基于动态资源选择策略的微服务工作流调度算法

刘洋睿[1]，江凌云[1,2+]

（1. 南京邮电大学 通信与信息工程学院，江苏 南京 210003；
2. 南京邮电大学 物联网研究院，江苏 南京 210003）

摘 要：为解决现有云中工作流调度算法在面对大量微服务任务组成的工作流时出现整体调度成本偏高的问题，提出一种基于动态资源选择策略（dynamic resource selection strategy，DRSS）的微服务工作流调度算法——DRSS 调度算法。利用任务在工作流中的位置确定任务的子截止期以及调度优先级，采用动态资源选择策略对任务进行调度，获得任务执行的最优资源，在此基础上更新任务状态以及虚拟机实例的资源向量。实验结果表明，该算法在调度成功率与成本方面上较同类算法更优。

关键词：云环境；工作流；调度算法；微服务；成本优化；动态资源选择策略；虚拟机实例
中图法分类号：TP302 **文献标识号**：A **文章编号**：1000-7024（2023）05-1313-07
doi：10.16208/j.issn1000-7024.2023.05.005

Microservice workflow scheduling algorithm based on dynamic resource selection strategy

LIU Yang-rui[1], JIANG Ling-yun[1,2+]

(1. School of Communication and Information Engineering, Nanjing University of Posts and Telecommunications, Nanjing 210003, China; 2. Internet of Things Research Institute, Nanjing University of Posts and Telecommunications, Nanjing 210003, China)

Abstract: To solve the problem of high overall scheduling cost of existing cloud-based workflow scheduling algorithms when dealing with workflows consisting of a large number of microservice tasks, a microservice workflow scheduling algorithm based on dynamic resource selection policy, DRSS scheduling algorithm, was proposed. The sub-deadlines and scheduling priorities of tasks were determined using their positions in the workflow, and the tasks were scheduled using the dynamic resource selection policy to obtain the optimal resources for task execution, and the resource vectors of task states and virtual machine instances were updated on this basis. Experimental results show that the proposed algorithm outperforms similar algorithms in terms of scheduling success rate and cost.

Key words: cloud environments; workflows; scheduling algorithms; microservices; cost optimization; dynamic resource selection strategies; virtual machine instances

0 引 言

在过去几年中，随着互联网用户数量的激增，传统的单片式应用已经无法应对日益增长的用户流量，越来越多的研究者开始将目光转向微服务架构[1]。如今，商业或者科学应用大多都包含着大量的计算任务，这些应用往往都以工作流的形式表示。对于工作流来说，将工作流中的任务调度到相应的资源上执行总是不可或缺的一个环节。

相较于传统工作流中的任务，微服务任务要小得多，所需要的计算资源也少得多，这意味着如果按照传统云平台的工作流调度方案，一对一地将微服务任务分配到虚拟机实例上执行，将造成大量的资源浪费与非必要的成本。轻量级虚拟机技术容器的出现正好解决了这一问题，容器具有轻量级、快速启动、良好的资源隔离和快速部署等特

收稿日期：2022-04-18；**修订日期**：2023-04-13
基金项目：江苏省重点研发基金项目（BE2020084-4）
作者简介：刘洋睿（2000-），男，江西上饶人，硕士研究生，研究方向为工作流调度；+通讯作者：江凌云（1971-），女，安徽安庆人，硕士，副教授，研究方向为下一代网络。E-mail：ql2816149426@163.com

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

点，非常适合于微服务任务的部署。将微服务任务部署到容器中，再将装有任务的容器部署到虚拟机上，则可以实现多个微服务任务在同一台虚拟机实例上执行，有效地利用剩余资源。

由于不同的微服务任务有着不同的资源需求，不同的虚拟机实例提供资源的能力也是不一样的，所以如何设计一个既能满足微服务架构下任务与资源多对一关系，又能尽可能地降低调度总成本的算法是本文研究的重点。

1 相关工作

工作流调度问题是一个 NP-hard 问题，在目前的研究中，许多 QoS（quality of service）指标常常被视为问题的优化目标和约束条件，如成本、时间、能量消耗、安全，本文研究的是截止期约束下优化成本的工作流调度问题。由于 NP-hard 问题无法在可接受的计算时间范围内找到大规模问题实例的最优解。所以，目前的研究通常采取启发式算法来解决这一问题。

蚁群算法[2]、遗传算法[3]、粒子群算法[4]是求解约束问题常用的元启发式算法。文献［5］设计了一种基于历史数据的任务估计模型用于体现工作流中任务执行时间的不确定性，并提出了一种基于自适应蚁群算法的工作流调度算法。文献［6］提出了免疫粒子群算法，该算法克服了粒子群算法收敛速度慢、易陷入局部最优的问题，有效地提高了优化的质量和速度。文献［7］同时考虑任务的最合适虚拟机类型集合、最佳调度序列、虚拟机实例上的空闲时隙 3 个方面，提出了结合空闲时隙感知规则的粒子群优化算法，并且提出了一种修复方法来修复由于经典粒子群算法的随机性导致的任务优先级无效的问题。除了 3 种常用的元启发式算法，蛙跳算法[8]、基于生物-地理的优化算法[9]等元启发式算法也被应用于解决工作流的调度问题。

基于其它启发式规则的调度算法也有很多，异构最早完成时间算法 HEFT 是应用最广泛的启发式调度算法之一。该算法最初是为了解决有限数量的异构处理器中的工作流调度问题而提出的，现在已经有许多改进版本的 HEFT 算法[10,11]被应用于云中工作流调度。Abrishami 等引入部分关键路径 PCP 的概念，提出了两种算法，分别为云中部分关键路径算法 IC-PCP 和结合截止期分配的云中部分关键路径算法 IC-PCPD2[12]，IC-PCP 算法通过在最便宜的虚拟机实例上调度整条部分关键路径上的任务来得到最小的工作流调度成本。Arabneja 等为了解决在动态供应的资源上具有截止期约束的工作流调度问题，提出了比例截止日期约束算法和截止日期约束关键路径算法。然而以上的文献都是基于传统云平台的调度算法，不适用于本文所研究的微服务架构下的工作流调度问题。

针对微服务架构下工作流调度问题的算法也有一些，文献［13］考虑到了容器层面，建立起以容器和虚拟机两层资源结构为资源模型的调度问题，并提出了一种融合了任务调度和自动扩展的弹性调度算法 ESMS（elastic scheduling for microservices）用于解决问题。文献［14］考虑了不同的虚拟机资源的类型、价格，用整数规划方法建立了调度问题的数学模型，并提出了 SMWC（scheduling microservice-based workflow to CaaS）算法用于解决该问题。

SMWC 算法在对任务进行调度时采用的策略为基于任务需求向量与虚拟机实例向量之间的相似度的策略。该策略存在一些缺陷，单纯的以向量相似度作为分配策略会导致调度后半段选择的虚拟机实例的单价偏高。为了尽可能降低成本，对文献［12］中基于成本的策略进行研究，发现在使用该策略的情况下调度的总成本仍达不到较好的效果。本文分析了单纯基于相似度策略与单纯基于成本的策略的缺陷，提出了一种动态资源选择策略，该策略能够在整个调度过程中都为任务选择更加合适的虚拟机实例，从而到达降低成本的目的。

2 问题模型与定义

2.1 工作流模型

工作流模型可以用一个有向无环图 DAG（directed acyclic graph）来表示，DAG＝(T,E)。有关 DAG 的基础定义请参见文献［15］，如果任务之间存在数据传输，则需要在边 e_{ij} 上加上 $data_{i,j}$ 属性，表示任务之间需要传输的数据量。此时，对任务 t_i 而言，只有当 t_i 的直接前驱任务集合 $pred(t_i)$ 中所有的任务都执行完，且数据都被传输完之后，t_i 才可以开始执行。所有微服务任务都事先给定执行任务中所需要的计算资源，用需求资源向量 $\overrightarrow{R_i^r}=(R_{i,1}^r,\cdots,R_{i,m}^r)$ 表示，其中 m 是计算资源种类的数目。

2.2 资源模型

本文采用亚马逊弹性容器服务（Amazon elastic container service，Amazon ECS）作为资源模型。弹性容器实例是无服务器化、容器化的弹性计算服务，即用户无需管理底层服务器，只需要提供打包好的镜像，即可运行容器，用户可以从云平台上租赁虚拟机用于编排这些容器实例。虚拟机实例的计费方式为按计费时间间隔即用即付，一个计费时间间隔通常为一小时，任何未满一小时的实例使用都按一小时计费。资源池由 R 表示，所有虚拟机实例的集合用 $R=\{v_1,v_2,\cdots v_p\}$ 表示。本文考虑异构资源场景，虚拟机的配置由内存、CPU 核数、存储、带宽等决定。虚拟机实例 v_j 具有的总资源向量由 $\overrightarrow{R_j^A}=(R_{j,1}^A,\cdots,R_{j,m}^A)$ 表示。

2.3 相关定义

t_i 的直接前驱任务集合用 $pred(t_i)$ 表示，t_i 的直接后继任务集合用 $succ(t_i)$ 表示。有的 DAG 可能存在多个入口节点和出口节点，为了使模型更加一般化，在工作流的开

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

始和末尾分别添加两个资源需求向量为零向量的虚拟任务节点 t_{entry} 和 t_{exit}。图 1 展示了一个微服务工作流的样例。

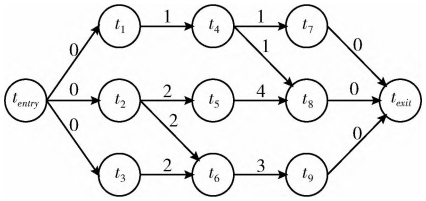

图 1　工作流示例

本文使用 $TT(t_i,t_k)$ 来表示任务 t_i 与任务 t_k 间的数据传输时间，定义如下

$$TT(t_i,t_k)=\begin{cases} data_{i,k}/BW, & v_i \neq v_k \\ 0, & v_i = v_k \end{cases} \tag{1}$$

即两个任务如果被分配到同一个虚拟机实例上执行，则这两个任务之间的数据传输时间为 0。否则，数据传输时间为任务 t_i 与任务 t_k 间需要传输的数据量 $data_{i,j}$ 和数据中心的平均带宽 BW 的比值。令 $ET(t_i,v_j)$ 表示任务 t_i 在虚拟机实例 v_j 上的执行时间，任务 t_i 的最早开始时间 $EST(t_i)$ 可以由下式定义

$$EST(t_i)=\begin{cases} 0, & t_i = t_{entry} \\ \max\limits_{t_k \in pred(t_i)} \{EST(t_k)+ET(t_k,v_j)+TT(t_i,t_k)\}, & otherwise \end{cases} \tag{2}$$

在初始化时，由于无法事先得到任务的实际执行时间，所以采用任务在所有虚拟机实例上的最短执行时间来替代 $ET(t_k,v_j)$ 进行初始计算，后续环节的初始化计算也采用此规则。

2.4　优化目标以及约束

本文假设微服务工作流的拓扑结构和任务所需的计算资源事先给定，并且在运行过程中不发生变化，且微服务任务在容器实例上执行时不能被抢占，即执行过程中任务与容器实例可以被视为一个整体。

由于同一时刻多个执行任务的容器可能被部署在同一台虚拟机实例上，所以无法从任务执行时间来计算总成本，应该从虚拟机实例的实际使用时间来进行计算。单个虚拟机实例 v_j 的执行成本 c_j 定义如下

$$c_j=\left\lceil \frac{UT_j}{interval} \right\rceil \times price_j \tag{3}$$

其中，UT_j 指的是虚拟机实例 v_j 的使用时间，$interval$ 指的是计费时间间隔，一般为 1 h，$price_j$ 指的是虚拟机实例 v_j 的单价。

所以，总成本可以定义为

$$Cost=\sum_{v_j \in R} c_j \tag{4}$$

工作流的实际完成时间定义为调度长度 $Makespan$，用户给定的截止期为 D。对于 DAG，由于 t_{exit} 的执行时间为 0，所以 t_{exit} 的实际完成时间为工作流的调度长度，即 $Makespan = FT(t_{exit})$，$FT(t_{exit})$ 为出口任务 t_{exit} 的完成时间。

综上，优化目标以及约束如下

$$\begin{aligned} &min\ Cost \\ &s.t.\ Makespan \leqslant D \end{aligned} \tag{5}$$

3　算法设计

3.1　子截止期划分

首先引入概念，任务 t_i 的升秩 $rank_u(t_i)$，该值表示任务 t_i 到任务 t_{exit} 之间的关键路径的长度，定义如下

$$rank_u(t_i)=ET(t_i,v_j)+\max_{t_k \in succ(t_i)}(TT(t_i,t_k)+rank_u(t_k)) \tag{6}$$

对于出口任务 t_{exit}，$rank_u(t_{exit})=0$。通过这个值，可以递归的求出每一个任务的升秩。

在定义升秩这个概念后，任务 t_i 的子截止期 sd_i 的定义为

$$sd_i = D \times \frac{rank_u(t_{entry})-rank_u(t_i)+ET(t_i,v_j)}{rank_u(t_{entry})} \tag{7}$$

式（7）使用了任务 t_i 与任务 t_{entry} 的升秩之间的比例来对截止期进行划分，升秩这个概念反映出了任务 t_i 所处于整个工作流的位置以及任务大小。任务距离入口任务越远，$rank_u(t_{entry})-rank_u(t_i)$ 的值就会越大，任务计算量越大，$ET(t_i,v_j)$ 就会越大。这两种情况都会导致子截止期的延长，正符合预期的逻辑，所以采取这种方法分配子截止期是较为合理的。

3.2　就绪任务排序

就绪任务的定义为：当一个任务的所有前驱任务都已经被分配到虚拟机实例上执行完毕，且所有数据都已经接收，该任务就被认为是就绪任务。任务一旦成为就绪任务，就会被放入到就绪队列。就绪队列是一个优先队列，本文使用紧急度这个概念来确定任务的优先级，定义如下

$$u_i=\frac{sd_i-EST(t_i)-ET(t_i,v_j)}{ut(t_i)} \tag{8}$$

其中，$ut(t_i)$ 表示任务 t_i 到出口任务 t_{exit} 之间的未调度的任务数。sd_i 与 $EST(t_i)+ET(t_i,v_j)$ 之间的距离越小，则表明该任务超过子截止期的风险越大。任务 t_i 之后的待调度任务越多，t_i 的完成时间超出子截止期后导致的延迟对后续的任务影响就会越大。因此，u_i 值越小，t_i 的调度紧迫性越高。

3.3　可用资源搜索

在完成任务子截止期的分配和就绪任务排序之后，就可以开始对就绪任务队列中优先级最高的任务进行可用资源搜索。合适的虚拟机实例需要满足：在容器生命周期内，虚拟机的可用空闲资源一直都满足任务的计算需求，并且任务的完成时间在子截止期之前，即满足

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

$$\overrightarrow{R_j^a}(t)=\overrightarrow{R_j^A}-\overrightarrow{R_j^u}(t)\geqslant\vec{0} \tag{9}$$

$$\overrightarrow{R_j^a}(t)-\overrightarrow{R_i^r}(t)\geqslant\vec{0} \tag{10}$$

$$EST(t_{cur})+ET(t_{cur},v_j)\leqslant sd_{cur} \tag{11}$$

其中，$\overrightarrow{R_j^a}(t)$ 代表虚拟机实例在 t 时刻的可用资源向量，$\overrightarrow{R_j^u}(t)$ 代表了虚拟机实例在 t 时刻的已用资源向量。式（9）代表虚拟机在任意时刻可用资源向量必须大于 0。式（10）表示在 t 时刻虚拟机的可用资源向量必须大于待分配任务的资源需求向量。由于本文默认了任务执行不会中断，所以只需要在任务分配开始时刻满足式（10）即可。遍历资源池 R 中的虚拟机实例，根据式（9）～式（11）去寻找可用的虚拟机实例，并添加至可用虚拟机实例集合 ψ 中。

3.4 动态资源选择策略

在完成可用资源搜索之后，如果 t_{cur} 的可用资源集合 ψ 中含有多个虚拟机实例供选择，此时则需要根据一些启发式的规则来选择最合适的虚拟机实例。

文献［12］中的算法在为任务选择虚拟机实例时采用的是为当前任务选择最便宜的虚拟机实例的策略。这种基于成本的选择策略仅仅考虑了每次选择的虚拟机的成本，忽略了任务异构的资源需求会导致虚拟机实例剩余资源不够均衡的情况，导致在选择时频繁选择新的虚拟机实例，造成成本的增加。

文献［14］在为任务选择虚拟机实例时采用的是选取可用资源向量与当前任务资源需求向量之间的余弦相似度最大的虚拟机实例。余弦相似度定义如下

$$CS(i,j,t)=\frac{\sum_{g=1}^{m}R_{j,g}^a(t)\times R_{i,g}^r}{\sqrt{\sum_{g=1}^{m}R_{j,g}^a(t)^2}\times\sqrt{\sum_{g=1}^{m}R_{i,g}^{r\,2}}} \tag{12}$$

余弦相似度越大，则说明两个向量越相似，意味着任务如果在这台虚拟机实例上执行，剩余的资源不会被过度消耗。基于余弦相似度的选择策略考虑到了基于成本选择策略的不足，在选择时考虑虚拟机实例内部剩余的资源情况，使得后续分配更加具有灵活性。但是这种方法过度地考虑了资源这一因素，而完全忽略了成本因素。随着任务分配过程的进行，后续剩余的未分配任务量越来越少，继续以余弦相似度这个指标进行分配导致的虚拟机数量减少所带来的成本降低远远不及选择高单价的虚拟机实例所带来的成本增加。

考虑到以上两种策略的缺陷，本文提出动态资源选择策略，定义虚拟机实例的质量 Q 如下

$$Q(i,j,t)=(1-\theta)\times CS(i,j,t)+\theta\times\frac{price_{max}-price_j}{price_{max}} \tag{13}$$

其中，$price_j$ 是虚拟机实例 v_j 的单价，$price_{max}$ 代表资源池最昂贵的虚拟机的单价。而 θ 是一个加权因子，计算公式如下

$$\theta=\frac{f}{n} \tag{14}$$

其中，f 代表已完成调度的任务数，n 代表有向无环图中的总任务数，用已调度任务与总任务数的比例来代表整个调度完成的程度。当处于调度前期时，θ 值较小，此时 Q 主要受到余弦相似度的影响，通过选取 Q 值大的虚拟机实例，就可以选取到余弦相似度大的虚拟机。此时后续还有较多待调度任务，选择余弦相似度较大的虚拟机就可以为后续的任务分配提供更大的灵活性，有助于减少虚拟机的使用数量。

随着调度过程的进行，当来到调度过程的后期时，θ 值较大，此时 Q 主要受到 $price_j/price_{max}$ 项的影响，通过选取 Q 值大的虚拟机实例，就可以选取到单价低的虚拟机实例。此时后续剩余的待调度任务数量较少，继续以余弦相似度进行分配带来的虚拟机数量减少带来的成本降低远远不及虚拟机的高单价所带来的成本增加。在此时，选取单价较低的虚拟机来完成任务，可以从虚拟机单价方面减少掉不必要的成本。

本文提出的动态虚拟机实例选择策略，依靠随调度进度变化的加权因子 θ，既在调度前期保证了后续任务分配的灵活性，也在调度后期降低了非必须的虚拟机单价成本，使得最终的总成本能够优于其它的选择方案。

3.5 在线调整

在任务调度环节中完成对就绪任务 t_i 的调度后，任务 t_i 的实际执行时间即可得到。由于任务 t_i 的实际执行时间会对其直接后继任务集合 $succ(t_i)$ 中的任务的最早开始时间产生影响，所以此时需要对 $succ(t_i)$ 集合中任务的各类时间参数进行更新。除此之外，还需遍历 $succ(t_i)$ 集合中的所有任务节点，检查集合中每一个任务节点 t_a 是否已经转变成就绪任务，即检查直接前驱任务集合 $pred(t_a)$ 中的所有任务是否已完成调度。若 t_a 已经成为就绪任务，则将其添加到就绪队列之中进行新一轮的排序。当任务分配到虚拟机实例上时，则需要将该虚拟机的可用资源向量 $\overrightarrow{R_j^a}(t)$ 更新成 $\overrightarrow{R_j^a}(t)-\overrightarrow{R_j^u}(t)$。任务执行完成时，任务所占用虚拟机的资源也会得到释放。

3.6 算法流程与时间复杂度

3.6.1 算法整体流程

算法 1：DRSS 调度算法的执行流程

输入：工作流 DAG，截止期 D

输出：调度方案列表 SL

(1)根据式（1）、式（2）、式（6）、式（7）计算出各个任务的 EST 参数与子截止期 sd

(2)将 t_{entry} 加入到就绪队列

(3)循环开始：当就绪队列不为空时

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

(4)使用式（8）计算就绪队列中所有任务的紧急度 u_i

(5)取出紧急度 u_i 最小的任务 t_{cur}，同时将该任务从就绪队列中移除，根据式（9）～式（11）找到可用的虚拟机实例集合 ψ

(6)如果 ψ 为空，则为该任务分配最贵的虚拟机实例。如果 ψ 不为空，根据式（12）～式（14），使用动态资源选择策略在 ψ 中为任务 t_{cur} 选择虚拟机实例 v_{select}，同时更新虚拟机实例 v_{select} 的可用资源向量

(7)将 $< t_{cur}, v_{select} >$ 添加到调度方案列表 SL 中

(8)检测正在执行的任务，若有任务已经完成，获取该任务的实际执行时间并更新该任务的后继任务的 EST 参数，同时更新对应虚拟机实例的可用资源向量，并将任务执行之后受影响而转变状态的就绪任务添加到就绪队列中

(9)循环结束

算法 1 给出了 DRSS 调度算法的整体执行流程。算法的输入是一个代表微服务工作流的 DAG 以及用户给出的截止期 D，输出为最终的调度方案列表 SL。算法首先为每个任务计算初始的 EST 和子截止期 sd 步骤（1）。为了使循环满足条件开始进行，首先将虚拟节点 t_{entry} 加入到就绪队列中步骤（2），接下来开始循环步骤（3），为当前就绪队列中的任务根据紧急度 u_i 进行排序步骤（4）。就绪队列采用的是优先队列，优先队列基于小根堆实现，每次添加元素时可以根据比较规则自动将队列最小的元素移动至队首，每次只需要取出队首元素，该元素即是 u_i 最小的任务。接下来为取出的任务寻找可用资源步骤（5），在得到可用资源集合 ψ 后，判断是否有可用资源步骤（6），如果存在可用资源，通过本文提出的动态资源选择策略，选择为任务选择 Q 值最大的虚拟机实例。如果不存在可用资源，说明此时无论选择哪种虚拟机实例都会超出任务的子截止期，此时需要选用最贵的虚拟机实例来尽可能的降低超出子截止期对后续任务的调度产生的影响。在为当前任务分配完虚拟机实例后，将结果记录下来步骤（7）。接下来对受任务分配完成所影响的一系列参数进行更新操作步骤（8），并将转变为就绪任务的任务加入到就绪队列之中进行新一轮的排序。循环截止条件为就绪队列不为空，每次处理完一个任务，该任务的后续任务就会被加入到就绪队列中，所以退出循环时，所有存在后续任务的任务都已经调度完成，说明整个工作流已调度完成。

3.6.2　算法时间复杂度

考虑任务数量为 n 的有向无环图 $G=(T,E)$，假设该有向无环图为全连通图，则有向边的最大数量为 $n(n-1)/2$，处理所有任务的时间复杂度为 $O(n^2)$。算法首先需要为所有的任务分配子截止期，其时间复杂度为 $O(n^2)$，接下来则是对就绪任务进行排序，其时间复杂度为 $O(nlogn)$。来到第（3）步，为队首任务寻找可用资源时，需要遍历资源池，找到其中的可用实例，然后再从这些可用虚拟机实例中选取 Q 值最大的虚拟机实例进行任务分配。资源池中虚拟机实例的数量为 p，整个第（3）步的时间复杂度为 $O(n\times p^2)$，第（4）步中，每调度完一个任务就需要对其余受影响的任务的 EST 进行更新，同时也对资源池中所使用的虚拟机实例的资源向量进行更新，由于任务数量为 n，所以该步骤的时间复杂度为 $O(2n)$。综上所述，算法整体时间复杂度为 $O(n^2)$。

4　仿真与结果分析

4.1　实验环境与参数

本文采用仿真的方法来评估提出算法的有效性。首先，采用 Java 编程语言完成了 DRSS 调度算法代码的编写，再将代码嵌入到基于 Java 语言的仿真平台 CloudSim 上进行仿真实验。其次，为了模拟调度场景与云资源平台，实验中构建了一个数据中心以及 8 种类型的虚拟机实例，其中虚拟机实例的详细参数参考 Amazon EC2 设置，见表 1。资源间的平均带宽参考 Amazon AWS 设置为 20 MBps。

表 1　资源参数

资源类型	计算单元 ECU	vCPU	内存/GB	价格/$
m4.large	6.5	2	8	0.100
m5d.large	8	2	8	0.113
m4.xlarge	13	4	16	0.200
m5d.xlarge	16	4	16	0.226
m4.2xlarge	26	8	32	0.400
m5d.2xlarge	31	8	31	0.452
m4.4xlarge	53.5	16	64	0.800
m5d.4xlarge	60	16	64	0.904

为了评估算法处理真实工作流应用程序的性能，本文使用了如图 2 所示的 4 种不同的科学工作流应用程序对所提算法进行测试，包括被广泛应用于工作流调度的研究 CyberShake、Epigenomics、LIGO 和 Montage 这 4 种科学工作流应用程序。Bharathi 等开发了一个工作流生成器，用于生成与这些工作流应用程序非常相似的合成工作流。他们使用这个工作流合成器创建了大量具有不同任务数的工作流应用程序，以 DAX（directed acyclic graph in XML）文件的格式放置在他们的网站上供研究者们下载使用。DAX 格式文件提供了工作流的详细信息，其中包括任务名称，任务计算工作量，任务传输工作量以及任务之间的依赖关系。本文借助文献［16］中提供的源代码，编写了一个用于对 DAX 文件进行信息提取的工具类，并使用提取出的数据完成了仿真实验。

为了衡量调度不同工作流的每种算法的性能，本文引入两个基准：

（1）最慢（最便宜）的调度方案，即在不考虑截止期的前提下，为所有的任务分配最便宜的虚拟机实例，得到

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

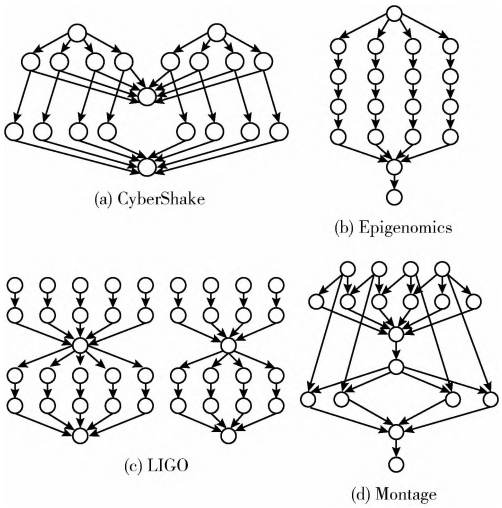

图 2 4 种科学工作流的结构

该调度方案花费的总时间 M_S 和总成本 C_S。

（2）最快（最昂贵）的调度方案，即为所有的任务分配最昂贵的虚拟机实例，得到该调度方案花费的总时间 M_F 和总成本 C_F。

为了评估算法的敏感度，本文设置了不同的截止期约束，截止期由严格变化至宽松。工作流的截止期 D 定义如下

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

其中，λ 表示截止期因子，范围从 0 到 1。工作流的截止期 D 由 λ 来决定，使得截止期可以在相对的严格到宽松之间变化，λ 越大，则代表截止期越宽松。通过以上的设定，可以通过控制截止因子 λ 的值来设置一个合理的截止期以测试不同算法的性能。

4.2 性能指标

本文评估算法性能的指标是：

（1）调度成功率 SR：算法调度成功率表示成功调度的工作流的数量与工作流总数的比值，定义如下

$$SR = \frac{\text{成功调度工作流数量}}{\text{实验中的工作流总量}} \tag{16}$$

（2）标准化成本 NC：标准化成本为调度方案的总成本 $Cost$ 与最便宜调度方案的成本 C_S 的比值，定义如下

$$NC = \frac{Cost}{C_S} \tag{17}$$

4.3 结果分析

整个数据集中的 DAX 文件共有 10 种，按任务节点数量对工作流的 DAX 文件进行分类，每一种工作流有 20 个 DAX 文件，任务节点数量分别为 100 个到 1000 个。为了减少实验的随机性，本文将实验分为 20 组，一组中包含 10 个不同节点数量的工作流。

本文选取的对比算法为 IC-PCP 算法和 SMWC 算法。前者在调度任务时采取基于成本的策略，后者在调度任务时采取基于余弦相似度的策略。

4.3.1 算法调度成功率结果分析

图 3 给出了截止期因子 λ 从 0.05 开始，以 0.05 的步长，变化至 0.5 时各个算法的平均成功率。

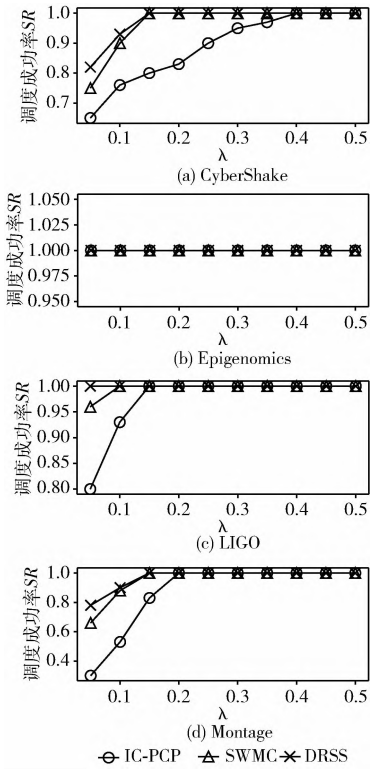

图 3 不同 λ 值下的调度成功率

从图 3 中可以看出，当截止期因子 λ 变大时，各个算法都能获得更高的成功率。在 Epigenomics 工作流中，所有算法都能取得 100%的调度成功率。在除 Epigenomics 工作流的另外 3 种工作流中，IC-PCP 算法在 λ 较小时取得的调度成功率都比较低，不过随着 λ 的变大，最终也能取得 100%的调度成功率。SMWC 算法和 DRSS 算法在截止期因子较小的时候也能取得高于 60%的调度成功率，其中 DRSS 算法取得的调度成功率会略高于 SMWC 算法。

4.3.2 算法标准化成本结果分析

图 4 给出了不同截止期因子下的各个算法的平均标准化成本。

从图 4 中可以看出，工作流的执行成本会随着截止期因子的增加而降低，这是因为在截止期宽松时各个算法可以选择更加便宜的虚拟机实例来在子截止期前完成任务。相比于 IC-PCP 算法和 SMWC 算法，本文提出的算法在大部分情况下能够取得更低的成本。

同时，不同工作流下的成本也有较大差距。从图中可以看出，Montage 和 Cybershake 工作流所需要花费的成本远远大于其它 3 种工作流下所需要的成本，这与工作流的结构有关。在 Montage 工作流和 Cybershake 工作流中，大

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

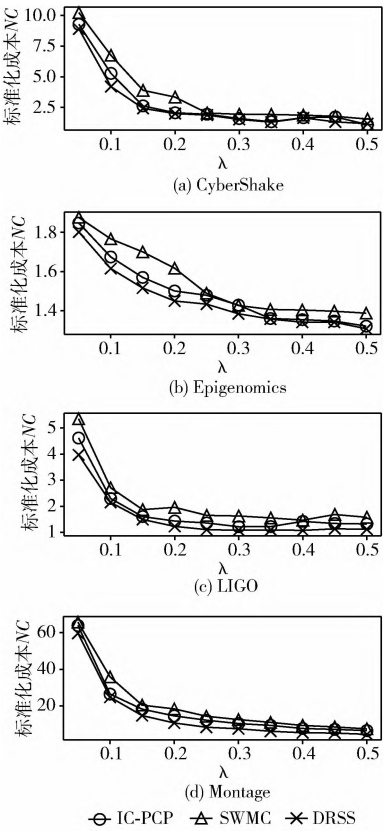

图 4　不同 λ 值下的标准化成本

部分的任务分布在第二层，而且任务的平均执行时间很短。当截止期较紧时，为了保证第二层的任务在子截止期前完成，算法会为任务选择昂贵的虚拟机实例，从而导致总成本较高。

5　结束语

由于现有云中工作流调度算法采用任务与资源实例一一对应的模式，而这类算法在面对大量微服务任务组成的工作流时，现有的应对策略会出现资源实例使用数量过多从而使得调度的成本升高的问题。针对这一问题，提出了一种基于动态资源选择策略的微服务工作流调度算法——DRSS 调度算法。该算法通过使用根据调度所处的相应阶段而动态变化的权重因子来对调度过程中的指标选取策略进行最优处理，以达到降低成本的目的。

实验结果表明，本文提出的 DRSS 调度算法与 IC-PCP 算法以及 SWMC 算法相比，DRSS 调度算法能够在保证调度成功率的前提下降低调度的总成本。

参考文献：

[1] O'Connor RV, Elger P, Clarke PM. Continuous software engineering—A microservices architecture perspective [J]. Journal of Software: Evolution and Process, 2017, 29 (11): e1866.

[2] HU Hongyu, CHEN Zheng. Cloud workflow scheduling based on fusion of chemical reaction optimization and ant colony optimization [J]. Computer Applications and Software, 2020, 37 (11): 229-238 (in Chinese). [胡红宇，陈政. 基于化学反应优化与蚁群优化融合的云工作流调度 [J]. 计算机应用与软件，2020，37 (11)：229-238.]

[3] 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, 20 (5): 1-12.

[4] XUE Fan, WU Zhijian. Cloud workflow task scheduling based on particle swarm optimization [J]. Microelectronics and Computers, 2018, 35 (8): 122-127 (in Chinese). [薛凡，吴志健. 基于粒子群优化的云工作流任务调度 [J]. 微电子学与计算机，2018，35 (8)：122-127.]

[5] Jia YH, Chen WN, Yuan HQ, et al. An intelligent cloud workflow scheduling system with time estimation and adaptive ant colony optimization [C] //IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2018: 1-16.

[6] Wang Pengwei, Lei Yinghui, Promise Ricardo, et al. Makespan-driven workflow scheduling in clouds using immune-based PSO algorithm [J]. IEEE Access, 2020, 8: 1.

[7] Wang Yun, Zuo Xingquan. An effective cloud workflow scheduling approach combining PSO and idle time slot-aware rules [J]. IEEE/CAA Journal of Automatica Sinica, 2021, 8 (5): 1079-1094.

[8] Kaur P, Mehta S. Resource provisioning and workflow scheduling in clouds using augmented shuffled frog leaping algorithm [J]. Journal of Parallel & Distributed Computing, 2017, 101: 41-50.

[9] Tong Z, Chen HJ, Deng XM, et al. A novel task scheduling scheme in a cloud computing environment using hybrid biogeography-based optimization [J]. Soft Computing, 2019, 23: 11035-11054.

[10] Faragardi HR, Saleh Sedghpour MR, Fazliahmadi S, et al. GRP-HEFT: A budget-constrained resource provisioning scheme for workflow scheduling in IaaS clouds [J]. IEEE Transactions on Parallel and Distributed Systems, 2020, 31 (6): 1239-1254.

[11] Zhou XM, Zhang GX, Sun J, et al. Minimizing cost and makespan for workflow scheduling in cloud using fuzzy dominance sort based HEFT [J]. Future Generation Computer Systems, 2019, 93: 278-289.

[12] Abrishami S, Naghibzadeh M, Epema DHJ. Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds [J]. Future Generation Computer Systems, 2013, 29 (1): 158-169.

[13] Wang S, Ding Z, Jiang C. Elastic scheduling for microservice applications in clouds [J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 32 (1): 98-115.

[14] Li W, Li X, Ruiz R. Scheduling microservice-based workflows to containers in on-demand cloud resources [C] //IEEE 24th International Conference on Computer Supported Cooperative Work in Design, 2021: 61-66.

[15] ZHANG Qiuxia, ZHANG Laishun. Workflow task scheduling meeting deadline constraints in cloud environment [J]. Computer Engineering and Design, 2019, 40 (2): 425-432 (in Chinese). [张秋霞，张来顺. 云环境中满足期限约束的工作流任务调度 [J]. 计算机工程与设计，2019，40 (2)：425-432.]

[16] Wu Q, Ishikawa 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.

(C)1994-2023 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net