·云计算专题· 文章编号:1000-3428(2018)08-0030-08 文献标志码:A 中图分类号:TP393

基于期限约束与关键路径的云工作流调度

刘雨潇,王 毅,袁 磊,吴 钊

(湖北文理学院 数学与计算机科学学院,湖北 襄阳 441053)

摘 要: 为优化云工作流任务的调度过程,提出基于期限约束与关键路径的工作流调度算法 WS-DCCP。结合云资源的异质与弹性特征对任务进行逻辑分层,在此基础上,正比例重分配工作流期限,通过改进的任务升秩与降秩值之和赋予任务优先级,并根据任务优先级构造工作流的约束关键路径,将约束关键路径上的任务集调度至同一资源以降低通信代价。在满足约束关键路径子期限的条件下寻找执行代价最小化的资源,进而获得满足期限约束的代价最小调度解。仿真结果表明,与 IC-PCP 算法和 JIT 算法相比,WS-DCCP 算法可以在满足期限约束的同时降低工作流调度代价,提高调度成功率。

关键词: 云计算;工作流调度;期限约束;任务优先级;约束关键路径

中文引用格式: 刘雨潇,王 毅,袁 磊,等. 基于期限约束与关键路径的云工作流调度[J]. 计算机工程,2018,44(8):30-37.

英文引用格式: LIU Yuxiao,WANG Yi,YUAN Lei,et al. Cloud workflow scheduling based on deadline constraint and critical path[J]. Computer Engineering,2018,44(8):30-37.

Cloud Workflow Scheduling Based on Deadline Constraint and Critical Path

LIU Yuxiao,WANG Yi,YUAN Lei,WU Zhao

(School of Mathematical and Computer Science,Hubei University of Arts and Science,Xiangyang,Hubei 441053,China)

【Abstract】 To optimize the scheduling process of cloud workflow tasks,a workflow scheduling algorithm based on deadline constraint and Critical Path(CP) is presented,which named WS-DCCP. Firstly,combined with the heterogenous and elastic features of cloud resources,all tasks are partitioned into different logic levels. Based on this logic levels,the workflow deadline is proportionally re-distributed. Then,tasks are prioritized by the improved sum of task upwark rank and downward rank,and the Constrainted CP(CCP) are constructed based on task priority. Finally,the tasks set on the CCP are scheduled on the same resource so as to reduce the communication cost. The next is to find the resource minimizing the workflow execution cost under meeting the sub-deadline constraint of the CCP,which can obtain the scheduling solution minimizing the execution cost under meeting deadline constraint. Through the simulation experiments,the performance evaluations are conducted compared with IC-PCP and JIT. The results show that WS-DCCP can reduce the workflow scheduling cost and improve the scheduling success rate while meeting deadline constraints.

【Key words】 cloud computing;workflow scheduling;deadline constraint;task priority;Constrainted Critical Path(CCP)

DOI: 10.19678/j.issn.1000-3428.0047754

0 概述

云计算可以融合大规模计算能力应用于实际问题求解,如工业、医疗、商业和科学计算等领域。就调度而言,以上领域的应用通常涉及复杂且多阶段的操作处理过程,即工作流模式[1]。云计算正是通过其弹性的资源提供模式及即付即用的资源支付方式,使得各领域下的工作流任务可以高效完成,类似 Globus Galaxies 平台[2]的云工作流应用正使得云计算环境成为构建科学工作流调度与分析的主流方法。

将工作流调度至可用资源,同时满足任务依赖关系及用户定义的相关服务质量(Quality of Service,QoS)约束即为工作流调度问题,通常为 NP-完全问题。云资源上的工作流调度包括 2 个阶段:资源提供与任务调度[3]。资源提供阶段旨在决定任务所需资源的类型和数量,并预留至工作流执行。工作流任务调度阶段旨在决定任务最优执行序列和满足用

基金项目: 国家自然科学基金面上项目(61272296,61172084);湖北省自然科学基金面上项目(2014CFB634);湖北省襄阳市科技计划项目(2015zd26)。

作者简介: 刘雨潇(1982—),男,讲师、硕士,主研方向为云计算、大数据;王 毅,副教授、硕士;袁 磊,教授;吴 钊,教授、博士。

收稿日期: 2017-06-29 **修回日期:** 2017-09-08 **E-mail:** liuyuxiao82@126.com

户与工作流约束的任务部署[4]。目前的研究工作多集中于第二阶段,即对于预定义的资源池(通常为同质资源),以最小化工作流执行时间为目标,未考虑资源使用代价。

本文提出一种基于期限约束与关键路径的云工作流调度算法,用于动态云资源提供环境中的调度优化。作为一种线性启发式调度方法,该算法包括 2 个阶段:任务优先级确定和任务分配。第 1 个阶段为各个工作流任务分配升秩/降秩值,并基于秩值和对任务进行调度排序;第 2 个阶段则寻找最优执行资源。本文通过两阶段的工作流调度,实现期限约束下工作流执行代价的最小化。

1　相关工作

在独立型包任务与依赖型工作流任务调度领域中,常见算法有启发式算法、搜索算法和元启发式算法。任务至资源间的分配可划分为调度阶段和提供阶段。传统的 GAIN 算法[5]可归类为纯任务调度阶段的算法,而 DRIVE 算法[6]更侧重于资源提供阶段。仅考虑某一阶段的优化通常较适用于传统的分布式计算环境,而云调度系统则同时包含了调度与提供 2 个阶段。

工作流调度算法有 2 个主要的分类:尽力服务调度与 QoS 约束调度[7]。尽力服务调度算法的目标是最小化工作流执行跨度 makespan,如 HEFT 算法[8]、Suffrage 算法[9]、Min-Min 算法[9]和 Max-Min 算法[9]等,这类算法通常忽略执行代价因素,并不符合云资源的使用特征。QoS 约束调度算法主要是在满足用户定义的相关约束的条件下优化 QoS 参数,如考虑预算约束算法[10]和考虑期限约束算法[11]。相比尽力服务调度算法,QoS 约束调度算法更适应于现实世界中科学工作流的应用特征。然而,该类算法在约束条件和优化较多时,会因较高的时间复杂度而不具实用性。

元启发式算法是解决多约束调度的常用方法,如遗传算法 GA[12]、蚁群优化算法 ACO[13]和粒子群优化算法 PSO[14-15]等随机搜索算法。该类算法可以通过种群初始化和大量解空间,以较高的时间代价求解云环境中的有效调度解。然而,该类算法寻找可行调度解时的开销会随着工作流规模的增加而增大,因此,在面对复杂大规模的实时工作流调度时效率不高。

在与本文类似的其他相关研究中,文献[16]提出一种基于局部关键路径的调度算法 IC-PCP,算法目标是满足截止时间约束下最小化执行代价。该算法将处于局部关键路径的所有任务优先调度至费用最低的资源上,避免每个 PCP 上的通信代价,但算法忽略了资源的启动和部署时间。文献[17]在 IC-PCP 算法的基础上,提出增强算法 EIPR,利用资源预分配空闲时间和预算盈余进行任务复制,以降低代价。然而,该算法会因为任务复制增加用户代价。文献[18]提出一种分割平衡时间调度算法 PBTS,在满足期限约束的同时最小化执行代价,算法主要通过估算所需资源的最小数量最小化执行代价。文献[19]提出一种满足给定预算和截止时间约束的最大化工作流执行数量的算法。然而,与本文算法相比,该算法仅考虑了一种资源类型,本文则考虑了多类型的资源使用实例。文献[20]提出一种满足期限约束的动态代价最小化算法 JIT,将管道任务集合并为单个任务以降低协作任务间的数据通信代价,但该算法并未优化目标资源寻找过程。

2　问题模型与定义

本文以有向无循环图(Directed Acyclic Graph,DAG)表示工作流,定义为 $G=(T,E)$。其中:T 表示图的节点,即任务集;E 表示图的有向边,即任务间的依赖关系集;边 $e_{i,j}\in E$ 表示任务 t_i 与 t_j 间的有向边,即执行顺序约束,仅当完成任务 t_i 并接收 t_i 的所有数据后,才开始执行 t_j,此时,t_i 称为 t_j 的直接前驱或父任务,t_j 称为 t_i 的直接后继或子任务。在工作流 DAG 中,每个任务可拥有一个或多个父任务或子任务,仅当 t_i 的所有父任务完成后,t_i 才可以开始执行。若任务没有任一父任务,则称该任务为 DAG 的入口任务(entry task);若任务没有任一子任务,则称该任务为 DAG 的出口任务(exit task)。若工作流 DAG 拥有多个入口或出口任务,为了确保 DAG 仅有单个输入与输出路径,通常增加 2 个傀儡任务作为入口任务 t_{entry} 和出口任务 t_{exit},且傀儡任务的执行代价及与其他任务间的通信数据均为 0。

在云环境中调度工作流时,用户通常会提供相应的 QoS 约束,本文考虑工作流执行的期限约束 D_{user} 和执行代价 $cost$,优化目标是在满足期限约束的同时,追求工作流执行代价的最小化,形式化为:

$$\begin{cases}\min\ cost\\ \text{s. t. }\ makespan\leqslant D_{\text{user}}\end{cases}\tag{1}$$

其中,$makespan$ 表示整个工作流的执行时间。

3　WS-DCCP 算法设计

3.1　任务优先级确定

将所有工作流任务根据并行程度和同步需求划

分为不同的逻辑层次(level)。为了最大化任务的并行程度,在划分任务分层时,考虑同一层次中的任务间不存在依赖性,从而使每一层次可考虑为包括独立任务集的包任务。定义一种向下分层法对任务进行分层,即定义任务 t_i 的分层为任务 t_i 至出口任务之间路径的边的最大数量,如图 1 所示。

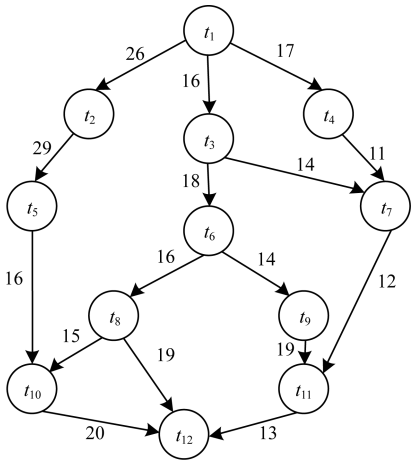

图 1 工作流 DAG

在图 1 中,边上的值表明任务间的通信时间(代价),分层表明任务所处的任务包(Bag of Tasks, BoT)。对于出口任务,其分层恒为 1。对于其他任务,分层计算方式为:

$$level_{\text{num}}(t_i)=\max_{t_j\in succ(t_i)}\{level_{\text{num}}t_j)+1\} \tag{2}$$

其中,$succ(t_i)$ 表示任务 t_i 的直接后继集,$level_{\text{num}}(t_i)$ 表示任务 t_i 所在层数。

根据任务分层,将所有拥有相同分层的任务组成任务分层集合 TLS,即:

$$TLS(l)=\{t_i\mid level_{\text{num}}(t_i)=l\} \tag{3}$$

其中,l 表示分级数,且 $l\in[1,2,\cdots,level_{\text{num}}(t_{\text{entry}})]$。

3.1.1 正比例期限重分配

任务分层后,需要将工作流期限 D_{user} 以正比例方式在各个分层上进行重分配。令 $level_{\text{sub-deadline}}$ 表示单个分层的子期限。为满足全局期限 D_{user} 的约束,需要确保单个分层中每个任务在其分配的子期限内完成。

首先,定义每个分层 l 的初始子期限估算值为:

$$level^{l}_{\text{sub-deadline}}=\max_{t_i\in TLS(l)}\{ECT(t_i)\} \tag{4}$$

在式(4)中,$ECT(t_i)$ 为任务 t_i 在所有资源上的最早完成时间,定义如下:

$$ECT(t_i)=exe_{\min}(t_i)+\max_{t_k\in pred(t_i)}\{level^{l_{ik}}_{\text{sub-deadline}}+ACT_{i,k}\} \tag{5}$$

其中,$pred(t_i)$ 表示任务 t_i 的直接前驱集,$exe_{\min}(t_i)$ 表示任务 t_i 的最小执行时间,$ACT_{i,k}$ 表示任务 t_i 与其父任务 t_k 间的平均通信时间,l_{ik} 表示父任务 t_i 的分层。由于入口任务 t_{entry} 没有父任务,因此 $ECT(t_{\text{entry}})=0$。

式(4)表明,一个分层中所有任务的最大 ECT 可视为该分层的全局完成时间估算值,该时间可作为所处分层中所有任务并行执行时要求的绝对最小完成时间。

计算所有分层的期限估算值后,需要以正比于 $level^{l}_{\text{sub-deadline}}$ 的方式,将用户期限 D_{user} 在所有任务上作非均匀重分配:

$$\pi_{\text{sub-deadline}}=\frac{D_{\text{user}}-level^{l}_{\text{sub-deadline}}}{level^{l}_{\text{sub-deadline}}} \tag{6}$$

其中,$level^{l}_{\text{sub-deadline}}$ 表示包含出口任务的分层。

然后,基于每个分层的期限长度,将以上比例加至每个分层中:

$$level^{l}_{\text{sub-deadline}}=level^{l}_{\text{sub-deadline}}+(\pi_{\text{sub-deadline}}\times|level^{l}_{\text{sub-deadline}}|) \tag{7}$$

显然,拥有越长任务的分层将得到越长的期限分配。

3.1.2 约束关键路径

定义 1(关键路径) 工作流 DAG 中入口任务至出口任务间的最长路径称为关键路径(Critical Path, CP)。

定义 2(关键路径的长度) 表示工作流 DAG 的关键路径上任务的计算时间与通信时间之和,即调度工作流的时间下限。

定义 3(约束关键路径) 仅包括就绪任务的任务集构成的路径称为约束关键路径(Constrainted Critical Path,CCP)。

定义 4(就绪任务) 若某任务的所有父任务已经执行,且所有数据均已接收,则称该任务为就绪任务。

传统方法基于升秩和降秩的方式寻找工作流 DAG 的所有 CCP。任务 t_i 的升秩表示从任务 t_i 至 t_{exit} 间的关键路径的长度,定义为:

$$rank_{\text{up}}(t_i)=AET_i+\max_{t_j\in succ(t_i)}(ACT_{i,j}+rank_{\text{up}}(t_j)) \tag{8}$$

其中,AET_i 表示任务 t_i 的平均执行时间,$ACT_{i,j}$ 表示任务 t_i 的平均通信时间。从式(8)可以看出,任务的升秩需要从出口任务开始计算,然后递归计算至工作流 DAG 的入口任务。对于出口任务 t_{exit}:

$$rank_{\text{up}}(t_{\text{exit}})=AET_{\text{exit}} \tag{9}$$

任务的降秩需要从入口任务开始计算,然后递归计算至工作流 DAG 的出口任务,定义为:

$$rank_{\text{down}}(t_i)=\max_{t_k\in pred(t_i)}(AET_k+ACT_{k,i}+rank_{\text{down}}(t_k)) \tag{10}$$

同时,有:

$$rank_{\text{down}}(t_{\text{entry}})=0 \tag{11}$$

$rank_{\text{down}}(t_i)$ 表示任务 t_{entry} 至任务 t_i 的最长距离,

但不包括任务本身的计算时间,而 $rank_{up}(t_i)$ 则表示任务 t_i 至 t_{exit} 间关键路径的长度,包括任务本身的计算时间。

WS-DCCP 算法对任务升秩与降秩计算方式进行改进,如式(12)和式(13)所示。

$$MRank_{up}(t_i) = AET_i + \sum_{t_j \in succ(t_i)} (ACT_{i,j} + \max_{t_j \in succ(t_i)} (rank_{up}(t_j))) \quad (12)$$

$$MRank_{down}(t_i) = \sum_{t_k \in pred(t_i)} ACT_{k,i} + \max_{t_k \in pred(t_i)} (AET_k + rank_{down}(t_k)) \quad (13)$$

与传统方法不同,改进升秩与降秩计算任务的前驱或后继任务的通信时间总量而不是选择其中的最大值。通过这种方式,可以使得拥有更高出度或入度的任务拥有更高的优先级,从而使得该类任务优先被执行,且在下一条约束关键路径 CCP 上的更多任务可被置为就绪任务。

同时,WS-DCCP 算法通过任务的升秩与降秩之和寻找所有工作流 DAG 的关键路径:

$$rank_{sum} = rank_{up} + rank_{down} \quad (14)$$

首先,基于任务的 $rank_{sum}$ 值对所有任务进行排序;然后,将拥有最高 $rank_{sum}$ 值的任务选择为第一条关键路径。第一条关键路径上的所有任务标识为已访问任务,以同样的方式,可以找到工作流的所有关键路径。该过程的执行步骤如算法 1 所示。

算法 1　关键路径查找算法

输入　工作流 $G=(T,E)$

输出　关键路径

```
1. procedureFindCP(DAG G)
2. for all task t_i ∈ G do//遍历工作流的所有任务
3. calculate rank_up ,rank_down and rank_sum//计算各任务秩值
4. end for
5. CPlist←Ø//初始化关键路径列表为空
6. while there is an univisited task in G do//寻找未访问任务
7. t_i←biggest rank_sum//寻找最大秩值和
8. CP←Ø//置关键路径为空
9. while t_i is not null do
10. add t_i to CP//添加任务 t_i至关键路径
11. t_i← max_{t_j∈pred(t_i)} (rank_sum t_j)//寻找最大父任务秩值和
12. end while
13. add CP to CPlist//添加关键路径至关键路径列表
14. end while
15. end procedure
```

3.1.3　算例说明

本节以一个实例,分别说明传统升秩与降秩方法和 WS-DCCP 算法的改进秩值方法寻找约束关键路径 CCP 的不同。考虑图 1 所示的工作流结构,边上数值代表任务间的数据通信时间,每个任务的平均计算时间 AET_i、升秩值 $rank_{up}$、降秩值 $rank_{down}$ 以及升降秩值之和 $rank_{sum}$ 如表 1 所示。

表 1　各任务计算时间、标准升降秩及其和

任务	AET_i	$rank_{up}$	$rank_{down}$	$rank_{sum}$
t_1	22	190	0	190
t_2	29	142	48	190
t_3	22	150	38	188
t_4	20	96	39	135
t_5	27	84	106	190
t_6	21	110	78	188
t_7	9	65	74	139
t_8	14	70	115	185
t_9	12	75	113	188
t_{10}	11	41	149	190
t_{11}	21	44	144	188
t_{12}	10	10	180	190

首先,根据选取任务 $rank_{sum}$ 最高的原则,可以得到第一条关键路径为 $t_1 \rightarrow t_2 \rightarrow t_5 \rightarrow t_{10} \rightarrow t_{12}$;然后,剔除已访问的第一条关键路径中的任务,以同样的 $rank_{sum}$ 最高原则,可依次得到其他关键路径;最后,需要通过遍历关键路径 CP,以循环方式寻找约束关键路径 CCP。在第 1 条 CP 中,仅有任务 t_1、t_2 可就绪,其他任务均无法就绪,则第 1 条 CCP 为 $t_1 \rightarrow t_2$。例如:考虑第 1 条关键路径中的任务 t_5,该任务未添加至该 CCP,由于其父任务之一 t_3 仍未添加至任意 CCP 中。当无法在第 1 条 CP 中找到更多的就绪任务时,即可考虑通过第 2 条CP 建立新的 CCP。在第 2 条CP 中,由于任务 t_3 的唯一父任务 t_1 已经添加至第 1 条 CCP 中,则任务 t_3 为就绪任务。依此类推,可知第 2 条 CCP 包括 3 个任务:$t_3 \rightarrow t_6 \rightarrow t_9$。从第 2 条CP 中排除任务 t_{11} 的原因在于它的一个父任务 t_7 仍未添加至任一 CCP 中。同理,其他的 CCP 可利用剩余的 CP 进行构造。通过上述过程得到的 CP 与 CCP 如表 2 所示。

表 2　通过标准升降秩得到的 CP 与 CCP

序号	CP	CCP
1	$t_1 \rightarrow t_2 \rightarrow t_5 \rightarrow t_{10} \rightarrow t_{12}$	$t_1 \rightarrow t_2$
2	$t_3 \rightarrow t_6 \rightarrow t_9 \rightarrow t_{11}$	$t_3 \rightarrow t_6 \rightarrow t_9$
3	$t_4 \rightarrow t_7$	$t_4 \rightarrow t_7$
4	t_8	t_8
5	—	$t_5 \rightarrow t_{10}$
6	—	t_{11}
7	—	t_{12}

基于同样的 CP 和 CCP 构造方法,改进的任务升秩与降秩值及得到的 CP 和 CCP 如表 3 和表 4 所示,其中,AET_i 表示每个任务的平均计算时间,$rank_{up}$ 表示改进升秩值,$rank_{down}$ 表示改进降秩值,$rank_{sum}$ 表示升降秩值之和。

表 3 各任务计算时间、改进升降秩及其和

任务	AET_i	$rank_{up}$	$rank_{down}$	$rank_{sum}$
t_1	22	284	0	284
t_2	29	142	48	190
t_3	22	203	38	241
t_4	20	96	39	135
t_5	27	84	115	199
t_6	21	140	78	218
t_7	9	65	85	150
t_8	14	89	115	204
t_9	12	75	113	188
t_{10}	11	41	173	214
t_{11}	21	44	156	200
t_{12}	10	10	236	246

表 4 通过改进升降秩得到的 CP 与 CCP

序号	CP	CCP
1	$t_1 \rightarrow t_3 \rightarrow t_6 \rightarrow t_8 \rightarrow t_{12}$	$t_1 \rightarrow t_3 \rightarrow t_6 \rightarrow t_8$
2	$t_2 \rightarrow t_5 \rightarrow t_{10}$	$t_2 \rightarrow t_5 \rightarrow t_{10}$
3	$t_4 \rightarrow t_7 \rightarrow t_{11}$	$t_4 \rightarrow t_7$
4	t_9	t_9
5	—	t_{11}
6	—	t_{12}

3.2 任务分配

任务分配阶段旨在寻找最佳资源执行 CCP 上的任务集。同时，为了避免增加通信代价，WS-DCCP 算法规定一条约束关键路径上的所有任务均执行于同一资源。任务分配的优化目标是在满足约束关键路径子期限的情况下最小化工作流执行代价。

令 $ECT(CCP_i, p_j)$ 为当前约束关键路径 CCP_i 在资源 p_j 上的最早完成时间，在单个任务的情况下，其值由式(5)决定。分层的子期限估算与当前 CCP 在资源 p_j 上的最早完成时间之差为：

$$Time_{CCP_i}^{p_j} = level_{\text{sub-deadlline}}^{l_{t_i}} - ECT(CCP_i, p_j) \quad (15)$$

其中，$level_{\text{sub-deadline}}$ 为分配至包含当前 CCP 中最后任务 t_i 的分层的子期限。如果当前 CCP 的最早完成时间超过分层子期限，即：

$$ECT(CCP_i, p_j) > level_{\text{deadlline}}^{l_{t_i}} \quad (16)$$

则表明式(15)可能为负值。同时，令 $Cost_{CCPi,pj}$ 表示在资源 p_j 上执行当前 CCP 上所有任务时的代价。

算法 2 给出了寻找最佳目标资源的执行过程。该过程需要考虑以下 3 种情况：

情况 1 由于云资源使用是基于账单数量的付费模式，以 Amazon EC2 为例，使用时间未达到 1 h 均按 1 h 间隔付费，如果任务可执行于还剩余使用账单时间的资源上，其执行代价为 0。因此，在分配资源时，算法优先选择还剩余空闲付费间隔的资源，即：算法的第 1 步是在确保 CCP 的最早完成时间不超过分层子期限的情况下，优先考虑无代价的资源执行 CCP，然后，选择最早完成时间最小的资源(即最快资源)。

情况 2 如果无法找到满足情况 1 的资源，需要提供一个新资源，此时算法在资源集中搜索满足分层子期限约束并且最低价的资源进行分配。

情况 3 对于较严格的期限约束，可能存在无法找到满足任务所在分层子期限的资源(即式(15)恒为负值)。如果存在某个 CCP 满足该条件，并不一定表明无法满足全局期限约束，而仅仅表明会违背子期限约束。此时，算法选择最佳性能的可用资源。

算法 2 寻找最佳资源算法

输入 关键路径与子期限

输出 代价最小化的最优资源

```
1. procedureResourceSelection(CCP_i)
2. F←find all resources that have zero cost for CCP_i
3. M←find all resources that can meet sub-deadline for CCP_i
4. if (F∩M)//情况 1
5. SelectResource←minECT(F∩M)
6. else if(M) then//情况 2
7. SelectResource←minCost(M)
8. else//情况 3
9. SelectResource←minCost(all resources)
10. end if
11. end procedure
```

3.3 算法时间复杂度

考虑任务数量为 n 的工作流 DAG $G=(T,E)$，假设 DAG 为全连通，则有向边的最大数量为 $n(n-1)/2$，处理所有任务及其依赖关系的时间复杂度为 $O(n^2)$。对于 WS-DCCP 算法的第一阶段，需要将所有 n 个就绪任务在 p 个可用资源上进行遍历，其时间复杂度为 $O(np)$，而选择所有工作流任务的时间复杂度为 $O(n^2p)$。对于 WS-DCCP 算法的第二阶段，由于前一阶段的所选任务需要在所有可用资源上进行计算，其时间复杂度为 $O(p)$。因此为任务选择资源的时间复杂度为 $O(np)$。另外，在计算关键路径时需要计算任务的升秩和降秩值，其时间复杂度为 $O(n^2p)$。综上，WS-DCCP 算法的时间复杂度为 $O(n^2+n^2p+np+n^2p)=O(n^2p)$。

4 仿真实验

本节对算法性能进行仿真评估，构建 2 种形式

的 WS-DCCP 算法:基于式(8)、式(10)的标准升秩与降秩算法 WS-DCCP(SR),基于式(12)、式(13)的改进升秩与降秩算法 WS-DCCP(MR)。另外,源于优化机制的相似性,选择 IC-PCP[16] 和 JIT[20] 作为基准算法。WS-DCCP(MR)同样使用逐层方式寻找关键路径,即:首先选择第一分层所有任务中拥有最高秩值的任务,然后在第二分层被选任务的子任务中,选择拥有最高秩值的任务。重复该过程,直到达到最后分层,找到所有其他关键任务,即可得到第 1 条关键路径。第 2 条关键路径重新回到剔除第 1 条关键路径后的子 DAG 中重新寻找。

4.1　实验环境与参数

通过仿真平台 CloudSim[21] 构造一个云数据中心和 6 种资源类型,资源的详细参数参考 Amazon EC2 设置,如表 5 所示。资源间的平均带宽参数参考 Amazon AWS 设置为 20 MB/s。EC2 单元的计算能力以每秒百万浮点操作次数 MFLOPS 进行度量。在实际的云计算环境中进行资源分配时,由于诸如时延、操作系统执行、资源类型、数据中心地理位置和资源请求量等因素的存在,可能导致资源启动时存在时延,因此,为了满足实际情况需求,仿真实验中为资源设置 97 s 的启动时间。

表 5　资源参数

资源类型	计算单元/MFLOPS	内存/GB	代价/$
m3.medium	3.0	3.75	0.067
m4.large	6.5	8.00	0.126
m3.xlarge	13.0	15.00	0.266
m4.2xlarge	26.0	32.00	0.504
m4.4xlarge	53.5	64.00	1.008
m4.10xlarge	124.5	160.00	2.520

同时,为了评估算法处理真实负载的性能,实验中使用 4 种现实科学工作流进行测试,包括 CyberShake、Montage、LIGO 和 SIPHT,利用 Pegasus 工作流产生器创建这 4 种合成工作流结构,同时将工作流规模设置为 200 个任务。

另外,为了评估算法的敏感性,实验设置不同的期限约束,其约束范围从较严格变化至较宽松。为了得到期限间隔,计算 2 种基准调度的长度:最快调度与最慢调度。

1)如果工作流的关键路径上的所有任务均执行于最快资源类型上,即可得到最快调度长度:

$$FS = \sum_{t_i \in CP} EC_i^j \tag{17}$$

其中,EC_i^j 表示任务 t_i 在最快资源 p_j 上的执行代价。

2)如果工作流的关键路径上的所有任务均执行于最慢资源类型上,即可得到最慢调度长度:

$$SS = \sum_{t_i \in CP} EC_i^k \tag{18}$$

其中,EC_i^k 表示任务 t_i 在最快资源 p_k 上的执行代价。

基于 FS 和 SS,将工作流的期限定义为:

$$D_{\text{user}} = FS + \alpha \times (SS - FS) \tag{19}$$

其中,α 表示期限因子,$\alpha \in [0.1, 1]$,α 以步长 0.1 进行递增,其值越小,期限约束越严格,其值越大,期限约束越宽松。

4.2　实验结果

为了比较算法的代价,引入满足期限的失效代价,即考虑任务执行失效对执行代价的影响。当执行结果无法满足期限约束时,视为一次失效。引入一个权重分配至算法返回的平均代价,令 k 表示满足期限约束的成功调度集合,则该权重代价可定义为:

$$Weighted\_Cost = \sum_k Cost(k)/SR \tag{20}$$

在式(20)中,$Cost(k)$ 表示满足期限约束的代价(式(1)得到的最小值),SR 表示算法的调度成功率,即满足调度期限的仿真实验次数与总的仿真运行次数的比率,定义为:

$$SR = n(k)/n_{\text{Tot}} \tag{21}$$

其中,n_{Tot} 表示总仿真次数,$n_{\text{Tot}} = 50$,$n(k)$ 为集合 k 的基数。将最低价调度考虑为所有任务均调度至最低价资源上执行,为了代价标准化处理,算法获得的权重代价为代价除以最低价调度。

图 2 显示了标准化代价的比较结果。可以看出,在多数情况下,WS-DCCP 算法均拥有比 IC-PCP 更低的代价。在最严格期限约束下($\alpha = 0.1$),IC-PCP 算法在 SIPHT 和 CyberShake 2 种工作流中拥有较低的代价,但在 Montage 中代价较高,在 LIGO 中 100% 失效。同时,对于 LIGO 和 Montage 工作流,WS-DCCP 算法在多数时间下能够以接近 IC-PCP 一半的代价调度工作流。此外,在所有工作流中,WS-DCCP(MR)算法在性能上优于 WS-DCCP(SR)算法,除了 CyberShake 工作流的最严格期限约束的情况。总体而言,JIT 算法降低通信代价的方式较IC-PCP 可以降低一定代价,但由于其资源选择并非最优,因此在4 种工作流中的代价均高于 WS-DCCP 算法。

图 3 显示了算法调度成功率的比较结果。可以看出,除了 CyberShake 工作流,WS-DCCP 算法几乎可以成功满足所有期限约束。在最严格期限约束下($\alpha = 0.1$),IC-PCP 算法和 JIT 算法的调度成功率表现较差,其中,IC-PCP 算法在 LIGO 工作流中几乎 100% 失效,而 JIT 算法的调度成功率略高于 IC-PCP 算法。结果还表明,WS-DCCP 算法对

于期限约束远没有IC-PCP算法和JIT算法表现得敏感,这反映WS-DCCP算法拥有比较稳定的性能。IC-PCP算法的不稳定性部分是由其较高的失效率导致的,源于该算法在选择资源分配时并没有做到最优选择。

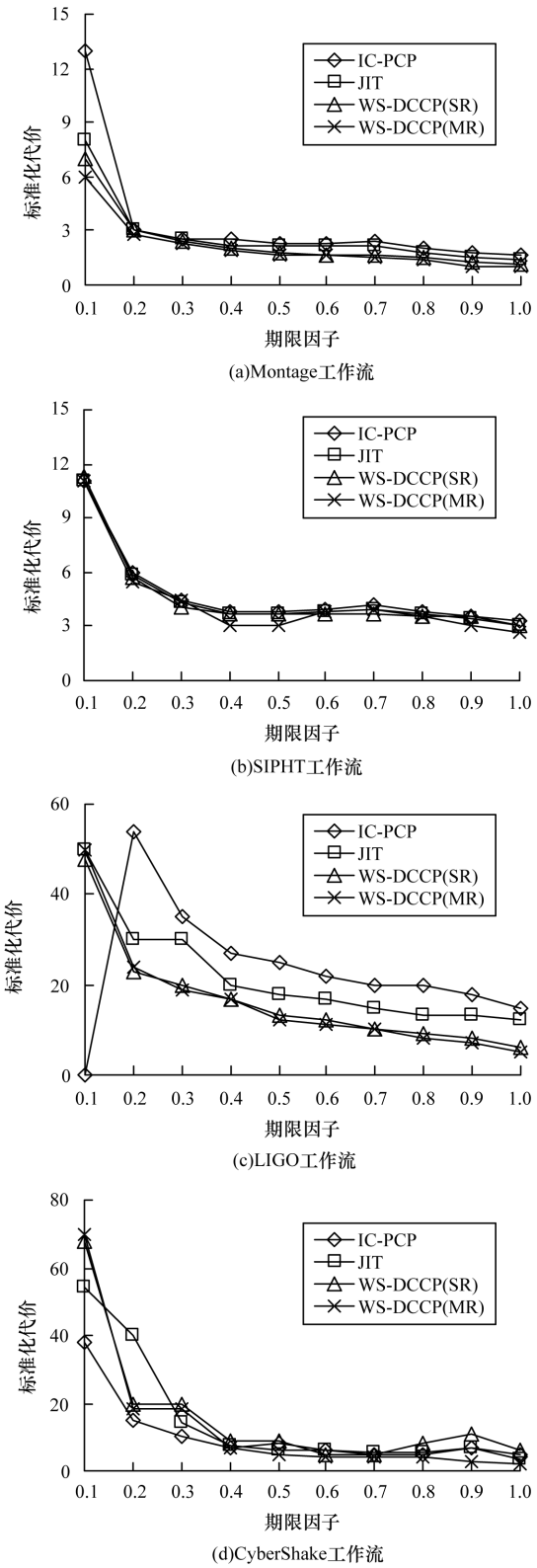

图 2 算法代价比较

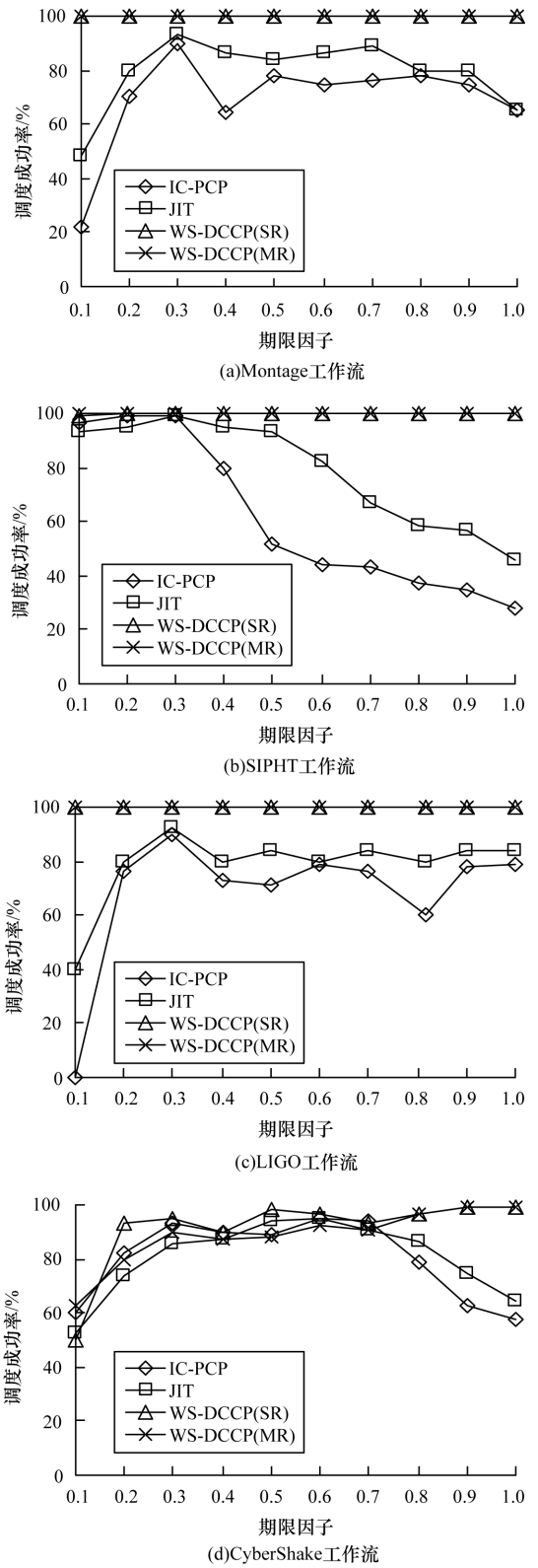

图 3 算法调度成功率比较

总体来看,WS-DCCP算法虽然无法保证在所有工作流类型和所有程度的期限约束下达到工作流执行代价的最小,但可以在降低代价的情况下仍然拥有更高的调度成功率,其综合性能更好,性能也更稳定。

5　结束语

为实现期限约束下的执行代价最小化,本文提出一种基于关键路径的云工作流调度算法。通过求解工作流结构的约束关键路径,并将约束关键路径上的任务执行于同一资源上,降低资源间的通信代价。同时,设计基于改进任务升秩与降秩之和的约束关键路径求解方法,引入基于任务秩值与关键路径的机制,以降低工作流执行代价。实验结果表明,本文算法可以在满足期限约束的同时,降低工作流执行代价,提高调度成功率。下一步研究将集中于多约束条件下的关键路径求解,如将预算约束或资源能耗因素考虑在内,求解多目标的工作流调度优化。

参考文献

[1] KASHLEV A, LU S. A system architecture for running big data workflows in the cloud[C]//Proceedings of IEEE International Conference on Services Computing. Washington D. C., USA: IEEE Press, 2014: 51-58.

[2] MADDURI R, CHARD K, CHARD R, et al. The Globus Galaxies platform: delivering science gateways as a service[J]. Concurrency and Computation: Practice and Experience, 2015, 27(16): 4344-4360.

[3] WU F, WU Q, TAN Y. Workflow scheduling in cloud: a survey[J]. Journal of Supercomputing, 2015, 71(9): 3373-3418.

[4] CALHEIROS R N, BUYYA R. Meeting deadlines of scientific workflows in public clouds with tasks replication[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(7): 1787-1796.

[5] SAKELLARIOU R, ZHAO H, TSIAKKOURI E, et al. Scheduling workflows with budget constraints[M]// GORLATCH S, DANELUTO M. Integrated Research in Grid Computing. Berlin, Germany: Springer-Verlag, 2012: 34-43.

[6] CHARD K, BUBENDORFER K, KOMISARCZUK P. High occupancy resource allocation for grid and cloud systems, a study with drive[C]//Proceedings of ACM International Symposium on High Performance Distributed Computing. New York, USA: ACM Press, 2013: 73-84.

[7] YU J, BUYYA R, RAMAMOHANARAO K. Workflow scheduling algorithms for grid computing[M]// XHAFA F, ABRAHAM A. Metaheuristics for Scheduling in Distributed Computing Environments. Berlin, Germany: Springer-Verlag, 2012: 173-214.

[8] 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.

[9] MAHESWARAN M, ALI S, SIEGEL H J, et al. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems[J]. Journal of Parallel and Distributed Computing, 2012, 59(2): 107-131.

[10] WU C Q, LIN X, YU D, et al. End-to-end delay minimization for scientific workflows in clouds under budget constraint[J]. IEEE Transactions on Cloud Computing, 2015, 3(2): 169-181.

[11] MIRZAYI S, RAFE V. A hybrid heuristic workflow scheduling algorithm for cloud computing environments[J]. Journal of Experimental and Theoretical Artificial Intelligence, 2015, 27(6): 721-735.

[12] YU J, KIRLEY M, BUYYA R. Multi-objective planning for workflow execution on grids[C]//Proceedings of the 8th IEEE/ACM International Conference on Grid Computing. Washington D. C., USA: IEEE Press, 2013: 10-17.

[13] 马俊波,殷建平. 云计算环境下带安全约束的工作流调度问题的研究[J]. 计算机工程与科学, 2014, 36(4): 607-614.

[14] 杨玉丽,彭新光,黄名选,等. 基于离散粒子群优化的云工作流调度[J]. 计算机应用研究, 2014, 31(12): 3677-3681.

[15] 王　月,刘　亚,郭继峰,等. 基于离散粒子群优化的云计算 QoS 调度算法[J]. 计算机工程, 2017, 43(6): 111-117.

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

[17] CALHEIROS R, BUYYA R. Meeting deadlines of scientific workflows in public clouds with tasks replication[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(7): 1787-1796.

[18] BYUN E K, KEE Y S, KIM J S, et al. Cost optimized provisioning of elastic resources for application workflows[J]. Future Generation Computer Systems, 2012, 27(8): 1011-1026.

[19] MALAWSKI M, JUVE G, DEELMAN E, et al. Cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds[C]//Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis. Washington D. C., USA: IEEE Press, 2012: 1-11.

[20] SAHNI J, VIDYARTHI D. A cost-effective deadline-constrained dynamic scheduling algorithm for scientific workflows in a cloud environment[J]. IEEE Transactions on Cloud Computing, 2018, 6(1): 2-18.

[21] GOYAL T, SINGH A, AGRAWAL A. Cloudsim: simulator for cloud computing infrastructure and modeling[J]. Procedia Engineering, 2012, 38(4): 3566-3572.

编辑　金胡考