改进CS算法结合决策树的云工作流调度

陈 超

(四川理工学院计算机学院 四川 自贡 643000)

【摘要】对云计算环境下工作流任务调度的现有方案进行分析，针对存在运行时间长、资源利用率低等不足，提出一种结合改进型布谷鸟搜索算法和决策树的工作流任务调度方案。首先，根据工作流任务属性分配截止期限；其次，利用改进型布谷鸟搜索算法将工作流分割成多个子工作流，最小化数据依赖性，再利用决策树选择出满足任务QoS约束的资源；最后，根据任务的计算时间、排队时间和通信延迟的总和来判断是否满足截止期限约束，以此配置相应的资源。实验结果表明，该方案具有较短的总运行时间和较高的任务完成率。

关 键 词 云计算； 布谷鸟搜索； 决策树； 工作流划分； 工作流调度

中图分类号 TP311 文献标志码 A doi:10.3969/j.issn.1001-0548.2016.06.017

Workflow Task Scheduling in Cloud Computing Based on Hybrid Improved CS Algorithm and Decision Tree

CHEN Chao

(School of Computer Science, Sichuan University of Technology & Engineering Zigong Sichuan 643000)

Abstract The existing workflow task scheduling schemes in cloud computing environment are analyzed, For the issues of the long operation time and low resource utilization, a workflow task scheduling scheme base on hybrid improved cuckoo search and decision tree in cloud computing is proposed. First, the deadline is assigned according to the work-flow task attribute; then, the improved cuckoo search algorithm is used to split the workflow into several sub workflow, minimizing data dependent; then, the decision tree is used to choose the resources which meet the QoS constraints of tasks; finally, the deadline constraints to be satisfied is judged according to satisfy according to the sum of task computing time, queuing time and communication delay, so as to configure the appropriate resources. Experimental results show that the proposed scheme has shorter total running time and higher task completion rate.

Key words cloud computing; cuckoo search; decision tree; workflow partition; workflow task scheduling

云计算[1]是一种新的计算技术，用户可以利用云计算租借软件、硬件、基础设施和计算资源作为每个用户的基础资源，将工作提交给云计算处理或存储。不同的用户有不同的QoS需求，在云计算中，云调度程序必须能够以最大化方式对工作流进行调度。科学工作流[2]是指将一系列在科学研究中遇到的数据管理、计算、分析、展现等工作变成独立的服务，再把这些服务通过数据链接组合在一起，满足研究人员科学实验和数据处理中的需要。传统的计算环境已很难满足科学工作流的需要，云计算以高性能的计算资源为科学工作流应用提供了一种全新的部署和执行方式。

目前，云计算环境下工作流任务调度方案作为云计算工作流技术的重要组成部分，已经成为该领域内的研究热点。工作流任务调度的主要目标是减少任务执行的总时间和资源的空闲时间，提高资源利用率[3]。为此，本文提出一种云计算环境下的工作流任务调度方案。该方案根据工作流任务属性分配截止期限，利用布谷鸟搜索(cuckoo search, CS)算法将工作流分割成多个子工作流，利用决策树选择满足任务QoS约束的资源，最后根据截止时间配置相应资源。实验结果表明，本文方案具有较短的总运行时间和较高的任务完成率。

1 相关研究

云计算环境中，何时租借云虚拟资源以及如何

收稿日期：2015－11－28；修回日期：2016－05－04
基金项目：四川省教育厅重点项目(15ZA0224)；人工智能四川省重点实验室(2014RYJ01)；四川省智慧旅游研究基地规划项目(ZHZ14-01)
作者简介：陈超(1969－)，男，副教授，主要从事云计算、大数据和高性能计算等方面的研究.

租借做出有效决策是一个难题。现有的一些调度策略主要以作业等待时间作为决策依据，缺乏对资源动态服务能力的有效评估。目前，也有学者以截止时间为约束提出一些任务调度方案，如，文献[4]描述了一种异态最早结束时间(HEFT)列表调度方案，该方案首先为工作流图中的节点和边赋权值，生成一个有序的任务列表，然后根据任务列表分配资源，但在后续调度过程中没有优化调度顺序的机制。文献[5]融入了回退机制对HEFT进行了改进(SHEFT)，使其能够动态地预分配和释放资源。然而，这些方案都以分配的截止时间为标准，没有估算任务的具体执行时间。文献[6]提出了一种混合计算环境下的任务调度策略(Aneka)，考虑了工作流截止时间的要求，其策略主要根据任务大小来预测作业完成时间，从而判断是否超出时间限制，该策略没有考虑任务的队列等待时间。

另外，现有大多数云计算工作流调度方案都是利用特定算法，直接将工作流进行调度，对工作流分割成子工作流进行调度的研究不多。文献[7]运用图形分割算法，尽量减少工作流执行期间的数据移动，最小化中间节点通信。但是，并没有研究划分后的工作流资源调度问题，不适合多资源的云计算环境。文献[8]提出一种支配队列任务调度算法，根据工作流中任务节点的计算负载和传输负载确定关键路径，将关键路径上的任务进行聚集，并安排在同一个资源上执行。该方法一定程度上减少了执行任务时资源之间的通信开销，但其从大规模图结构角度出发，延长了任务的调度时间。在云计算应用中，工作流中的任务会分配到不同的资源上，由于这些任务之间存在依赖性，所以资源之间需要数据通信。如果能将工作流中相对独立的较小任务分离出来，再将这些子工作流尽量分配到集中的资源上，将会大大降低资源之间的通信量，提高任务执行效率。为此，本文利用一种启发式智能算法将工作流进行划分，最小化子工作流间的数据依赖性。

本文的主要创新在于：

1) 采用CS搜索算法，将工作流划分成子工作流，最小化数据依赖性，以此提高后续任务调度的效率。

2) 对CS搜索算法进行改进，提出了一种适用于工作流划分的变异和交叉机制，避免算法陷入局部最优，提高划分能力。

3) 采用决策树来分类和选择候选资源，根据任务截止期限约束和排队时间来分配相应的资源。

2　工作流任务调度方案

本文将每个工作流建模为一个有向无环图(directed acyclic graph，DAG)[9]，其中节点表示工作流的任务，有向边表示任务之间的数据依赖关系。本文中，将无任何父代任务的任务称为入口任务，无任何子任务的任务称为出口任务。

该工作流调度方案主要分成3个阶段：1) 分配截止期限；2) 工作流划分；3) 资源配置。

2.1　分配截止期限

每个工作流具有明确的截止期限，工作流截止期限的分配包含每个子任务的子截止期限的分配。本文基于空闲时间计算子截止期限[10]，工作流的空闲时间是工作流在满足截止期限下，其关键路径的可增加时间：

$$\mathrm{SLT}(w)=\mathrm{DL}(w)-\mathrm{CPT}(w) \tag{1}$$

式中，DL(w) 为工作流的截止期限；CPT(w) 为工作流的关键路径。每个任务的子截止期限为：

$$\mathrm{DL}_{n_i}=\mathrm{ls}_{n_i}+\mathrm{rt}_{n_i}+\mathrm{SLT}(l_{n_i}) \tag{2}$$

式中，ls_{n_i} 为任务的最迟开始时间；rt_{n_i} 为任务执行时间；$\mathrm{SLT}(l_{n_i})$ 为该级别的空闲时间。

2.2　工作流划分

2.2.1　问题描述

对工作流进行划分是本文的主要创新之一，本文将每个工作流视作为DAG，所以划分工作流问题可视为图划分问题。图划分的目的是将DAG分割到近似大小的不相交子集中，使在不同子集间端点边的相关性最小，即划分工作流为子工作流，使它们之间数据传输总量最少，如图1所示。

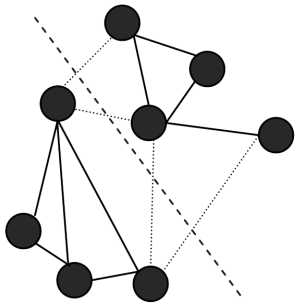

图1　工作流划分为两个子工作流的案例

由于划分子工作流为一个NP困难问题，所以可采用元启发式算法来求解。目前出现一种新兴的启发式智能算法：布谷鸟搜索(CS)算法。CS算法是一种通过模拟布谷鸟寄生育雏行为来求解最优化问题的算法。相对于其他智能算法，CS算法参数较少，可以大大削弱参数对实验效果的影响。同时，CS算

法具有更好的搜索效率和较少的求解次数[11]。另外，CS中的Levy飞行是模仿自然界的一种随机移动方式。相对于高斯和统一分布，Levy飞行更遵循自然规律，当解集较大时，Levy飞行的优势更大，适用于包含大量子任务的工作流划分情况。为此，本文对传统CS算法进行改进，使其能够应用于工作流划分。

本文通过改进型CS算法最小化子工作流中的数据依赖性，基本思想是：首先，将工作流随机分割成多个大小相等的子工作流，形成初始鸟巢；其次，计算每个子工作流中的每个任务与相邻子工作流之间的通信需求量，并将高于平均通信量的任务移动到该相邻子工作流中，实现一种交叉过程，获得新鸟巢；最后，以子工作流之间的总通信量为适应度函数，通过CS算法反复迭代，寻找最优划分方案，使各子工作流之间的整体数据依赖性最小。

2.2.2 *传统CS算法*

CS算法初始时，一定数量的鸟巢位置随机落在目标函数的可行解空间中，迭代进化时，适应度值最优的鸟巢位置得以保留，鸟巢位置按照莱维机制进行更新得到新的鸟巢位置，在此过程中，有一定概率的鸟巢会被抛弃并另建新巢。

设 $Y_l^t = Y_{l1}^t, Y_{l2}^t, \cdots, Y_{lp}^t$ 为第l个鸟巢在第t次进化时的鸟巢位置，其中，$l = 1,2,\cdots,m$，p为优化问题的维数。每一次迭代中，鸟巢位置按下式更新：

$$Y_l^{t+1} = Y_l^t + \alpha \oplus \boldsymbol{L}(\lambda) \tag{3}$$

式中，α为步长调节因子，一般取值为0.01；符号$\oplus$为点对点乘法；$\boldsymbol{L}(\lambda)$为服从莱维分布的随机搜索向量。

2.2.3 *改进CS算法*

传统CS算法受随机行走策略影响较大，存在后期收敛速度慢、会陷入局部极小等不足[12]。因此，本文提出一种变异机制来提高其全局搜索的能力。该变异机制采用了类似差分进化方法的简化算法和一个惯性权重w，来提高种群的多样性，从而降低迭代次数，提高全局寻优的速度。

本文变异机制的基本思想是应用当前种群中鸟巢的差异来重组种群，在鸟巢上增加两个随机数加权的鸟巢差执行变异过程。这样，在迭代初期，种群中鸟巢差异较大，变异操作提高CS算法全局搜索能力。随着迭代过程的进行，算法趋于收敛，此时鸟巢之间的差异较小，变异操作则能够加强局部搜索能力。

在t次迭代时，随机选取$q(q<m)$个鸟巢$\{Y_j, j = 1,2,\cdots,q\}$，对于鸟巢$Y_j$，在所有鸟巢中随机选取4个与之不同的鸟巢$Y_{r1}, Y_{r2}, Y_{r3}, Y_{r4}$，用于构建新巢，其位置为：

$$\hat{Y}_j^t = wY_{r1}^t + \text{rand}(0,1)(Y_{r2}^t - Y_{r3}^t) + \text{rand}(0,1)(Y_{\text{best}}^t - Y_{r4}^t) \tag{4}$$

式中，rand(0,1)为[0,1]上服从均匀分布的随机数；Y_{best}^t为当前最优鸟巢位置；$\hat{Y}_j^t = (\hat{y}_{j1}^t, \hat{y}_{j2}^t, \cdots, \hat{y}_{jp}^t)$；$w$为惯性权重，决定新鸟巢对先前鸟巢的继承程度。

式(4)中，$\text{rand}(0,1)(Y_{r2}^t - Y_{r3}^t)$能使微粒具有飞向自身最优位置的趋向，$\text{rand}(0,1)(Y_{\text{best}}^t - Y_{r4}^t)$能使微粒具有飞向全局最优的趋向。这两半部分即为本文的变异机制，变异机制能够提高CS算法的搜索能力。另外，wY_{r1}^t能够使微粒保持原有的飞行惯性，

惯性权重w的引入，可使CS算法有扩展搜索空间的趋势，有能力搜索新的区域。实验表明，较大的惯性权重w有利于跳出局部最优，进行全局寻优；较小的w值有利于局部寻优，加速算法收敛。为了平衡算法的全局和局部搜索能力，惯性权重w的值应随迭代次数的增加而递减。然而本文工作流划分搜索过程是非线性的，由于加入了变异机制，所以惯性权重线性递减策略不能反映实际的优化搜索过程。因此，本文设定惯性权重根据当前迭代次数(iter)非线性递减，有：

$$w = \left(\frac{2}{\text{iter}}\right)^{0.3} \tag{5}$$

位置更新后，会随机生成一个随机数r=[0,1]，与鸟窝的主人发现外来鸟的概率P_a对比，若$r > P_a$，则对Y_l^t进行随机改变，否则不变。本文设定概率P_a=0.25。

然后，对选取的种群$\{Y_j\}$及其变异种群$\{\hat{Y}_j\}$进行个体间交叉操作，传统CS算法是根据交叉率进行随机交叉，但这不适合本文工作流划分。因为工作流中的每个任务存在关联性，所以本文提出一种新型交叉操作，将最高移动增益的任务进行交叉，交叉过程描述为：计算一个子工作流w_i的任务t与相邻子工作流w_j之间的移动增益(t, w_j)；选择具有最高数据移动增益的任务t和t'，选择目标子工作流w_m和w_n，移动任务t到子工作流w_m，移动任务t'到子工作流w_n，并更新$W = (w_1, w_2, \cdots, w_k)$。获得最终的新鸟巢$\hat{Y}_j^{t+1}$。

最后，利用式(6)通过适应度值评估新鸟巢是否优于Y_l^t：

$$Y_j^{t+1}=\begin{cases}\hat{Y}_j^{t+1} & f(\hat{Y}_j^{t+1})<f(Y_j^t)\\ Y_j^t & \text{其他}\end{cases} \tag{6}$$

本文中，适应度函数f为各子工作流的端点边数目，基于改进CS算法划分工作流的步骤如下。

1) 初始化种群，本文设置种群规模为20，最大迭代次数为200。将工作流w随机划分成k=50个子工作流$W'=(w_1',w_2',\cdots,w_k')$，每种子工作流划分方案作为一个鸟巢位置。

2) 计算每个鸟巢位置的适应度值，保留最优者，并随机选取一定数量的鸟巢位置构建备选鸟巢位置，若备选鸟巢位置的适应度值优于原鸟巢位置，则保留较优者。

3) 执行改进型CS算法，得到最优适应度的鸟巢，若当前值大于保存的全局最优值，则更新全局最优值。

4) 重复步骤2)、步骤3)，直到满足搜索精度或达到最大搜索次数，输出最优划分方案。

2.3 资源分配

资源分配阶段中，主要分为4个部分：1) 客户端节点计算任务排名；2) 决策树分类资源；3) 报告节点选择候选资源；4) 根据截止期限分配任务到相应资源。

2.3.1 计算任务排名

在将客户端节点上的工作流划分为子工作流后，计算工作流中每个任务的向上排名：

$$\begin{cases}k_{n_i}=C_{n_i}+\max\limits_{n_j\in\text{children}(n_i)}(t_{ij}+k_{n_j})\\ k_{n_{\text{exit}}}=C_{n_{\text{exit}}}\end{cases} \tag{7}$$

式中，C_n为任务n_i的执行时间估计；t_{ij}为两个任务n_i和n_j之间的平均通信时间，根据平均网络带宽计算平均通信时间。

2.3.2 决策树分类资源

本文使用文献[13]提出的一种基于P2P的决策树对资源进行分类，选择报告节点。决策树使用5个属性值：CPU速度、RAM数、可用硬盘空间、操作系统、处理器模型来对资源进行分类。所以，决策树对应于每个资源有5个属性，每个属性有4个等级，因此，资源由决策树分为4^5=1 024个聚类。决策树架构中有3种节点类型：报告节点、主机节点和客户端节点。客户端节点导入工作流到系统中，并把它们发送到其中一个活动主节点进行调度；主机节点用来分配工作流和资源配置；报告节点收到请求后，从主机节点寻找合适的资源，并广播符合该请求QoS约束的资源。

2.3.3 选择候选资源

决策树中所选择的报告节点基于QoS约束来广播大量志愿资源，广播的资源数量等于子工作流数。按照资源CPU速度的权重，将资源的标识符进行排序并返回到注入节点。至此，注入节点具有一个能执行子工作流的资源集，可选择一些志愿资源作为候选。注入节点根据CPU速度、距客户端节点最小通信延迟来选择候选资源，找出等于子工作流数量的候选资源数。通信延迟根据基于排队理论的网络模型计算，有：

$$R=2S_p+\left(\frac{\sigma_p^2\lambda_p^2}{2((S_p^{-1})-\lambda_p)}\right)+\sum_{i=s+1}^{d}S_{P_i} \tag{8}$$

式中，

$$S_p=0.5\alpha_{\text{net}}+F\beta_{\text{net}} \tag{9}$$

式中，S_p为两个资源之间每个连接的服务时间；α_{net}、β_{net}分别为路由算法中两个相邻资源之间链路上的网络延迟和带宽倒数；F为两个资源之间的数据量；σ_p^2、λ_p^2分别为源对等队列中流量的方差和中间到达率。

2.3.4 资源分配

在资源分配过程中，注入节点根据各任务在各资源上的估计执行时间和任务截止时间来分配资源。每个子工作流的估计排队延迟会从候选资源发回到注入节点，注入节点运用算法，在志愿资源上寻找具有较低概率满足子截止期限的任务，该概率基于估计的排队和通信延迟来计算。然后，将这些低概率的任务在云资源上重新调度，并将其他任务正常派遣到志愿资源上。

算法：资源配置算法

输入：候选资源运行子工作流的排队和通信延迟(CANDID-RES)；

输出：错过截止期限的任务集(MISSDEAD-TASKS)。

1) For 每个子工作流s上任务j do
2) QueueDelay ← QueueDelay(CANDID – RES(s))
3) RequireExeTime ← Execution Time()
4) If (QueueDelay + RequiredExeTime > Sub_Deadline(j))
MISSDEAD – TASKS.add(j)
5) MaxQueueDelay ← 0
6) For 每个子工作流t上任务j的父代任

务P do

7) If (QueueDelay(CANDID – RES(t) > MaxQueueDelay))

MaxQueueDelay = QueueDelay(CANDID – RES(t))

8) End

9) If (MaxQueueDelay + RequiredExeTime > Sub_Deadline(j)) and notFoundOn(MISSDEAD – TASK, j)

10) MISSDEAD – TASK.add(j)

11) End

12) If (isEmpty(MISSDEAD – TASKS)) Return (NULL)

13) For 每个 MISSDEAD – TASKS(i) 的子任务 do

14) If (notFoundOn(MISSDEAD – TASK,c)) MISSDEAD – TASK.add(c)

15) End

算法中，注入节点首先在每个子工作流对任务进行搜索，找出执行其所需时间(所分配的志愿资源的排队延迟+执行时间)大于其子截止期限的任务，并将该任务添加到错过截止期限的任务中(算法步骤1)～4))。然后，由于每个任务须在工作流中父代任务之后运行，需要寻找子工作流内每个任务的父代任务最大排队延迟(算法步骤5)～8))。如果父代任务的最大排队延迟和该任务执行时间的总和大于其子截止期限，则将该任务添加到错过截止期限的任务中。如果该任务之前被添加过，则不会再添加(算法步骤9)～10))。如果错过截止期限的任务集为空，则算法返回空(算法步骤12))，否则，从所有子工作流中寻找每个错过截止期限任务的子任务，并将子任务添加到错过截止期限任务中，如果它们之前添加过(算法步骤13)～15))，则注入节点调整子工作流，并从子工作流移除错过截止期限的任务(算法步骤16))。

如果算法的输出的错过截止期限任务集合为空，则注入节点发送子工作流到候选资源，使其能够在志愿资源上运行并满足截止期限。将这些志愿资源称作运行节点，运行节点中，每个子工作流的任务按照 k_{n_i} 进行降序排列，并保存在优先运行队列中。如果算法的输出不为空，则注入节点发送调整过的子工作流到候选资源，并发送错过截止期限任务到公共云资源。

3 实验及分析

3.1 实验设置

使用可扩展的模拟工具CloudSim来模拟和评估结果，CloudSim被广泛用于构建和仿真云计算环境及评估云计算资源分配策略和调度算法的性能。在具备酷睿i5CPU、4 GB内存和Windows7系统的PC机平台上进行实验。

实验中，在CloudSim中构建10个不同的计算资源(虚拟机服务器)，资源的每秒执行百万指令数范围为[10, 150] MIPS，内存范围为[128, 1 024] MB，带宽范围为[10, 100] Mb/s。设定工作流的任务数量为20、40、60、80、100个不等的5种场景，任务计算量为[50, 200] MIPS。对于数据密集型工作流，设定工作流任务传输时间在[10, 200] s内随机生成，执行时间在[1, 20] s内，即传输时间约为计算时间的20倍。对于计算密集型工作流，设置任务执行时间在[10,200] s内，工作流任务传输时间在[1,20] s内，即计算时间约为传输时间的20倍。对于本文CS算法，设置种群规模为20，最大迭代次数为200，工作流划分成子工作流数k=50。

3.2 算法收敛性实验

首先，通过实验验证本文改进型CS算法求解问题的收敛性。设置具有100个任务的实验场景，分别执行传统CS算法和本文改进型CS算法，记录各迭代次数下的适应度函数值，实验结果如图2所示。可以看出，本文改进型CS算法的收敛速度明显比传统CS算法快，大约在150次迭代后，本文改进型CS算法几乎收敛到最优，而传统CS至少需要250次。另外，本文改进算法获得的解更优秀。这是因为，本文融入变异机制和非线性递减惯性权重，加强了种群的多样性和算法搜索能力，提高了搜索速度。

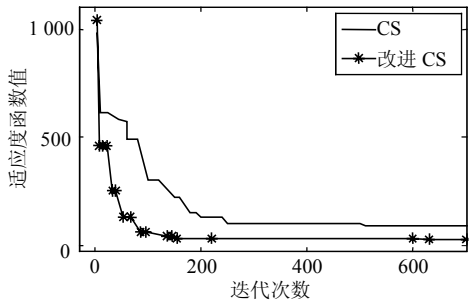

图2 本文改进型CS算法的收敛性分析

3.3 模拟工作流实验

在数据和计算密集型工作流中，将本文方法与SHEFT和Aneka方案在工作流完成时间上进行比较。

本文方案与上述两种方案的主要区别在于：1) 本文方案对工作流进行了划分；2) SHEFT方案以给定的截止期限进行分配，没有具体估算任务的执行时间，Aneka方案没有考虑任务的队列等待时间，而本文方案在以截止期限为约束下，考虑了任务的执行、通信和排队时间。

分别进行10次实验，记录工作流的完成时间并取平均值，结果如图3和图4所示。从图3和图4可以看出，完成时间随着任务数的增加而增加，这是因为完成时间主要依赖于可用资源总量，当任务规模较小时，各种调度方案的完成时间差异较小；当任务规模逐步增加时，不同方案的性能差异开始增大。其中，本文方案对工作流进行划分，减少了任务之间的通信，并合理分配资源，获得了最短的相对完成时间。当任务数量为20、40、60、80、100时，数据密集型工作流平均完成时间分别为：55、87、125、164和193 s，计算密集型工作流平均完成时间分别为：99、165、225、275和358 s。结果表明，对于数据和计算密集型工作流，本文方案在完成时间上具有明显优势。

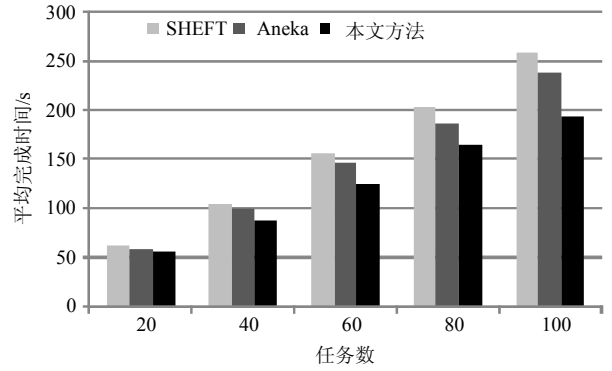

图3　不同任务数量下，数据密集型工作流完成时间相对值

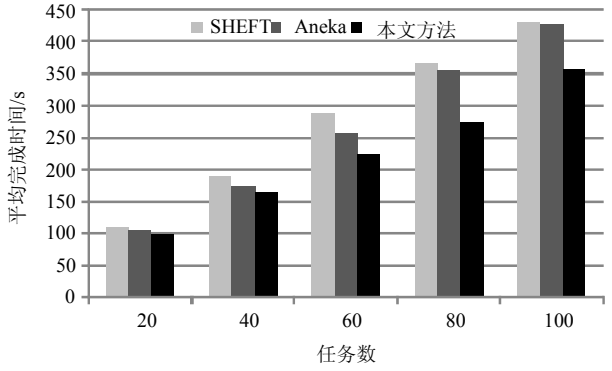

图4　不同任务数量下，计算密集型工作流完成时间相对值

3.4　实际工作流实验

为了评估数据密集型工作流调度方案的实际性能，使用CyberShake和Montage工作流为测试对象，CyberShake工作流是美国宇航局拼接太空图片的任务，是一种数据密集型工作流；Montage工作流是美国地震中心从表征区域中提取地震灾害特征的任务，是一种计算密集型工作流。

计算工作流的每个任务并分配子截止期限，以保证任务的顺利完成。为了评估截止期限对任务完成率的影响，本文实验中为所有任务设置7种不同的截止期限(2、5、10、15、20、25、30)，执行具有100个任务的数据流，分别执行10次，获得满足截止期限的平均工作流完成率，结果如图5和图6所示。

图5和图6中可以看出，当截止期限较小时，多个工作流同时调度，对资源竞争强度较大，3种方案在截止期限内的完成率都比较低，但本文方案仍然较高。另外由于Montage工作流为计算密集型，任务所需的计算时间较大，所以整体完成率比CyberShake工作流要低。随着截止期限的不断放大，各种方案的完成率也逐渐上升且差距也越大，当截止期限为30时，两种工作流中，本方案完成率分别达到98%和87%，高于SHEFT方案约19%，这是因为对工作流进行划分，根据截止期限对子任务进行排序并分配到相应资源进行运行，提高了任务的执行效率。

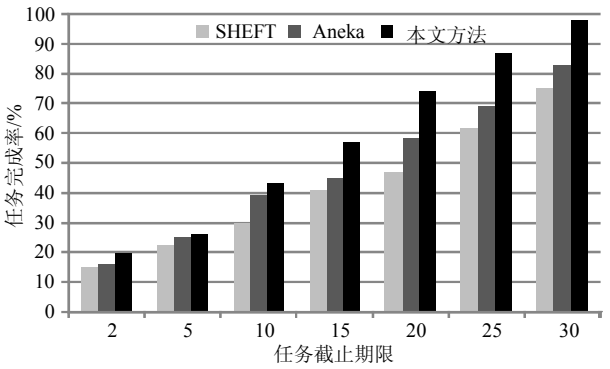

图5　不同截止期限下的CyberShake工作流完成率

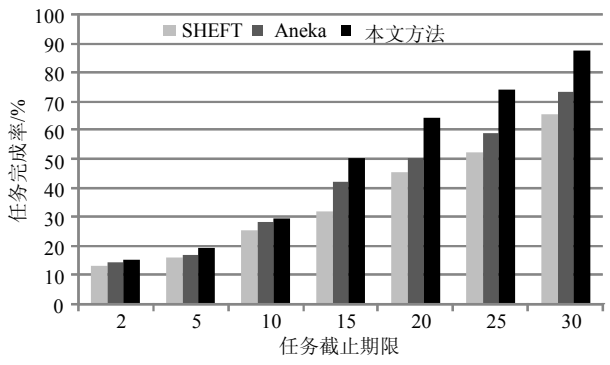

图6　不同截止期限下的Montage工作流完成率

4 结 束 语

本文提出一种结合CS算法和决策树的工作流任务调度方案。为工作流中每个任务分配截止期限，利用CS算法将工作流分割成多个子工作流，利用决策树选择合适资源，根据截止期限约束配置相应的资源。在数据和计算密集型工作流上进行实验，与SHEFT和Aneka方案相比，本文方案具有较短的总运行时间和较高的任务完成率。

未来研究中，将考虑以亚马逊现货实例为对象进行实际实验。

感谢四川理工学院高性能科学与工程计算中心提供测试平台和仿真模拟计算支持。

参 考 文 献

[1] 张鹏, 王桂玲, 徐学辉. 云计算环境下适于工作流的数据布局方法[J]. 计算机研究与发展, 2013, 50(3): 636-647.
ZHANG Peng, WANG Gui-ling, XU Xue-hui. A data placement approach for workflow in cloud[J]. Journal of Computer Research and Development, 2013, 50(3): 636-647.

[2] 刘少伟, 孔令梅, 任开军, 等. 云环境下优化科学工作流执行性能的两阶段数据放置与任务调度策略[J]. 计算机学报, 2011, 34(11): 2121-2130.
LIU Shao-Wei, KONG Ling-Mei, REN Kai-jun, et al. A two-step data placement and task scheduling strategy for optimizing scientific workflow performance on cloud computing platform[J]. Chinese Journal of Computers, 2011, 34(11): 2121-2130.

[3] LI W, WU J, ZHANG Q, et al. Trust-driven and QoS demand clustering analysis based cloud workflow scheduling strategies[J]. Cluster Computing, 2014, 17(3): 1-18.

[4] CANON L C, JEANNOT E. Evaluation and optimization of the robustness of DAG schedules in heterogeneous environments[J]. IEEE Transactions on Parallel & Distributed Systems, 2010, 21(4): 532-546.

[5] MAO Y, ZHU L, CHEN X, et al. Associate task scheduling algorithm based on delay-bound constraint in cloud computing[C]//2012 13th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES). [S.l.]: IEEE, 2012: 92-96.

[6] CALHEIROS R N, VECCHIOLA C, KARUNAMOORTHY D, et al. The aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds[J]. Future Generation Computer Systems, 2012, 28(6): 861-870.

[7] AHMAD S G, LIEW C S, RAFIQUE M M, et al. Data-intensive workflow optimization based on application task graph partitioning in heterogeneous computing systems[C]//2014 IEEE Fourth International Conference on Big Data and Cloud Computing (BdCloud). [S.l.]: IEEE, 2014: 129-136.

[8] MALAWSKI M, JUVE G, DEELMAN E, et al. Algorithms for cost-and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds[C]//Proceedings of the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis. [S.l.]: IEEE Computer Society, 2012: 1-11.

[9] 田国忠, 肖创柏, 谢军奇. 有期限约束的多DAG共享资源的调度及公平费用优化方法[J]. 计算机学报, 2014, 37(7): 1607-1619.
TIAN Guo-zhong, XIAO Chuang-bai, XIE Jun-qi. Scheduling and fair cost-optimizing methods for concurrent multiple DAGs with deadline sharing resources[J]. Chinese Journal of Computers, 2014, 37(7): 1607-1619.

[10] CHEN W, DEELMAN E. Integration of workflow partitioning and resource provisioning[C]//IEEE/ACM International Symposium on Cluster, Cloud & Grid Computing. [S.l.]: IEEE Computer Society, 2012: 764-768.

[11] LI Xiang-tao, YIN Ming-hao. A hybrid cuckoo search via Lévy flights for the permutation flow shop scheduling problem[J]. International Journal of Production Research, 2013, 51(16): 4732-4754.

[12] LARUMBE F, SANSO B. A Cuckoo search algorithm for the location of data centers and software components in green cloud computing networks[J]. Transactions on Cloud Computing IEEE, 2013, 1(1): 22-35.

[13] GHAFARIAN T, DELDARI H, JAVADI B, et al. Cycloid grid: a proximity-aware P2P-based resource discovery architecture in volunteer computing systems[J]. Future Generation Computer Systems, 2013, 29(6): 1583-1595.

编 辑 漆 蓉