地理分布式数据中心动态重排任务调度算法

吴延畅[1],敬 超[1,2]

(1. 桂林理工大学 计算机科学与工程学院;2. 广西嵌入式技术与智能系统重点实验室,广西 桂林 541006)

摘 要: 与集中式数据中心相比,地理分布式数据中心通过资源就近部署与负载均衡机制,可显著降低广域网传输延迟并提升系统容灾性。然而,其庞大规模、异构资源以及跨地域电价和网络成本的差异,使得地理分布式数据中心在保证服务质量的同时优化成本,成为一项关键挑战。为此,提出一种截止时间感知且成本驱动的启发式调度算法CADR。CADR的核心机制是通过放缩预留时间和预期成本,动态调整列表调度中的任务优先级,从而在满足截止时间的前提下优先选择低成本资源。实验结果表明,与PRESTO、HEFT、PEFT、MSL等主流算法相比,CADR在运行时间略增的情况下,最多可将平均成本降低47%,显著提升了地理分布式数据中心的成本效益。

关键词: 云计算;地理分布式数据中心;成本最小化;任务重排

DOI: 10. 11907/rjdk. 251289 扫描二维码阅读全文:

中图分类号: TP393 **文献标识码:** A **文章编号:** 1672-7800(2026)007-0031-09

Cost-Aware Dynamic Task Reranking Scheduling Algorithm on Geo-Distributed Cloud Data Centers

WU Yanchang[1], JING Chao[1,2]

(1. *School of Computer Science and Engineering, Guilin University of Technology*;
2. *Guangxi Key Laboratory of Embedded Technology and Intelligent System, Guilin* 541006, *China*)

Abstract: Compared to centralized data centers, Geographically Distributed Data Centers (GDDC) can significantly reduce Wide Area Network (WAN) transmission latency and enhance system fault tolerance through proximity-based resource deployment and load balancing mechanisms. However, their vast scale, heterogeneous resources, and the differences in inter-regional electricity prices and network costs make it a key challenge for GDDC to optimize costs while ensuring quality of service. In response, this paper proposes a deadline-aware and cost-driven heuristic scheduling algorithm, Cost-effective Aware Dynamic Job Ranking (CADR). The core mechanism of CADR is to dynamically adjust the priority of tasks in list scheduling by scaling the reserved time and expected cost, thereby prioritizing the selection of low-cost resources while meeting task deadlines. Experimental results show that compared with mainstream algorithms such as PRESTO, HEFT, PEFT, and MSL, CADR can reduce the average cost by up to 47% with only a slight increase in execution time, significantly improving the cost-effectiveness of Geographically Distributed Data Centers.

Key Words: cloud computing; geo-distributed data centers; cost minimization; task re-ranking

0 引言

随着云计算和大数据技术的飞速发展,地理分布式数据中心已成为支持大规模分布式计算和机器学习的关键基础设施。通过在不同地理位置部署数据中心,能够显著降低延迟并提高数据处理效率。然而,这种分布式架构也增加了任务调度的复杂性。

地理分布式数据中心在应对海量任务时面临多重挑战。一方面,数据中心通过异构网络相互连通,但由于不同链路的容量差异以及多个应用共享资源,导致单个数据中心的上下行链路带宽存在显著不对称,且传输成本各异[1];另一方面,地理分布式数据中心因所处地区的电价、散热成本不同,在计算成本与计算资源方面也存在差

收稿日期: 2025-05-23

基金项目: 国家自然科学基金地区项目(62362018);广西重点研发计划项目(桂科AB23075116)

作者简介: 吴延畅(2000-),男,桂林理工大学计算机科学与工程学院硕士研究生,研究方向为云计算、地理分布式数据中心任务调度和成本优化;敬超(1983-),男,博士,桂林理工大学计算机科学与工程学院教授,研究方向为云计算、大数据处理和调度。本文通讯作者:吴延畅。

异[2]。针对上述挑战,学术界在云计算任务调度领域围绕时间效率与经济成本的平衡开展了广泛研究。

早期工作聚焦于最小化任务的执行时间,例如文献[3]考虑到任务间存在依赖关系,以及计算资源异构的计算效率,提出了一种启发式方法用于计算最小化任务的完成时间。文献[4]进一步揭示了跨数据中心网络带宽异构性对传输延迟的影响,构建了动态带宽分配模型,以降低端到端时延。随着云计算商业化进程的深入,能耗、成本越来越受到重视。文献[5]的研究认为任务执行延迟也会影响响应时间,特别是在热点数据中心,并提出了一种在线启发式方法,通过协同重新分配数据集与任务,以平衡数据中心间的工作负载,进而优化整体响应时间。文献[6]基于粒子群优化(PSO),设计了面向科学工作流的云资源供给与调度策略,以在满足截止时间约束的同时最小化工作流执行成本。文献[7]考虑到数据中心能耗,针对多变的电价提出了一种自适应搜索策略来进行任务的调度。文献[8]针对数据传输瓶颈问题,设计基于任务划分的并行传输机制,在降低传输时间的同时减少跨数据中心流量费用。

本文将综合考虑计算任务的依赖关系、计算资源的异构性以及网络的异构情况,研究在截止时间约束下的成本优化问题。为了解决这一问题,本文采用的方法主要基于元启发式算法和启发式算法。其中,元启发式算法的概念是以特定的方式对解进行迭代采样。例如,现有的元启发式调度算法大多基于遗传算法(GA)[9-11]、粒子群算法[10]和布谷鸟搜索算法[12-13]。文献[10]提出了一种基于多群协同进化的混合智能优化算法,该算法用于多任务调度,旨在最小化总完工时间与成本,并严格满足各任务的截止时间约束。为提高算法的全局搜索能力,该方法引入了模拟退火的Metropolis接受准则,用于更新各粒子群的局部引导解,从而有效避免算法过早陷入局部最优。刘陈伟等[14]基于反向学习的混沌映射自适应粒子群算法(OAPSO)解决云数据中心能耗优化任务调度问题,采用反向学习的方法产生初始种群,在粒子更新方式中引入非线性递减的动态惯性权重策略和混沌映射策略,在最优解位置进行扰动变异产生新解。史爱武等[15]提出一种基于混合策略鲸鱼优化算法的云计算任务调度方法。元启发式虽然具有较强的全局搜索能力,但因无法充分利用环境的信息,执行速度通常较慢。

启发式算法作为一种近似求解方法,由于其高度依赖启发式规则而非迭代搜索过程,因此执行速度通常优于元启发式算法[16]。典型的列表调度启发式算法流程包含3个阶段:任务排序、子截止时间分配及任务部署。其中,任务排序阶段通过优先级排序确定任务执行序列;截止时间分配阶段将全局截止时间分解为各任务的子截止时间;任务部署阶段则将任务分配至合适的计算实例。文献[16]提出的W-LA启发式算法融合了任务优先级划分、截止时间分配、前瞻深度确定以及前瞻实例选择机制。然而,由于需要在每次任务部署时更新前瞻信息,因此其时间复杂度较高。Senapati等[17]设计的启发式算法整合了MMSH策略与子截止时间调节机制,当首次尝试中截止时间大于完工时间时,算法直接返回可行解。然而,当子截止时间以相同速率递减且未考虑成本因素时,此类算法中任务优先级排序的固定性将限制子截止时间分配的决策空间。启发式算法虽然快,但高度依赖先验规则,容易陷入局部最优解,且过去的列表调度算法在排序阶段往往只考虑时间,未考虑成本,导致容易陷入局部最优解。

相较于已有研究,本文提出一种面向地理分布式云数据中心的成本敏感型动态任务优先级调度算法(Cost-effective Aware Dynamic Job Ranking, CADR)。该算法的创新性在于聚焦截止时间约束下,地理分布式环境中的多类型作业调度成本最小化问题。CADR基于经典的列表调度框架,但与传统方法仅将时间作为排序指标不同,CADR创新性地将任务的优先级指标扩展为时间和成本的综合考量,并通过调度结果进行动态调整。这种同时兼顾时间和成本的决策机制,能够实现高效的动态排序,从而在保证截止时间的同时有效地优化成本。具体而言,算法首先将多类型作业分解为任务集合,随后基于任务预估执行时间与传输成本,进行动态优先级重排序,生成任务序列。在确保所有任务满足截止时间约束的前提下,迭代降低成本;反之若超时,则迭代降低时间。实验采用Random[18]、高斯消元[19]、Epigenomics[20]和快速傅里叶变换[21]等真实基准测试集验证算法性能。实验结果表明,在截止时间约束下,CADR在成本优化方面显著优于现有经典算法[3, 17, 21, 22]。

综上,本文的主要贡献如下:

(1)将地理分布式数据中心中的作业调度问题,形式化为一个受截止期限约束的优化问题。通过构建成本、作业与地理分布式数据中心模型,本文综合考量了截止期限、异构网络、任务数据依赖与多类型作业等一系列关键因素,以实现成本最小化为目标。

(2)提出了一种在截止期限约束下有效最小化成本的调度算法CADR。该算法的主要工作流程如下:首先,将作业分解为一组任务进行预处理;其次,依据预估的完成时间和成本对任务进行调整并排序;最终,设计并应用一种考虑松弛时间的成本最小化分配方案,充分利用松弛时间以优化成本。若所有任务均能成功分配且满足截止期限,则对任务进行重新排序以进一步降低成本;反之,若存在违反截止期限的情况,则通过调整参数重新计算任务分配,直至满足截止期限要求。

(3)采用真实世界基准进行仿真实验。在随机生成的4种不同类型的任务中,将本文提出的CADR与其他知名算法(HEFT[22]、MSL[3]、PEFT[21]和PRESTO[17])进行对比。实验结果表明,在最小化成本方面,CADR优于其他知名算

法，但其运行时间有所延长。

1　问题模型

本文首先构建了一个涵盖任务、地理分布式数据中心系统以及成本的综合模型，系统架构如图 1 所示。首先，具有依赖关系的任务被建模为有向无环图（DAG）；其次，调度器会按照以下关键步骤依次执行：任务汇总、开销、时间评估，以及生成调度方案；最后，资源管理器应用该调度方案。在此模型的基础上，本文针对地理分布式数据中心，定义并形式化了一个在截止时间约束下的任务调度成本最小化问题。

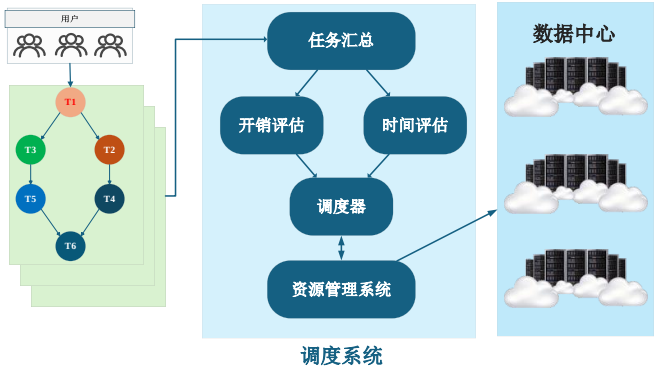


Fig. 1　System architecture

图 1　系统架构图

1.1　任务模型

任务建模为有向无环图（Directed Acyclic Graph，DAG）$\mathcal{G}=<\mathcal{T},\mathcal{E}>$。其中，顶点集合 $\mathcal{T}=\{\tau_i\}$ 表示任务的集合，边集合 $e_{i,j}\in\mathcal{E}$ 表示任务间的数据依赖关系。有向边 $e_{i,j}\in\mathcal{E}$ 表示任务 τ_j 需要在 τ_i 执行结束后才能开始，同时需要以 τ_i 的输出数据作为输入依赖。$e_{i,j}$ 的文件大小为 $d_{i,j}$。任务必须在截止时间 D 内完成。

假定，整个任务有一个开始任务 τ_{start} 和一个结束任务 τ_{end}。为统一处理这些独立的任务流（多 DAG），可以通过引入虚拟的 τ_{start}（作为全局源点）和 τ_{end}（作为全局汇点），将多 DAG 的任务转换为单源点、单汇点的统一 DAG 任务。具体操作如下：首先，添加从 τ_{start} 到每个独立 DAG 的开始任务的边；随后，添加从每个独立 DAG 的汇点到 τ_{end} 的边。整个任务有截止时间 D 的限制，所有任务都要在给定的时间内完成。

1.2　地理分布式数据中心模型

模型有多个数据中心，其地理分布式数据中心由多个物理分布的数据中心组成。每个数据中心内部部署异构的虚拟机（Virtual Machine，VM）资源池，定义全局 VM 集合为 $\mathcal{W}=\{w_1,w_2,\cdots,w_{|\mathcal{W}|}\}$。假设，所有数据中心之间可以通信，且每个数据中心内部的 VM 之间也可以通信。各数据中心之间通过广域网（Wide Area Network，WAN）建立全互联拓扑结构，数据中心内部 VM 间则通过局域网（Local Area Network，LAN）连接，形成层次化通信架构。任意两个 VMw_m 与 VMw_n 之间的有效传输带宽记为 $b_{m,n}$，该参数表征单位时间内可传输的最大数据量。从在 VMw_m 的任务 τ_i 传输到在 VMw_n 的任务 τ_j 的时间记为 $tt_{i,j}^{m,n}$，具体计算如公式（1）所示。

$$tt_{i,j}^{m,n}=\begin{cases}0 & \text{if } m=n,\\ \dfrac{data_{i,j}}{b_{m,n}} & \text{otherwise.}\end{cases}\tag{1}$$

1.3　成本模型

成本模型主要包含两个部分：执行成本和传输成本。由于数据中心所在地的散热效率和用电价格不同，因此数据中心中的 VM 的计算成本也存在差异。记任务 τ_i 在 w_n 的最坏完成时间是 et_i^n，w_n 的执行价格为 pe_n，w_n 传输到 w_m 的传输价格为 $pc^{n,m}$，任务 τ_i 在 VMw_n 上执行的成本记为 ec_i^n，具体计算如公式（2）所示。传输成本为 $\mathrm{cc}_{\mathrm{i,j}}^{\mathrm{m,n}}$，具体计算如公式（3）所示。

$$ec_i^n=et_i^n pe_n\tag{2}$$

$$cc_{i,j}^{m,n}=d_{i,j}pc^{m,n}\tag{3}$$

1.4　问题描述

本文的调度决定为 $y_{j,n,t}\in Y$。若值为 1，表示在 t 时间片任务 τ_j 在 VMw_n 上运行，则 $y_{i,n,t}$ 为 1，否则为 0。为了后面的表述方便，记 τ_i 所在的 VM 为 $aloc[\tau_i]$，其表示如公式（4）所示。记任务 τ_i 开始执行的时间为 $ST[\tau_j]$，其表示如公式（5）所示。

$$aloc[\tau_j]=\arg\max_n y_{j,n,t}\tag{4}$$

$$ST[\tau_j]=\arg\min_{t\in[0,D]}\max_n y_{j,n,t}\tag{5}$$

任务间存在数据依赖，其表示如公式（6）所示。其中，$avail(w_n)$ 表示 w_n 的最早空闲时间，$\mathrm{pred}(\tau_i)$ 表示任务的前驱任务。

$$AST[\tau_i]\geqslant\max\left(avail(w_n),ct_{k,i}^{aloc[\tau_k],aloc[\tau_i]}+\max_{\tau_k\in\mathrm{pred}(\tau_i)}AFT[\tau_k]\right)\tag{6}$$

每个任务仅在 VM 中执行一次，如公式（7）所示。

$$\forall j\in\{1,2,3,\cdots,|\mathcal{T}|\},\ \sum_{n\in\{1,2,3,\cdots,|\mathcal{W}|\}}\sum_t y_{i,n,t}=1\tag{7}$$

截止时间约束如公式（8）所示。

$$AFT[\tau_{end}]<D\tag{8}$$

任务的总开销等于执行成本和传输成本的和，如公式（9）所示。

$$cost(Y)=\sum_{j=1}^{|V|}ec_{j,aloc[\tau_j]}+\sum_{\tau_i\in pred(\tau_j)}cc_{i,j}^{aloc[\tau_i],aloc[\tau_j]}\tag{9}$$

因此，本文的总目标如公式（10）所示。

$$\min_{\mathrm{Y}}\ \mathrm{cost(Y)}$$
$$\mathrm{st.(6)(7)(8)(9)}\tag{10}$$

2　算法设计

CADR 旨在解决地理分布式数据中心环境下的作业调

度优化问题，其核心流程如算法1所示。算法的超参数包括开销—时间的步长因子α，以及最大迭代次数*maxiter*。算法通过多个阶段实现任务到数据中心VM的分配。首先，进行数据的预处理，包括预估剩余时间和开销。随后，将当前预测结果输入到可放缩剩余时间和开销的任务调度算法（LTCA）中，根据调度结果反馈调整预估值。值得注意的是，与传统仅基于时间进行排序的列表调度法不同，CADR能够根据当前的调度结果进行动态调整，从而在调度过程中不断优化排序策略。

算法1 CADR

输入：G，D，W，α，δ，maxiter

输出：intime，cost，makespan

1.用公式（11）、（12）计算EFT、MFC

2.LFT←ERT，LFC←MFC，TLR←1.0，α←0

3.用公式（13）计算rank

4. intime，cost，makespan，minLenSched=LTCAG，LFT，LFC，rank，TLR

5.If intime=True then

6.α=α

7.While itercount≤maxiter do

8.itercount=itercount+1，α=α+α

9.用公式（14-15）计算LRT，LRC的值，然后根据公式（13）计算rank的值

10. intimenew，costnew，makespannew，=LTCAG，LRT，LRC，rank，TLR

11.If intimenew=False or cost<costnew then

12.If 4×α>α then

13.Break

14.Else

15.α=α-α，α=α，α=2α

16.Else if

17.Else

18.α=2α

19. intime，cost，makespan←intimenew，costnew，makespannew

20.End if

21.End while

22.Else

23. While intime = False and minLenSched=False and itercount<maxiter

24.根据公式（16）计算α的值，TLR=1-α

25.intime，cost，makespan，minLenSched=LTCAG，LRT，LRC，rank，TLR

26.itercount = itercount+1

27.End while

28.End if

2.1 数据预处理

对数据进行预处理，计算*ERT*（Minimum Remain Finishing Time）和*MRC*（Minimum Remain Finishing Cost）。*ERT*和*MRC*分别表示在不考虑VM占用的情况下，最少的剩余时间和最少的剩余开销。计算如公式（11）—（12）所示。

$$ERT\left[\tau_j, w_n\right] = et_{j,n} + \max_{\tau_k \in succ(\tau_j)} \min_{w_r \in Q} \left(ERT\left[\tau_k, w_r\right] + ct_{j,k}^{n,r} \right) \tag{11}$$

$$MRC\left[\tau_j, w_n\right] = et_{j,n} \cdot p_e(n) + \min_{\tau_k \in succ(\tau_j)} \min_{w_r \in Q} \left(MRC\left[\tau_k, w_r\right] + c_{j,k}^{n,r} \cdot p_c(n,r) \right) \tag{12}$$

其中，$ct_{j,k}^{n,r}$表示VMw_n任务τ_j传输到VMw_n的任务τ_k，$p_c(n,r))$表示VMw_n传输到VMw_r的价格。

2.2 重排任务调度算法

任务τ_j的优先级$rank\left[t_j\right]$由其在所有VM上的$LRT\left[t_j, w_n\right]$平均值确定。*LRT*是一个动态指标，初始化为*ERT*，随后会根据λ进行动态更新。算法在每次迭代中根据*rank*降序选择任务进行调度。$rank\left[\tau_j\right]$如公式（13）所示。若首次LTCA结果满足截止时间要求，则进入成本改进阶段；否则，进入时间改进阶段。

$$rank\left[\tau_j\right] = \frac{1}{|\mathcal{W}|} \sum\nolimits_{p_i \in Q} LRT\left[\tau_j, p_i\right] \tag{13}$$

在成本改进阶段，*LRT*和*LRC*分别通过公式（14）—（15）计算得出。*LRT*和*LRC*会随着迭代参数$\bar{\alpha}$的递增而同步放大。在每次迭代中，调度器利用*LRT*和*LRC*的值进行任务的排序和资源选择，以最小化成本。$\bar{\alpha}$的增长量记为$\hat{\alpha}$。为加速算法收敛，$\hat{\alpha}$采用指数递增，并在迭代结果变差时进行调整。

$$LRT\left[\tau_j, w_n\right] = ERT\left[\tau_j, w_n\right] + \bar{\alpha} \frac{ERT\left[\tau_j, w_n\right] \times MRC\left[\tau_j, w_n\right]}{scale_{EFT}\left[\tau_j\right]} \tag{14}$$

$$scale_{ERT}\left[\tau_j\right] = \frac{1}{|\mathcal{W}|} \sum_{p_i \in Q} EFT\left[\tau_j, p_i\right]$$

$$LRC\left[\tau_j, w_n\right] = MFC\left[\tau_j, w_n\right] + \bar{\alpha} \frac{ERT\left[\tau_j, w_n\right] \times MRC\left[\tau_j, w_n\right]}{scale_{MRC}\left[\tau_j\right]} \tag{15}$$

$$scale_{MRC}\left[\tau_j\right] = \frac{1}{|\mathcal{W}|} \sum_{p_i \in Q} MRC\left[\tau_j, p_i\right]$$

算法会努力寻找一个解决方案以缩短完成时间（makespan）。在时间改进阶段，迭代参数$\bar{\alpha}$的更新方式应反映当前makespan超出截止时间D的程度，因此，$\bar{\alpha}$根据公式（16）进行递增。其中，⌈ ⌉表示向上取整函数，δ被设置为固定值0.001。在每次迭代中，时间松弛率（TLR）随$\bar{\alpha}$的增加而减少。

$$\bar{\alpha} \leftarrow \bar{\alpha} + \alpha \log_2 \left\lceil \frac{(makespan - D + \delta)}{D} \right\rceil \tag{16}$$

2.3　基于放缩剩余时间和成本的任务调度算法（LTCA）

通过调用LTCA算法计算当前参数下的调度结果，其输出的*LRT*和*LRC*分别用于估算完成时间和成本。算法LACT是基于松弛时间和成本的调度算法，其流程如算法2所示，旨在根据给定参数*LRT*、*LRC*和*TRR*生成尽可能合理的分配方案。

算法 2　LTCA

输入：G，D，W，rank，LRT，LRC，TLR

输出：intime，cost，makespan，minLenSched

1.minLenSched←True

2.FOR τ_j in降序 rank DO：

3. FOR wn∈W DO：

4. 根据公式（17）计算 EAST$[\tau_j, w_n]$

5. 根据公式（18）计算 LT$[\tau_j, w_n]$

6.End For

7. Ls=$\{w_n|LT[\tau_j, w_n]\geqslant 0\}$

8. 根据公式（19）计算 aloc$[\tau_j]$

9.If Ls≠∅ Then

10.minLenSched=Fasle

11.End if

12.Update AST，AFT

13.End For

首先，计算*EAST*，即当前调度中将任务τ_i分配到VMw_n上执行的开始时间，具体如公式（17）所示。

$$EAST\left(\tau_j, w_n\right)=max\left(avail\left[w_n\right], \max_{\tau_i\in pred\left(\tau_j\right)}\left(AFT\left[\tau_i\right]+ct_{\tau_i,\tau_j}^{aloc[\tau_i],w_n}\right)\right) \tag{17}$$

其次，计算*LT*（Left Time），具体如公式（18）所示。

$$LT\left[\tau_j, w_n\right]=D-\left(EAST\left[\tau_j, w_n\right]+LFT\left[\tau_j, w_n\right]\right)/\mathrm{TLR} \tag{18}$$

然后，根据*LT*计算*aloc*。若未超时，则从L_s中选择*LRC*代价最小的方案；反之，在所有VM $\mathcal{W}$中选择*LT*最大的方案，以尽可能减少任务的结束时间。具体如公式（19）所示。

$$aloc\left[\tau_j\right]=\begin{cases}\arg\min\limits_{w_n\in L_s}\{LRC[\tau_j, w_n]\} & \text{if } Ls\neq\phi,\\ \arg\max\limits_{w_n\in\mathcal{W}}\{LT[\tau_j, w_n]\} & \text{if } Ls=\phi.\end{cases} \tag{19}$$

最后，根据算法第12行更新*AST*和*AFT*，二者分别表示任务的实际开始时间与实际结束时间。具体如公式（20）—（21）所示。

$$AST[\tau_j]=EAST[\tau_j, \mathrm{alloc}[\tau_j]] \tag{20}$$

$$FT[\tau_j]=ST[\tau_j]+et_j^{aloc[\tau_j]} \tag{21}$$

2.4　复杂度分析

本文提出的CADR主要由3个部分构成：初始化阶段、成本改进阶段以及完工时间改进阶段。总的算法复杂度为$O\left(maxiter|\mathcal{W}|\left(|\mathcal{T}|+|\mathcal{E}|\right)\right)$。

在初始化阶段，初始化*EFT*，*MFC*的复杂度为$O\left(|\mathcal{W}|\left(|\mathcal{T}|+|\mathcal{E}|\right)\right)$。

在成本改进阶段和完工时间改进阶段，均需要执行多次LTCA算法，次数不超过*maxiter*次。在每次迭代中，计算*LRT*、*LRC*的算法复杂度为$O(|\mathcal{W}||\mathcal{T}|)$。计算任务优先级*rank*的算法复杂度为$O(|\mathcal{W}|(|\mathcal{T}|+|\mathcal{E}|))$，对*rank*进行排序的算法复杂度为$O\left(|\mathcal{T}|\log|\mathcal{T}|\right)$。随后执行LTCA算法，计算*EAST*、*LT*、*Ls*的算法复杂度都是$O(|\mathcal{W}|(|\mathcal{T}|+|\mathcal{E}|))$。因此，CADR的算法单次迭代的总算法复杂度为$O(|\mathcal{W}|(|\mathcal{T}|+|\mathcal{E}|))$。考虑到*maxiter*是常数，因此，CADR的总算法复杂度为$O\left(maxiter|\mathcal{W}|\left(|\mathcal{T}|+|\mathcal{E}|\right)\right)$，可以化简为$O(|\mathcal{W}|(|\mathcal{T}|+|\mathcal{E}|))$。

3　实验

本文将通过一系列基于仿真的实验来评估所提出的CADR算法的性能。

3.1　实验设置

本文实验中的具体任务通过以下4种方式生成：

（1）随机（Random）[18]：采用文献[18]随机生成工作流，给定任务数量以及边的概率来生成图。其中，边的生成概率 p = 0.1，通过作业数量$|\mathcal{T}|\in\{40, 70, 100, 130\}$来生成作业。

（2）高斯消元法（Gaussian elimination）[19]：基于解线性方程组的算法流程，其工作流中任务点和边的数量由矩阵大小m决定。在本次实验中，m设定为{5, 8, 11, 14}。基于m值，图中的顶点数和边数分别为$|\mathcal{T}|\in\{54, 119, 209, 324\}$和$|\mathcal{E}|\in 89, 209, 379, 599$。

（3）Epigenomics[20]：一种数据处理操作流程，其作业图规模取决于并行分支数量（pb）。在本次实验中，pb设定为{30, 60, 90, 120}。基于pb值，图中的顶点数和边数分别为$|\mathcal{T}|\in\{124, 244, 364, 484\}$和$|\mathcal{E}|\in\{152, 302, 452, 602\}$。

（4）FFT（Fast Fourier Transform）[21]：FFT算法的工作图规模取决于输入向量大小m。在本次实验中，本文将m设定为{3, 4, 5, 6}，由此产生的工作图规模分别为$|\mathcal{T}|\in\{39, 95, 223, 511\}$、$|\mathcal{E}|\in\{62, 158, 382, 894\}$的结果。

首先，关于资源规模，VM总数量$|\mathcal{W}|$被设定在集合{4, 8, 16, 32}内取值，以此模拟不同资源密度的场景。其次，为体现任务执行时间的不确定性和处理器性能的差异，任务的平均最坏情况执行时间$\overline{et}$取自集合{100, 200, 300}。单个任务的实际最坏情况执行时间从具有均值$\mu=\overline{et}$和标准差$\sigma=\beta\overline{et}$的正态分布中实例化，其中，时间变异系数β设定为{0.2, 0.5, 0.8}，用以量化执行时间的范围。

任务间的数据传输量与通信计算比（CCR）相关。CCR被定义为任务中平均通信成本与平均计算成本的比

率，其取值集合为{0.1, 0.5, 1, 2}，旨在模拟从计算密集型（低 *CCR*）到通信密集型（高 *CCR*）的不同工作负载。任务的平均数据大小 $\overline{data}$ 依据 *CCR*、平均执行时间 $\overline{et}$ 和平均带宽 $\overline{B}$，通过公式（22）计算得出。

$$\overline{data} = CCR \cdot \overline{et} \cdot \overline{B} \tag{22}$$

生成数据大小 *data* 则从均值 $\mu = \overline{data}$、标准差 $\sigma = c\,\overline{data}$ 的正态分布中随机生成，其中，*c* 为带宽变异系数。此外，为确保物理意义上的合理性，所有低于 1 的数据传输量抽样结果均被设定为 1。

网络异构性通过带宽配置和价格模型体现。区域间的带宽从 $\left[(1-c)\overline{B}, (1+c)\overline{B}\right]$ 的均匀分布中随机生成，其中，平均带宽 $\overline{B}$ 固定为 100 Mbps，带宽变异系数 c 被实例化为 {0.2, 0.5, 0.8}。在价格模型方面，执行价格 *pe* 和通信价格 *pc* 用于模拟跨地域价格差异。执行价格 *pe* 从范围 [0.1, 1] 的均匀分布中随机生成，而通信价格 *pc* 则从范围 [0.1, 0.5] 的均匀分布中随机生成，确保了模拟环境中的成本要素具有足够的动态性和随机性，符合分布式数据中心的实际运行特征。

为了定量评估截止时间约束对调度结果的影响，本文引入截止期限扩展率 ∂。该比率定义为作业图 $\mathcal{G}$ 的规定截止期限 *D*，相对于某基准算法生成的总完成时间 $makespan_{alg}$ 的比率，如公式（23）所示。实验中，∂ 设定为 {1, 1.2, 1.4, 1.6}。$makespan_{alg}$ 通常选用 HEFT 算法的完工时间作为基准。但在评估截止时间限影响的特定实验中，$makespan_{alg}$ 选择了需要对比的算法 HEFT、PEFT 和 MSL 所能生成的最小完工时间作为基准。

$$D = \partial makespan_{alg} \tag{23}$$

本文实验基于以下 4 个方面进行：

（1）性能比较。将 CADR 算法与具有截止期限约束的成本最小化算法进行对比分析，选取 PRESTO 作为对比算法，以评估 CADR 在成本最小化方面的性能表现。

（2）评估截止期限约束的影响。为深入探究截止期限约束对调度性能的影响，本文将 CADR 算法与领域内最先进的 HEFT、PEFT 及 MSL 算法进行对比，并以这 3 种算法所能生成的最短任务完成时间作为基准。在此基础上，设置参数 $\partial = 1$，即将对比算法生成的最短完成时间设定为截止时间，进而由 CADR 算法解决在该截止时间约束下的调度问题。

（3）评估 CCR 的影响。鉴于 CCR 在实际作业中的重要性，本文对不同 CCR 值下的算法性能进行了统计分析。分析表明，CCR 值越大，数据传输所需时间越长。因此，通过观察不同 CCR 条件下的算法表现，可以更好地理解 CCR 对调度性能的影响。

（4）参数设置的影响评估。CADR 算法包含一个关键超参数 α，该参数对算法的性能具有显著影响。因此，本文对这些参数的影响进行了详细评估，以确定最优参数配置，从而提升算法的整体性能。

本文的实验框架使用 Python 3.12 编写，并在配备 64 位英特尔酷睿 i9-11900K 处理器与 125 GB 内存的计算机上运行。

3.2 对比算法

为进行评估，本文将 CADR 与下列先进算法进行比较：

（1）PRESTO[17]：一种静态列表调度算法，能够在考虑截止日期的情况下最小化成本。它引入了乐观完成时间（Optimistic Finish Time）来预测作业的完成时间，并提出了不考虑惩罚的最小化完成时间算法 MMSH。

（2）HEFT[22]：目前最著名的列表调度方案之一。它是一种用于最小化作业完成时间的启发式算法。算法首先找到关键路径，随后将关键路径上的任务分配到高优先级，以便进行任务分配。

（3）PEFT[21]：一种用于最小化时间成本的启发式算法。它通过估计的最低成本表预测剩余时间，并根据预测的完成时间对任务进行分配。

（4）MSL[3]：一种生成具有最小完工时间的作业图的启发式算法。该调度方案的设计会将满足前置约束条件的任务，独立于其级别进行分组。通过消除逐层约束，可以实现独立任务有效作业时间的最小化，这可能会促使 MSL（最小作业时间）调度策略生成更短跨度的作业计划。

3.3 评价指标

在本文实验中，采用了 3 种常用的性能指标进行对比，其具体描述如下：

（1）超时率（ToR）：用于衡量算法在所有无法满足时间截止期限的计划情况下所占的比例。数值越低，表明该算法更能生成满足时间截止期限的解决方案。

（2）平均开销（Cost）：用于衡量算法生成满足截止期限的计划的平均开销。数值越低，表明该算法所生成的解决方案开销越低。此指标仅在满足截止期限的情况下有效。为此，在统计分析过程中，仅在被比较算法中满足截止期限的案例中考虑计数，以确保统计数据的条件一致且都满足截止期限。

（3）运行时间（Running time）：用于评估算法的平均运行时间。该指标值越低，表明算法的效率越高。

通过综合评估上述指标，可以全面掌握所提算法在截止期限满足度、成本控制、执行效率及资源管理等方面的表现，从而为算法的进一步优化和实际应用提供有力依据。

4 实验结果

4.1 截止时间的影响

表 1 通过对比 CARD 与其他最小化完成时间的算法，验证了其在严格截止时间下实现成本最小化的有效性。

由表 1 可知，实验中的截止时间是根据对比算法计算

Table 1　The comparison of CADR with HEFT, MSL and PEFT when $\partial = 1$

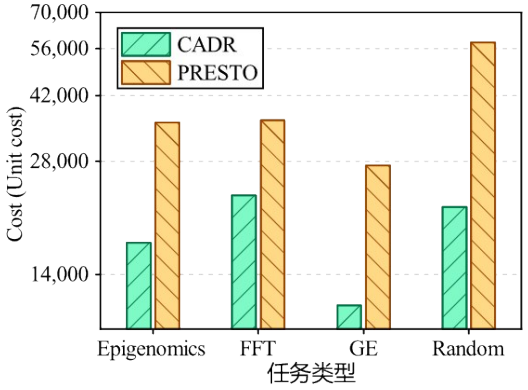
表 1　$\partial = 1$对比 CADR 与 HEFT，MSL PEFT

Algorithm	Cost(unit cost)	ToR/%	Running time/ms
CADR	19 859.87	0	346.05
HEFT	37 436.36	0	15.31
CADR	21 645.03	0	358.13
MSL	27 316.00	0	10.10
CADR	22 023.49	1.16	372.98
PEFT	27 738.94	0	106.95

的最短完成时间设定的，因此，所有对比算法的ToR均为0。CADR在与HEFT和MSL的对比中，ToR均为0，表明CADR在所有测试样例中都成功通过了测试。而在与PEFT的对比中，有1.16%的测试样例中，PEFT能够提供完成时间更低的解决方案，而CADR则无法匹配这种性能。在开销方面，CADR显著优于对比的最小化运行时间算法，尤其是相较于HEFT算法，CADR最多能实现47%的成本降低。CADR能够通过利用松弛时间来重新调度任务，从而最小化成本。而HEFT、MSL和PEFT等算法则专注于最小化任务的执行时间，未能考虑成本问题。尽管PEFT在某些情况下（约1.16%）能够实现更短的任务完成时间，但CADR在成本方面仍然保持优势。在运行时间上，CADR的运行时间比对比算法长，这是因为CADR不仅需要考虑如何最小化成本，还要在任务的截止时间约束下进行多次调度，以确保找到成本最优的解决方案。而HEFT、MSL和PEFT等对比算法只需要进行一次调度就能得出最终方案。

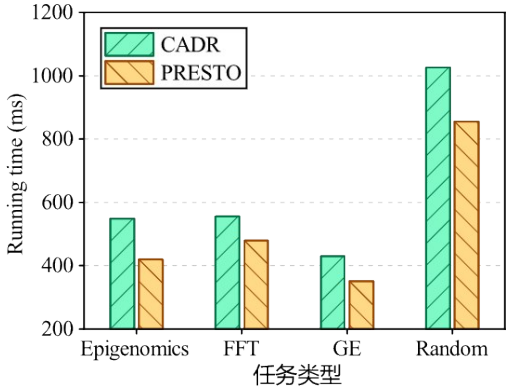
4.2　不同任务类型

本文对比了所提出的CADR算法与PRESTO算法在不同作业类型下的成本开销、ToR及运行时间，结果如图2所示。



(a)Cost of different types of workloads

(a)不同的任务类型的开销



(b) Running time of different types of workloads

(b)不同的任务类型的执行时间

Fig. 2　Results of CADR and PRESTO under different types of workloads

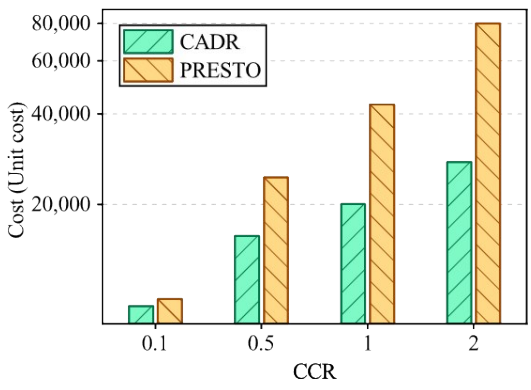
图 2　CADR与PRESTO在不同类型的工作负载的实验结果

由于CADR和PRESTO的ToR均为0，从图2(a)可以看出，CADR在成本上更具优势，表现为代价最小。这是因为CADR能够通过任务重排序来优化成本，而PRESTO由于采用固定任务顺序，导致其性能相对劣势。在图2(b)中，CADR的平均运行时间略高于PRESTO。这是由于CADR采用了更灵活的任务调度顺序，而PRESTO遵循固定顺序，导致在运行效率上存在差异。

4.3　CCR的影响

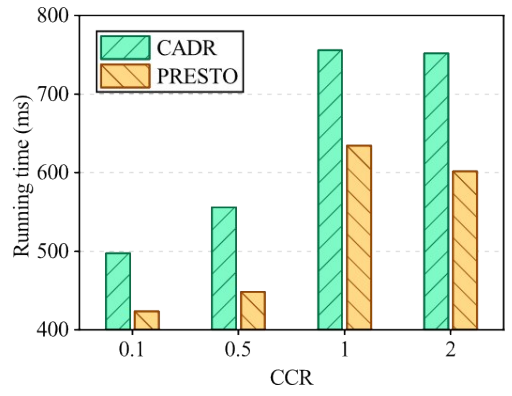
为验证CADR算法的有效性，本文选取了不同CCR值来模拟传输密集型与计算密集型任务，并对比分析了其与PRESTO算法的性能，结果如图3所示。

从图3(a)可以看出，随着CCR值的增加，本文提出



(a)Cost of different CRRs

(a)不同CCR的开销



(b)Running time of different CRRs

(b)不同CCR的执行时间

Fig. 3　Result of CADR and PRESTO under different CCRs

图 3　不同CCR下CADR与PRESTO的结果

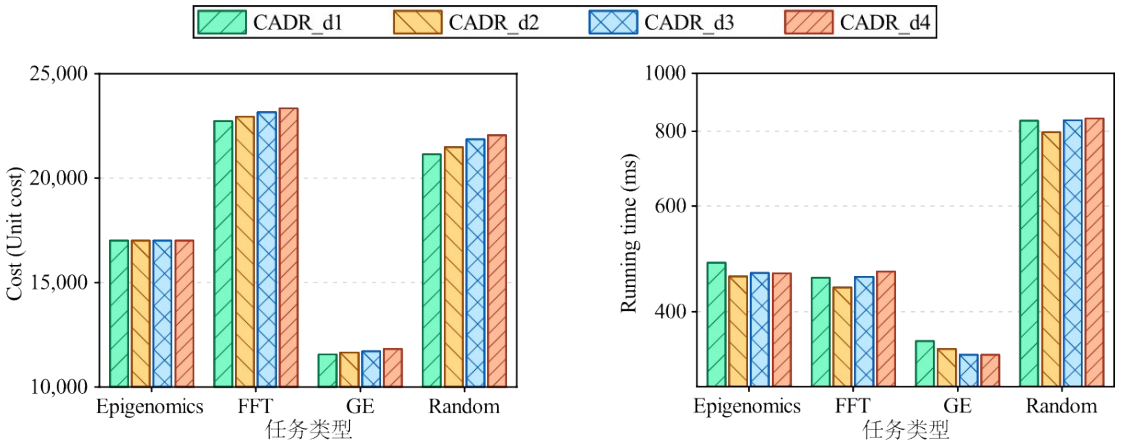
的算法 CADR 相比 PRESTO 优势更加明显。由于 CCR 的增加，传输数据对 rank 计算的影响变得越来越显著。PRESTO 采用固定 rank 顺序，而 CADR 则采用随迭代过程变化的 rank 顺序。因此，CADR 能够找到具有更低开销的方案。

图 3（b）展示了算法的平均调度时间对比情况。在所有 CCR 测试场景下，CADR 的平均调度时间均高于 PRESTO。这种时间差异源于两种算法内在机制的不同：PRESTO 采用的是轻量级的、基于静态优先级的调度策略，计算过程相对简单直接；而 CADR 为了实现成本最优解，必须执行额外的迭代重排和动态优先级计算，因此引入了更高的计算开销。

尽管随着 CCR 值的提高，通信开销有所增加，但 CADR 相较于 PRESTO 的额外时间开销并未出现爆炸性增长，始终维持在一个相对稳定且有限的范围内。这表明 CADR 算法的迭代和重排机制具有良好的计算效率，其增加的复杂度是可控的。因此，CADR 在调度时间上的少量增加，是为其卓越的成本优化能力所付出的合理代价，这成功地证明了 CADR 算法在调度时间开销和执行成本之间达成了一种高效的平衡。

4.4 CADR 参数的影响

如算法 1 所定义，α 是平衡算法收敛速度与结果质量的关键参数。当 α 增大时，尽管 CADR 能够最小化运行时间，但其结果也更容易陷入局部最优解。因此，为了合理地选取 α 的值，本文进一步将算法 CADR_d1、CADR_d2、CADR_d3、CADR_d4 的参数 α 设定为 {0.02, 0.04, 0.04, 0.08}，不同 α 下 CADR 的结果，如图 4 所示。



（a）Cost of different workflow types
（a）不同的任务类型的开销

（b）Running time of different workflow types
（b）不同的任务类型的执行时间

Fig. 4 Result of CADR under different α

图 4 不同 α 下 CADR 的结果

由图 4（a）可以看出，在作业类型 GE、FFT 和 Random 三种作业类型中，平均成本随着参数 α 的增大而呈现出增加的趋势。这与算法设计预期相符，α 增大意味着迭代步长增加，搜索精度降低，导致算法可能因步长过大而未能充分探索解空间，从而更容易陷入局部最优解。然而，参数 α 大小在 Epigenomics 上影响不明显，成本对 α 变化的敏感度较低，这与其特定的任务结构特性有关。

由图 4（b）可以看出，在不同作业类型上，CADR 的运行时间与 α 的关系和任务结构相关。对于 GE 任务，运行时间随 α 增加而减少；而在其他任务类型下，由于执行时间受任务依赖关系结构的影响更大，CADR 的运行时间随 α 呈现不规则的增减。

5 结语

为降低成本，本文设计了一种基于重新排序概念的 CADR（基于地理分布的数据中心任务分配器），能够将任务分配到合适的数据中心，同时严格遵守任务截止时间约束。具体而言，本文首先构建了一个系统模型，涵盖作业模型、成本模型以及地理分布式数据中心的各项要素，并严格定义了通过地理分布式数据中心实现成本最小化的调度问题。然后，在作业提交和预处理任务的过程中，利用 CADR 进行任务完成时间和成本的估算，并结合基于任务的最小化成本分配器 LTCA 对任务进行调度。若结果未超出截止期限，则对任务进行重新排序以最小化成本；否则，则增加参数 α 并重新调整 LTC-A，以减少完工时间，使其符合截止期限。最后，针对随机、高斯消元、表观基因组学以及快速傅里叶变换等不同来源生成多种类型的作业，并进行实验。实验结果表明，在截止期限约束条件下，与著名的算法 PRESTO、HEFT、PEFT 和 MSL 相比，CADR 的成本最低。同时，本文还进一步展示了 CADR 在不同的截止期限、CCR 和参数设置方面的有效性。本文提出的调度算法主要基于集中式的思想，对于处理中小规模的分布式数据中心任务调度取得了较好的效果。下一步的工作将进一步优化算法的分布式处理能力，尝试针对大规模任务调度场景设计低复杂度高并行性的调度策略，同时，考

虑任务需求的多样性建立多类型的任务模型。

参考文献:

[1] ZHOU A C,SHEN B,XIAO Y, et al. Cost-aware partitioning for efficient large graph processing in geo-distributed datacenters [J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 31(7): 1707-1723.

[2] YAO F, TAO Q, YU W, et al. Ragraph: a region-aware framework for geo-distributed graph processing [J]. Proceedings of the VLDB Endowment, 2023, 17(3): 264-277.

[3] SIRISHA D, KUMARI G V. A new heuristic for minimizing schedule length in heterogeneous computing systems[C]//2015 IEEE International Conference on Electrical, Computer and Communication Technologies, 2015:1-7.

[4] GUEVARA J C, FONSECA N L. Task scheduling in cloud-fog computing systems [J]. Peer-to-Peer Networking and Applications, 2021, 14(2): 962-977.

[5] JIN Y, GAO Y, QIAN Z, et al. Workload-aware scheduling across geo-distributed data centers[C]//2016 IEEE Trustcom/BigDataSE/ISPA, 2016: 1455-1462.

[6] RODRIGUEZ M A, BUYYA R. Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds [J]. IEEE Transactions on Cloud Computing, 2014, 2(2): 222-235.

[7] LI X, YU W, RUIZ R, et al. Energy-aware cloud workflow applications scheduling with geo-distributed data [J]. IEEE Transactions on Services Computing, 2020, 15(2): 891-903.

[8] WANG Q, GAO B, ZHOU Z, et al. DAG-aware optimization for geo-distributed data analytics[C]//Proceedings of the 52nd International Conference on Parallel Processing, 2023:472-481.

[9] LIU L, ZHANG M, BUYYA R, et al. Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing [J]. Concurrency and Computation: Practice and Experience, 2017, 29(5): e3942.

[10] LI H, WANG D, ZHOU M, et al. Multi-swarm co-evolution based hybrid intelligent optimization for bi-objective multi-workflow scheduling in the cloud [J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 33(9): 2183-2197.

[11] XIE Y, GUI F X, WANG W J, et al. A two-stage multi-population genetic algorithm with heuristics for workflow scheduling in heterogeneous distributed computing environments [J]. IEEE Transactions on Cloud Computing, 2021, 11(2): 1446-1460.

[12] ASGHARI A Y, HOSSEINI S M, RAHMANI A M. A hybrid bi-objective scheduling algorithm for execution of scientific workflows on cloud platforms with execution time and reliability approach [J]. The Journal of Supercomputing, 2023, 79(2): 1451-1503.

[13] SA′AD S, MUHAMMED A, ABDULLAHI M, et al. An optimised cuckoo-based discrete symbiotic organisms search strategy for tasks scheduling in cloud computing environment [DB/OL]. https://arxiv. org/abs/2311. 15358.

[14] LIU C W, SUN J, LEI B B, et al. Task scheduling strategy for energy consumption optimization of cloud data center based on improved particle swarm algorithm[J]. Computer Science, 2023, 50(7):246-253.
刘陈伟,孙鉴,雷冰冰, 等. 基于改进粒子群算法的云数据中心能耗优化任务调度策略[J]. 计算机科学,2023,50(7):246-253.

[15] SHI A W, HUANG H, LUO G. Research on cloud computing task scheduling based on mixed strategy whale optimization algorithm[J]. Software Guide, 2024, 23(10):104-111.
史爱武,黄河,罗干. 基于混合策略鲸鱼优化算法的云计算任务调度研究[J]. 软件导刊,2024,23(10):104-111.

[16] YANG L, YE L, XIA Y, et al. Look-ahead workflow scheduling with width changing trend in clouds [J]. Future Generation Computer Systems, 2023, 139: 139-150.

[17] SENAPATI D, SARKAR A, KARFA C. PRESTO: a penalty-aware real-time scheduler for task graphs on heterogeneous platforms [J]. IEEE Transactions on Computers, 2022, 71: 421-435.

[18] CORDEIRO D, MOUNIÉ G, PERARNAU S, et al. Random graph generation for scheduling simulations[C]//Malaga: International ICST Conference on Simulation Tools and Techniques, 2010.

[19] TOPCUOGLU H R, HARIRI S, WU M. Performance-effective and low-complexity task scheduling for heterogeneous computing [J]. IEEE Trans Parallel Distributed Syst, 2002, 13: 260-274.

[20] JUVE G, CHERVENAK A, DEELMAN E, et al. Characterizing and profiling scientific workflows [J]. Future Generation Computer Systems, 2013, 29(3): 682-692.

[21] ARABNEJAD H, BARBOSA J G. List scheduling algorithm for heterogeneous systems by an optimistic cost table [J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25: 682-694.

[22] TOPCUOGLU H, HARIRI S, WU M Y. Performance-effective and low-complexity task scheduling for heterogeneous computing [J]. IEEE Transactions on Parallel and Distributed Systems, 2002, 13 (3): 260-274.

(责任编辑:陈维捷)