华东理工大学学报(自然科学版)
Journal of East China University of Science and Technology
ISSN 1006-3080,CN 31-1691/TQ

《华东理工大学学报(自然科学版)》网络首发论文

题目：截止日期约束的云中资源调度成本优化算法
作者：韩伊琳，范贵生，虞慧群
DOI：10.14135/j.cnki.1006-3080.20230419001
收稿日期：2023-04-19
网络首发日期：2023-09-21
引用格式：韩伊琳，范贵生，虞慧群．截止日期约束的云中资源调度成本优化算法[J/OL]．华东理工大学学报(自然科学版).
https://doi.org/10.14135/j.cnki.1006-3080.20230419001

DOI: 10.14135/j.cnki.1006-3080.20230419001

截止日期约束的云中资源调度成本优化算法

韩伊琳， 范贵生， 虞慧群
（华东理工大学信息科学与工程学院，上海 200237）

摘要：随着越来越多的工作流应用程序部署在云端，如何在满足工作流截止期限约束的前提下优化资源调度成本成为一个热门研究领域。本文提出了一种截止日期约束的成本优化(CODC)算法。首先，合并工作流任务以减少不同实例之间的数据传输开销。其次，关注父任务和子任务对当前任务优先级的影响，并考虑任务子截止日期未被满足的情况，以选择最早完成任务执行的实例。最后，在五种工作流上与现有算法进行对比，实验结果表明，CODC算法与三种对照算法相比具有更低的工作流执行成本。

关键词：云计算；资源调度；截止日期约束；任务实例映射；成本优化

中图分类号：TP311 **文献标志码：**A

云计算(Cloud Computing)是将物理设备抽象为虚拟资源，以互联网为媒介向用户提供的一种新型分布式计算模式[1]。受益于服务器不断提升的处理能力以及高速网络的低延迟，用户能够随时随地通过服务接口动态地获得可扩展的便捷资源并根据实际使用量进行付费，而不需要大量的前期投资。成本优化是云资源调度的重要考虑因素之一，它试图在满足服务质量(Quality of Service, QoS)的同时获得最小的调度成本，然而任务之间的依赖约束使得可调度条件变得更加复杂[2]，QoS的要求也间接地提高了在任务响应时间和执行成本之间寻找平衡的难度。因此，设计一种有效的资源调度方法来均衡任务执行的时间和成本变得日益重要。

针对此问题，Chen等[3]提出了一种实时任务调度算法，旨在减少工作流响应时间、降低执行成本并提高服务实例的资源利用率。该算法基于任务的最早开始时间为任务分配调度优先级，并根据任务的最晚完成时间将任务分配到合适的租赁实例上。Toussi等[4]提出了一种分而治之的调度算法，将工作流分为多个子工作流，针对每个子工作流的关键路径上的任务选择满足截止日期和最低执行成本的服务资源，并将关键路径上的任务从工作流中删除。王旖旎[5]提出了一种主动响应式工作流调度算法，在新工作流到达时将当前所有工作流的就绪任务进行混合调度以满足期限约束，并优先选择具有最小调度成本的计算资源。Ahmad等[6]提出了一种截止日期感知的启发式算法，通过工作流最短执行时间确定其截止日期，并实时监控资源池动态为工作流任务分配执行成本最低的虚拟实例。张艮山等[7]提出了一种期限预算双重约束的工作流调度算法，根据工作流结构特征对其进行分层处理，将任务划分为若干个互无关联的任务包，以提高调度过程中的并行程度。

尽管上述方法考虑了工作流任务之间的数据依赖关系，但这些方法未进行任务合并以减少数据传输成本，并忽略了父任务对于当前任务优先级的影响。此外，它们很少关注任务子截止日期未被满足的情况。为了克服上述不足，本文提出了一种截止日期约束的成本优化算法(CODC)，包括四个调度阶段：任务合并，任务优先级分配，任务子截止日期分配和租赁实例选择，主要优势如下：

(1)明确云工作流任务合并条件，将满足条件的

收稿日期：2023-04-19
作者简介：韩伊琳(1998—)，女，河南开封人，硕士生，主要研究方向为云计算、软件工程。E-mail: 757333748@qq.com
通信联系人：范贵生，E-mail: gsfan@ecust.edu.cn

任务进行合并,以减少不同实例之间的数据传输开销;

(2)关注父任务以及子任务对于当前任务优先级的影响,给出优化任务优先级分配过程;

(3)设计每个任务的子截止日期,将工作流中的任务分离,以便在子截止日期的约束下分别调度任务,并选择最早完成当前任务执行的实例。

1 问题描述与形式化

1.1 时间模型

当任务 t_i 分配给虚拟机 (Virtual Machine, VM) 类型为 v_j 的实例时,执行时间 $\mathrm{ET}(t_i,v_j)$ 主要是由任务 t_i 的工作负载 $W(t_i)$ 和虚拟机 v_j 的处理能力 $C(v_j)$ 决定的,因此 t_i 在 v_j 上执行时间可以计算为,

$$\mathrm{ET}(t_i,v_j)=W(t_i)/C(v_j) \tag{1}$$

任务 t_i 的最小执行时间 $\mathrm{MET}(t_i)$ 可以通过将 t_i 分配到最快的实例上来计算,用公式表示为,

$$\mathrm{MET}(t_i)=\min_{v_j\in VM}\{\mathrm{ET}(t_i,v_j)\} \tag{2}$$

当一个任务完成执行时,它需要将数据传输给它的后继任务。任务 t_i 与其后继任务 t_s 之间的数据传输时间 $\mathrm{DTT}(t_i,t_s)$ 需要分为两类情况进行讨论:如果 t_i 与 t_s 被分配给不同的实例,则 $\mathrm{DTT}(t_i,t_s)$ 由传输数据量 $d_{i,s}$ 和网络带宽 bw 决定;否则 $\mathrm{DTT}(t_i,t_s)$ 为零。设 $v(t_i)$ 为分配给任务 t_i 的虚拟机实例,则 $\mathrm{DTT}(t_i,t_s)$ 可以计算为,

$$\mathrm{DTT}(t_i,t_s)=\begin{cases} d_{i,s}/bw, & \text{if } v(t_i)\neq v(t_s)\\ 0, & else\end{cases} \tag{3}$$

$\mathrm{SET}(t_i,v_j)$ 定义为 t_i 的开始执行时间,$\mathrm{FET}(t_i,v_j)$ 定义为 t_i 的完成执行时间。设 $\mathrm{ERT}(t_i,v_j)$ 为虚拟机 v_j 能够执行任务 t_i 的最早就绪时间,则 $\mathrm{SET}(t_i,v_j)$ 和 $\mathrm{FET}(t_i,v_j)$ 可以分别计算为,

$$\mathrm{SET}(t_i,v_j)=\max\left\{\max_{t_p\in PT(t_i)}\{\mathrm{FET}(t_p,v_p)+\mathrm{DTT}(t_p,t_i)\},\mathrm{ERT}(t_i,v_j)\right\} \tag{4}$$

$$\mathrm{FET}(t_i,v_j)=\mathrm{SET}(t_i,v_j)+\mathrm{ET}(t_i,v_j) \tag{5}$$

1.2 成本模型

工作流的执行成本用工作流执行期间云计算平台的租赁费用表示,本文根据按需租赁模型计算工作流任务的执行成本,租赁费用一般通过工作流使用云计算平台中服务的收费间隔数来计算,小于一个收费间隔 BI 会按照一个收费间隔计算[8]。

此外,实例 s_j 的租赁时间从部署在 s_j 上的第一个任务 t_f 开始从其前驱任务接收数据起,到部署在 s_j 上的最后一个任务 t_l 执行完毕并向其后继任务完成数据传输时结束,因此实例 s_j 的开始租赁时间 $\mathrm{SRT}(s_j)$ 以及结束租赁时间 $\mathrm{FRT}(s_j)$ 可以分别计算为,

$$\mathrm{SRT}(s_j)=\mathrm{SET}(t_f,v_j)-\max_{t_p\in PT(t_f)}\{\mathrm{DTT}(t_p,t_f)\} \tag{6}$$

$$\mathrm{FRT}(s_j)=\mathrm{FET}(t_l,v_j)+\max_{t_s\in ST(t_l)}\{\mathrm{DTT}(t_l,t_s)\} \tag{7}$$

实例 s_j 的租赁时间可以通过 $\mathrm{SRT}(s_j)$ 和 $\mathrm{FRT}(s_j)$ 得到,并将其转换为收费间隔数来计算实例 s_j 的租赁费用 $\mathrm{LC}(s_j)$,用 $\mathrm{Price}(s_j)$ 来表示实例 s_j 的租赁单价,那么实例 s_j 的租赁费用 $\mathrm{LC}(s_j)$ 可以计算为,

$$\mathrm{LC}(s_j)=\left\lceil\frac{\mathrm{FRT}(s_j)-\mathrm{SRT}(s_j)}{\mathrm{BI}}\right\rceil\times\mathrm{Price}(s_j) \tag{8}$$

1.3 问题形式化

给定一个工作流 w 和一个云计算平台,求解工作流任务调度问题主要是生成一个租赁实例集 $S=\{s_1,s_2,\ldots,s_{|S|}\}$ 以及一个任务实例映射结果集 $R=\{r_1,r_2,\ldots,r_n\}$。s_j 是 S 的一个租赁实例,可以表示为四元组 $<\mathrm{type}(s_j),id,\mathrm{SRT}(s_j),\mathrm{FRT}(s_j)>$,分别代表实例 s_j 的虚拟机类型,实例 id,实例 s_j 的开始租赁时间以及结束租赁时间。r_i 是任务 t_i 在虚拟机 v_j 上的调度结果,可以表示为 $<t_i,v_j,\mathrm{SET}(t_i,v_j),\mathrm{FET}(t_i,v_j)>$,即任务 t_i 在虚拟机 v_j 上从 $\mathrm{SET}(t_i,v_j)$ 到 $\mathrm{FET}(t_i,v_j)$ 时间段执行。工作流 w 的响应时间 $RT(w)$ 定义为所有任务完成执行的时间,用公式表示为,

$$\mathrm{RT}(w)=\max_{t_i\in T}\{\mathrm{FET}(t_i,v_j)\} \tag{9}$$

进一步地,工作流 w 的执行成本 $EC(w)$ 可计算为,

$$\mathrm{EC}(w)=\sum_{s_j\in S}\mathrm{LC}(s_j) \tag{10}$$

使用 D 表示工作流的截止日期。进行工作流任务调度的目的是在保证工作流响应时间 $RT(w)$ 不超过 D 的前提下,最小化工作流执行成本 $EC(w)$。根据上述定义,截止日期约束的工作流任务调度成本优化问题可以定义为:

$$\begin{aligned}&\text{Minimize} && \mathrm{EC}(w)\\ &\text{Subject to} && \mathrm{RT}(w)\leqslant D\end{aligned} \tag{11}$$

2 截止日期约束的云中资源调度成本优化算法

CODC 算法的整体框架如图 1 所示。CODC 包括 4 个调度阶段:任务合并,任务优先级分配,任务子截止日期分配和租赁实例选择,接下来将对每个调度阶段进行详细说明。

2.1 任务合并

为了减少工作流调度过程中不同实例之间的数

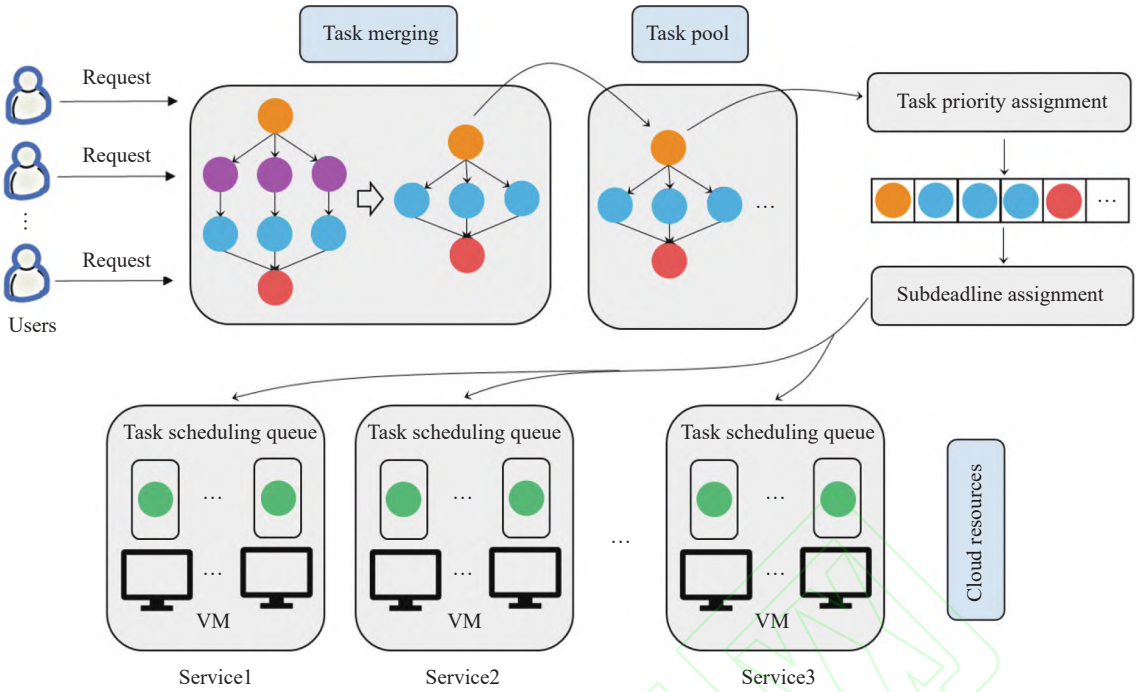

图 1 CODC 算法框架图

Fig. 1 Framework of CODC algorithm

据传输开销, 将多个相关联的工作流任务合并为单个任务。首先为每个工作流添加两个零工作负载、零传输数据的虚拟任务 t_{entry} 和 t_{exit}, 并将这两个任务作为工作流的入口和出口。其次根据条件进行任务合并, 合并规则如下: 任务 t_i 是任务 t_s 的父任务, 当 t_i 有唯一的子任务 t_s, 且 t_s 有唯一的父任务 t_i 时, 将 t_i 和 t_s 合并为 t_{i+s}。若 t_i 和 t_s 被分配到不同的实例上执行。由于任务间的数据依赖关系, 会不可避免地产生数据传输成本, 而任务合并保证相关联的任务分配到同一服务上, 因此避免了数据传输成本的产生。任务合并的具体步骤如下:

输入: 由 n 个任务组成的工作流 w

输出: 任务合并完成后的工作流 w'

(1) 在 w 中添加 t_{entry} 和 t_{exit}

(2) 通过 t_{entry} 生成任务序列 sequence ;

(3) **for** each 任务 t_i in sequence **do**

(4) **if** t_i 有唯一子任务 t_s **then**

(5) **if** t_s 有唯一父任务 t_i **then**

(6) 更新 t_s 的工作负载 $W(t_s)=W(t_i)+W(t_s)$;

(7) 更新 t_s 的父任务集合;

(8) 更新 t_i 的父任务的子任务集合;

(9) **end**

(10) **end**

(11) 将 t_i 从任务序列 sequence 移除;

(12) **end**

2.2 任务优先级分配

将工作流中当前任务 t_i 到出口任务 t_{exit} 的最长路径 $\text{LP}(t_i)$ 作为任务 t_i 的等级, $\text{LP}(t_i)$ 是通过出口任务 t_{exit} 向前遍历计算得到的, 并考虑了任务之间的数据传输时延。选择任务的最小执行时间 MET 是因为当前调度问题的首要目标是满足工作流的截止日期约束。因此任务等级 $\text{LP}(t_i)$ 可以计算为,

$$\text{LP}(t_i)=\begin{cases}0, & \text{if } t_i \text{ equal to } t_{\text{exit}}\\ \text{MET}(t_i)+\max\limits_{t_s\in \text{ST}(t_i)}\left\{\text{LP}(t_s)+\dfrac{d_{i,s}}{bw}\right\}, & \text{else}\end{cases} \tag{12}$$

最长路径越长的任务其优先级就越高, 如果存在任务等级相同的工作流任务, 则进一步比较每个任务的最早完成时间 EFT, 这里同样考虑任务之间的数据传输, 并将工作流任务的最小执行时间 MET 作为任务的执行时间, 因此任务 t_i 的最早完成时间 $\text{EFT}(t_i)$ 可以计算为,

$$\text{EFT}(t_i)=\text{MET}(t_i)+\max_{t_p\in PT(t_i)}\left\{\text{EFT}(t_p)+\frac{d_{p,i}}{bw}\right\} \tag{13}$$

规定最早完成时间越早的任务优先级越高, 并按照任务优先级由高到低进行排序生成工作流任务的调度队列。

2.3 任务子截止日期分配

任务子截止日期分配的主要目的是给每个任务一个子截止日期, 以便将工作流中的任务分离, 从而在子截止日期的约束下分别调度任务。任务 t_i 的子

截止日期 $SDL(t_i)$ 是根据任务等级 $LP(t_i)$ 和当前任务负载信息进行分配的，将任务 t_i 的子截止日期与其从入口任务 t_{entry} 到它的最长路径按比例地进行设置，因此 $SDL(t_i)$ 可以计算为，

$$SDL(t_i)=D\times\frac{LP(t_{entry})-LP(t_i)+MET(t_i)}{LP(t_{entry})} \tag{14}$$

2.4 租赁实例选择

候选租赁实例包括任务实例映射结果集中已经使用的所有实例，以及那些还没有使用但可以随时添加到租赁实例集 S 中的实例。租赁实例选择的首要标准是选择一个满足当前任务 t_i 的子截止日期并使成本增量最小化的租赁实例 s_j。需要注意的是，成本增量并不是实例 s_j 的租赁费用 $LC(s_j)$，而是通过将任务 t_i 分配给租赁实例 s_j 后的工作流执行成本减去分配之前的工作流执行成本计算的。

当没有实例可以满足当前任务 t_i 的子截止日期约束时，租赁实例选择的次要标准是最小化任务 t_i 的执行完成时间。通过这样的方法，工作流 w 的响应时间 $RT(w)$ 能够满足工作流截止日期 D 的可能性就会增加，从而提高算法的调度成功率。

2.5 工作流任务调度

基于上述计算过程，提出了一种截止日期约束的成本优化算法 CODC。CODC 算法描述如下：

输入：$S\leftarrow\varnothing, R\leftarrow\varnothing$

输出：任务实例映射结果 $<S,R>$

(1) 任务合并；

(2) 任务优先级分配；

(3)**for** each 任务 t_i in Queue **do**

(4) 计算 $SDL(t_i)$；

(5) $DTT_{max}(t_p,t_i)=t_i$ 的父任务与 t_i 之间最大的数据传输时间；CT 为当前时间；

(6) **for each** 虚拟机 v_j in VM **do**

(7) 计算 $ET(t_i,v_j)$；

(8) **if** $DTT_{max}(t_p,t_i)+ET(t_i,v_j)\leqslant SDL(t_i)-CT$ **then**

(9) 计算任务 t_i 在虚拟机 v_j 上执行的成本增量；

(10) **end**

(11) 选择一个成本增量最小的租赁实例 s_j；

(12) **if** S 中不存在满足上述子截止日期约束的实例 **then**

(13) $s_j=S$ 中最早完成 t_i 执行的租赁实例；

(14) **end**

(15) **end**

(16) 更新 $s_j=<type(s_j),id,SRT(s_j),FRT(s_j)>$；

(17) 更新 $r_i=<t_i,v_j,SET(t_i,v_j),FET(t_i,v_j)>$；

(18) **end**

首先，合并工作流任务以减少不同实例之间的数据传输开销。其次，关注父任务和子任务对当前任务优先级的影响进行任务优先级分配，并计算任务子截止日期。最后，考虑任务子截止日期未被满足的情况，以选择最早完成任务执行的实例。

具有 n 个任务的工作流最大数据依赖关系数为 $n(n-1)/2$，任务合并及优先级分配需要遍历每个任务和数据依赖关系，因此具有 $O(n^2)$ 的复杂度。接下来需要遍历 n 个任务和提供 m 个实例的服务资源，计算子截止日期并选择租赁实例，因此具有 $O(m*n)$ 的复杂度。当 $n>m$ 时，CODC 算法的总时间复杂度为 $O(n^2)$；否则为 $O(m*n)$。

3 性能评估

3.1 实验设置

本文使用五种不同类型的科学工作流应用程序(Montage, CyberShake, Epigenome, LIGO 和 SIPHT)对提出的 CODC 算法进行性能评估，这些工作流应用程序在结构、依赖关系、通信数据和计算特性等方面各不相同[9]，图 2 给出了每种工作流应用程序的一个简单结构图。本文为每种工作流应用程序选择三种规模，分别是 50、100 和 200 个任务，对于每种规模都使用了 20 个工作流应用程序实例。

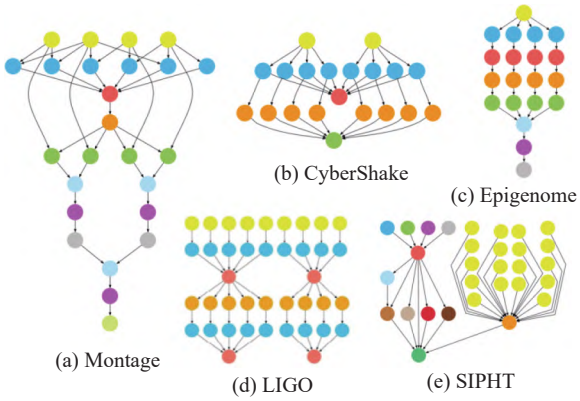

图 2 五种科学工作流结构图

Fig. 2 Structure diagram of five scientific workflows

由于很难在真实的数据中心上进行可重复的实验，因此实验在提供类似 Amazon EC2 服务的仿真平台上进行。假设使用了一个提供 5 种不同类型的服务的数据中心，每种服务都具有不同的处理能力和成本，如表 1 所示。将不同服务之间的平均带宽设置为 20MBps，即 Amazon EC2 提供的计算服务之间的近似平均带宽，并将收费间隔 BI 设置为 1 h，

表 1 不同服务类型的处理速度和成本表

Table 1 Processing speed and cost table for different service types

Type	ECUs(cores)	Processing Capacity	Cost/USD
1	1(1)	4.4	0.04
2	4(2)	17.6	0.16
3	5(2)	22.0	0.20
4	8(4)	35.2	0.32
5	20(8)	88.0	0.80

也与 Amazon EC2 相同。

此外, 引入松弛系数 λ 表示截止日期的宽松程度, M_F 和 M_c 表示工作流在效率最快的服务和成本最低的服务上执行的成本, 工作流截止日期可以计算为,

$$D = M_F + (M_C - M_F) \times \lambda \tag{15}$$

3.2 实验对比算法

为了测试提出算法的调度性能, 将 CODC 算法与 IC-PCP 算法 (IaaS Cloud Partial Critical Paths)[10]、PSO(Particle Swarm Optimization)[11] 和 ProLiS 算法 (Deadline-constrained Probabilistic List Scheduling)[12] 进行对比实验, 三种基准算法的主要原理如下:

(1) IC-PCP 算法通过递归方式从已分配的任务出发构造局部关键路径, 每个关键路径上的任务被安排到成本最低且满足子截止期限约束的虚拟机上。

(2) PSO 最初通过随机产生一组粒子作为问题的有效解, 在每一轮迭代中粒子根据自身信息以及当前全局最优解不断更新下轮迭代速度和位置, 循环往复直至输出最优解。

(3) ProLiS 算法为每个工作流任务分配子截止期限, 根据任务的紧急程度建立一个有序的任务列表, 将每个任务分配给满足其子截止期限且成本增量最少的服务。

3.3 实验结果与分析

将时间松弛系数 λ 按照步长 0.005 从 0.005 逐步递增到 0.05, 并对每组实验数据进行对比和具体分析, 以此验证 CODC 算法的性能优势。由于 50 和 100 个任务规模下三种对比算法的调度成功率过低, 限制了工作流执行成本的对比, 因此便不再展示。

200 个任务规模下四种算法在松弛系数由小到大过程中执行成本的变化情况如图 3 所示。可以发现随着松弛系数 λ 不断增大, 工作流截止日期不断放宽, 工作流任务所分配到的服务其执行能力也逐渐降低, 工作流执行成本也逐渐降低。(1)在 Montage

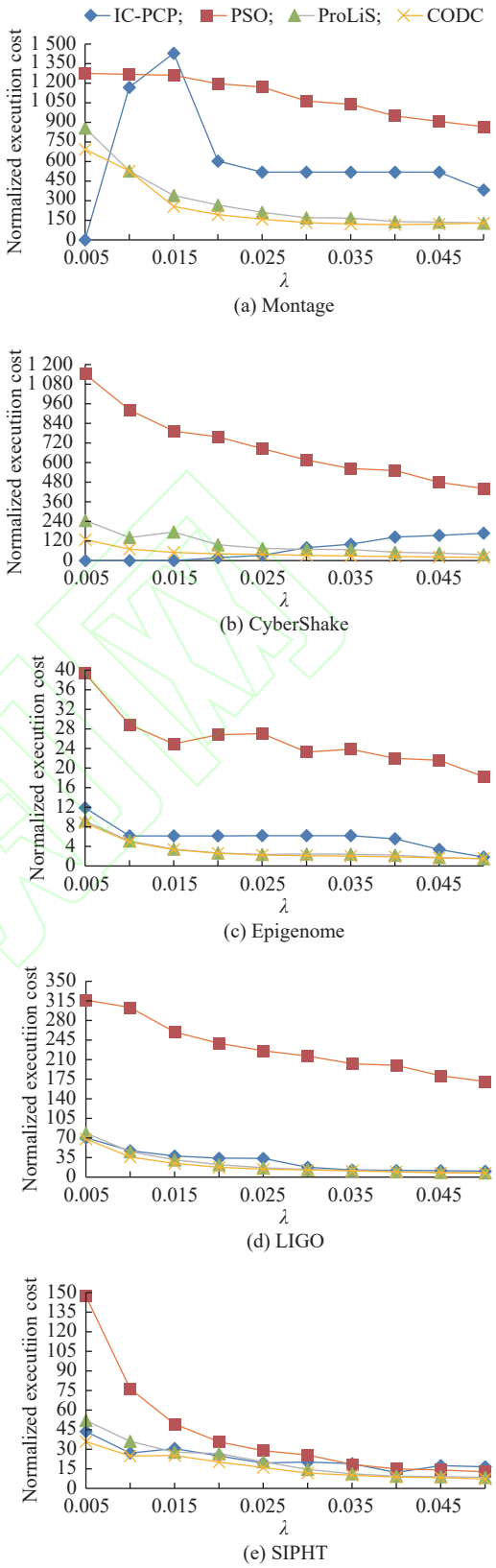

图 3 每种算法的执行成本

Fig. 3 The execution cost of each approach

上, PSO 的执行成本最高, IC-PCP 算法的执行成本低于 PSO 的执行成本但明显高于 ProLiS 算法和 CODC 算法的执行成本, CODC 算法的执行成本最

低。(2)在 CyberShake 上，PSO 的执行成本最高，CODC 算法的执行成本最低。(3)在 Epigenome 上，PSO 的执行成本最高，IC-PCP 算法的执行成本明显低于 PSO 但高于 ProLiS 算法和 CODC 算法的执行成本，CODC 算法的执行成本略低于 ProLiS 算法。(4)在 LIGO 上，PSO 具有最高的执行成本，CODC 算法的降低执行成本的能力略优于 IC-PCP 算法和 ProLiS 算法。(5)在 SIPHT 上，PSO 的执行成本最高，CODC 算法的执行成本低于 IC-PCP 算法和 ProLiS 算法的执行成本。

4 结束语

针对云工作流的部署问题，本文提出了一种截止日期约束的成本优化算法 CODC。该算法进行云工作流任务合并，优化任务优先级分配过程，为每个任务设计子截止日期，并选择最早完成当前任务执行的实例。在 5 种工作流上的实验结果表明：在同样的截止日期约束下，该算法进一步降低了云工作流调度过程中的执行成本。尤其在工作流 Montage 上，CODC 算法的执行成本与三种对照算法 (IC-PCP, PSO, ProLiS) 的执行成本相比分别降低了约 87%, 75% 和 16%。由于本文关注的是单工作流中的任务调度问题，所以在接下来的工作中可以对 CODC 算法进行优化，以处理并行工作流应用程序中任务的调度问题。

参考文献：

[1] BERA S, MISRA S, RODRIGUES J. Cloud computing applications for smart grid: A survey[J]. Parallel & Distributed Systems IEEE Transactions on, 2015, 26(5): 1477-1494.

[2] HAN P, DU C, CHEN J, *et al*. Minimizing monetary costs for deadline constrained workflows in cloud environments[J]. IEEE Access, 2020, 8: 25060-25074.

[3] CHEN H, WEN J, PEDRYCZ W, *et al*. Big data processing workflows oriented real-time scheduling algorithm using task-duplication in geo-distributed clouds[J]. IEEE Transactions on Big Data, 2020, 6(1): 131-144.

[4] TOUSSI G K, NAGHIBZADEH M. A divide and conquer approach to deadline constrained cost-optimization workflow scheduling for the cloud[J]. Cluster Computing, 2021, 24(3): 1711-1733.

[5] 王旖旎. 满足实时云任务需求的主动响应式工作流调度模型[J]. 计算机应用与软件, 2021, 38(8): 99-106+166.

[6] AHMAD W, ALAM B, AHUJA S, *et al*. A dynamic VM provisioning and de-provisioning based cost-efficient deadline-aware scheduling algorithm for Big Data workflow applications in a cloud environment[J]. Cluster Computing, 2021, 24(1): 249-278.

[7] 张艮山, 刘旭宁. 一种多重约束下确保成功率的云工作流调度方法[J]. 计算机应用与软件, 2021, 38(2): 278-284, 317.

[8] SINGH S P, NAYYAR A, KAUR H, *et al*. Dynamic task scheduling using balanced VM allocation policy for fog computing platforms[J]. Scalable Computing: Practice and Experience. 2019, 20(2): 433-456.

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

[10] ABRISHAMI S, NAGHIBZADEH M, EPEMA D. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds[J]. Future Generation Computer Systems, 2013, 29(1): 158-169.

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

[12] WU Q, ISHILAWA F, ZHU Q, *et al*. Deadline-constrained cost optimization approaches for workflow scheduling in clouds[J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 28(12): 3401-3412.

Deadline Constrained Cloud Resource Scheduling Cost Optimization Algorithm

HAN Yilin, FAN Guisheng, YU Huiqun
(School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China)

Abstract: As more and more workflow applications are deployed in the cloud, how to optimize resource scheduling cost while satisfying workflow deadline constraints is a popular research area. In this paper, we propose a cost optimization with deadline constraint (CODC) algorithm, firstly, merging workflow tasks to reduce the data transfer overhead between different instances. Secondly, paying attention to the impact of parent and subtasks on the priority of the current task, and the case of unmet task sub-deadlines is considered to select the instance that completes task execution earliest. Finally, the experimental results are compared with existing algorithms on five workflows and show that the CODC algorithm has a lower workflow execution cost compared to other three comparison algorithms.

Key words: cloud computing; resource scheduling; deadline constraints; task instance mapping; cost optimization