	0.9862	0.9886	0.9863	0.9832	0.9862	0.9857	0.9862
17	0.9863	0.9935	0.9886	0.9936	0.9832	0.9935	0.9857	0.9935
21	0.9881	0.9954	0.9901	0.9954	0.9854	0.9954	0.9876	0.9954
25	0.9899	0.9806	0.9916	0.9811	0.9876	0.9805	0.9895	0.9806
29	0.9779	0.9898	0.9820	0.9900	0.9730	0.9899	0.9770	0.9898

3.5　基于 MOEA/D-IFM 算法的抗衰策略生成结果分析

为评估 MOEA/D-IFM 算法在任务卸载决策中的性能, 本文设定了一系列实验比较各算法在仿真数据下制定卸载策略的效果. 本文的实验环境配置基于 EdgeCloudSim 仿真平台, 实验参数包括: 容器数量 M=30, 任务请求 T=100, 边缘服务器数量 N_e=200, 云服务器数量 N_c=100. 本节仿真实验通过在较大规模集群的云边协同场景下比较不同卸载算法在任务平均响应时间和负载标准差等指标上的表现, 用于从算法层面对 MOEA/D-IFM 在较大节点规模下的卸载效果和可扩展性进行补充评估. 仿真数据如表 8 所示.

表 8　任务卸载仿真数据配置

参数	数值
任务计算需求 (MIPS)	1–1 000上随机分布
任务内存需求 (MB)	50–2 000上随机分布
节点CPU规格 (MIPS)	1 000–3 000上随机分布
节点内存剩余 (MB)	1 000–8 000上随机分布
节点带宽 (MB/s)	50–200上随机分布
要传输的任务数据大小 (MB)	50–1 000上随机分布
任务对时延的敏感程度	0, 1, 2

为了评估算法性能, 本文采用归一化 Pareto 前沿的倒数距离 (*IGD*) 和超体积指标 (*HV*) 两个评价指标. 其中 *IGD* 用于衡量算法求得的近似 Pareto 前沿与真实 Pareto 前沿之间的平均距离, 以此反映解集的多样性与均匀性, 具体计算方式如公式 (3) 所示, 其中 *PF* 代表解的真实 Pareto 前沿, *A* 为算法得到的近似 Pareto 前沿, $d(PF, A)$ 代表 *PF* 与 *A* 之间的最小欧氏距离, $|p|$表示分布在真实 Pareto 前沿的个体数量. 算法性能越好, 真实 Pareto 前沿与近似 Pareto 前沿之间的距离越小, *IGD* 数值也随之越小. 另一方面, *HV* 指标通过衡量由 Pareto 前沿解围成的空间大小, 以此反映解集的全局优化能力, 如公式 (4) 所示, 其中 $f(x)$ 代表 Pareto 前沿中的点, *Z* 代表参考点.

$$IGD(PF,A)=\frac{\sum_{i=1}^{|p|}d(PF,A)}{|p|} \tag{3}$$

$$HV=\int_{-\infty}^{Z}f(x)\mathrm{d}x \tag{4}$$

为验证 MOEA/D-IFM 效果, 实验选取 MOEA/D 算法、第 3 代非支配排序遗传算法 (non-dominated sorting genetic algorithm III, NSGA3)[37]和第 2 代强度 Pareto 进化算法 (strength Pareto evolutionary algorithm II, SPEA2)[38]作为基准. MOEA/D 计算高效, NSGA3 擅长维持种群多样性, SPEA2 环境选择突出; MOEA/D-IFM 通过引入动态信息反馈与协同进化, 在保持高效性的基础上增强了应对复杂约束能力, 4 种算法对比结果见表 9.

表 9　4 种多目标优化算法在仿真数据上的 *IGD* 和 *HV* 结果对比

算法	*IGD*	*HV*
MOEA/D-IFM	**0.0103**	**174.423**
MOEA/D	0.0192	174.215
NSGA3	0.0334	172.331
SPEA2	0.0409	15.087

从表 9 观察到, 在 *IGD* 指标方面, MOEA/D-IFM 的值为 0.0103, 要优于其他对比算法, 表现最好, 说明其生成的 Pareto 前沿最接近真实前沿, 且算法在收敛性方面表现较好. MOEA/D 的 *IGD* 值为 0.0192, NSGA3 的 *IGD* 值为 0.0334, 这两种算法虽然相较于 MOEA/D-IFM 略高, 但都要好于 SPEA2 算法. SPEA2 的 *IGD* 值为 0.0409, 明显高于其他算法, 说明其在收敛性方面表现较差, 难以有效逼近真实 Pareto 前沿.

在 *HV* 指标方面, MOEA/D-IFM 的值为 174.423, 同样要优于其他对比算法, 表现最好, 说明其具更优的解集多样性和目标空间覆盖能力. MOEA/D 的表现与 MOEA/D-IFM 十分接近, 其 *HV* 值为 174.215, 说明两者在多样性方面差距较小. NSGA3 要低于前两者, 其 *HV* 值为 172.331. SPEA2 要显著低于其他算法, 其 *HV* 值仅为 15.087, 表明其解集在多样性和整体质量方面存在明显不足. 综合两个指标来看, MOEA/D-IFM 算法均优于其他对比算法. 紧随其后的是 MOEA/D 和 NSGA3 算法, 而相对较差的是 SPEA2 算法, 可能需要通过参数调整或优化来提高算法性能. 上述结果验证了本文所提出的基于 MOEA/D-IFM 的任务卸载策略能够生成适应卸载场景和需求的高质量方案, 为系统抗衰提供了有效手段.

在 MOEA/D-IFM 算法生成任务卸载策略后, 算法会执行任务卸载操作, 本文通过任务平均响应时间和负载标准差两个指标对卸载的效果进行评估. 任务平均响应时间是指容器接收任务到返回结果所经历的时间, 该值越小代表算法效果越好; 负载标准差则用于衡量节点负载均衡度, 其值越小代表负载越均衡、资源利用率越均匀. MOEA/D-IFM 方法与 MOEA/D、NSGA3 算法和 SPEA2 算法在两个指标上的对比结果如表 10 所示.

表 10　4 种多目标优化算法进行任务卸载效果对比

算法	任务平均响应时间 (s)	负载标准差
MOEA/D-IFM	**0.83**	**0.21**
MOEA/D	0.96	0.29
NSGA3	0.90	0.27
SPEA2	1.45	0.42

从表 10 观察到, 在任务平均响应时间和负载均衡两个指标上, MOEA/D-IFM 都要优于对比算法. 具体来说, 在任务平均响应时间方面, MOEA/D-IFM 为 0.83 s, MOEA/D 为 0.96 s, SPEA2 为 1.45 s, NSGA3 为 0.90 s, MOEA/D-IFM 明显优于 MOEA/D 和 SPEA2, 略优于 NSGA3. 在负载标准差方面, MOEA/D-IFM 为 0.21, MOEA/D、NSGA3 和 SPEA2 分别为 0.29、0.27 和 0.42, MOEA/D-IFM 表现最好. 虽然 NSGA3 在任务平均响应时间上与 MOEA/D-IFM 接近, 但其在负载均衡能力方面要比 MOEA/D-IFM 差. SPEA2 在任务平均响应时间和负载均衡两个指标上均表现最差, 不仅无法有效缩短任务平均响应时间, 也没有实现负载均衡分布. 综上所述, 在加速任务完成和平衡节点负载两个方面上 MOEA/D-IFM 均表现最佳, 与其他 3 种算法相比较, 是最优的解决方案, 进一步证明了 MOEA/D-IFM 算法是有效的.

3.6 GIP-MI 的整体抗衰效果分析

在本节实验中, 使用 GIP-MI 框架所生成的抗衰策略进行任务卸载. 由于第 3.5 节的实验已经验证了 MOEA/D-IFM 的有效性, 因此本节实验将从整体验证在执行任务卸载后, 边缘系统能否由老化状态恢复到正常状态. 实验将在 KubeEdge 边缘节点上部署更多类型的任务, 任务描述如表 11 所示.

表 11　应用程序的介绍和来源

程序	程序介绍	程序来源
程序1	程序计数器, KubeEdge平台提供的案例	https://github.com/KubeEdge/examples/tree/master/KubeEdge-counter-demo
程序2	温度感应器, KubeEdge平台提供的案例	https://github.com/KubeEdge/examples/tree/master/temperature-demo
程序3	音频翻译服务 (ATS)	改写文献[39]中的部分代码
程序4	心血管健康监测 (CHM)	改写文献[39]中的部分代码
程序5	图像分类程序	自己编写

在完成以上任务部署, 且边缘节点进入老化状态后, 使用 GIP-MI 框架制定卸载策略进行抗衰操作, 轻量级抗衰执行完成后, 系统的各项指标变化情况如图 9–图 11 所示.

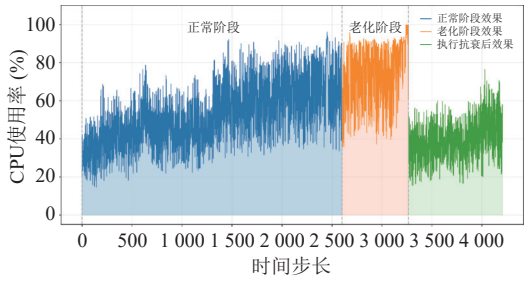

图 9　执行抗衰操作后的 CPU 变化情况

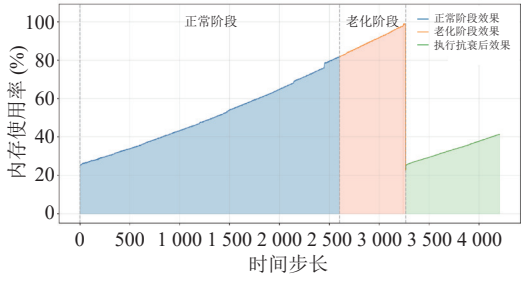

图 10　执行抗衰操作后的内存变化情况

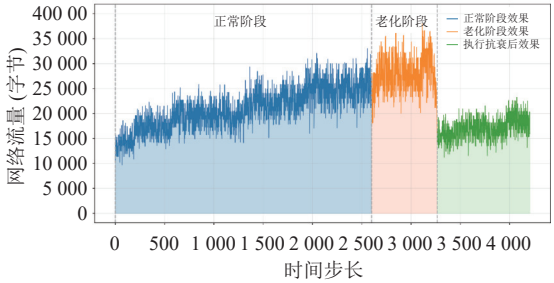

图 11　执行抗衰操作后的网络变化情况

从图 9–图 11 中的实验数据可以观察到 GIP-MI 综合方法的抗衰效果. 可以从 CPU 变化情况图 (图 9) 观察到, 在实施抗衰操作前 CPU 使用率的波动较为明显; 在实施抗衰操作后, CPU 使用率趋于平稳, 并在大部分时间保持

在较低区间. 从内存变化情况图 (图 10) 观察到, 内存使用率数据也表现出类似情况, 在实施抗衰操作前系统内存占用持续攀升, 甚至接近满载; 在实施抗衰操作后, 内存使用率快速下降, 并且稳定在较低水平, 这表明内存管理得到明显提升, 内存泄漏问题得到缓解. 从网络变化情况图 (图 11) 中的网络接收字节数变化可知, 实施抗衰操作可以有效降低网络流量, 减轻系统负载. 综上所述, 基于卸载的轻量级抗衰策略能够有效恢复系统性能, 使系统退出老化状态.

本文还在停机时间和抗衰时间两个指标上对传统基于不同粒度重启的抗衰操作进行对比, 对应实验结果如表 12 所示.

表 12　不同抗衰操作在停机时间与抗衰时间上的比较

抗衰策略	停机时间 (s)	抗衰时间 (s)
任务卸载	**0**	30
重启容器	10	45
重启虚拟机	125	100
检查点恢复	24	**27**

在停机时间方面, 采用 GIP-MI 框架中的轻量级抗衰方法能使系统在不中断服务的条件下完成抗衰操作, 这一点对边缘系统尤为关键, 因为边缘系统中运行的经常是对时延敏感的且需要与终端交互频繁的实时任务, 中断服务将会严重影响用户体验, 因此传统基于不同粒度的重启操作并无法满足该需求.

在抗衰时间方面, 检查点恢复方法的时间为 27 s, 用时最短, 但会伴随较高的停机成本, 与终端设备的交互可能会受影响. 相比之下, 通过任务卸载这种方式需要的抗衰时间仅为 30 s, 与检查点恢复方法接近, 系统可以较快退出老化状态, 同时还不会造成停机. 另外, 重启容器也可以在短时间内恢复系统状态, 但重启虚拟机则需要耗费 100 s, 需要更长的时间. 综上所述, 本文所提出的 GIP-MI 方法能够有效恢复系统状态, 并且在停机时间上表现最好, 在抗衰时间上表现次优, 在两个指标上均表现较好. 整体而言, 本文所提出的 GIP-MI 方法相比于传统方法更适用于保证边缘系统的可靠性. 需要说明的是, GIP-MI 方法的主要计算模块 (包括老化预测、老化状态判定和抗衰策略生成) 均部署在监控节点上, 边缘节点仅承担指标采集和执行卸载操作的开销, 且卸载仅在节点被判定为老化时触发, 因此不会在正常运行阶段持续叠加额外负载.

4 总　结

本文对边缘智能场景中 KubeEdge 平台系统软件老化问题进行了深入研究, 提出了名为 GIP-MI 的综合方法, 为解决边缘系统的老化问题提供了全面的解决方案. 该方法结合了基于 GCN-Informer 模型的老化预测方法、基于 ParNet 模型的老化状态判定方法, 以及作为轻量级恢复策略的基于 MOEA/D-IFM 算法的抗衰策略生成方法. 这些方法在实验中表现良好, 不仅提升了老化预测和状态判定的准确性, 还能有效恢复系统状态. 通过合理预测、识别和处理软件老化问题, 该方法可以有效增强边缘系统的可靠性与稳定性. 在实际应用场景中, 只需将该方案部署到监控节点, 并提供各被监控边缘节点的相关指标, 即可应对边缘系统的老化现象. 未来工作中, 我们将进一步引入设备连接状态、云边通信异常事件等更细粒度的 KubeEdge 运行指标, 并结合不同类型边缘设备与业务场景构建更精细的任务-设备依赖模型, 以更准确反映平台老化过程和业务侧约束; 同时, 推进与运营商合作, 在真实边缘算力环境中落地 GIP-MI, 并评估其在大规模场景下的可扩展性和实用性. 此外, 目前实验主要以图像推理任务为负载, 后续将在保持平台层系统指标建模的前提下, 针对视频流处理、IoT 数据分析等更多类型边缘任务构建数据集并开展老化预测、状态判定与抗衰效果的系统实验, 以验证方法在不同业务模式与负载结构下的适用性和鲁棒性.

References

[1] Khan WZ, Ahmed E, Hakak S, Yaqoob I, Ahmed A. Edge computing: A survey. Future Generation Computer Systems, 2019, 97: 219–235. [doi: 10.1016/j.future.2019.02.050]

[2] Andrade E, Machida F, Pietrantuono R, Cotroneo D. Software aging in image classification systems on cloud and edge. In: Proc. of the 31st IEEE Int'l Symp. on Software Reliability Engineering Workshops. Coimbra: IEEE, 2020. 342–348. [doi: 10.1109/ISSREW51248.2020.00099]

[3] Pietrantuono R, Russo S. Software aging and rejuvenation in the cloud: A literature review. In: Proc. of the 2018 IEEE Int'l Symp. on Software Reliability Engineering Workshops. Memphis: IEEE, 2018. 257–263. [doi: 10.1109/ISSREW.2018.00016]

[4] da Costa JT, de S. Matos R, de Araujo JCT, Maciel PRM. Systematic mapping of literature on software aging and rejuvenation research trends. In: Proc. of the 2021 Annual Reliability and Maintainability Symp. Orlando: IEEE, 2021. 1–6. [doi: 10.1109/RAMS48097.2021.9605775]

[5] Dias D, Machida F, Andrade E. Analysis of software aging in a blockchain platform. In: Proc. of the 2022 IEEE Int'l Symp. on Software Reliability Engineering Workshops. Charlotte: IEEE, 2022. 170–177. [doi: 10.1109/ISSREW55968.2022.00064]

[6] Bai J, Chang XL, Machida F, Trivedi KS, Han Z. Analyzing software rejuvenation techniques in a virtualized system: Service provider and user views. IEEE Access, 2020, 8: 6448–6459. [doi: 10.1109/ACCESS.2019.2963397]

[7] Cotroneo D, de Simone L, Natella R, Pietrantuono R, Russo S. A configurable software aging detection and rejuvenation agent for Android. In: Proc. of the 2019 IEEE Int'l Symp. on Software Reliability Engineering Workshops. Berlin: IEEE, 2019. 239–245. [doi: 10.1109/ISSREW.2019.00078]

[8] Yakhchi M, Alonso J, Fazeli M, Bitaraf AA, Patooqhy A. Neural network based approach for time to crash prediction to cope with software aging. Journal of Systems Engineering and Electronics, 2015, 26(2): 407–414. [doi: 10.1109/JSEE.2015.00047]

[9] Shi FD, Yuan Z, Wang M, Cui J. Research on cloud platform software aging prediction method based on VMD-ARIMA-BiLSTM combined model. Integrated Ferroelectrics, 2023, 237(1): 297–309. [doi: 10.1080/10584587.2023.2192665]

[10] Jia YF, Zhou ZQ, Wu RB. A workload-based nonlinear approach for predicting available computing resources. Journal of Systems Engineering and Electronics, 2020, 31(1): 224–230. [doi: 10.21629/JSEE.2020.01.21]

[11] Meng HN, Tong XY, Shi YK, Zhu L, Feng K, Hei XH. Cloud server aging prediction method based on hybrid model of auto-regressive integrated moving average and recurrent neural network. Journal on Communications, 2021, 42(1): 163–171 (in Chinese with English abstract). [doi: 10.11959/j.issn.1000-436x.2021015]

[12] Meng HN, Tong XY, Xie G, Zhang BB, Hei XH. Cloud server aging prediction method based on RUL and SVs-GFF. Acta Automatica Sinica, 2024, 50(10): 2036–2048 (in Chinese with English abstract). [doi: 10.16383/j.aas.c211112]

[13] Ning GR, Zhao J, Lou YL, Alonso J, Matias R, Trivedi KS, Yin BB, Cai KY. Optimization of two-granularity software rejuvenation policy based on the Markov regenerative process. IEEE Trans.on Reliability, 2016, 65(4): 1630–1646. [doi: 10.1109/TR.2016.2570539]

[14] Zheng JJ, Okamura H, Li L, Dohi T. A Comprehensive evaluation of software rejuvenation policies for transaction systems with markovian arrivals. IEEE Trans.on Reliability, 2017, 66(4): 1157–1177. [doi: 10.1109/TR.2017.2741526]

[15] Zheng JJ, Okamura H, Dohi T. A transient interval reliability analysis for software rejuvenation models with phase expansion. Software Quality Journal, 2020, 28(1): 173–194. [doi: 10.1007/s11219-019-09458-1]

[16] Carnevali L, Paolieri M, Reali R, Scommegna L, Vicario E. A Markov regenerative model of software rejuvenation beyond the enabling restriction. In: Proc. of the 2022 IEEE Int'l Symp. on Software Reliability Engineering Workshops. Charlotte: IEEE, 2022. 138–145. [doi: 10.1109/ISSREW55968.2022.00060]

[17] Torquato M, Maciel P, Vieira M. A Model for availability and security risk evaluation for systems with VMM rejuvenation enabled by VM migration scheduling. IEEE Access, 2019, 7: 138315–138326. [doi: 10.1109/ACCESS.2019.2943273]

[18] Liu J, Tan XY, Wang Y. CSSAP: Software aging prediction for cloud services based on ARIMA-LSTM hybrid model. In: Proc. of the 2019 IEEE Int'l Conf. on Web Services. Milan: IEEE, 2019. 283–290. [doi: 10.1109/ICWS.2019.00055]

[19] Jia K, Yu X, Zhang C, Hu WH, Zhao DD, Xiang JW. Software aging prediction for cloud services using a gate recurrent unit neural network model based on time series decomposition. IEEE Trans.on Emerging Topics in Computing, 2023, 11(3): 580–593. [doi: 10.1109/TETC.2023.3258503]

[20] Andrade E, Pietrantuono R, Machida F, Cotroneo D. A comparative analysis of software aging in image classifiers on cloud and edge. IEEE Trans.on Dependable and Secure Computing, 2023, 20(1): 563–573. [doi: 10.1109/TDSC.2021.3139201]

[21] Andrade E, Machida F. Analysis of software aging impacts on plant anomaly detection with edge computing. In: Proc. of the 2019 IEEE Int'l Symp. on Software Reliability Engineering Workshops. Berlin: IEEE, 2019. 204–210. [doi: 10.1109/ISSREW.2019.00073]

[22] Wang L, Liu JY, He Q. Concept drift-based checkpoint-restart for edge services rejuvenation. IEEE Trans.on Services Computing, 2023, 16(3): 1713–1725. [doi: 10.1109/TSC.2022.3210086]

[23] Vinícius L, Rodrigues L, Torquato M, Silva FA. Docker platform aging: A systematic performance evaluation and prediction of resource consumption. The Journal of Supercomputing, 2022, 78(10): 12898–12928. [doi: 10.1007/s11227-022-04389-4]

[24] Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. In: Proc. of the 5th Int'l Conf. on Learning Representations. Toulon: OpenReview.net, 2017. 1–14.

[25] Zhou HY, Zhang SH, Peng JQ, Zhang S, Li JX, Xiong H, Zhang WC. Informer: Beyond efficient Transformer for long sequence time-series forecasting. In: Proc. of the 35th AAAI Conf. on Artificial Intelligence. AAAI, 2021. 11106–11115. [doi: 10.1609/aaai.v35i12.17325]

[26] Goyal A, Bochkovskiy A, Deng J, Koltun V. Non-deep networks. In: Proc. of the 36th Int'l Conf. on Neural Information Processing Systems. New Orleans: Curran Associates Inc., 2022. 492.

[27] Zhang Y, Wang GG, Li KQ, Yeh WC, Jian MW, Dong JY. Enhancing MOEA/D with information feedback models for large-scale many-objective optimization. Information Sciences, 2020, 522: 1–16. [doi: 10.1016/j.ins.2020.02.066]

[28] Zhang QF, Li H. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. IEEE Trans.on Evolutionary Computation, 2007, 11(6): 712–731. [doi: 10.1109/TEVC.2007.892759]

[29] Wu HX, Xu JH, Wang JM, Long MS. Autoformer: Decomposition Transformers with auto-correlation for long-term series forecasting. In: Proc. of the 35th Int'l Conf. on Neural Information Processing Systems. Curran Associates Inc., 2021. 1717.

[30] Kitaev N, Kaiser Ł, Levskaya A. Reformer: The efficient Transformer. In: Proc. of the 8th Int'l Conf. on Learning Representations. Addis Ababa: OpenReview.net, 2020.

[31] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I. Attention is all you need. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 6000–6010.

[32] Zeng AL, Chen MX, Zhang L, Xu Q. Are Transformers effective for time series forecasting? In: Proc. of the 37th AAAI Conf. on Artificial Intelligence. Washington: AAAI Press, 2023. 11121–11128. [doi: 10.1609/aaai.v37i9.26317]

[33] Challu C, Olivares KG, Oreshkin BN, Rzmirez FG, Canseco MM, Dubrawski A. NHITS: Neural hierarchical interpolation for time series forecasting. In: Proc. of the 37th AAAI Conf. on Artificial Intelligence. Washington: AAAI Press, 2023. 6989–6997.

[34] Wu HX, Hu TG, Liu Y, Zhou H, Wang JM, Long MS. TimesNet: Temporal 2D-variation modeling for general time series analysis. In: Proc. of the 11th Int'l Conf. on Learning Representations. Kigali: OpenReview.net, 2023. 1–13.

[35] Ke GL, Meng Q, Finley T, Wang TF, Chen W, Ma WD, Ye QW, Liu TY. LightGBM: A highly efficient gradient boosting decision tree. In: Proc. of the 31st Int'l Conf. on Neural Information Processing Systems. Long Beach: Curran Associates Inc., 2017. 3149–3157.

[36] He KM, Zhang XY, Ren SQ, Sun J. Deep residual learning for image recognition. In: Proc. of the 2016 IEEE Conf. on Computer Vision and Pattern Recognition. Las Vegas: IEEE, 2016. 770–778. [doi: 10.1109/CVPR.2016.90]

[37] Jain H, Deb K. An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, Part II: Handling constraints and extending to an adaptive approach. IEEE Trans.on Evolutionary Computation, 2014, 18(4): 602–622. [doi: 10.1109/TEVC.2013.2281534]

[38] Zitzler E, Laumanns M, Thiele L. SPEA2: Improving the strength Pareto evolutionary algorithm. Zurich: ETH Zurich, 2001. [doi: 10.3929/ethz-a-004284029]

[39] Mahmud R, Pallewatta S, Goudarzi M, Buyya R. iFogSim2: An extended iFogSim simulator for mobility, clustering, and microservice management in edge and fog computing environments. Journal of Systems and Software, 2022, 190: 111351. [doi: 10.1016/j.jss.2022.111351]

附中文参考文献

[11] 孟海宁, 童新宇, 石月开, 朱磊, 冯锴, 黑新宏. 基于 ARIMA-RNN 组合模型的云服务器老化预测方法. 通信学报, 2021, 42(1): 163–171. [doi: 10.11959/j.issn.1000-436x.2021015]

[12] 孟海宁, 童新宇, 谢国, 张贝贝, 黑新宏. 基于 RUL 和 SVs-GFF 的云服务器老化预测方法. 自动化学报, 2024, 50(10): 2036–2048. [doi: 10.16383/j.aas.c211112]

作者简介

刘靖, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件可靠性, 启发式算法, 分布式计算.

谭雪勇, 博士生, CCF 学生会员, 主要研究领域为软件可靠性工程, 软件老化与抗衰.

唐志伟, 硕士生, 主要研究领域为软件可靠性, 时间序列预测.

牛超煜, 博士生, CCF 学生会员, 主要研究领域为软件可靠性.