doi：10.13413/j.cnki.jdxblxb.2023418

基于偏好和虚拟适应度的两阶段依赖任务卸载算法

董立岩[1,2]，齐竞则[1]，刘元宁[1,2]，冯嘉辉[1]
（1. 吉林大学 计算机科学与技术学院，长春 130012；
2. 吉林大学 符号计算与知识工程教育部重点实验室，长春 130012）

摘要：针对云边端协同环境中依赖任务卸载时效率低以及任务卸载失败的问题，提出一种基于偏好和虚拟适应度的两阶段依赖任务卸载算法. 第一阶段，根据提出的二维卸载偏好因子对依赖任务的部分子任务进行直接卸载决策，从而有效缩小遗传算法初始种群的规模. 第二阶段，提出基于虚拟适应度的启发式交叉方法，并对基于参考点的快速非支配排序遗传算法(non-dominated sorting genetic algorithm Ⅲ，NSGA-Ⅲ)的交叉算子进行改进，保留了种群多样性并提升了算法收敛速度，最后使用改进的算法对所有依赖任务的子任务进行最优卸载决策集的搜索. 实验结果表明，与其他算法相比，该算法在任务完成时间、任务能耗和边缘云集群成本方面平均优化了 10.2%～18.3%，并且将任务失败率平均降低了 10.7%～25.6%.

关键词：云边端协同环境；依赖任务卸载；多目标优化；虚拟适应度；遗传算法
中图分类号：TP393 **文献标志码**：A **文章编号**：1671-5489(2024)04-0923-10

Two-Stage Dependent Task Offloading Algorithm Based on Preference and Virtual Fitness

DONG Liyan[1,2], QI Jingze[1], LIU Yuanning[1,2], FENG Jiahui[1]
(1. *College of Computer Science and Technology, Jilin University, Changchun* 130012, *China*;
2. *Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun* 130012, *China*)

Abstract: Aiming at the problem of low efficiency and failure of dependent task offloading in the cloud-edge-end architecture, we proposed a two-stage dependent task offloading algorithm based on preference and virtual fitness. In the first stage, based on the proposed two-dimensional offloading preference factor, direct offloading decisions were made for some sub-tasks of the dependent tasks, thus effectively reducing the size of the initial population of the genetic algorithm. In the second stage, we proposed a heuristic crossover method based on virtual fitness to improve the crossover operator of the fast non-dominated sorting genetic algorithm Ⅲ (NSGA-Ⅲ) based on reference points, which preserved the diversity of population and improved the convergence speed of the algorithm. Finally, we used the improved algorithm to search for the optimal offloading decision set for the

收稿日期：2023-10-09.

第一作者简介：董立岩(1966—)，男，汉族，博士，教授，博士生导师，从事人工智能、数据挖掘和云计算的研究，E-mail：dongly@jlu.edu.cn.

基金项目：国家自然科学基金(批准号：61471181)和吉林省科技发展计划项目(批准号：20230101054JC).

subtasks of all dependent tasks. The experimental results show that compared with other algorithms, the proposed algorithm optimizes task completion time, task energy consumption and edge cloud cluster cost by 10.2%—18.3% on average and reduces the task failure rate by 10.7%—25.6% on average.

Keywords: cloud-edge-end architecture; dependent task offloading; multi-objective optimization; virtual fitness; genetic algorithm

随着智能应用和物联网设备的快速发展[1]，应用程序产生的任务对计算资源和存储资源的需求显著激增[2]. 但本地移动设备资源和边缘计算资源[3-4]相对有限，无法支持所有任务的资源需求，并且传统云计算范式存在高延迟问题[5]，因此云边端协同计算架构是一个更具前景的解决方案[6-7]，其中边缘云集群能满足高实时性和低延迟的任务需求[8-9]，中心云集群能分担边缘云集群的负载压力[10].

真实场景中，许多应用任务由多个有前驱后继关系的子任务组成，这样的任务称为依赖任务[11]，而任务卸载是指将本地移动设备产生的任务卸载到云环境的虚拟机中执行. Xu 等[12]提出了一种基于博弈论的两阶段依赖任务卸载算法，但仅以能耗优化为主要目标. Yuan 等[13]提出了一种基于遗传模拟退火的粒子群优化算法，解决了云边端协同环境中依赖任务的卸载问题，但仅以降低成本为主要目标. 文献[14-16]针对边缘计算范式中依赖任务卸载问题都给出了有效的改进方法，但缺乏对边缘计算资源相对有限性的考虑，没有将边缘云与中心云相结合. Liu 等[17]提出了一种基于改进强度 Pareto 进化算法的多目标卸载算法，以解决依赖任务卸载问题，但仅考虑了延迟和能耗的目标，未考虑边缘服务器成本. Shahidinejad 等[18]利用基于拥挤度的快速非支配排序遗传(non-dominated sorting genetic algorithm Ⅱ，NSGA-Ⅱ)算法提出了一种基于元启发式的加载机制，对云边端协同环境中任务卸载问题进行以能耗和执行时间为目标的优化，但未考虑边缘云集群成本和任务失败率指标. 相比 NSGA-Ⅱ算法，NSGA-Ⅲ算法[19]引入了参考点对种群个体进行选择操作，从而增强其全局搜索能力[20]，更适合高维目标的优化.

本文将云边端协同环境中依赖任务的卸载问题视为多目标优化问题，构建任务完成时间、能耗和成本的多维优化目标，提出一种基于偏好和虚拟适应度的两阶段依赖任务卸载算法. 第一阶段，根据所有依赖任务信息和设备环境信息构建二维卸载偏好因子，从而得到部分子任务的卸载位置，进而对初始种群的部分基因进行预设定；第二阶段，本文提出基于虚拟适应度的交叉方法，从而改进 NSGA-Ⅲ算法的交叉操作，最后通过改进的 NSGA-Ⅲ算法对所有依赖任务的卸载决策进行 Pareto 最优解集的搜索.

1 模型定义

1.1 系统模型

云边端协同环境模型如图 1 所示. 本地移动设备层会产生大量待处理任务，可卸载至中心云和边缘云. 该体系架构将整个中心云集群层作为一个整体. 边缘云集群层包括 R 个边缘云集群，每个边缘云集群下划分出 U 个虚拟机、1 个基站并服务于 M 个本地移动设备 DEV_r. 依赖任务可表示为有向无环(DAG)图[21]，如图 2 所示. 本文所用符号及其含义如下：R 表示边缘云集群层下边缘云集群的数量；ec_r 表示边缘云集群层中编号为 r 的边缘云集群，$r\in\{1,2,\cdots,R\}$；M 表示 ec_r 所服务本地移动设备的数量；$T_n^{m,r}$ 表示 dev_r^m 需要执行的编号为 n 的依赖任务，$n\in\{1,2,\cdots,N\}$；X 表示 $T_n^{m,r}$ 子任务数；$t_{n_x}^{m,r}$ 表示编号为 n 的 $T_n^{m,r}$ 子任务，$x\in\{1,2,\cdots,X\}$；$\mathrm{pre}_{n_x}^{m,r}$ 表示 $t_{n_x}^{m,r}$ 的直接前驱任务集合；$\mathrm{suc}_{n_x}^{m,r}$ 表示 $t_{n_x}^{m,r}$ 的直接后继任务集合；$\mathrm{cpu}_{n_x}^{m,r}$ 表示完成 $t_{n_x}^{m,r}$ 执行所需的 CPU 周期数；$\mathrm{ram}_{n_x}^{m,r}$ 表示执行 $t_{n_x}^{m,r}$ 所需的内存量；$\mathrm{ip}_{n_x}^{m,r}$ 表示 $t_{n_x}^{m,r}$ 的输入数据量；$\mathrm{vm}_{n_x}^{m,r}$ 表示 $t_{n_x}^{m,r}$ 卸载到 ec_r 上后被安排执行的虚拟机编号；α 表示路径损耗因子；$E_n^{m,r}$ 表示 $T_n^{m,r}$ 各子任务的前驱后继关系；$K_n^{m,r}$ 表示 $T_n^{m,r}$ 的卸载决策集；$k_{n_x}^{m,r}$ 表示 $t_{n_x}^{m,r}$ 的卸载决策；$\varepsilon_{r,u}$ 表示 vm_u^r 的单位时间成本的系数；dev_r^m 表示 ec_r 所服务编号为 m 的本地移动设备，

$m\in\{1,2,\cdots,M\}$；$\mathrm{cpu}_{r,m}^{\mathrm{total}}$表示 dev_r^m 的 CPU 总量；$\mathrm{cpu}_{r,m}^{\mathrm{rest}}$表示 dev_r^m 的 CPU 剩余量；$\mathrm{ram}_{r,m}^{\mathrm{total}}$表示 dev_r^m 的 RAM 总量；$\mathrm{ram}_{r,m}^{\mathrm{rest}}$表示 dev_r^m 的 RAM 剩余量；N 表示 dev_r^m 需要执行的依赖任务数；ξ 表示设备芯片架构的有效电容系数[15]；f 表示设备的计算能力；P 表示设备的发射功率；U 表示 ec_r 下虚拟机数量；vm_u^r 表示 ec_r 下编号为 u 的虚拟机，$u\in\{1,2,\cdots,U\}$；$\mathrm{cpu}_{r,u}^{\mathrm{total}}$表示 vm_u^r 的 CPU 总量；$\mathrm{cpu}_{r,u}^{\mathrm{rest}}$表示 vm_u^r 的 CPU 剩余量；$\mathrm{ram}_{r,u}^{\mathrm{total}}$表示 vm_u^r 的 RAM 总量；$\mathrm{ram}_{r,u}^{\mathrm{rest}}$表示 vm_u^r 的 RAM 剩余量；δ 表示 ec_r 虚拟机资源的最佳利用率上限[22]；UR_u^r 表示 ec_r 中编号为 u 的虚拟机的当前资源利用率.

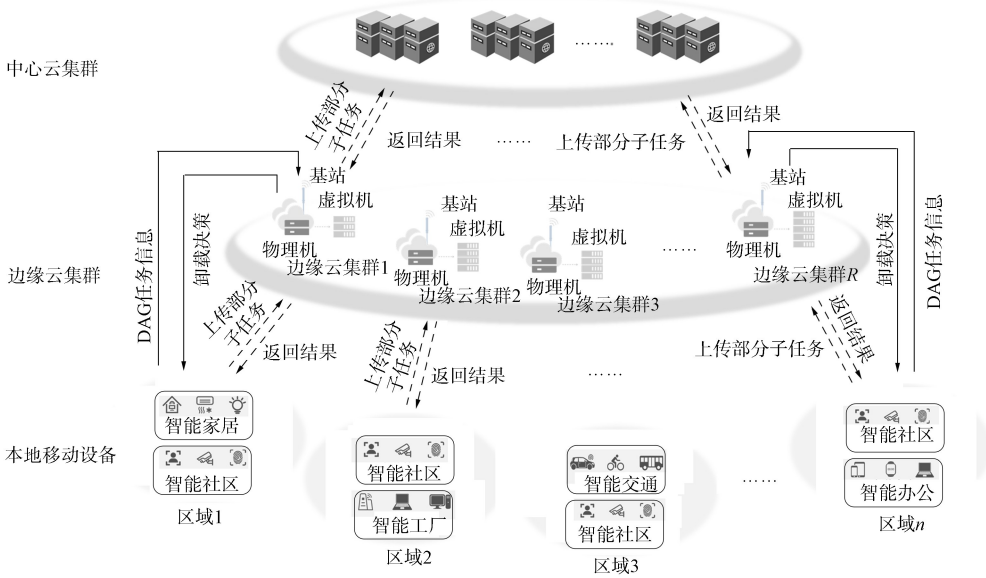

图 1　云边端协同体系架构

Fig. 1　Cloud-edge-end collaborative architecture

1.2　执行模型

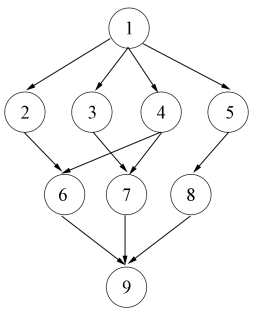

图 2　依赖任务 DAG

Fig. 2　Dependent task DAG

本文执行模型包含执行时间和执行能耗. 若依赖任务 $T_n^{m,r}$ 的子任务 $t_{n_x}^{m,r}$在计算能力为 f 且有效电容系数为 ξ 的设备 d 上执行，$\mathrm{cpu}_{n_x}^{m,r}$为 $t_{n_x}^{m,r}$执行所需的 CPU 周期数，则其执行时间 $TE_{n_x,d}^{m,r}$和执行能耗 $EE_{n_x,d}^{m,r}$分别为

$$TE_{n_x,d}^{m,r}=\frac{\mathrm{cpu}_{n_x}^{m,r}}{f},\tag{1}$$

$$EE_{n_x,d}^{m,r}=\xi^2(f)\cdot\mathrm{cpu}_{n_x}^{m,r}.\tag{2}$$

1.3　通信模型

当多个设备同时占用一个信道时，由 Shannon 公式可得设备 a 到设备 b 的传输速率 V_a^b 为

$$V_a^b=W\log_2\left(1+\frac{S_a^b}{N_a}\right),\tag{3}$$

其中 W 为信道带宽，S_a^b 为信号功率，N_a 为噪声功率. 则 $t_{n_x}^{m,r}$的输入数据 $\mathrm{ip}_{n_x}^{m,r}$从设备 a 卸载到设备 b 的传输时间 $TT_{n_x,a,b}^{m,r}$和传输能耗 $ET_{n_x,a,b}^{m,r}$分别为

$$TT_{n_x,a,b}^{m,r}=\frac{\mathrm{ip}_{n_x}^{m,r}}{V_a^b},\tag{4}$$

$$ET_{n_x,a,b}^{m,r}=TT_{n_x,a,b}^{m,r}\cdot P_a,\tag{5}$$

其中 P_a 为设备 a 的发射功率.

2 算法设计

2.1 优化目标构建

$t_{n_x}^{m,r}$的卸载策略$k_{n_x}^{m,r}$为

$$k_{n_x}^{m,r}=\begin{cases}-1, & \text{中心云集群执行},\\ 0, & \text{本地移动设备执行},\\ 1, & \text{边缘云集群执行}.\end{cases} \tag{6}$$

其中：$k_{n_x}^{m,r}=-1$，表示$t_{n_x}^{m,r}$卸载到中心云集群执行；$k_{n_x}^{m,r}=0$，表示$t_{n_x}^{m,r}$在本地移动设备dev_r^m执行；$k_{n_x}^{m,r}=1$，表示$t_{n_x}^{m,r}$卸载到边缘云集群ec_r执行.

由执行模型可得$t_{n_x}^{m,r}$的执行时间$TE_{n_x}^{m,r}$和执行能耗$EE_{n_x}^{m,r}$：

$$TE_{n_x}^{m,r}=\left\lfloor\frac{k_{n_x}^{m,r}+1}{2}\right\rfloor\cdot TE_{n_x,\mathrm{ec}_r}^{m,r}+\left\lfloor\frac{1-k_{n_x}^{m,r}}{2}\right\rfloor\cdot TE_{n_x,c}^{m,r}+\left|\left|k_{n_x}^{m,r}\right|-1\right|\cdot TE_{n_x,l}^{m,r}, \tag{7}$$

$$EE_{n_x}^{m,r}=\left\lfloor\frac{k_{n_x}^{m,r}+1}{2}\right\rfloor\cdot EE_{n_x,\mathrm{ec}_r}^{m,r}+\left\lfloor\frac{1-k_{n_x}^{m,r}}{2}\right\rfloor\cdot EE_{n_x,c}^{m,r}+\left|\left|k_{n_x}^{m,r}\right|-1\right|\cdot EE_{n_x,l}^{m,r}. \tag{8}$$

若$t_{n_x}^{m,r}$在ec_r执行，则执行时间和执行能耗分别为$TE_{n_x,\mathrm{er}_r}^{m,r}$和$EE_{n_x,\mathrm{er}_r}^{m,r}$；若$t_{n_x}^{m,r}$在$\mathrm{dev}_r^m$执行，则执行时间和执行能耗分别为$TE_{n_x,l}^{m,r}$和$EE_{n_x,l}^{m,r}$；若$t_{n_x}^{m,r}$在中心云集群执行，则执行时间和执行能耗分别为$TE_{n_x,c}^{m,r}$和$EE_{n_x,c}^{m,r}$.

由通信模型，$\mathrm{ip}_{n_x}^{m,r}$的传输时间和传输能耗分别为$TT_{n_x}^{m,r}$和$ET_{n_x}^{m,r}$：

$$TT_{n_x}^{m,r}=\left\lfloor\frac{k_{n_x}^{m,r}+1}{2}\right\rfloor\cdot TT_{n_x,\mathrm{ec}_r}^{m,r}+\left\lfloor\frac{1-k_{n_x}^{m,r}}{2}\right\rfloor\cdot TT_{n_x,c}^{m,r}, \tag{9}$$

$$ET_{n_x}^{m,r}=\left\lfloor\frac{k_{n_x}^{m,r}+1}{2}\right\rfloor\cdot ET_{n_x,\mathrm{ec}_r}^{m,r}+\left\lfloor\frac{1-k_{n_x}^{m,r}}{2}\right\rfloor\cdot ET_{n_x,c}^{m,r}. \tag{10}$$

若$t_{n_x}^{m,r}$在ec_r执行，则$\mathrm{ip}_{n_x}^{m,r}$的传输时间和传输能耗分别为$TT_{n_x,\mathrm{ec}_r}^{m,r}$和$ET_{n_x,\mathrm{ec}_r}^{m,r}$；若$t_{n_x}^{m,r}$在中心云集群执行，则$\mathrm{ip}_{n_x}^{m,r}$的传输时间和传输能耗分别为$TT_{n_x,c}^{m,r}$和$ET_{n_x,c}^{m,r}$.

就绪时间$TR_{n_x}^{m,r}$是指$t_{n_x}^{m,r}$可以开始执行的最早时间，即$t_{n_x}^{m,r}$前驱子任务最长完成时间与$\mathrm{ip}_{n_x}^{m,r}$传输时间相比的最大值，即

$$TR_{n_x}^{m,r}=\max\{TT_{n_x}^{m,r},\ \max_{i\in \mathrm{per}_{n_x}^{m,r}}\{TR_{n_i}^{m,r}+TE_{n_i}^{m,r}\}\}. \tag{11}$$

$T_n^{m,r}$的完成时间$TF_n^{m,r}$为结束任务$t_{n_x}^{m,r}$的就绪时间$TR_{n_x}^{m,r}$与执行时间$TE_{n_x}^{m,r}$的总和，即

$$TF_n^{m,r}=TR_{n_x}^{m,r}+TE_{n_x}^{m,r}. \tag{12}$$

$t_{n_x}^{m,r}$的能耗$E_{n_x}^{m,r}$为$\mathrm{ip}_{n_x}^{m,r}$传输能耗与执行能耗的总和，$T_n^{m,r}$的总能耗$E_n^{m,r}$为所有子任务的总能耗与计算依赖任务卸载方案算法能耗$E_{n_{\mathrm{DAG}}}^{m,r}$的总和，即

$$E_n^{m,r}=\sum_{i=1}^{X}E_{n_x}^{m,r}+E_{n_{\mathrm{DAG}}}^{m,r}=\sum_{i=1}^{X}(ET_{n_i}^{m,r}+EE_{n_i}^{m,r})+E_{n_{\mathrm{DAG}}}^{m,r}. \tag{13}$$

$T_n^{m,r}$的边缘云集群成本根据子任务执行时间计算，即其所有子任务的边缘云集群成本之和$\mathrm{CEC}_n^{m,r}$：

$$\mathrm{CEC}_n^{m,r}=\sum_{i\in\{x\mid k_{n_x}^{m,r}=1\ \&\ x\in\{1,2,\cdots,X\}\}}\mathrm{CEC}_{n_x}^{m,r}=$$

$$\sum_{i\in\{x\mid k_{n_x}^{m,r}=1\ \&\ x\in\{1,2,\cdots,X\}\}}TE_{n_x}^{m,r}\cdot\varepsilon_{r,\mathrm{vm}_{n_i}^{m,r}}. \tag{14}$$

基于上述分析，本文将云边端协同环境中依赖任务卸载问题形式化为一个多目标优化问题P，构建任务完成时间$TF_n^{m,r}$、任务能耗$E_n^{m,r}$和边缘云集群成本$\mathrm{CEC}_n^{m,r}$的多维优化目标，即

$$\mathrm{P}:\ \min_{k_{n_x}^{m,r},\mathrm{vm}_{n_x}^{m,r}}\{TF_n^{m,r},E_n^{m,r},\mathrm{CEC}_n^{m,r}\},$$

$$\text{s.t. } C_1: k_{n_x}^{m,r}=\begin{cases}-1, & \min\limits_{i=\{1,2,\cdots,U\}}\{UR_i^r\}\geqslant\delta,\\ 0, & \text{本地移动设备执行},\\ 1, & \min\limits_{i=\{1,2,\cdots,U\}}\{UR_i^r\}<\delta,\end{cases}\tag{15}$$

$$C_2:\ \mathrm{vm}_{n_x}^{m,r}=\begin{cases}j, & k_{n_x}^{m,r}=1,\\ 0, & k_{n_x}^{m,r}=0\ \text{或}\ k_{n_x}^{m,r}=-1,\end{cases}\qquad\forall j\in\{1,2,\cdots,U\},$$

其中：约束 C_1 表示 $k_{n_x}^{m,r}$ 卸载决策的取值范围和约束，UR_i^r 表示 ec_r 中编号为 i 虚拟机的资源利用率，ec_r 中虚拟机资源利用率的最小值表示 ec_r 资源利用率，当 ec_r 资源利用率未达到最优上限 δ 时，将任务卸载到边缘云集群执行，否则，将任务卸载到中心云集群执行；约束 C_2 表示当卸载决策 $k_{n_x}^{m,r}\neq1$ 时，子任务的虚拟机卸载编号 $\mathrm{vm}_{n_x}^{m,r}$ 为 0，否则为 $\mathrm{vm}_{n_x}^{m,r}$ 分配数值，数值范围为边缘云集群 ec_r 中的虚拟机编号集合.

2.2　基于偏好的卸载决策算法

依赖任务可划分为数据密集型和计算密集型[23]，这两种类型不互斥. 本文提出二维卸载偏好因子对子任务进行类型偏好设置，设 $KR_n^{m,r}$ 为 $T_n^{m,r}$ 子任务的二维卸载偏好因子集合，$kr_{n_x}^{m,r}$ 表示 $t_{n_x}^{m,r}$ 的二维卸载偏好因子，$kr_{n_x}^{m,r}$ 由一个二元组$(\mathrm{compute}_{n_x}^{m,r},\mathrm{data}_{n_x}^{m,r})$表示：

$$kr_{n_x}^{m,r}=\begin{cases}(1,1), & \mathrm{cpu}_{n_x}^{m,r}<\mathrm{cpu}l_{r,m}^{\mathrm{rest}},\ \mathrm{ram}_{n_x}^{m,r}<\mathrm{ram}l_{r,m}^{\mathrm{rest}},\ \mathrm{ip}_{n_x}^{m,r}<\overline{\mathrm{ip}_{n_x}^{m,r}},\\ (1,0), & \mathrm{cpu}_{n_x}^{m,r}<\mathrm{cpu}l_{r,m}^{\mathrm{rest}},\ \mathrm{ram}_{n_x}^{m,r}<\mathrm{ram}l_{r,m}^{\mathrm{rest}},\ \mathrm{ip}_{n_x}^{m,r}\geqslant\overline{\mathrm{ip}_{n_x}^{m,r}},\\ (-1,1), & \mathrm{cpu}l_{r,m}^{\mathrm{rest}}\leqslant\mathrm{cpu}_{n_x}^{m,r}<\mathrm{cpu}l_{r,m}^{\mathrm{total}},\ \mathrm{ram}l_{r,m}^{\mathrm{rest}}\leqslant\mathrm{ram}_{n_x}^{m,r}<\mathrm{ram}l_{r,m}^{\mathrm{total}},\ \mathrm{ip}_{n_x}^{m,r}<\overline{\mathrm{ip}_{n_x}^{m,r}},\\ (-1,0), & \mathrm{cpu}l_{r,m}^{\mathrm{rest}}\leqslant\mathrm{cpu}_{n_x}^{m,r}<\mathrm{cpu}l_{r,m}^{\mathrm{total}},\ \mathrm{ram}l_{r,m}^{\mathrm{rest}}\leqslant\mathrm{ram}_{n_x}^{m,r}<\mathrm{ram}l_{r,m}^{\mathrm{total}},\ \mathrm{ip}_{n_x}^{m,r}\geqslant\overline{\mathrm{ip}_{n_x}^{m,r}},\\ (0,1), & \mathrm{cpu}_{n_x}^{m,r}\geqslant\mathrm{cpu}l_{r,m}^{\mathrm{total}},\ \mathrm{ram}_{n_x}^{m,r}\geqslant\mathrm{ram}l_{r,m}^{\mathrm{total}},\ \mathrm{ip}_{n_x}^{m,r}<\overline{\mathrm{ip}_{n_x}^{m,r}},\\ (0,0), & \mathrm{cpu}_{n_x}^{m,r}\geqslant\mathrm{cpu}l_{r,m}^{\mathrm{total}},\ \mathrm{ram}_{n_x}^{m,r}\geqslant\mathrm{ram}l_{r,m}^{\mathrm{total}},\ \mathrm{ip}_{n_x}^{m,r}\geqslant\overline{\mathrm{ip}_{n_x}^{m,r}},\end{cases}\tag{16}$$

$$\overline{\mathrm{ip}_n^{m,r}}=\frac{1}{X}\sum_{i=1}^{X}\mathrm{ip}_{n_x}^{m,r},\tag{17}$$

其中：$\mathrm{compute}_{n_x}^{m,r}$ 表示是否为计算密集型，1 为否，0 为是，−1 趋于中间；$\mathrm{data}_{n_x}^{m,r}$ 表示是否为数据密集型，1 为否，0 为是；$\overline{\mathrm{ip}_n^{m,r}}$ 为 $T_n^{m,r}$ 子任务输入数据的平均值.

首先根据式(16)得到 $KR_n^{m,r}$，然后 dev_r^m 将所有依赖任务信息和 $KR_n^{m,r}$ 发送到 ec_r 基站，根据 ec_r 的计算属性条件 $\Theta_{n_x}^{m,r}$，得到部分子任务的卸载决策 $K_n^{m,r}$. 计算属性条件 $\Theta_{n_x}^{m,r}$ 定义为

$$\Theta_{n_x}^{m,r}=\begin{cases}1, & \mathrm{cpu}_{n_x}^{m,r}<\delta\cdot\mathrm{cpu}_{\min}^r,\ \mathrm{ram}_{n_x}^{m,r}<\delta\cdot\mathrm{ram}_{\min}^r,\\ -1, & \mathrm{cpu}_{n_x}^{m,r}\geqslant\delta\cdot\mathrm{cpu}_{\max}^r,\ \mathrm{ram}_{n_x}^{m,r}\geqslant\delta\cdot\mathrm{ram}_{\max}^r,\\ 0, & \text{其他},\end{cases}\tag{18}$$

其中 $\mathrm{ram}_{\max}^r$ 和 $\mathrm{cpu}_{\max}^r$ 分别为 ec_r 中虚拟机的最大剩余内存容量和最大剩余 CPU 容量，$\mathrm{ram}_{\min}^r$ 和 $\mathrm{cpu}_{\min}^r$ 分别为 ec_r 中虚拟机的最小剩余内存容量和最小剩余 CPU 容量.

综上，基于偏好的卸载决策算法如下.

算法 1　基于偏好的卸载决策算法.

输入：任务信息 $T_n^{m,r}$，本地移动设备信息 dev_r^m，边缘云集群信息 ec_r；

输出：二维卸载偏好因子集合 $KR_n^{m,r}$，$T_n^{m,r}$ 部分子任务的卸载决策集合 $K_n^{m,r}$；

步骤 1) 初始化 $KR_n^{m,r}$，$K_n^{m,r}$，$\Theta_n^{m,r}$ 和 $Q_n^{m,r}$，根据式(17)计算 $\overline{\mathrm{ip}_n^{m,r}}$；

步骤 2) For $t_{n_x}^{m,r}\in T_n^{m,r}$ do：

步骤 3) 根据式(16)计算 $kr_{n_x}^{m,r}=(\mathrm{compute}_{n_x}^{m,r},\mathrm{data}_{n_x}^{m,r})$并且$\{kr_{n_x}^{m,r}\}\cup KR_n^{m,r}$；

步骤 4) If $ip_{n_x}^{m,r}<\overline{ip_n^{m,r}}$ then $data_{n_x}^{m,r}=1$ 并且$\{t_{n_x}^{m,r}\}\cup Q_n^{m,r}$;

步骤 5) Else $data_{n_x}^{m,r}=0$;

步骤 6) If $compute_{n_x}^{m,r}=1$ then $k_{n_x}^{m,r}=0$ 并且$\{k_{n_x}^{m,r}\}\cup K_n^{m,r}$;

步骤 7) Else $\{t_{n_x}^{m,r}\}\cup Q_n^{m,r}$;

步骤 8) End if

步骤 9) End if

步骤 10) End for

步骤 11) For $t_{n_x}^{m,r}\in Q_n^{m,r}$ do:

步骤 12) 根据式(18)计算 $\Theta_{n_x}^{m,r}$;

步骤 13) If $\Theta_{n_x}^{m,r}=1$, ($kr_{n_x}^{m,r}=(0,1)$或 $kr_{n_x}^{m,r}=(0,0)$) then $k_{n_x}^{m,r}=1$ 并且$\{k_{n_x}^{m,r}\}\cup K_n^{m,r}$;

步骤 14) Else if $\Theta_{n_x}^{m,r}=-1$, ($kr_{n_x}^{m,r}=(0,1)$或 $kr_{n_x}^{m,r}=(0,0)$) then $k_{n_x}^{m,r}=-1$ 并且$\{k_{n_x}^{m,r}\}\cup K_n^{m,r}$;

步骤 15) End if

步骤 16) End for.

2.3 基于虚拟适应度交叉方法的 NSGA-Ⅲ算法

本文对 NSGA-Ⅲ算法的初始种群、交叉操作和变异操作进行优化和改进.

构建初始种群：首先由随机的方式产生初始种群，其中每个个体包含 N 条染色体，染色体数量等于依赖任务的数量，每条染色体代表该依赖任务的子任务的一组决策，每条染色体上基因的数量等于该染色体对应的依赖任务的子任务数量，每个基因代表每个子任务的决策. 将 2.2 节得到的部分子任务卸载决策填入初始种群，即算法会根据得到的部分卸载策略对每个个体的部分基因进行预设定，通过这种方式对初始种群进行预处理，从而优化初始种群.

交叉操作：本文将交叉操作分为对初始种群交叉操作和对非初始种群交叉操作. 对于非初始种群，使用基于虚拟适应度的启发式交叉方法. 在对种群个体进行非支配排序的过程中，需要给每个非支配层指定一个虚拟适应度值，非支配排序层级越低，虚拟适应度值越大；反之，虚拟适应度值越小. 这样可以保证在交叉操作中层级较低的非支配个体有更多机会被选择进入下一代，使算法以最快的速度收敛于最优区域. 对基因为 P_1 和 P_2 的父代个体，其虚拟适应度分别为 V_1 和 V_2，设 P_1 的非支配排序层级低于 P_2 的非支配排序层级，则 $V_1>V_2$，其交叉操作如下，并生成后代个体 S_1 和 S_2：

$$\begin{cases} S_1=\dfrac{V_1}{V_1+V_2}\cdot P_1+\dfrac{V_2}{V_1+V_2}\cdot P_2, \\ S_2=\dfrac{V_2}{V_1+V_2}\cdot P_1+\dfrac{V_1}{V_1+V_2}\cdot P_2. \end{cases} \tag{19}$$

变异操作：在依赖任务卸载问题的场景下，变异率太高会导致收敛过慢，退化为随机搜索，变异率太低则会导致陷入局部最优；当变异算子保持在[0.010,0.025]时，结果能保持在较优的状态[24].

基于虚拟适应度交叉的 NSGA-Ⅲ算法如下.

算法 2 基于虚拟适应度交叉方法的 NSGA-Ⅲ算法.

输入：$KR_n^{m,r}$, $K_n^{m,r}$, $T_n^{m,r}$, dev_r^m, ec_r, 中心云集群信息；

输出：依赖任务卸载决策的 Pareto 最优解集；

步骤 1) 随机创建初始种群 P，根据 $K_n^{m,r}$ 对 P 部分基因进行预设定，得到种群 P^*，$t=1$；

步骤 2) While $t\leqslant$最大迭代次数 do:

步骤 3) If 首次迭代 then

步骤 4) 进行随机交叉和变异操作，得到新的子代种群 Q_t，获取父子代并集 $R_t=P^*\cup Q_t$；

步骤 5) Else 通过式(19)进行交叉和变异操作，得到新的子代种群 Q_t，获取父子代并集 $R_t=P_t\cup Q_t$；

步骤 6) 对 R_t 进行快速非支配等级划分(F_1, F_2, …)=FNS(R_t)，初始化精英选择的子代集合 S_t，$i=1$；

步骤 7)　while $|S_t|<N_0$ do://每次迭代要选出 N_0 个个体作为新一代种群

步骤 8)　$S_t=S_t\bigcup F_i$；

步骤 9)　$i=i+1$；

步骤 10)　End while

步骤 11)　得到最后一个前沿 $l=i-1$；

步骤 12)　If $|S_t|=N$ then $P_{t+1}=S_t$；

步骤 13)　Else 基于参考点进行选择最后一个前沿的操作[18]；

步骤 14)　End if

步骤 15)　$t=t+1$；

步骤 16) End while.

3　实验结果与分析

3.1　对比算法

将本文算法与下列 5 种算法进行对比实验.

1) 完全本地执行算法(AL)：将所有依赖任务的子任务放置在本地移动设备上处理.

2) 完全边缘云集群卸载算法(AM)：将所有依赖任务的子任务卸载到边缘云集群处理.

3) 随机卸载算法(R)：所有依赖任务的子任务通过随机算法进行任务卸载.

4) NSGA-Ⅱ算法：所有子任务通过 NSGA-Ⅱ算法得到卸载决策的 Pareto 最优解集.

5) NSGA-Ⅲ算法：所有子任务通过 NSGA-Ⅲ算法得到卸载决策的 Pareto 最优解集.

3.2　仿真环境设置

基于图 1 模型，本文设计一个由中心云集群、边缘云集群和本地移动设备组成的仿真环境. 所有实验测试均在如表 1 所示的 3 种型号设备中进行，仿真环境相关参数列于表 2. 本文实验任务数据集由仿真环境任务生成器随机生成得到 300～900 个依赖任务.

表 1　实验设备信息

Table 1　Experimental equipment information

设备型号	CPU	RAM/GB
Legion R9000p2021H	AMD Ryzen 7 5800H	16.0
Legion XiaoXin16Pro	Intel Core i5-12500H	16.0
Legion XiaoXinRui7000	Intel Core i7-7700HQ	12.0

表 2　仿真环境参数设置

Table 2　Simulation environment parameter settings

参数	取值	参数	取值
边缘云服务器数量 R	[5,10]	边缘云服务器计算能力 f^r_{edge}/GHz	[1,2]
每个边缘云服务器下的本地设备数量 M	[20,100]	本地设备计算能力 $f^{m,r}_{\text{local}}$/GHz	10
本地设备需要处理的依赖任务数 N	[4,12]	边缘云服务器 CPU 总量 $\text{cpu}^{\text{total}}_r$/Mips	[50 000,80 000]
每个依赖任务中的子任务数 X	[4,12]	边缘云服务器 RAM 总量 $\text{ram}^{\text{total}}_r$/B	[8 000,20 000]
中心云集群服务器发射功率 P_c/W	[0.5,1]	边缘云服务器中虚拟机 CPU 总量 $\text{cpu}^{\text{total}}_{r,u}$/Mips	[2 000,5 000]
边缘云服务器发射功率 P_r/W	[1,2]	边缘云服务器中虚拟机 RAM 总量 $\text{ram}^{\text{total}}_{r,u}$/MB	[500,1 000]
本地设备发射功率 P^r_m/W	[2,4]	边缘云服务器中虚拟机单位时间成本 $\varepsilon_{r,u}$	[0.1,0.5]
路径损耗因子∂	4	本地设备 CPU 总量 $\text{cpu}l^{\text{total}}_{r,m}$/Mips	[1 000,2 000]
中心云集群服务器计算能力 f_{cloud}/GHz	[0.3,0.8]	本地设备 RAM 总量 $\text{ram}l^{\text{total}}_{r,m}$/MB	[250,500]

3.3　模拟实验结果分析

将本文算法与其他 5 种基线算法的实验结果进行比较，实验结果均为 5 次实验取平均值. 本文首先统计了平均完成时间、平均完成能耗以及边缘云集群成本这 3 个指标的性能对比，结果列于表 3 和表 4.

表3 不同算法本地移动设备数量对实验指标的影响

Table 3 Impact of number of local mobile devices on experimental indicators by different algorithms

对比指标	算法	本地移动设备数量				
		20	40	60	80	100
平均任务完成时间/ms	AL	785.81	786.68	827.32	808.71	846.06
	AM	248.59	265.67	307.12	360.54	438.39
	R	367.06	286.86	405.82	368.14	535.91
	NSGA-Ⅱ	263.55	276.35	293.64	329.66	377.15
	NSGA-Ⅲ	214.99	223.54	248.65	287.81	353.91
	本文	209.45	213.64	232.77	265.46	310.66
平均任务能耗/(kW·h)	AL	5.91	5.53	5.84	6.04	5.74
	AM	7.85	8.43	11.18	13.95	15.98
	R	8.23	6.68	12.44	9.86	13.59
	NSGA-Ⅱ	6.31	7.34	8.76	10.49	12.01
	NSGA-Ⅲ	6.22	6.93	7.67	9.54	11.59
	本文	6.15	6.61	7.21	8.83	10.12
平均任务边缘云集群成本	AL	0.00	0.00	0.00	0.00	0.00
	AM	16.45	30.47	45.76	58.68	73.79
	R	15.23	10.35	34.45	29.13	46.17
	NSGA-Ⅱ	10.77	17.86	24.27	31.76	42.17
	NSGA-Ⅲ	8.25	14.75	21.32	28.16	38.39
	本文	7.73	12.98	18.46	24.54	32.99

表4 不同算法子任务数量对实验指标的影响

Table 4 Impact of number of subtasks on experimental indicators by different algorithms

对比指标	算法	依赖任务的子任务数量				
		4	6	8	10	12
平均任务完成时间/ms	AL	533.685	779.81	1 042.21	1 385.50	1 625.54
	AM	215.06	337.82	484.67	570.12	689.54
	R	231.17	352.39	401.86	496.52	584.07
	NSGA-Ⅱ	174.63	233.9	316.95	393.88	507.16
	NSGA-Ⅲ	156.99	214.92	293.54	365.10	452.91
	本文	150.12	196.78	261.37	320.54	389.66
平均任务能耗/(kW·h)	AL	3.24	5.98	8.84	13.04	20.74
	AM	4.94	7.85	11.18	16.95	23.98
	R	5.01	8.23	12.44	15.36	21.59
	NSGA-Ⅱ	3.71	6.31	8.46	10.49	13.01
	NSGA-Ⅲ	3.50	6.02	7.87	9.54	11.69
	本文	3.31	5.65	7.21	8.83	10.52
平均任务边缘云集群成本	AL	0.00	0.00	0.00	0.00	0.00
	AM	8.47	16.45	34.76	46.68	61.79
	R	6.93	15.23	31.05	30.13	46.17
	NSGA-Ⅱ	5.77	10.77	17.87	27.66	40.97
	NSGA-Ⅲ	5.45	9.25	15.72	23.56	35.89
	本文	5.02	8.43	14.06	21.04	32.09

由表3和表4可见：AL算法中所有任务都在本地执行，其边缘云集群成本为0且平均任务完成时间最长，随着依赖任务的子任务数量增多，平均任务完成时间和能耗显著增长；AM算法中所有任务都在边缘云服务器执行，其边缘云集群成本最高，当本地移动设备或依赖任务子任务数的数量增加时，边缘云服务器的负载也越来越高；R算法通过随机方式选择卸载的位置，其指标通常有较大的波动；NSGA-Ⅱ算法、NSGA-Ⅲ算法和本文算法都属于多目标优化算法，与上述3种算法相比，这3种

算法性能更好. 本文算法基于 NSGA-Ⅲ算法，通过二维卸载偏好因子优化初始种群，基于虚拟适应度改进了遗传算法的交叉因子，在保证种群多样性的基础上增加算法收敛速度，减少算法搜索时间，使实验指标呈现出更优的任务卸载性能. 相比其他两种多目标优化算法，本文算法对上述 3 个指标平均优化了 10.2%～18.3%.

为进一步评估本文算法，下面对 6 种算法的任务失败率指标进行对比实验. 实验设置平均任务完成时间的 2 倍作为任务最长容忍完成时间，通过判断任务实际执行时间是否超出任务最长容忍完成时间判断任务是否失败. 实验结果如图 3 和图 4 所示. 由图 3 和图 4 可见，相比其他多目标优化算法，本文算法将任务失败率平均降低了 10.7%～25.6%，并且随着本地移动设备数量和子任务数量的上升，本文算法任务失败率的上升趋势明显缓于其他算法的上升趋势.

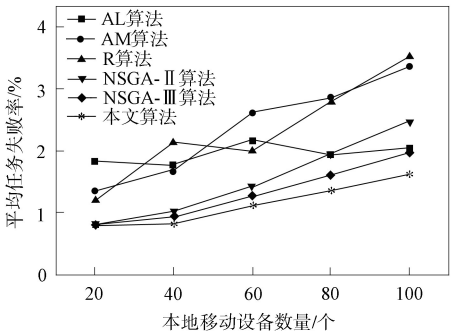

图 3　不同算法本地移动设备数量对任务失败率的影响

Fig. 3　Impact of number of local mobile devices on task failure rate by different algorithms

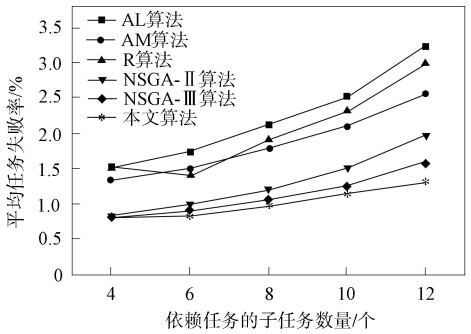

图 4　不同算法依赖任务的子任务数量对任务失败率的影响

Fig. 4　Impact of number of subtasks of dependent tasks on task failure rate by different algorithms

综上所述，针对云边端协同环境中依赖任务卸载时效率低以及任务卸载失败的问题，本文在云边端协同环境中提出了一种基于偏好和虚拟适应度的两阶段依赖任务卸载算法. 首先根据实时环境计算出依赖任务的二维卸载偏好因子，并通过二维卸载偏好因子对部分子任务进行直接偏好决策，使用该部分决策对初始种群进行预处理，最后通过基于虚拟适应度的启发式交叉操作的 NSGA-Ⅲ算法对全部任务的卸载决策进行最优解集的搜索，从而在缩短算法搜索时间的情况下，获取 Pareto 最优解集，同时降低了任务失败率. 实验结果表明，本文算法能在合理卸载子任务的基础上加快算法收敛速率，呈现更好的性能和效率.

参　考　文　献

[1] GOUDARZI M, WU H, PALANISWAMI M, et al. An Application Placement Technique for Concurrent IoT Applications in Edge and Fog Computing Environments [J]. IEEE Transactions on Mobile Computing, 2021, 20(4): 1298-1311.

[2] BHARADWAJ H K, AGARWAL A, CHAMOLA V, et al. A Review on the Role of Machine Learning in Enabling IoT Based Healthcare Applications [J]. IEEE Access, 2021, 9: 38859-38890.

[3] SUN Z J, YANG H, LI C, et al. Cloud-Edge Collaboration in Industrial Internet of Things: A Joint Offloading Scheme Based on Resource Prediction [J]. IEEE Internet Things Journal, 2022, 9(18): 17014-17025.

[4] 罗新刚，王万银. 基于正则化思想的 tilt-Euler 法在边缘深度反演中的应用 [J]. 吉林大学学报(地球科学版)，2024，54(2)：633-646. (LUO X G, WANG W Y. Application of Tilt-Euler Method Based on Regularization in Edge Depth Inversion [J]. Journal of Jilin University (Earth Science Edition), 2024, 54(2): 633-646.)

[5] WANG S Z, WANG W L, JIA Z T, et al. Flexible Task Scheduling Based on Edge Computing and Cloud Collaboration [J]. Computing System Science and Engineering, 2022, 42(3): 1241-1255.

[6] ZHOU H, WANG Z N, CHENG N, et al. Stackelberg-Game-Based Computation Offloading Method in Cloud-Edge Computing Networks [J]. IEEE Internet Things Journal, 2022, 9(17): 16510-16520.

[7] GAO J X, CHANG R, YANG Z P, et al. A Task Offloading Algorithm for Cloud-Edge Collaborative System Based on Lyapunov Optimization [J]. Cluster Computing: The Journal of Networks Software Tools and

Applications, 2023, 26(1): 337-348.
[8] TONG Z, DENG X M, MEI J, et al. Response Time and Energy Consumption Co-offloading with SLRTA Algorithm in Cloud-Edge Collaborative Computing [J]. Future Generation Computer Systems, 2022, 129: 64-76.
[9] SUN X, TIAN C L, HU C H, et al. Privacy-Preserving and Verifiable SRC-Based Face Recognition with Cloud-Edge Server Assistance [J]. Computing and Security, 2022, 118: 102740-1-102740-14.
[10] GUO K, ZHANG R L. Fairness-Oriented Computation Offloading for Cloud-Assisted Edge Computing [J]. Future Generation Computer Systems, 2022, 128: 132-141.
[11] ZHANG Y F, CHEN J, ZHOU Y C, et al. Dependent Task Offloading with Energy-Latency Tradeoff in Mobile Edge Computing [J]. IET Communications, 2022, 16(17): 1993-2001.
[12] XU F, XIE Y, SUN Y Y, et al. Two-Stage Computing Offloading Algorithm in Cloud-Edge Collaborative Scenarios Based on Game Theory [J]. Computing Electrical Engineering, 2022, 97: 107624-1-107624-15.
[13] YUAN H T, HU Q L, WANG M J, et al. Cost-Minimized User Association and Partial Offloading for Dependent Tasks in Hybrid Cloud-Edge Systems [C]//IEEE 18th International Conference on Automation Science and Engineering. Piscataway, NJ: IEEE, 2022: 1059-1064.
[14] KHALID M H, AHMED I A, MARWA M K, et al. New Improved Multi-objective Gorilla Troops Algorithm for Dependent Tasks Offloading Problem in Multi-access Edge Computing [J]. Journal of Grid Computing, 2023, 21(2): 21-1-21-24.
[15] SONG F H, XING H L, WANG X H, et al. Offloading Dependent Tasks in Multi-access Edge Computing: A Multi-objective Reinforcement Learning Approach [J]. Future Generation Computer Systems, 2022, 128: 333-348.
[16] WANG P, LI K L, XIAO B, et al. Multiobjective Optimization for Joint Task Offloading, Power Assignment, and Resource Allocation in Mobile Edge Computing [J]. IEEE Internet Things Journal, 2022, 9(14): 11737-11748.
[17] LIU L, CHEN H M, XU Z T. SPMOO: A Multi-objective Offloading Algorithm for Dependent Tasks in IoT Cloud-Edge-End Collaboration [J]. Information, 2022, 13(2): 75-1-75-15.
[18] SHAHIDINEJAD A, GHOBAEI-ARANI M. A Metaheuristic-Based Computation Offloading in Edge-Cloud Environment [J]. Journal of Ambient Intelligence and Humanized Computing, 2022, 13(5): 2785-2794.
[19] DEB K, JAIN H. An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part Ⅰ: Solving Problems with Box Constraints [J]. IEEE Transactions on Evolutionary Computation, 2014, 18(4): 577-601.
[20] 张西林, 林兵. 考虑多重不确定性因素的研发任务分配优化 [J]. 计算机系统应用, 2023, 32(7): 219-225. (ZHANG X L, LIN B. Task Allocation Optimization for Product Development Project Considering Multiple Uncertainties [J]. Computer Systems and Applications, 2023, 32(7): 219-225.)
[21] LIU J G, ZHANG Y M, REN J, et al. Auction-Based Dependent Task Offloading for IoT Users in Edge Clouds [J]. IEEE Internet of Things Journal, 2022, 10(6): 4907-4921.
[22] ZHU L L, FENG J H, LIU D, et al. Balanced Cloud Edge Resource Allocation Based on Conflict Conditions [J]. IEEE Access, 2020, 8: 193449-193461.
[23] YU M Y, LIU A F, XIONG N N, et al. An Intelligent Game-Based Offloading Scheme for Maximizing Benefits of IoT-Edge-Cloud Ecosystems [J]. IEEE Internet Things Journal, 2020, 9(8): 5600-5616.
[24] YI J H, DEB S, DONG J Y, et al. An Improved NSGA-Ⅲ Algorithm with Adaptive Mutation Operator for Big Data Optimization Problems [J]. Future Generation Computer Systems, 2018, 88: 571-585.

(责任编辑: 韩 啸)