文章编号： 1006-3080(2026)02-0265-11 **DOI：** 10.14135/j.cnki.1006-3080.20250509001

云边协同环境下可靠低延迟的任务卸载方法

方卓越[1]， 虞慧群[1,2]， 范贵生[1,2]
（1. 华东理工大学计算机科学与工程系, 上海 200237； 2. 上海智慧能源工程技术研究中心, 上海 201103）

摘要：针对云边协同环境下的任务卸载问题，本文提出了一种基于树状遗传编程的超启发式算法(Tree-Based Genetic Programming Hyper-Heuristic，TBGP-HH)，该算法能够根据资源配置和输入任务，动态生成服务放置与任务卸载策略，以提高任务处理可靠性并降低延迟。首先，结合任务卸载目标设计了一组低层次启发式算法；接着，将其作为基因构建个体编码树，以初始化种群；然后，通过选择、交叉和变异操作迭代进化种群，并采用精英保留策略保留优秀个体；最终，生成在可靠性与延迟优化方面具有良好性能的启发式算法，用于指导服务放置与任务卸载。通过在现实应用场景中与对比算法进行实验比较表明，TBGP-HH在不同场景下均能有效提高可靠性并降低延迟，整体性能优于最近的任务卸载算法。

关键词：云边协同环境；树状遗传编程；超启发式算法；可靠低延迟；任务卸载

中图分类号：TP393 **文献标志码：**A

随着时代的发展，物联网作为一种新兴技术，正逐渐融入人们的日常生活。典型的物联网应用，如智能穿戴设备、智能监控和智慧医疗等[1-2]，已经被广泛应用。然而，随着用户需求的不断增长，移动设备有限的计算能力逐渐难以满足用户的需求，在面对大规模数据计算任务时，通常需要将这些任务卸载到其他计算资源上处理。

云计算作为一种计算服务模型，能够为用户提供近乎无限的计算与存储资源，充分满足了用户的计算需求[3]。然而，云计算通常需要用户将本地数据上传至云端处理，再由云端将计算结果返回，这一过程的传输时间开销相当可观，可能达到用户难以接受的程度。

边缘计算作为一种新兴的计算服务模型，通过将数据传输给靠近用户的边缘端处理，不仅有效解决了移动设备计算能力不足的问题，也避免了过高的延迟。然而，边缘计算也存在不足之处：(1)相比于云计算，边缘计算的计算资源相对有限，难以满足大数据量任务的计算需求；(2)边缘计算将原本卸载到云端的任务负载转移至边缘端，当任务负载过多时，可能导致边缘端超载，从而使得边缘服务器的任务处理性能下降[4]。

面对上述问题，将云计算与边缘计算的优势相结合，构建一个云边协同的网络体系结构，是合理且必要的。对于复杂且延迟敏感度低的任务，可以卸载到云端处理；而对于简单且延迟敏感度高的任务，可以卸载到边缘端处理。在任务卸载过程中，需要考虑以下两个关键问题：(1)服务放置：云边协同环境中存在多个资源节点，如何选择合适的节点放置应用程序服务；(2)任务卸载：用户提交的任务在处理过程中需要依次传递给不同的服务执行，只有放置了相应服务的资源节点才能完成任务的处理。因此，如何合理选择卸载节点，也是需要考虑的问题。

云边协同环境下的服务放置与任务卸载问题已经被证明属于NP-hard问题[5]，难以在多项式时间内精确求解。为了解决这一难题，许多学者展开了广泛研究，并取得了一定的研究成果。Fan等[1]提出了一种联合服务放置、任务调度、计算资源和传输速率

收稿日期： 2025-05-09

作者简介： 方卓越 (1999—)，男，浙江人，硕士生，主要研究方向为云计算和边缘计算。E-mail：fang_zhuoyue@163.com

通信联系人： 虞慧群，E-mail：yhq@ecust.edu.cn

引用本文： 方卓越，虞慧群，范贵生. 云边协同环境下可靠低延迟的任务卸载方法 [J]. 华东理工大学学报(自然科学版)，2026，52(2)：265-275.

Citation： FANG Zhuoyue, YU Huiqun, FAN Guisheng. Reliable and Low-Latency Task Offloading Approach in Cloud-Edge Collaborative Environments[J]. Journal of East China University of Science and Technology, 2026, 52(2): 265-275.

分配的优化算法来降低任务处理延迟。Fang 等[2] 提出了一种启发式的动态任务处理算法和改进的粒子群优化算法，旨在满足用户低延迟需求的同时降低能耗。为了联合优化延迟、能耗与计算资源，黄如等[6] 提出了一个基于蒙特卡洛树搜索的多通道探索算法。Sarrafzade 等[7] 提出了一种基于遗传算法的服务放置方法，该方法通过引入惩罚机制来最小化延迟，并减少应用程序服务的放置数量。针对数据密集型应用在云边环境中的服务放置和请求调度问题，Farhadi 等[8] 提出了一种双时间尺度的联合优化框架。Li 等[9] 考虑到深度神经网络应用的计算密集型和数据密集型特性，提出了一种混合混沌进化算法，以满足截止时间和预算约束的同时降低能耗。钟传江等[10] 考虑到边缘服务器之间的协作，提出了一种面向多边缘服务器合作博弈和离散粒子优化的计算卸载算法，以降低服务成本和延迟。Wu 等[11] 提出了一种基于李雅普诺夫优化的算法，旨在满足云边环境下任务卸载的平均响应时间约束，并降低平均能量消耗。

虽然云边协同环境下的任务卸载问题已经取得了显著进展，但是仍存在一些未能充分解决的问题。在云边协同环境下，资源节点通常分布在不同的地理位置，这些节点之间通过回程网络进行通信协作，在数据传输过程中，网络通信可能存在不可靠的问题[12]。同时，边缘服务器在执行任务的过程中也有可能发生故障，这些故障都将影响用户请求的处理，进而造成重大财产损失。例如，2017 年 AWS 的 4 h 停机给前 100 名在线零售商中的 54 家造成了 1.5 亿美元的损失[13]。

因此，在上述问题中，可靠性的重要性愈加突出。然而，目前的研究大多侧重于延迟、能耗和成本等目标，较少有研究将可靠性作为任务卸载的关键考量因素。为此，本文提出了一种基于树状遗传编程的超启发式算法(Tree-Based Genetic Progarmming Hyper-Heuristic, TBGP-HH)，它是通过动态选择和组合多个精心设计的低层次启发式算法生成，在可靠性与延迟优化方面具有良好性能，用于指导服务放置与任务卸载。

1 问题描述与分析

1.1 云边协同网络体系结构

云边协同网络体系结构如图 1 所示，包含三种不同类型的计算资源：远程的云端，靠近用户的边缘端以及边缘端服务范围内的移动设备。三者通过网络互相通信，并协作处理用户提交的任务。

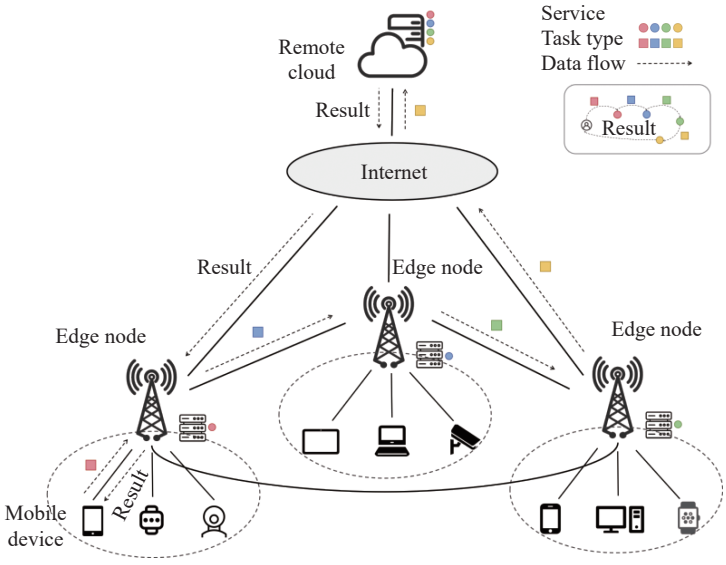


图 1 云边协同网络体系结构

Fig. 1 Cloud-edge collaborative network architecture

云边协同网络体系结构的资源节点集合用 $S=\{s_1,s_2,\cdots,s_k,\cdots,s_{|S|}\}$ 表示。s_k 包含多个属性：资源类型 $\text{type}(s_k)$ ($\text{type}(s_k)\in\{0,1,2\}$，0 表示云端，1 表示边缘端，2 表示移动设备)，计算资源 $C(s_k)$，内存资源 $M(s_k)$，存储资源 $D(s_k)$，执行性能参数 $\lambda_e(s_k)$ (表示 s_k 在执行任务时，单位时间内发生故障的平均次数) 和通信质量参数 $\lambda_t(s_k)$ (表示 s_k 在与其他节点进行通信时，单位时间内发生通信故障的平均次数)。

云边协同网络体系结构中的节点之间通过网络互相通信，对于节点 s_m 和 s_n，传输带宽为 $\text{BW}(s_m,s_n)$。

1.2 应用程序服务与任务模型

应用程序由一系列服务模块组成, 用 $\mathrm{App}=\{a_1,a_2,\cdots,a_i,\cdots,a_{|A|}\}$ 表示。其中 a_i 表示应用程序 App 的第 i 个模块, a_i 部署所需的计算资源为 $c(a_i)$, 内存资源为 $m(a_i)$, 存储资源为 $d(a_i)$。服务模块之间的数据依赖关系用 $E=\{e_{i,j}\big|1\leqslant i,j\leqslant n,i\neq j\}$ 表示。

用户提交的任务在处理过程中会被依次传输给不同的服务模块执行, 任务类型在传输过程中会不断变化。为了便于描述, 用 $T=\{t_1,t_2,\cdots,t_i,\cdots,t_{|T|}\}$ 表示任务类型的集合, 其中 t_i 表示任务的第 i 种类型。对于任务 t_i, 它只能被 a_i 执行。t_i 的计算负载为 $w(t_i)$, 数据传输量为 $\mathrm{data}(t_i)$。

1.3 服务放置与任务卸载模型

对于第 i 种服务模块 a_i 和第 k 个资源节点 s_k, 放置关系如式 (1) 所示。

$$C1: x(a_i,s_k)\in\{0,1\},\forall a_i\in\mathrm{App},\forall s_k\in S \tag{1}$$

如果服务模块 a_i 被放置在资源节点 s_k 上, 则 $x(a_i,s_k)=1$, 否则 $x(a_i,s_k)=0$。

由于应用程序模块可以放置在多个资源节点上, 所以用 $S_i=\{s_k|a_i\in\mathrm{App},x(a_i,s_k)=1\}$ 表示放置了 a_i 的节点集合, $S_i\subseteq S$。对于服务放置问题, 需要满足以下约束条件: 放置在节点上的服务模块的计算资源、内存资源和存储资源需求不能超过节点本身所能提供的资源容量, 即:

$$C2: x(a_i,s_k)\cdot c(a_i)\leqslant C(s_k),\forall a_i\in\mathrm{App},\forall s_k\in S \tag{2}$$

$$C3: \sum_{\forall a_i\in\mathrm{App}} x(a_i,s_k)\cdot m(a_i)\leqslant M(s_k),\forall s_k\in S \tag{3}$$

$$C4: \sum_{\forall a_i\in\mathrm{App}} x(a_i,s_k)\cdot d(a_i)\leqslant D(s_k),\forall s_k\in S \tag{4}$$

对于任务 t_i, 其只能被相应的服务模块 a_i 执行。如前所述, 放置了 a_i 的节点集合用 S_i 表示, 因此, 任务 t_i 只能被卸载到 S_i 的节点上执行。任务和节点之间的卸载关系如式 (5) 所示。

$$C5: y(t_i,s_k)\in\{0,1\},\forall t_i\in T,\forall s_k\in S_i \tag{5}$$

其中, $y(t_i,s_k)=1$ 表示任务 t_i 被卸载到放置了模块 a_i 的节点 s_k 上执行, 否则 $y(t_i,s_k)=0$。由于任务 t_i 只可能被卸载到某个节点上执行, 所以

$$C6: \sum_{\forall s_k\in S_i} y(t_i,s_k)=1 \tag{6}$$

1.4 问题描述

任务处理延迟由执行延迟和数据传输延迟两部分组成。当任务 t_i 卸载到节点 s_m 上执行时, 执行延迟为

$$\mathrm{ET}(t_i,s_m)=\frac{w(t_i)}{C(s_m)},\ y(t_i,s_m)=1 \tag{7}$$

假设任务 t_i 在节点 s_m 上完成执行后, 转变为下一种类型 t_j, 而 t_j 需要在 a_j 所在的节点 s_n 上执行, 则数据传输延迟为

$$\begin{aligned}&\mathrm{TT}(t_i,t_j,s_m,s_n)=\begin{cases}\dfrac{\mathrm{data}(t_i)}{\mathrm{BW}(s_m,s_n)}, & \text{if } s_m\neq s_n\\ 0, & \text{Otherwise}\end{cases}\\ &e_{i,j}\in E,\ y(t_i,s_m)=1,\ y(t_j,s_n)=1\end{aligned} \tag{8}$$

因此, 任务 t_i 的延迟如式 (9) 所示。

$$L(t_i)=\mathrm{ET}(t_i,s_m)+\mathrm{TT}(t_i,t_j,s_m,s_n) \tag{9}$$

对于用户提交的任务, 总延迟为 $L(T)=\sum_{\forall t_i\in T}L(t_i)$。

任务处理可靠性由执行可靠性和数据传输可靠性两部分组成。本文采用泊松分布[14] 对任务处理可靠性进行建模。

对于执行可靠性, 假设任务 t_i 在节点 s_m 上执行时故障发生次数为 $N(\mathrm{ET}(t_i,s_m))$, 则在任务执行期间发生 ξ 次故障的概率为

$$\Pr\{N(\mathrm{ET}(t_i,s_m))=\xi\}=\frac{(\lambda_e(s_m)\cdot\mathrm{ET}(t_i,s_m))^{\xi}}{\xi!}\mathrm{e}^{-\lambda_e(s_m)\cdot\mathrm{ET}(t_i,s_m)} \tag{10}$$

对于数据传输可靠性, 假设在数据传输期间, 故障发生次数为 $N(\mathrm{TT}(t_i,t_j,s_m,s_n))$, 则在数据传输期间发生 ξ 次故障的概率为

$$\begin{aligned}&\Pr\{N(\mathrm{TT}(t_i,t_j,s_m,s_n))=\xi\}=\\ &\frac{(\lambda_t(s_m,s_n)\cdot\mathrm{TT}(t_i,t_j,s_m,s_n))^{\xi}}{\xi!}\mathrm{e}^{-\lambda_t(s_m,s_n)\cdot\mathrm{TT}(t_i,t_j,s_m,s_n)}\end{aligned} \tag{11}$$

如果任务可靠执行且数据可靠传输, 则表明任务在处理期间没有发生任何故障。则任务执行可靠性为

$$R_{\mathrm{exec}}(t_i,s_m)=\Pr\{N(\mathrm{ET}(t_i,s_m))=0\}=\mathrm{e}^{-\lambda_e(s_m)\cdot\mathrm{ET}(t_i,s_m)} \tag{12}$$

数据传输可靠性为

$$\begin{aligned}R_{\mathrm{trans}}(t_i,t_j,s_m,s_n)=&\Pr\{N(\mathrm{TT}(t_i,t_j,s_m,s_n))=0\}=\\ &\mathrm{e}^{-\lambda_t(s_m,s_n)\cdot\mathrm{TT}(t_i,t_j,s_m,s_n)}\end{aligned} \tag{13}$$

因此, 任务 t_i 在节点 s_m 上处理可靠性为

$$\begin{aligned}R(t_i,s_m)=&R_{\mathrm{trans}}(t_i,t_j,s_m,s_n)\cdot R_{\mathrm{exec}}(t_i,s_m)=\\ &\mathrm{e}^{-\left(\lambda_t(s_m,s_n)\cdot\mathrm{TT}(t_i,t_j,s_m,s_n)+\lambda_e(s_m)\cdot\mathrm{ET}(t_i,s_m)\right)}\end{aligned} \tag{14}$$

对于用户提交的任务, 处理可靠性为 $R(T)=\prod_{\forall t_i\in T}R(t_i,s_m)$。

综上所述, 云边协同环境下关注延迟与可靠性目标的服务放置与任务卸载问题, 可以表述为:

$$\text{Minimize } L(T)=\sum_{\forall t_i\in T}L(t_i)$$

$$\text{Maximize } R(T)=\prod_{\forall t_i\in T}R(t_i,s_m)$$

$$\text{s.t. } C1-C6 \tag{15}$$

2 基于树状遗传编程的超启发式算法

2.1 终端集合

在超启发式算法中，由低层次启发式算法构成的集合通常被称为终端集合，其中，低层次启发式算法也可被称为终端。终端的设计需要充分考虑研究问题的特性和目标，以确保其能够有效发挥作用。由于云边协同环境下的任务卸载问题实际包含服务放置与任务卸载两个阶段。因此，针对这两个阶段，分别设计了相应的终端，具体内容如表 1 所示。

表 1 终端集合

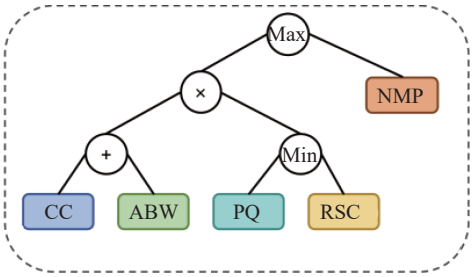
Table 1 Terminal set

No.	Terminal	Description	Service placement	Task offloading
1	CC	Computing capacity	√	
2	RMC	Remaining memory capacity	√	
3	RSC	Remaining storage capacity	√	
4	ABW	Average bandwidth	√	
5	PQ	Processing quality	√	
6	ACQ	Average communication quality	√	
7	NMP	Number of modules placed	√	
8	RCC	Remaining computing capacity		√
9	ET	Execution time		√
10	ATT	Average transmission time		√
11	R	Reliability		√

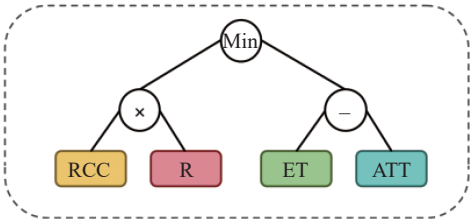
2.2 树状编码表示

如图 2 所示是启发式算法的树状编码示例，其中，第 1 棵树用于服务放置，其叶子节点从终端集合的 1~7 号终端(表 1)中选取；第 2 棵树用于任务卸载，其叶子节点从终端集合的 8~11 号终端(表 1)中选取。通过构建这两棵树，从而形成一个完整的启发式算法。

图 2 所示的启发式算法的两个资源选择规则(R_1 和 R_2) 如式 (16) 所示。这两个规则分别是在对算法的两棵编码树执行中序遍历得到的。其中，+、−、×、/、Max 和 Min 等运算符用于连接不同的终端。



(a) Resource selection rule (Phase 1: service placement)



(b) Resource selection rule (Phase 2: task offloading)

图 2 启发式算法的树状编码示例

Fig. 2 Tree-based heuristic encoding example

$$\begin{cases} R_1=\text{Max}((\text{CC}+\text{ABW})\times\text{Min}(\text{PQ},\text{RSC}),\text{NMP}) \\ R_2=\text{Min}(\text{RCC}\times R,\text{ET}-\text{ATT}) \end{cases} \tag{16}$$

值得注意的是，在服务放置阶段，资源选择规则 R_1 适用于云边协同环境中的所有资源节点；在任务卸载阶段，资源选择规则 R_2 只适用于已经放置了相应服务的资源节点。

2.3 TBGP-HH 算法框架

TBGP-HH 算法框架如图 3 所示，主要包括训练阶段和执行阶段两部分。其中，训练阶段包括几个关键步骤：种群初始化、个体适应值评估以及种群进化操作。

2.3.1 *种群初始化* 为了确保初始种群具备足够的多样性，采用 Ramped-half-and-half 方法[15] 初始化种群个体。具体而言，种群中一半个体采用 Full 方法生成，另一半个体采用 Grow 方法生成。种群初始化的伪代码如 Algorithm 1 所示。

Algorithm 1: Ramped-half-and-half population initialization

Input: population size ps, terminal set $\mathbb{T}$, minimum tree depth $d_{\min}$, maximum tree depth $d_{\max}$

Output: initial population $\mathbb{P}$

$\mathbb{P}=\varphi$

for $i=1$ to ps/2 do

generate resource selection rules R_1 and R_2 using the Full$(\mathbb{T},d_{\max})$ method

add individual $I=\{R_1,R_2\}$ to $\mathbb{P}$

end for

for $i=\text{ps}/2+1$ to ps do

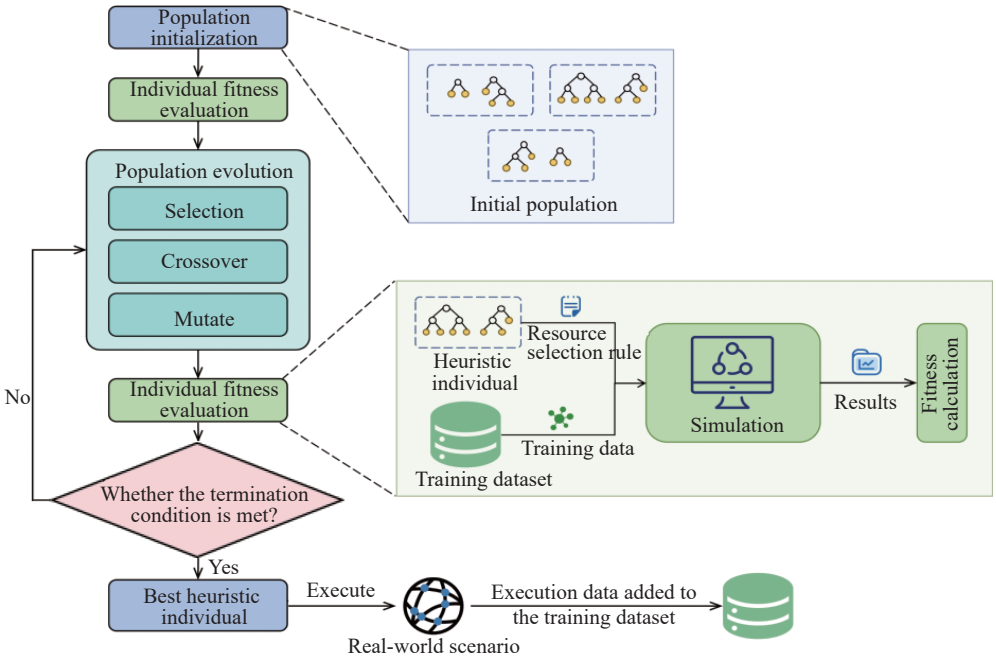


图 3　TBGP-HH 算法框架

Fig. 3　Framework of the TBGP-HH

generate resource selection rules R_1 and R_2 using the Grow$(\mathbb{T}, d_{\max}, d_{\max})$ method

add individual $I=\{R_1,R_2\}$ to $\mathbb{P}$

end for

return $\mathbb{P}$

2.3.2 *适应值函数和选择操作*　适应值函数用于评估生成的启发式算法个体的优劣, 在种群进化过程中发挥引导和优化选择的作用。采用如式 (17) 所示的适应值函数对个体进行评估, 该函数综合考虑了可靠性与延迟目标, 通过在可靠性目标前引入动态权重系数 $\alpha(|T|)$, 动态调整可靠性在适应值评估中的重要程度。其中, R(Reliability) 代表可靠性, L(Latency) 代表延迟, λ 是惩罚系数, λ 的值越大, 对高延迟的容忍度越低。

$$\mathrm{Fit}(I)=\alpha(|T|)\cdot R+(1-\alpha(|T|))\cdot \mathrm{e}^{-\lambda L} \tag{17}$$

$\alpha(|T|)$ 的计算公式如式 (18) 所示。当 $|T|$ 较小时, $\alpha(|T|)$ 接近初始值 α_0; 随着 $|T|$ 的增大, $\alpha(|T|)$ 逐渐趋近于 1, 表明可靠性的重要程度不断提高。

$$\alpha(|T|)=\alpha_0+(1-\alpha_0)\cdot\left(1-\mathrm{e}^{-k(|T|-1)}\right) \tag{18}$$

其中, k 是斜率参数, k 值越大, $\alpha(|T|)$ 趋近于 1 的速度越快。选择操作以个体的适应值为依据, 本文采用锦标赛选择法[16] 选择个体。

2.3.3 *模拟执行过程*　如式 (17) 所示, 个体的适应值计算涉及可靠性与延迟指标, 而这些指标通过将个体应用于模拟环境中进行服务放置与任务卸载计算得到。

在服务放置阶段, 放置决策取决于资源选择规则 R_1。具体而言, 首先判断资源节点 s_k 的剩余资源能否满足服务模块 a_i 的计算资源、内存资源和存储资源需求。如果满足, 则将 s_k 的相关属性作为 R_1 的输入进行计算, 以确定其优先值 $R_1(s_k)$。由于应用程序服务可以放置在多个资源节点上, 因此, 当 $R_1(s_k)$ 大于设定阈值 χ 时, 服务 a_i 即可放置在 s_k 上。

在任务卸载阶段, 任务 t_i 只能被卸载到放置了对应服务 a_i 的资源节点上。因此, 使用资源选择规则 R_2 遍历放置了 a_i 的资源节点集合 S_i, 以确定其中各节点的优先值 $R_2(s_k)$。随后, 选择将任务 t_i 卸载到优先值 $R_2(s_k)$ 最高的节点上执行。

模拟执行过程的伪代码如 Algorithm 2 所示。

Algorithm 2: Simulation process

Input: individual I, resource node set S, application App, task type set T

Output: latency L, reliability R

$L=0$, $R=1$

for each a_i in App do

for each s_k in S do

if $c(a_i)\leqslant C(s_k)$ and $m(a_i)\leqslant$ remaining $M(s_k)$ and $d(a_i)\leqslant$ remaining $D(s_k)$ then

calculate $R_1(s_k)$

end if

if $R_1(s_k)\geqslant \chi$ then

place a_i on s_k, $x(a_i,s_k)=1$

end if

end for
end for
for each t_i in T do
for each s_k in S_i do
calculate $R_2(s_k)$
end for
offloading t_i to the node with the highest $R_2(s_k)$, update L, R
end for
return L, R

2.3.4 交叉操作 交叉操作的具体步骤为：首先，从种群中选择两个个体 A 和 B；然后，如果生成的随机数小于交叉概率(cr)，分别对 A 和 B 的两棵编码树执行交叉操作，从而生成两个新的后代个体 A' 和 B'。交叉操作示例如图 4 所示。交叉操作的伪代码如 Algorithm 3 所示。

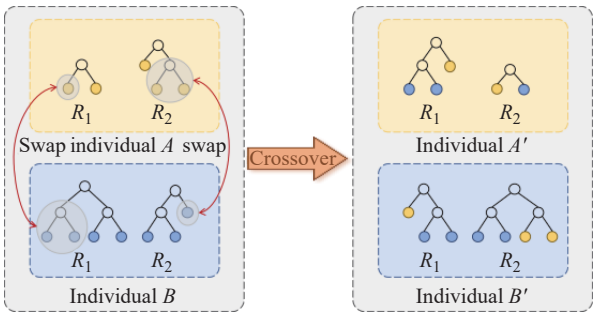


图 4 交叉操作示例

Fig. 4 An example of crossover

Algorithm 3: Crossover operation
Input: parent individuals A, B
Output: offspring individuals A', B'
rand$(0,1) <$ cr then
randomly select a node $\text{node}_{A(R_1)}$ from $A(R_1)$, and a node $\text{node}_{A(R_2)}$ from $A(R_2)$
randomly select a node $\text{node}_{B(R_1)}$ from $B(R_1)$, and a node $\text{node}_{B(R_2)}$ from $B(R_2)$
swap the subtrees rooted at $\text{node}_{A(R_1)}$ and $\text{node}_{B(R_1)}$ to construct $A'(R_1)$ and $B'(R_1)$
swap the subtrees rooted at $\text{node}_{A(R_2)}$ and $\text{node}_{B(R_2)}$ to construct $A'(R_2)$ and $B'(R_2)$
end if
return A', B'

2.3.5 变异操作 变异操作的具体步骤为：首先，从个体的编码树中随机选择一棵子树；然后，如果生成的随机数小于变异概率(mr)，则将该子树用 Grow$(\mathbb{T}, d_{\max}, d_{\max})$ 方法生成的随机树替换，从而得到新的编码树。变异操作示例如图 5 所示，变异操作的伪代码如 Algorithm 4 所示。

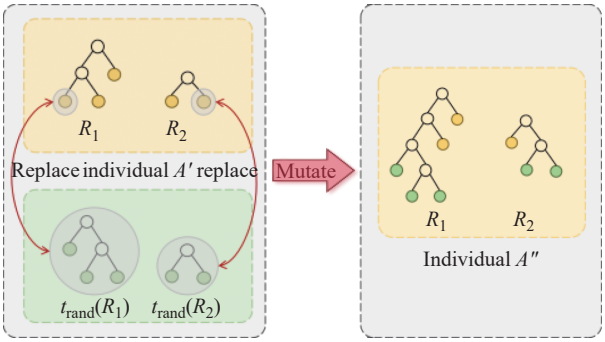


图 5 变异操作示例

Fig. 5 An example of mutation

Algorithm 4: Mutation operation
Input: parent individual A'
Output: offspring individual A''
if rand$(0,1) <$ mr then
randomly select a node $\text{node}_{A'(R1)}$ from $A'(R_1)$, and a node $\text{node}_{A'(R_2)}$ from $A'(R_2)$
generate random trees $t_{\text{rand}}(R_1)$ and $t_{\text{rand}}(R_2)$ using the Grow$(\mathbb{T}, d_{\max}, d_{\max})$ method
replace the subtree rooted at $\text{node}_{A'(R1)}$ with $t_{\text{rand}}(R_1)$ to construct $A''(R_1)$, and replace the subtree rooted at $\text{node}_{A'(R_2)}$ with $t_{\text{rand}}(R_2)$ to construct $A''(R_2)$
end if
return A''

2.3.6 TBGP-HH 算法 TBGP-HH 算法为：首先采用 Ramped-half-and-half 方法初始化种群；利用适应值函数对初始种群中的个体进行评估；根据个体适应值执行选择、交叉和变异操作，从而生成新的后代个体。同时，采用精英保留策略保留前 μ 个最优个体，对新生成的种群个体进行适应值评估。重复上述过程，直到满足终止条件。最终得到的就是算法输出的最优启发式算法，将该算法应用于现实应用场景中的服务放置与任务卸载，以实现高可靠性与低延迟的优化目标。本算法的伪代码如 Algorithm 5 所示。

Algorithm 5: TBGP-HH Algorithm
Input: population size ps, terminal set $\mathbb{T}$, minimum tree depth $d_{\min}$, maximum tree depth $d_{\max}$, crossover rate cr, mutation rate mr, number of elitism μ
Output: optimal heuristic individual I^*
initialize population $\mathbb{P}$ using the Ramped-half-and-half method
for each I in $\mathbb{P}$ do
evaluate fitness of I using fitness function Fit
end for

while the termination condition is not met do
  $\Omega=\varphi$, $\mathbb{P}'=\varphi$
  for $i=1$ to $ps-\mu$ do
    select the winner of tournament selection and add it to Ω
  end for
  for each pair $\{A,B\}$ in Ω do
    apply the crossover operation to $\{A,B\}$ to generate offspring $\{A',B'\}$, and add them to $\mathbb{P}'$
  end for
  for each I in $\mathbb{P}'$ do
    apply the mutation operation to I
  end for
  add the top μ individuals from $\mathbb{P}$ to $\mathbb{P}'$
  $\mathbb{P}=\mathbb{P}'$
  for each I in $\mathbb{P}$ do
    evaluate fitness of I using fitness function Fit
  end for
end while
$I^*=$ the best individual in $\mathbb{P}$
return I^*

3　实验结果与分析

3.1　实验设置

3.1.1　*实验环境*　本文采用仿真平台 iFogSim[17] 模拟云边协同环境下的服务放置与任务卸载过程。所有模拟实验均在配置为 Core i7 2.7 GHz CPU、32 GB RAM 的 Windows 11 操作系统的电脑上进行, 所有算法均使用 Java 8 Standard Edition V1.8.0 编写。

3.1.2　*数据集*　选取现实场景中的虚拟现实[18] 和智能监控[19] 评估算法的性能。其中, 虚拟现实通过计算机模拟构建一个三维立体环境, 用户能够借助专业设备获得沉浸式体验。虚拟现实的应用程序模块如图 6 所示。

智能监控通过将计算机视觉算法集成到摄像头中, 从而实现对监控场景的自动化与智能化分析。智能监控的应用程序模块如图 7 所示。

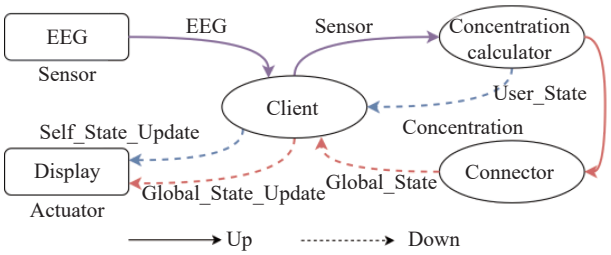


图 6　虚拟现实应用程序模块

Fig. 6　Virtual reality application module

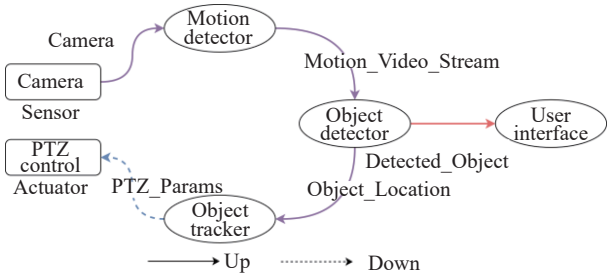


图 7　智能监控应用程序模块

Fig. 7　Intelligent surveillance application module

3.1.3　*资源配置*　云边协同环境的资源配置如表 2 所示。在云边协同环境下, 不同节点通过回程网络进行通信。本文设置边缘端和云端之间的带宽为 100 Mbps, 边缘端之间的带宽为 500 Mbps, 边缘端和移动设备之间的带宽为 1 000 Mbps[1]。

3.1.4　*参数设置*　根据已有文献研究[20] 和实验模拟结果, 当实验参数按照表 3 设置时, 树状遗传编程算法能够在解空间中进行充分搜索, 具备较好的分布性和收敛性。

3.2　结果分析

为了验证 TBGP-HH 算法的有效性, 将其与 HDTP-IDPSO(Heuristic Dynamic Task Processing-Improved Discrete Particle Swarm Optimization)[2]、HCEA(Hybrid Chaotic Evolutionary Algorithm)[9]、TBGP-HH-NSP(Tree-Based Genetic Programming Hyper-Heuristic No Service Placement) 和 TBGP-HH-NTO(Tree-Based Genetic Programming Hyper-Heuristic No Task Offloading) 等算法进行比较。其中, TBGP-HH-NSP 算法在 TBGP-HH的基础上禁用了服务放置策略; TBGP-HH-NTO 算法在 TBGP-HH 的基础上禁

表 2　云边协同环境资源配置[1-2]

Table 2　Resource configuration in cloud-edge collaborative environment[1-2]

Name	Resource type	Computing resource/MIPS	Memory resource/GB	Storage resource/GB	λ_e	λ_t
Cloud	0	44 800	inf	inf	0.000 1	0.000 1
Edge	1	[14 000, 16 000]	[100, 200]	[3 000, 5 000]	[0.001, 0.01]	[0.001, 0.01]
Mobile device	2	[3 000, 5 000]	[24, 40]	[800, 1 200]	[0.01, 0.05]	[0.01, 0.05]

表 3 云边协同环境资源配置

Table 3 Parameter settings for the proposed algorithm

Parameter	Definition	Value
ps	Population size	100
d_{min}	Minimum tree depth	2
d_{max}	Maximum tree depth	7
cr	Crossover rate	0.9
mr	Mutation rate	0.01
μ	Number of elitism	10

用了任务卸载策略。

3.2.1 不同用户数量 首先,研究不同用户数量对算法性能的影响,选取用户数量分别为 10、30、50、100、200、300 和 500 进行实验,实验结果如图 8 所示。

从图 8 可以看出,在用户数量较少的情况下,TBGP-HH 在延迟方面与 HDTP-IDPSO 和 HCEA 表现相近,但在可靠性方面表现更优。这是因为 TBGP-HH 能够通过精心设计的终端生成合理的服务放置与任务卸载策略,从而更有效地利用节点资源。随着用户数量的增加,任务处理可靠性的重要程度随之上升。此时,TBGP-HH 会通过牺牲部分延迟性能以提高任务处理的可靠性。因此可以观察到,当用户数量超过 300 时,TBGP-HH 的任务处理延迟开始高于 HDTP-IDPSO 和 HCEA,然而,其可靠性显著优于 HDTP-IDPSO 和 HCEA。这表明在用户数量较多的情况下,TBGP-HH 仍能可靠处理用户的任务,展现出较强的适应性。

对于 TBGP-HH-NSP,其在延迟方面显著高于其他算法,主要原因在于该算法缺乏有效的服务放置策略,难以充分利用节点资源,从而产生较高的延迟。对于 TBGP-HH-NTO,虽然其在用户数量较少时表现出较好的性能,但随着用户数量的增加,该算法的性能明显下降。这是因为 TBGP-HH-NTO 缺乏有效的任务卸载策略,可能导致任务难以被卸载到最合适的节点上处理,进而影响整体的延迟与可靠性。相比之下,TBGP-HH 通过服务放置与任务卸载策略的有效协作,不仅将延迟控制在合理范围内,还能持续保持较高的可靠性。

3.2.2 不同资源配置 为研究资源配置由紧缺向充足变化过程中算法的性能表现,选取资源节点数量分别为 5、7、9、16、19、25 的多组配置进行实验,各配置下资源节点的具体分配如表 4 所示。用户数量统一设置为 100,实验结果如图 9 所示。从图 9 可以看出,在不同资源配置条件下,TBGP-HH 在延迟方面的表现与 HDTP-IDPSO 和 HCEA 相当,并在可靠

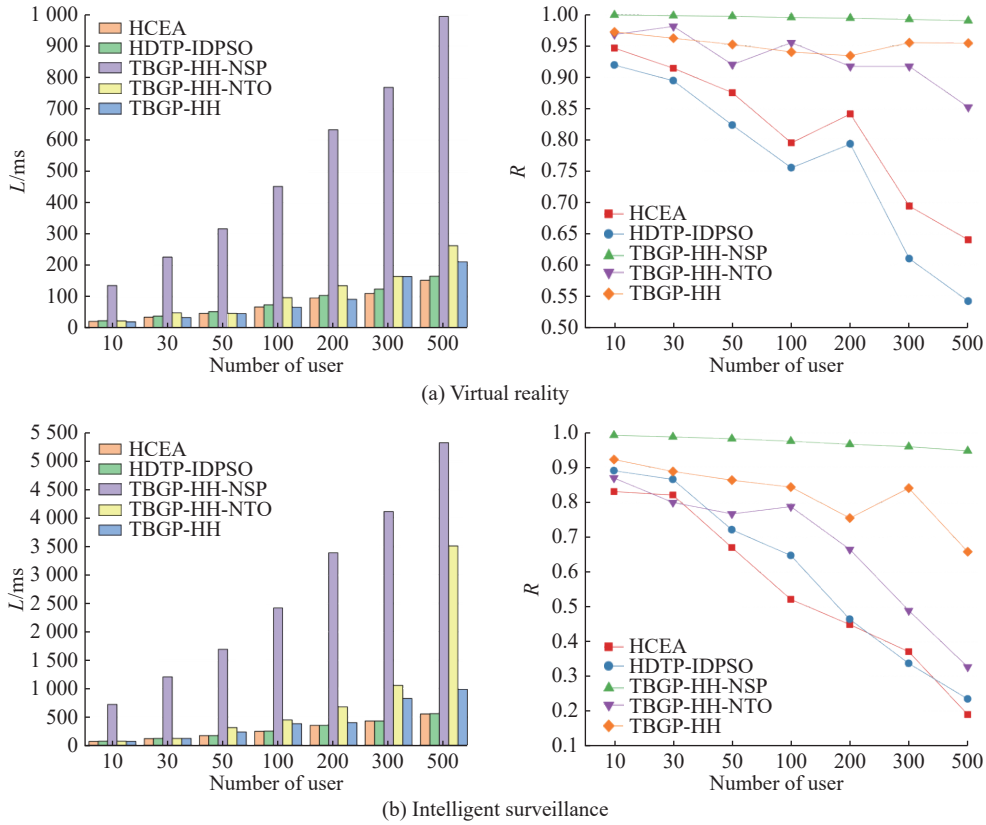


图 8 不同用户数量下的延迟与可靠性

Fig. 8 Latency and reliability under different number of users

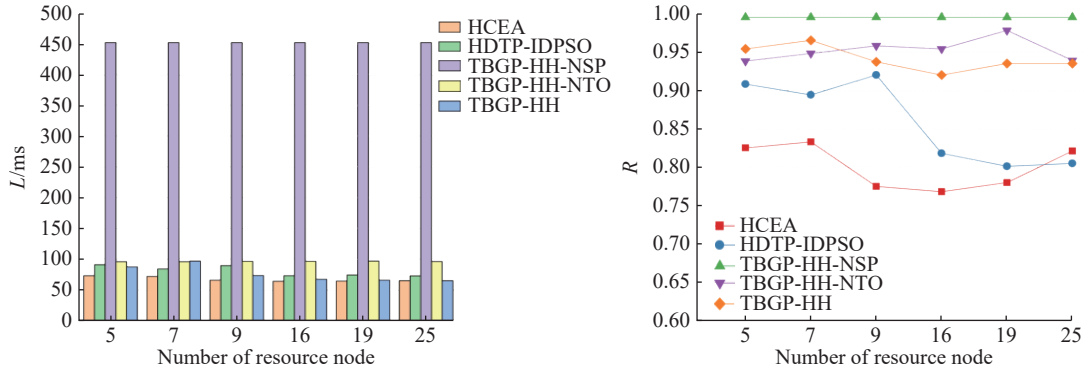
性方面优于这两种算法。这主要得益于 TBGP-HH 生成的服务放置与任务卸载策略，能够在资源节点充足的条件下进行更合理的服务放置与任务卸载，从而确保延迟在合理范围内的同时，有效提高任务处理的可靠性。

对于 TBGP-HH-NSP 和 TBGP-HH-NTO，由于分别缺乏有效的服务放置与任务卸载策略，其在不同资源配置条件下的任务处理延迟与可靠性表现相对接近，难以实现进一步优化。相比之下，TBGP-HH 能够随着资源节点数量的增加，进行更为合理的服务放置与任务卸载，从而有效优化任务处理延迟与可靠性，表现出更好的性能。

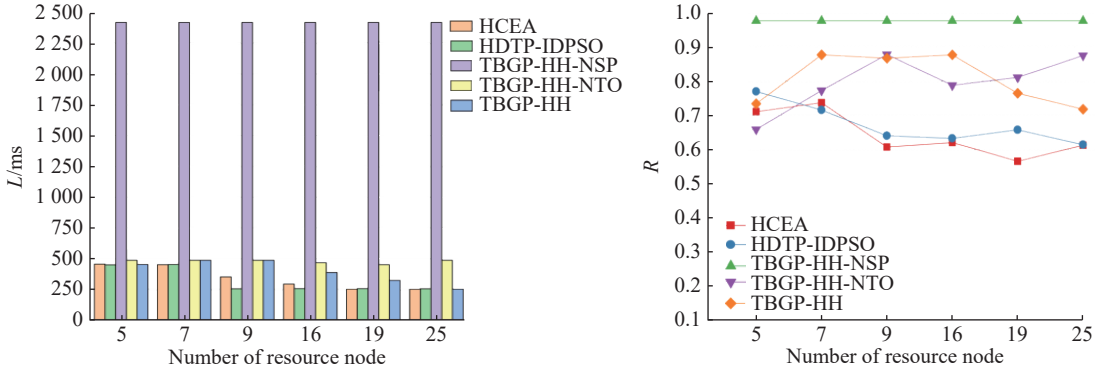
表 4　不同资源配置的具体设置

Table 4　Specific settings for different resource configurations

Number of resource node	Number of cloud	Number of edge node	Number of mobile device
5	1	2	2
7	1	2	4
9	1	4	4
16	1	5	10
19	1	6	12
25	1	8	16



(a) Virtual reality



(b) Intelligent surveillance

图 9　不同资源配置下的延迟与可靠性

Fig. 9　Latency and reliability under different resource configurations

3.2.3　最优个体示例　为进一步探究 TBGP-HH 算法生成的启发式算法在服务放置与任务卸载中的指导作用，选取用户数量为 100，资源节点数量为 16 的场景进行分析。该场景下生成的最优个体如图 10 所示。

从图 10 可以看出，在服务放置阶段，终端出现的频率依次为 CC、ACQ、NMP、PQ、ABW、RMC 和 RSC。其中，CC、NMP 和 ABW 主要与延迟相关，ACQ 和 PQ 则主要与可靠性相关。这表明，在用户数量为 100，资源节点数量为 16 的场景下，TBGP-HH 在服务放置阶段综合考虑了延迟与可靠性两个指标，通过平衡二者，将服务放置在合适的资源节点上。

在任务卸载阶段，终端出现的频率依次为 ET、ATT、R 和 RCC，其中，ET、ATT 和 RCC 主要与延迟相关，R 则主要与可靠性相关。这表明，在该场景下，TBGP-HH 在任务卸载阶段更侧重于优化延迟。这是因为此时用户数量相对较少，延迟被视为更关键的性能指标。

综上所述，通过从不同用户数量和资源配置两个方面，将 TBGP-HH 与其他比较算法的实验结果进行对比分析，验证了 TBGP-HH 的适应性和有效性。具体而言，在用户数量较少的情况下，TBGP-HH 能在保持较低延迟的同时，表现出更好的可靠性。随

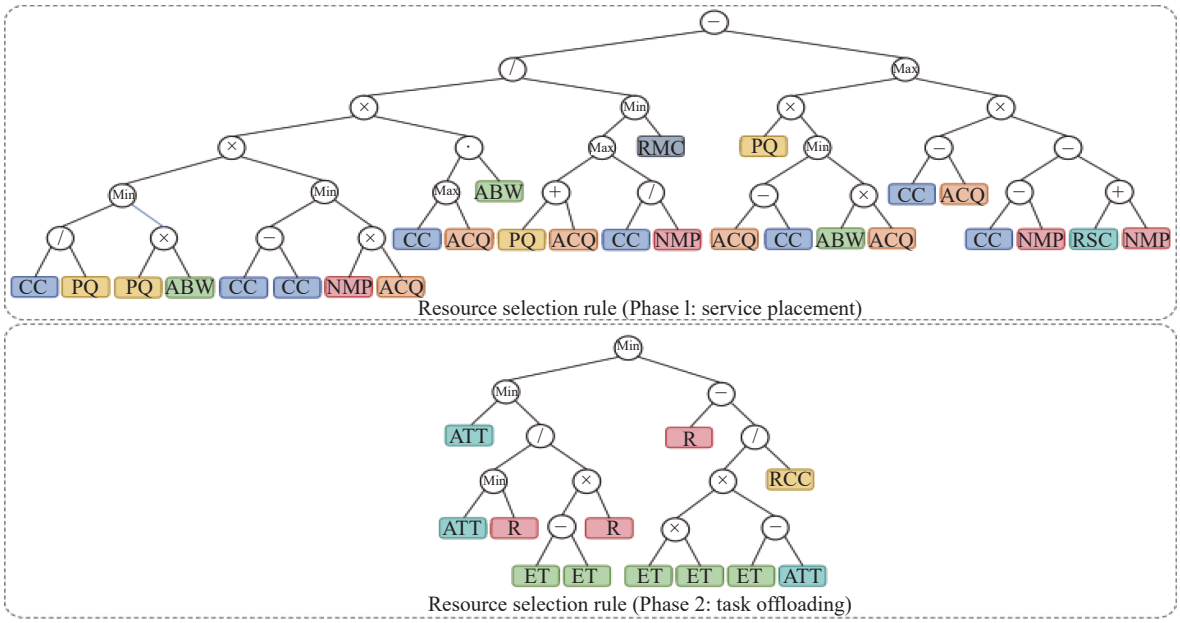


图 10 用户数量为 100、资源节点数量为 16 的场景下的启发式算法

Fig. 10 The heuristic algorithm obtained under the scenario with 100 users and 16 resource nodes

着用户数量的增加，该算法能够在牺牲部分延迟性能的前提下，始终保证任务的可靠处理。同时，随着资源节点数量的增加，TBGP-HH 能够进行更合理的服务放置与任务卸载，不仅确保任务处理延迟维持在合理范围内，还有效提高了任务处理可靠性。

4 结束语

针对云边协同环境下的服务放置与任务卸载问题，本文提出了一种基于树状遗传编程的超启发式算法 TBGP-HH。首先，根据研究问题的特性和目标，设计了一组低层次启发式算法。然后，基于树状遗传编程，对低层次启发式算法进行动态选择和组合。最终，经过种群迭代进化后，生成一个在可靠性与延迟优化方面具有良好性能的启发式算法，用于指导服务放置与任务卸载。

本文从不同用户数量和资源配置两个方面，结合虚拟现实与智能监控等现实应用场景，验证了 TBGP-HH 算法的适应性和有效性。通过将 TBGP-HH 与四种比较算法(HDTP-IDPSO、HCEA、TBGP-HH-NSP 以及 TBGP-HH-NTO)的实验结果进行对比分析，结果表明，TBGP-HH 在延迟与可靠性优化方面表现更优。

参考文献：

[1] FAN W, ZHAO L, LIU X, *et al*. Collaborative service placement, task scheduling, and resource allocation for task offloading with edge-cloud cooperation[J]. IEEE Transactions on Mobile Computing, 2022, 23(1): 238-256.

[2] FANG J, MA A. Iot application modules placement and dynamic task processing in edge-cloud computing[J]. IEEE Internet of Things Journal, 2020, 8(16): 12771-12781.

[3] 田倬璟, 黄震春, 张益农. 云计算环境任务调度方法研究综述 [J]. 计算机工程与应用, 2021, 57(2): 1-11.

[4] WANG B, LI M. Cooperative edge computing task offloading strategy for urban internet of things[J]. Wireless Communications and Mobile Computing, 2021, 2021(1): 9959304.

[5] SHEN S, HAN Y, WANG X, *et al*. Computation offloading with multiple agents in edge-computing-supported iot[J]. ACM Transactions on Sensor Networks (TOSN), 2019, 16(1): 1-27.

[6] 黄如, 宋国梁. 基于卸载策略的物联网边缘计算任务调度优化 [J]. 华东理工大学学报 (自然科学版), 2024, 50(2): 264-273.

[7] SARRAFZADE N, ENTEZARI-MALEKI R, SOUSA L. A genetic-based approach for service placement in fog computing[J]. The Journal of Supercomputing, 2022, 78(8): 10854-10875.

[8] FARHADI V, MEHMETI F, HE T, *et al*. Service placement and request scheduling for data-intensive applications in edge clouds[J]. IEEE/ACM Transactions on Networking, 2021, 29(2): 779-792.

[9] LI Z, YU H, FAN G, *et al*. Energy-efficient offloading for dnn-based applications in edge-cloud computing: A hybrid chaotic evolutionary approach[J]. Journal of Parallel and Distributed Computing, 2024, 187: 104850.

[10] 钟传江, 虞慧群, 范贵生. 云边场景下基于合作博弈的数

据上传优化 [J]. 华东理工大学学报 (自然科学版), 2025, 51(2): 250-259.

[11] WU H, SUN Y, WOLTER K. Energy-efficient decision making for mobile cloud offloading[J]. IEEE Transactions on Cloud Computing, 2018, 8(2): 570-584.

[12] LONG T, MA Y, XIA Y, *et al*. A mobility-aware and fault-tolerant service offloading method in mobile edge computing[C]//2022 IEEE International Conference on Web Services (ICWS). Honolulu, HI, USA: IEEE, 2022: 67-72.

[13] ARAL A, BRANDIC I. Learning spatiotemporal failure dependencies for resilient edge computing services[J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 32(7): 1578-1590.

[14] LIU J, ZHOU A, LIU C, *et al*. Reliability-enhanced task offloading in mobile edge computing environments[J]. IEEE Internet of Things Journal, 2021, 9(13): 10382-10396.

[15] NGUYEN S, MEI Y, ZHANG M. Genetic programming for production scheduling: A survey with a unified framework[J]. Complex & Intelligent Systems, 2017, 3: 41-66.

[16] MILLER B L, GOLDBERG D E. Genetic algorithms, selection schemes, and the varying effects of noise[J]. Evolutionary Computation, 1996, 4(2): 113-131.

[17] GUPTA H, VAHID D A, GHOSH S K, *et al*. Ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments[J]. Software: Practice and Experience, 2017, 47(9): 1275-1296.

[18] POULARAKIS K, LLORCA J, TULINO A M, *et al*. Service placement and request routing in mec networks with storage, computation, and communication constraints[J]. IEEE/ACM Transactions on Networking, 2020, 28(3): 1047-1060.

[19] CHEN J, LI K, DENG Q, *et al*. Distributed deep learning model for intelligent video surveillance systems with edge computing[J]. IEEE Transactions on Industrial Informatics, 2019.

[20] SUN Z, MEI Y, ZHANG F, *et al*. Multi-tree genetic programming hyper-heuristic for dynamic flexible workflow scheduling in multi-clouds[J]. IEEE Transactions on Services Computing, 2024, 17(5): 2687-2703.

Reliable and Low-Latency Task Offloading Approach in Cloud-Edge Collaborative Environments

FANG Zhuoyue[1], YU Huiqun[1,2], FAN Guisheng[1,2]

(1. Department of Computer Science and Engineering, East China University of Science and Technology, Shanghai 200237, China; 2. Shanghai Engineering Research Center of Smart Energy, Shanghai 201103, China)

Abstract: In recent years, cloud computing has been widely adopted by individuals and organizations. However, as cloud resources are typically deployed far from end users, processing tasks may incur considerable latency. Therefore, it is both reasonable and necessary to leverage the advantages of edge computing to develop a cloud-edge collaborative network architecture. A tree-based genetic programming hyper-heuristic algorithm (TBGP-HH) is proposed to address the task offloading problem in cloud-edge collaborative environments. The algorithm can dynamically generate service placement and task offloading strategies based on resource configurations and input tasks, thereby enhancing task processing reliability and reducing latency. First, a set of low-level heuristic algorithms is designed according to the objectives of task offloading. These heuristics are then encoded as genes to construct individual encoding trees for population initialization. Next, the population evolves iteratively through selection, crossover, and mutation operations, with an elitism strategy adopted to preserve high-quality individuals. Finally, a heuristic algorithm with strong performance in both reliability and latency optimization is generated to guide service placement and task offloading. Experimental comparisons with benchmark algorithms in real-world application scenarios demonstrate that TBGP-HH consistently improves task processing reliability and reduces latency across diverse scenarios, with overall performance outperforming recent task offloading algorithms.

Key words: cloud-edge collaborative environment; tree-based genetic programming; hyper-heuristic algorithm; reliable and low-latency; task offloading

（责任编辑：王晓丽）