计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220100100

云中使用竞价实例的截止时间约束的工作流调度优化算法

潘纪奎[1,2]　董心仪[1,2]　卢政昊[1,2]　王子健[1]　孙福权[1]
1 东北大学秦皇岛分校　河北 秦皇岛 066000
2 东北大学研究生院　沈阳 110000
(panjikuidxy@163.com)

摘　要　近年来,由于按需资源供应和即付即用付费模式具有的明显优势,在云环境中执行大规模工作流应用程序越来越流行。云服务提供商以不同的价格提供不同性能的资源。为了提高资源的利用率,许多云服务商提供的瞬时资源的价格远低于正常资源的价格,Amazon EC2 提供的竞价实例,可以大大降低工作流的执行成本。云中工作流调度的主要问题之一是在满足用户给定的截止时间约束的前提下,找到一种更廉价的调度方法。为解决这个问题,提出了一种使用竞价实例的截止时间约束工作流调度优化算法(Spot-ProLis)。该算法考虑了同一虚拟机上数据传输时长为零的情况,使用概率向上排序的方法对任务进行排序。在资源配置阶段,增加了竞价实例作为候选资源,有效降低了执行成本。实验结果表明,相比经典算法 ProLis,所提算法在降低执行成本上具有显著优势。

关键词:云环境;工作流调度;竞价实例;截止时间;执行成本;优化
中图法分类号　TP393

Deadline Constrained Scheduling Optimization Algorithm for Workflow in Clouds Using Spot Instance

PAN Jikui[1,2],DONG Xinyi[1,2],LU Zhenghao[1,2],WANG Zijian[1] and SUN Fuquan[1]
1 Northeastern University at Qinhuangdao,Qinhuangdao,Hebei 066000,China
2 Graduate School of Northeastern University,Shenyang 110000,China

Abstract　In recent years,due the advantages of on-demand resource provisioning and pay-as-you-go billing model,it is increasingly popular to execute large-scale workflow applications in cloud environments. Cloud service providers offer resources with different capabilities at different prices. In order to improve resource utilization,many cloud service providers provide transient resources at a much lower price than normal resources. Spot instance provided by Amazon EC2 can greatly reduce the execution cost of workflow. One of the main problems of workflow scheduling in cloud is to find a cheaper scheduling method on the premise of meeting the deadline. To solve this problem,a deadline constrained scheduling optimization algorithm for workflow in clouds using spot instance(Spot-ProLis) is proposed. The algorithm takes into account the case that the data transmission time of the same virtual machine is zero,and uses the method of probabilistic upward rank to order tasks. In the resource allocation stage,spot instances are added as candidate resources,which effectively reduces the execution cost. Experiment results show that compared with the classical ProLis algorithm,Spot-ProLis has significant advantages in reducing the execution cost.

Keywords　Cloud,Workflow scheduling,Spot instance,Deadline,Cost,Optimization

1　引言

工作流常被用于生物信息学、天文学和物理学等大规模建模科学问题[1]。一个工作流通常包含数百或数千个具有依赖关系的计算任务。一般用有向无环图(DAG)来描述一个工作流模型,其中节代表示任务,边表示任务之间的依赖关系[2]。工作流调度的目的是在满足服务质量(QoS)约束的同时找到合适的计算资源来执行工作流中的任务。传统的分布式计算平台不仅需要昂贵的构建和维护成本,而且不能适应资源需求的激增。然而,云计算的出现使越来越多的大规模工作流应用程序被转移到云领域。云环境中包含各种类型的资源,用户可以基于按次付费的模式租用云资源[3],租用性能越高的资源可以带来更快的处理速度,同时也需要承担更加昂贵的费用。因此云工作流调度不仅需要考虑完工时间,

到稿日期:2022-01-12　返修日期:2022-06-20
基金项目:国家重点研发计划(2018YFB1402800)
This work was supported by the National Key R&D Program of China(2018YFB1402800).
通信作者:孙福权(2224765938@qq.com)

还需要考虑执行工作流应用程序的成本，这是一个已知的NP难问题[4]。多年来，这一问题一直是分布式计算社区的热点[5]。

实际上，云服务商为了满足用户的峰值需求，往往会过度提供云资源，这就使得云数据中心的计算资源没有得到充分的利用。为了充分利用计算资源，云服务商为用户提供了更加经济高效的计算资源。例如，Amazon在2009年提出了一种基于出价的实例类型，即竞价实例(Spot Instance)[6]。当用户出价不低于市场价格时，可以成功启动实例；当市场价格高于用户出价时，将撤销该实例。虽然这类资源可靠性较低，但配置相同的竞价实例与按需实例具有相同的性能，而竞价实例却具有极大的价格优势。与按需实例相比，竞价实例最高可以享受90%的折扣。当有可用的资源时，用户可以以更低的价格(通常便宜50%～90%)访问与按需实例相同性能的资源。这对云服务提供商和用户来说是互利的过程，因为它不仅降低了闲置资源带来的操作和维护成本，而且还显著降低了用户运行大规模工作负载的成本。尽管如此，竞价实例也存在一个非常严重的问题，即它可能被随时中断。在使用竞价实例时，突然撤销正在运行的实例将严重降低所提交的应用程序的性能。因此，研究人员在使用竞价实例时，特别是在截止日期限制下的工作流应用程序中，需要非常小心。

本文提出了一种云中使用竞价实例的截止时间约束工作流调度优化算法(Spot-ProLis)。该算法首先利用概率向上排序的方法对任务进行排序[7]，然后为每一个任务分配合适的计算资源。在资源分配阶段，引入了具有高成本效益的竞价实例。用户根据不同的需求，在可用的按需实例和竞价实例中选择更加低廉的计算资源，以降低成本。实验结果表明，相比经典算法ProLis[7]，Spot-ProLis在降低执行成本上有非常明显的优势。

2 相关研究

工作流调度问题一开始都是围绕多处理器系统[8]和阵列系统[9]进行研究，因为这些系统中的资源的使用都是免费的，所以最初的研究都是以最小化工作流的完工时间为目的。云计算的兴起和云环境的出现，使得云工作流调度成为了新的研究课题。由于云计算资源是以租用的形式使用，这使得执行成本成为了不得不考虑的问题。云工作流调度一般需要同时优化多个目标，通常的做法是在满足其他目标约束的前提下，将其中的一个目标作为主要的优化对象。这个问题是已知的NP难问题，因此研究者们提出了大量的算法来解决这一问题。文献[10]利用了遗传算法，文献[11-12]基于粒子群算法，文献[13]基于蚁群算法。文献[14]则在传统粒子群算法的基础上进行了改进，提出了免疫粒子群算法，该算法加快了粒子群算法的收敛速度，从而提高了算法的质量。

自Amazon在2009年提出了竞价实例后，很多关于竞价实例价格预测的研究相继出现。文献[15]使用3种统计方法来分析典型的竞价实例定价模型。文献[16]使用LSTM网络来预测竞价实例价格。LSTM基于递归神经网络(Recurrent Neural Network，RNN)，解决了RNN的长期依赖问题。文献[17]提出了一种随机森林回归算法，通过计算机拟合可以准确预测次日和一周的竞价实例价格。文献[18]提出了时间序列中的持续时间协议(DrAFTS)方法，该方法使用历史价格作为训练集，可以在用户指定的截止日期约束和资源需求下预测最低价格。文献[19]利用semi-Markov链模拟竞价实例价格波动，并通过枚举和贪婪策略开发出最优出价方法。文献[20]利用竞价实例的价格历史，通过建立一个k-Nearest Neighbors(kNN)回归模型来预测未来的价格。

竞价实例的稳定性不高，可能被随时中断，因此很多研究人员对竞价实例的容错机制做了大量研究。文献[21]介绍了一种调度算法，用于部署在现场实例上的科学工作流。该算法采用了备份复制和检查点机制，提高了容错能力。文献[22]引入了一个新的框架MASA-CUDAlign，该框架即使在多个实例被撤销的情况下也能满足截止时间。文献[23]介绍了一种在基于云的GPU集群上进行分布式模板计算的容错框架。该框架中，计算任务在Amazon Web服务GPU云上执行，并利用其竞价实例来提高成本效率。

3 问题描述

3.1 工作流模型

一个工作流通常用有向无环图$DAG=(V,E)$来表示。工作流中的每一个任务都可以被理解为一段独立的、不能并行执行的程序，在有向无环图中用一个节点表示。有向无环图中的所有节点构成一个集合$V=\{t_1,t_2,\cdots,t_n\}$。用w_i代表一个给定的任务t_i的计算量，E代表有向无环图中所有边的集合。E中连接任务t_i和任务t_j的边$e_{ij}=\{t_i,t_j\}$代表两个任务之间的一种依赖关系，即任务t_j只有在任务t_i执行结束以后才能开始执行。如果任务t_i和任务t_j之间有数据传输，则用d_{ij}代表由任务t_i向任务t_j的数据传输量。这时的任务t_i被称为父任务，任务t_j被称为子任务。如果某个任务没有父任务，则把这个任务称为起始任务；如果某个任务没有子任务，则把这个任务称为终止任务。有些工作流可能同时拥有多个起始任务或者终止任务，大多数的调度算法不能在这种情况下使用。为了一般化工作流模型，可以在开始处和结束处分别添加一个负载为零且没有数据传输的虚拟任务t_{entry}和t_{exit}，用以表示整个工作流的起始任务和终止任务。图1给出了一个工作流示例。

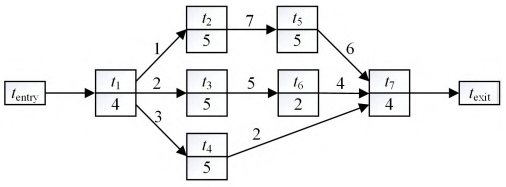


图1 工作流示例图

Fig. 1 Example of workflow

3.2 云资源模型

云资源模型采用基于Amazon EC2实际产品的资源模型。为方便统一管理和为用户提供服务，云服务商把各类资源虚拟化为资源池的形式。云服务商提供具有不同计费模式的虚拟机，如按需实例、预留实例和竞价实例。本文使用了按需实例和竞价实例两种资源实例。除计费模式不同外，具有

相同配置的按需实例和竞价实例的另一区别就是按需实例可靠性更高，因为竞价实例可能被随时收回。

用 $R=\{r_1,r_2,\cdots,r_n\}$ 代表用户可以租用的资源类型的集合，不同类型资源的处理能力和价格也不同。租用性能越高的资源可以带来更快的处理速度，同时也需要承担更加昂贵的费用。一个具体的资源实例或虚拟机用 vm_k^{type} 表示，其中 $k\in R$ 代表这个虚拟机的资源类型；$type\in\{d,s\}$ 被用来区分竞价实例和按需实例。则 k 类型的按需实例用 vm_k^d 表示，k 类型的竞价实例用 vm_k^s 表示。每一个 vm_k^{type} 都属于 $VMS=\{vm_1^d,vm_2^d,\cdots,vm_n^d,vm_1^s\ vm_2^s,\cdots,vm_n^s\}$。

工作流中的任务是在虚拟机上执行的，一个任务只能在一个虚拟机上执行，而一个虚拟机可以按照顺序执行多个任务。使用 $P(vm_k^{type})$ 代表 k 类型资源的处理能力，所有的资源都采用即付即用的定价模式，虚拟机的收费由其被租用的单元时间数量决定。即使虚拟机没有使用完一个完整的单元时间，也会收取一个完整单元时间的费用。如图 2 所示，T 表示虚拟机的单元时长，此虚拟机的租用开始时刻为 0，租用结束时刻为 $3.2T$，那么用户会被收取 4 个单元时长的费用。

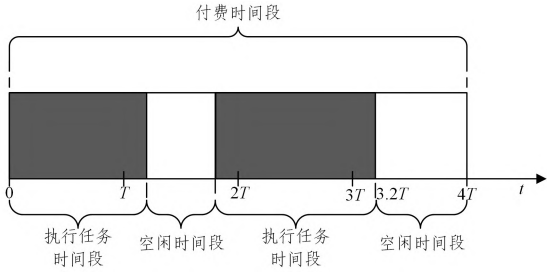


图 2 付费时间段示例图

Fig. 2 Example of a paid time period

图 3 给出了竞价实例的价格波动示例。其中实线表示竞价实例的市场价格，虚线表示用户的出价，只有用户的出价高于竞价实例的市场价格时才能访问竞价实例。例如，B 为用户的出价，竞价实例的价格在 T_b 和 T_d 之间小于用户的出价，用户在 T_b 时刻可以访问竞价实例，在 T_d 时刻终止访问，在这段时间里用户支付租用费用并获得竞价实例。需要注意的是，用户需要支付的费用不是用户的出价 B，而是竞价实例的市场价格。

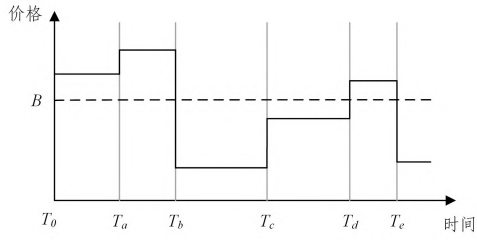


图 3 竞价实例的价格波动示例图

Fig. 3 Examples of fluctuations about spot insance price

假设任务 t_i 被安排到虚拟机 vm_k^{type} 上执行，用 $ET_{i,k}^{type}$ 表示 t_i 在 vm_k^{type} 上的执行时长，则：

$$ET_{i,k}^{type}=\frac{w_i}{P(vm_k^{type})} \tag{1}$$

假设所有的虚拟机都位于相同的物理区域并且虚拟机的平均带宽大致相等，虚拟机之间的数据传输是免费的。将虚拟机的虚拟带宽设置为 bw，则任务 t_i 向任务 t_j 的数据传输时长为：

$$TT_{i,j}=\begin{cases}\dfrac{d_{i,j}}{bw}, & \text{if } vm(t_i)\neq vm(t_j)\\ 0, & \text{otherwise}\end{cases} \tag{2}$$

其中，$vm(t_i)$ 代表被用来执行任务 t_i 的虚拟机。当两个任务被安排到同一虚拟机上执行时，这两个任务之间的传输时长为零。

用 $ST_{i,k}^{type}$ 代表任务 t_i 在虚拟机 vm_k^{type} 上的执行开始时刻，$FT_{i,k}^{type}$ 代表任务 t_i 在虚拟机 vm_k^{type} 上的执行完成时刻，则：

$$ST_{i,k}^{type}=\max\{RT_{i,k}^{type},\max_{t_j\in t_i\text{'parents}}\{FT_{j,k}^{type}+TT_{j,i}\}\} \tag{3}$$

$$FT_{i,k}^{type}=ST_{i,k}^{type}+ET_{i,k}^{type} \tag{4}$$

其中，$RT_{i,k}^{type}$ 代表 vm_k^{type} 的可执行 t_i 的最早时刻，即 vm_k^{type} 上排列在 t_i 之前的任务的完成时刻。若 t_i 是 vm_k^{type} 上的第一个任务，则 $RT_{i,k}^{type}$ 为 vm_k^{type} 的启动完成时刻。

虚拟机 vm_k^{type} 的租用开始时刻和租用结束时刻分别用 LST_k^{type} 和 LFT_k^{type} 表示，则：

$$LST_k^{type}=ST_{i,k}^{type}-\max_{t_j\in t_i\text{'s parents}}\{TT_{j,i}\} \tag{5}$$

$$LFT_k^{type}=FT_{i,k}^{type}+\max_{t_j\in t_i\text{'s children}}\{TT_{i,j}\} \tag{6}$$

其中，LST_k^{type} 为虚拟机 vm_k^{type} 上的第一个任务开始接收数据的时刻，LFT_k^{type} 为 vm_k^{type} 上最后一个任务完成数据传输的时刻。假设租用 vm_k^{type} 一个单位时长（TI）所需要承担的租赁费用为 UC_k^{type}，式(7)给出了 vm_k^{type} 的租用费用。

$$EC_k^{type}=\left\lceil\frac{LFT_k^{type}-LST_k^{type}}{TI}\right\rceil\times UC_k^{type} \tag{7}$$

一般来说，云工作流成本包含执行成本和传输成本。本文的数据传输时间包含在虚拟机的租用时间中，使用虚拟机的租用时间计算执行成本，因此传输成本包含在执行成本中。

3.3 调度模型

调度的目的是把工作流中的任务分配给具体的虚拟机，并合理安排执行顺序。租用的虚拟机的集合用 $I=\{vm_1^d,vm_2^d,\cdots vm_I^d,vm_1^s,vm_2^s,\cdots,vm_J^s\}$ 表示。任务到虚拟机的映射的集合用 M 表示，一个由 vm_k^{type} 执行任务 t_i、执行开始时间为 $ST_{i,k}^{type}$、执行结束时间为 $FT_{i,k}^{type}$ 的映射可以用 $m=\langle t_i,vm_k^{type},ST_{i,k}^{type},FT_{i,k}^{type}\rangle$ 描述。根据前文所述，一次调度的结果可以描述为 $\langle I,M\rangle$。工作流调度结果的完工时间 $makespan(I,M)$ 和执行成本 $cost(I,M)$ 分别使用式(8)和式(9)计算。

$$makespan(I,M)=\max_{k\in I}\{FT_k^{type}\} \tag{8}$$

$$cost(I,M)=\sum_{i=1}^{|I|}EC_i^d+\sum_{j=1}^{|J|}EC_j^s \tag{9}$$

其中，FT_k^{type} 表示虚拟机 vm_k^{type} 的租用结束时刻，$\sum_{i=1}^{|I|}EC_i^d$ 表示租用的全部按需实例的执行成本之和，$\sum_{j=1}^{|J|}EC_j^s$ 表示租用的全部竞价实例的执行成本之和，将两部分相加得到执行成本的总和。如果用 D 代表一个已知工作流给出的最后期限，则工作流调度问题可描述为：

$$\begin{aligned}&\min\ cost(I,M)\\&\text{s.t.}\ \ makespan(I,M)\leqslant D\end{aligned}\tag{10}$$

4 算法设计

本文算法考虑了当两个任务部署到同一虚拟机上时，这两个任务之间的传输时间会变为零的情况。采用一个简单的概率向上排序的方法对任务进行排序，将截止时间分配给各个任务；充分利用了竞价实例的价格优势，提出了一种云中使用竞价实例的截止时间约束工作流调度优化算法(Spot-ProLis)，用于基于截止时间约束的成本优化。

4.1 子截止日期分配方法

将从任务 t_i 到任务 t_{exit} 的最长路径用任务向上排序 r_i 表示，r_i 可以递归定义为式(11)。因为首要目标是满足截止时间约束，所以 vm^* 选择处理速度最快的虚拟机。任务 t_{entry} 的任务向上排序是工作流的关键路径，用 r_{entry} 表示。

$$r_i=\max_{t_j\in t_i\text{'s children}}\left\{r_j+\frac{d_{i,j}}{bw}\right\}+\frac{w_i}{p(vm^*)}\tag{11}$$

$$sd_i=D\times\frac{r_{\text{entry}}-r_i+w_i/p(vm^*)}{r_{\text{entry}}}\tag{12}$$

一种直观的截止时间分配方法是将任务的子截止时间按照从入口节点到任务节点的最长路径比例设置，如式(12)所示。但是这种方法没有考虑到当两个连续执行的任务部署到同一虚拟机上时，这两个任务之间的传输时间会变为零，这种情况下会导致不合理的截止时间分配[7]。例如：图 4 中工作流的截止时间为 103.1，这种分配方法将 t_1，t_2，t_3，t_4 的子截止时间分别设置为 1，1，102 和 103.1。虽然每个任务的执行时间是相同的，但是 t_3 获得的截止时间比其他任务长得多。这可能会导致即使把 t_3 分配给最快的服务器，其他 3 个任务也不能按时完成。

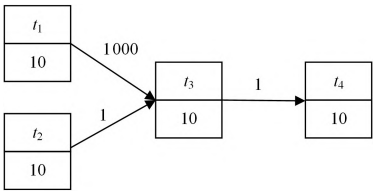


图 4 用于说明的工作流示例图

Fig. 4 Workflow example for illustration

由于 t_1 和 t_3 之间的传输时间很长，如果在服务分配阶段考虑把 t_1 和 t_3 分配到同一个虚拟机上将更加合理，因此引入任务的概率向上排序来分配截止时间，如式(13)所示：

$$pr_i=\max_{t_j\in t_i\text{'s children}}\left\{pr_j+\gamma_j\times\frac{d_{i,j}}{bw}\right\}+\frac{w_i}{p(vm*)}\tag{13}$$

$$\gamma_j=\begin{cases}0, & \text{if } 1-\theta^{-ccr_j}<rand()\\1, & \text{otherwise}\end{cases}\tag{14}$$

其中，γ_j 表示计算 pr_i 时是否考虑任务 t_i 到 t_j 的数据传输时间，具体如(14)所示。ccr_j 是 t_j 的计算与通信比，$rand()$ 是一个返回[0，10)中随机数的函数，θ 是一个大于 1 的参数。因此，ccr_j 越小，pr_i 返回 0 的概率越大，反之亦然。

基于概率向上排序的截止时间分布通过式(15)计算，其中用 pr_i 代替 r_i。

$$psd_i=D\times\frac{pr_{\text{entry}}-pr_i+w_i/p(vm*)}{pr_{\text{entry}}}\tag{15}$$

4.2 使用竞价实例的 Spot-ProLis 算法

尽管可以用来对任务进行排序的方法很多，如任务向上排序、静态级别排序和 ALAP(As-Late-As-Possible)等，但考虑到在同一虚拟机顺序执行的两个任务之间的传输时间可能为零的情况，本文选择使用概率向上排序的方法对任务进行排序，分配子截止期限更加合理。

Spot-ProLis 算法对云工作流调度具有十分明显的效果。此算法首先使用概率向上排序的方法对任务进行排序，再进行截止时间的分配，将截止时间 D 分配给每一个任务，使每一个任务得到一个子截止时间 psd_i。然后依次为每一个任务配置一个满足子截止期限并且尽可能使租用费用最低的虚拟机。

算法 1 给出了虚拟机的选择算法。此算法参考了 ProLis 服务选择算法[7]。算法的输入为一个经过拓扑排序后的任务列表 L。算法需要为 L 中的每一个任务选择虚拟机，原则是满足任务的子期限且执行成本的增量最小(第 3 行)。候选的虚拟机包括所有已选择的虚拟机(集合 I)以及尚未被选择但是可以随时被添加到 I 中的虚拟机。将找到的任务到虚拟机的映射并入集合 M 中并计算执行开始时刻和执行结束时刻(第 4－5 行)。如果所有虚拟机均不能满足一个任务的子期限，则将该任务配置到能使其最早完成任务的虚拟机中，并将此虚拟机升级为执行速度最快的类型(第 6－11 行)。再次将找到的任务到虚拟机的映射并入集合 M 中并计算执行开始时刻和执行结束时刻(第 12 行)。将找到的虚拟机分别设为按需实例和竞价实例来比较租用费用，如果按需实例的租用费用高于竞价实例的租用费用，则虚拟机选择竞价实例，否则选择按需实例(第 13－17 行)。至此虚拟机选择完毕，生成分配方案(第 19－21 行)，直到所有的任务均配置好虚拟机后返回最终的调度结果(第 23 行)。

算法 1 虚拟机选择算法

输入：一个任务列表 L

输出：调度结果 S=〈I，M〉

1. I←Ø，M←Ø
2. for ti∈L do
3. vm_k^{type}←选择一个满足子截止期限 psd_i 并租用费用增量最小的虚拟机(通过算法 2 计算)
4. if vm_k^{type}!=null then
5. M=M∪{〈t_i，vm_k^{type}，$ST_{i,k}^{type}$，$FT_{i,k}^{type}$〉}←$ST_{i,k}^{type}$，$FT_{i,k}^{type}$ 分别通过式(3)和式(4)计算
6. else
7. vm_k^{type}←选择一个结束时间最早的虚拟机
8. while 不能满足 psd_i && 不是最快类型的虚拟机 do
9. 将 vm_k^{type} 的类型上升到一个更快的级别
10. 更新 vm_k^{type} 的完成时间
11. end while
12. M=M∪{＜t_i，vm_k^{type}，$ST_{i,k}^{type}$，$FT_{i,k}^{type}$＞}←$ST_{i,k}^{type}$，$FT_{i,k}^{type}$ 分别通过式(3)和式(4)计算

13. if $EC_k^d > EC_k^s$ then
14. 将虚拟机设置成竞价实例
15. else
16. 将虚拟机设置成按需实例
17. end else
18. end else
19. if $vm_k^{type} \notin I$ then
20. $I = I \cup \{vm_k^{type}\}$
21. end if
22. end for
23. return $S = \langle I, M \rangle$

算法 2 给出了租用费用增量最小的虚拟机的选择算法。算法的输入是一个任务 t_i。在所有已租用的虚拟机中选择增量小的虚拟机(第 2－4 行)。如果还没有已租用的虚拟机，则租用一个新的虚拟机。分别计算每一个类型的按需实例和竞价实例的租用费用增量 inC_k^d 和 inC_k^s(第 7－8 行)，将 inC_k^d 和 inC_k^s 分别与之前选择的虚拟机的租用费用增量进行比较，选择增量更小的虚拟机(第 9－16 行)。遍历所有的虚拟机类型，得到一个租用费用增量最小的虚拟机(第 6－18 行)。

算法 2 租用费用增量最小的虚拟机的选择算法

输入：一个任务 t_i

输出：租用费用增量最小的虚拟机 $vm_{selected}$

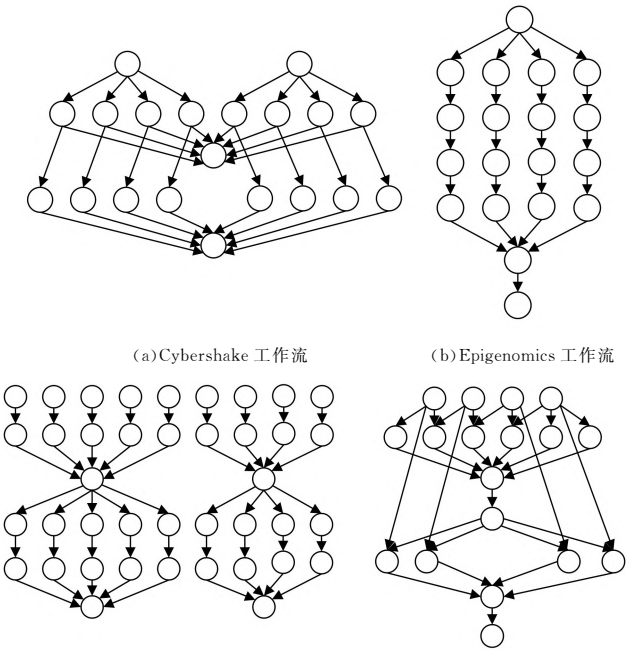
1. $vm_{selected} = null$，$inC = MAX$
2. for $vm_k^{type} \in I$ do
3. $vm_{selected} = vm_k^{type}$ ←通过式(7)计算增加 t_i 前后的虚拟机的租用费用，取差值得到租用费用的增量，选择增量最小的虚拟机
4. end for
5. if $vm_{selected} = null$ then
6. for $ri \in R$ do
7. $k = r_i$ ←将虚拟机类型赋值给 k
8. 通过式(7)分别计算按需实例 vm_k^d 和竞价实例 vm_k^s 增加 t_i 前后的虚拟机的租用费用，取差值得到租用费用的增量 inC_k^d 和 inC_k^s
9. if $inC > inC_k^d$ then
10. $vm_{selected} = vm_k^d$
11. $inC = inC_k^d$
12. end if
13. if $inC > inC_k^s$ then
14. $vm_{selected} = vm_k^s$
15. $inC = inC_k^s$
16. end if
17. end for
18. return $vm_{selected}$

5 仿真实验

5.1 实验设置

此部分将介绍对所提算法进行验证的仿真实验设置情况。仿真实验在一台 PC 机上进行。PC 机的硬件环境：Intel(R) Core(TM) i5，8 GB 内存。PC 机软件环境：Windows 10 的操作系统，Java 语言，Eclipse 开发环境。仿真程序利用了文献[7]所提出的开源程序，并在该基础上重新对竞价实例进行建模。

实验使用了 CyberShake，Epigenomics，LIGO 和 Montage 这 4 种工作流模型，它们各具特点。CyberShake 属于需要大量的 CPU 资源的数据密集型工作流，通常用于刻画地震灾害；Epigenomics 本质上是一个用来自动执行各种各样的基因组排序操作的数据处理管道，被应用于信息生物学；LIGO 也属于 CPU 密集型工作流，主要应用于引力物理学；Montage 的任务间存在大量的 I/O，CPU 资源需求不高，主要被应用于天文学。图 5 给出了上述 4 种工作流的结构特点，可以参考文献[1]获得更具体的描述。实验使用 Bharathi 等提供的工作流生成器生成的工作流。仿真实验对每个工作流模型都选择了 50 个任务、100 个任务、200 个任务、…、900 个任务和 1 000 个任务 11 个不同规模的工作流。



(a) Cybershake 工作流　(b) Epigenomics 工作流

(c) LIGO 工作流　(d) Montage 工作流

图 5 4 种工作流的结构特征

Fig. 5 Structural characteristics of four workflows

仿真实验采取了 5 种不同类型的虚拟机，分别为 r5.large，r5.xlarge，r5.2xlarge，r5.4xlarge 和 r5.8xlarge。虚拟机的租用费用参照 Amazon EC2 真实的收费标准按照 r5.8xlarge 按需实例的价格进行归一化处理。5 种类型的虚拟机按照按需资源类型进行收费时，它们的处理能力和归一化收费标准如表 1 所列。

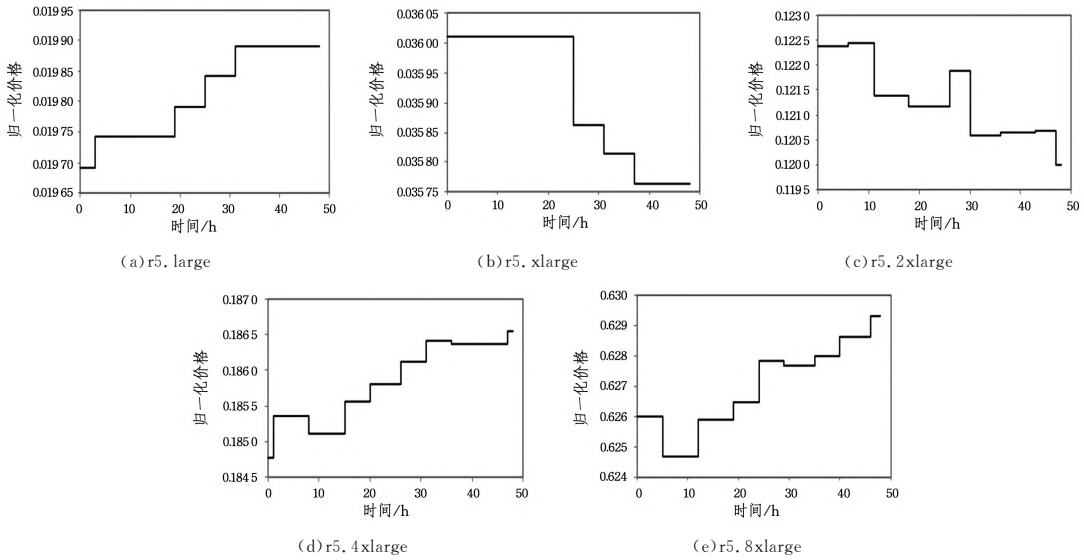
表 1 5 种按需实例的处理速度和价格

Table 1 Processing speed and cost of 5 on-demand instances

类型	处理速度	成本
r5.large	1	0.0625
r5.xlarge	2	0.1250
r5.2xlarge	4	0.2500
r5.4xlarge	8	0.5000
r5.8xlarge	16	1.0000

自 2017 年以来，Amazon 已经调整了竞价实例的竞价

策略。当前的价格波动已经趋于稳定，未来的价格也很容易预测。因此本文假设所有的竞价实例在收费期间都是可靠的，没有考虑更多的投标策略。5 种类型的竞价实例的处理速度与按需实例的处理速度相同，收费标准参照了 2021 年 12 月 6 日 0 时到 7 日 24 时之间 48 h 的历史价格并进行归一化处理，如图 6 所示。



(a) r5. large (b) r5. xlarge (c) r5. 2xlarge

(d) r5. 4xlarge (e) r5. 8xlarge

图 6 5 种按需实例的价格波动图

Fig. 6 Price fluctuations of 5 on-demand instances

选择 ProLis 作为对比算法，使用调度成功率和执行成本两个指标对两种算法进行评估。性能越好的算法成功率越高并且执行成本越低。最后期限的设置对调度算法的结果也有重要的影响，最后期限越紧张，调度的成功率越低且成本越高。为了更加合理地评估所提算法的性能，引入以下两个基准对不同最后期限设置下的算法进行验证。

(1) 廉价调度：仅在一台最廉价的虚拟机上使用 HEFT[24] 算法进行工作流调度。M_c 和 C_c 分别表示廉价调度的完工时间和执行成本。

(2) 快速调度：仅选择最昂贵的虚拟机进行工作流调度，调度算法为 HEFT[24]。M_f 和 C_f 分别表示快速调度的完工时间和执行成本。

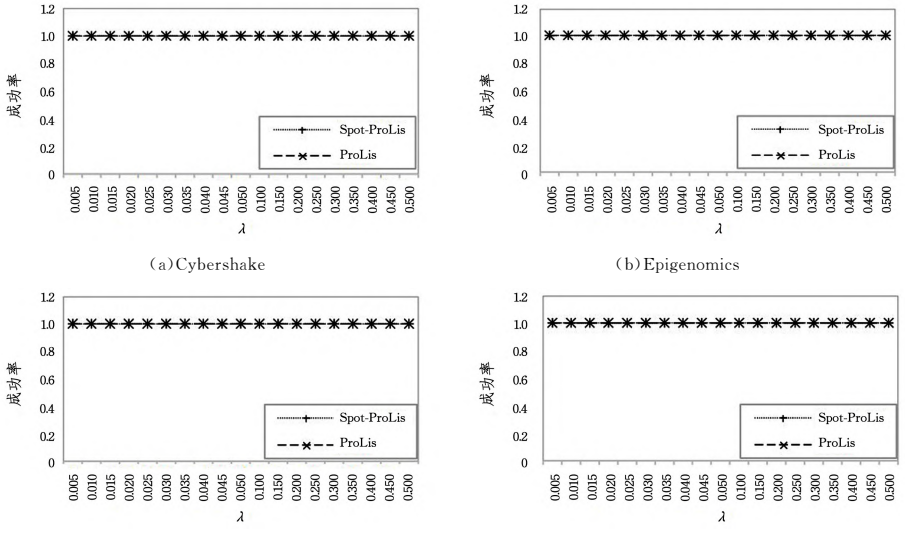
将最后期限表示为：

$$D=M_F+(M_C-M_F)\times\lambda \tag{16}$$

其中，$\lambda(\lambda\in[0,1])$ 为最后期限的宽松程度。不同宽松程度下各工作流调度算法的表现可以通过设置不同的 λ 值来观察。

5.2 实验结果

先令 λ 从 0.005 变化至 0.05，步长为 0.005 进行实验，观察在相对紧张的最后期限设置下各算法的表现；又令 λ 从 0.1 变化至 0.5，步长为 0.05，观察在相对宽松的最后期限设置下各算法的表现。图 7 给出了 λ 从 0.005 变化至 0.5 时各算法的平均成功率。由实验结果可知，对于 λ 的所有取值，两种算法在 4 种不同的工作流模型下的成功率都为 100%，都表现优异。



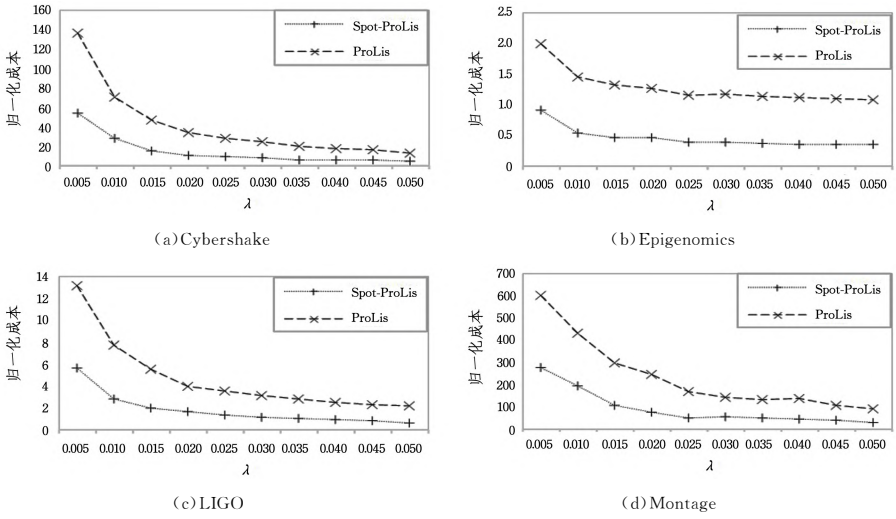
(a) Cybershake (b) Epigenomics

(c) LIGO (d) Montage

图 7 不同 λ 时每种方法的成功率

Fig. 7 Success rate of each method with different λ

图 8 给出了 λ 从 0.005 变化至 0.05 时各算法的归一化化执行成本。由实验结果可知，随着 λ 由小变大，两种算法在 4 种工作流模型下的归一化执行成本都逐渐降低，但 Spot-ProLis 算法在 λ 取任何值时的表现均优于 ProLis 算法，在 Epigenomics 模型下的表现尤其优异。图 9 给出了 λ 从 0.1 变化至 0.5 时各算法的归一化执行成本。由实验结果可知，随着 λ 由小变大，两种算法在 Cybershake 和 Montage 这两种模型下的归一化执行成本都逐渐降低，在 Epigenomics 和 LIGO 两种模型下的归一化执行成本变化相对不明显。但 Spot-ProLis 算法在 λ 取任何值时的表现均优于 ProLis 算法。

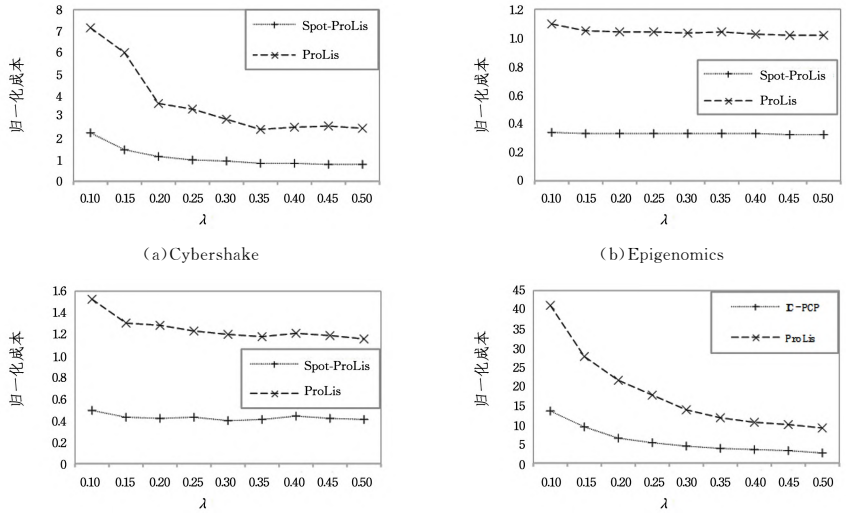


(a)Cybershake (b)Epigenomics

(c)LIGO (d)Montage

图 8 不同 λ 时每种方法的成本(0.005<λ<0.05)

Fig. 8 Cost of each method with different λ(0.005<λ<0.05)



(a)Cybershake (b)Epigenomics

(c)LIGO (d)Montage

图 9 不同 λ 时每种方法的成本(0.1<λ<0.5)

Fig. 9 Cost of each method with different λ(0.1<λ<0.5)

结束语 本文提出了 Spot-ProLis 算法，该算法使用了更加低廉的竞价实例，主要解决云工作流的调度问题。对于一个给定的工作流、最后期限约束和云模型，使用该算法可以找到一个满足最后期限约束且优化了执行成本的调度方案。我们使用 Amazon EC2 提供的现实数据进行实验仿真，并与经典的调度算法 ProLis 进行了对比。实验结果表明，Spot-ProLis 在降低执行成本上很有竞争力。

在接下来的工作中，一方面我们将尝试改进所提算法，提出更加有效且合理的算法来解决此类问题；另一方面我们也将尝试使用其他方式来解决本文所研究的问题。除此之外，我们还将围绕多目标的工作流调度进行研究。

参 考 文 献

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

[2] HEE K V. Workflow Management: Models, Methods, and Systems[M]. Cambridge: The MIT Press, 2004.

[3] SU S, LI J, HUANG Q J, et al. Cost-efficient task scheduling for executing large programs in the cloud[J]. Parallel Computing, 2013, 39(4/5): 177-188.

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

[5] MICHZEL L, PINED O. Scheduling: Theory, Algorithms, and Systems[M]. Berlin: Springer, 2012.

[6] CAO S J, DENG K F, REN K J, et al. An optimizing algorithm for deadline constrained scheduling of scientific workflows in IaaS clouds using spot instances[C]// 2019 IEEE International Conference on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking. 2019: 1421-1428.

[7] WU Q W, FUYUKI I, ZHU Q S, et al. Deadline-constrained cost optimization approaches for workflow scheduling in clouds [J]. IEEE Transactions on Parallel and Distributed Systems, 2017, (12): 3401-3412.

[8] YU-KWONG K, ISHFAQ A. Static scheduling algorithms for allocating directed task graphs to multiprocessors [J]. ACM Computing Surveys(CSUR), 1999, 31(4): 406-471.

[9] JIA Y, RAJKUMAR B, KOTAGIRI R. Workflow scheduling algorithms for grid computing[J]. Studies in Computational Intelligence, 2008, 146: 173-214.

[10] JIA Y, RAJKUMAR B. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms[J]. Scientific Programming, 2006, 14(3/4): 217-230.

[11] WU Z J, NI Z W, GU L C, et al. A revised discrete particle swarm optimization for cloud workflow scheduling[C]// 2010 International Conference on Computational Intelligence and Security. 2010: 184-188.

[12] MARIA A R, RAJKUMAR B. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds [J]. IEEE Transactions on Cloud Computing, 2014, 2(2): 222-235.

[13] CHEN Z G, ZHAN Z H, LI H H, et al. Deadline constrained cloud computing resources scheduling through an ant colony system approach[C]// 2015 International Conference on Cloud Computing Research and Innovation(ICCCRI). 2016: 112-119.

[14] WANG P W, LEI Y H, PROMISE R, et al. Makespan-driven workflow scheduling in clouds using immune-based PSO algorithm[J]. IEEE Access, 2020, 8: 29281-29290.

[15] MARKUS L, MOHAN B C, QUOC B V, et al. On Estimating Minimum Bids for Amazon EC2 Spot Instances[C]// Cluster, Cloud and Grid Computing. 2017: 391-400.

[16] MATT B, CHRISTIAN H, RICH W, et al. Predicting Amazon Spot Prices with LSTM Networks[C]// Scientific Cloud Computing. 2018: 1-7.

[17] VEENA K, ANAND K C, CHANDRA P G. Amazon EC2 spot price prediction using regression random forests [J]. IEEE Transactions on Cloud Computing, 2020, 8(1): 59-72.

[18] RICH W, JOHN B, RYAN C, et al. Probabilistic guarantees of execution duration for Amazon spot instances[C]// Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Association for Computing Machinery. 2017: 1-11.

[19] GUO W H, CHEN K, WU Y W, et al. Bidding for Highly Available Services with Low Price in Spot Instance Market[C]// High-Performance Parallel and Distributed Computing. 2015: 191-202.

[20] LIU W Q, WANG P W, MENG Y, et al. Cloud spot instance price prediction using kNN regression[J]. Human-centric Computing and Information Sciences, 2020, 10(34): 1-14.

[21] DEEPAK P, KOTAGIRI R, RAJKUMAR B. Enhancing Reliability of Workflow Execution Using Task Replication and Spot Instances[J]. ACM Transactions on Autonomous and Adaptive Systems, 2016, 10(4): 1-21.

[22] BRUM R C, SOUSA W P, MELO A, et al. A Fault Tolerant and Deadline Constrained Sequence Alignment Application on Cloud-Based Spot GPU Instances[C]// Parallel Processing. 2021: 317-333.

[23] ZHOU J, ZHANG Y, WONG W F. Fault Tolerant Stencil Computation on Cloud-based GPU Spot Instances[J]. IEEE Transactions on Cloud Computing, 2017, 7(4): 1013-1024.

[24] 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.

PAN Jikui, born in 1998, postgraduate. His main research interest is workflow scheduling.

SUN Fuquan, born in 1964, Ph.D, professor. His main research interests include cloud resource scheduling and allocation and big data analysis.

(责任编辑:何杨)