一种云计算环境下的工作流双向调度方法

张佩云 凤 麒
(安徽师范大学数学计算机科学学院 芜湖 241003)

摘 要 为降低云计算中工作流调度的时间和成本,提出了一种双向调度算法,以实现后向Backward和前向Forward的双向调度。首先,Backward算法按照每个任务的最迟开始时间进行后向调度;此基础上,为降低虚拟机调度费用,Forward算法尽可能地提前调度每个任务,且在前向调度过程中充分考虑到工作流deadline、最大cost及传输时间的限制,从而实现对虚拟机的动态调度。由实验可知,本算法比BDA算法以及ICPCP算法更节约虚拟机调度成本,提高了调度的灵活性。

关键词 云计算,虚拟机,工作流,双向调度
中图法分类号 TP391 **文献标识码** A

Method of Workflow Bi-directional Scheduling in Cloud Computing Environment

ZHANG Pei-yun FENG Qi
(School of Mathematics and Computer Science, Anhui Normal University, Wuhu 241003, China)

Abstract To reduce the time and cost of workflow scheduling in cloud computing, we proposed a bi-directional scheduling algorithm including two sub-algorithms, which are Backward and Forward. Firstly, the Backward algorithm achieve a scheduling according to the deadline start time for each task scheduling. Then, to reduce the cost of scheduling, the Forward algorithm schedules each task in advance as much as possible. In the process of forward scheduling, taking the deadline and the biggest cost and transmission time into consideration, the algorithm achieves dynamic scheduling. The experiment results show that our algorithm is better than BDA algorithm and ICPCP algorithm for lower rent cost and higher scheduling flexibility.

Keywords Cloud computing, Virtual machine, Workflow, Bi-directional scheduling

1 引言

调度问题有着广泛的应用前景,国内外研究者针对不同的应用领域,不断探索新的调度策略和方法。随着云计算的兴起,需要根据云计算的特点、用户的偏好以及实际的情况为用户动态地提供调度服务。当用户通过租用云端的虚拟机来完成一系列工作流调度任务时,如果租用资源过少,执行时间可能会过长,出现一些不可预知的突发情况(如节点失效等问题)的概率增加,鲁棒性则会降低,影响系统的性能,最坏情况可能是违反SLA[1](Service Level Agreement)协议,从而因违约要对用户做出赔偿;而当用户租用的资源过多时,就会使大量的空闲时间没有得到利用,对资源造成浪费。因此,亟需适用于资源自动调节策略并且同时满足系统性能的调度算法。

2 研究现状分析

工作流调度是一种多约束满足问题,例如对于工作流中的任务之间相互依赖关系、任务与任务之间的传输时间、任务类型的约束满足。云资源提供商往往会基于租赁的销售模式以及基于使用量和性能标准的计费模式对用户应用所提供的计算服务进行收费。不同的调度算法直接关系到云计算的整体稳定性,并且在保证整个工作流按时完成(deadline)的情况下[2],尽可能为用户省钱[3]。从云服务供应商角度出发,既要保证云端QoS[4],又要考虑到负载均衡问题[5],以及利益最大化问题[6]。因此,需要设计良好的调度算法,以实现云计算资源更好的共享。另外,在调度过程中,往往需要考虑到用户偏好问题。

工作流任务调度是一个NP-hard问题[7],目前大多数调度算法从云服务商[8]角度出发,最大化地使用云端的固定资源数量,从而产生最大效益。但这些算法存在把工作流中的任务理想化成同一种类型任务[9]、没有考虑到云端虚拟机按小时收费的策略[10]、把云中资源想象成同构资源[11]、没有考虑到预算的限制[2]以及未考虑到任务之间的通讯量[12]等问题,容易造成在实际执行过程中用户成本的增加。此外,文献[13]认为云资源是单一化的,而这与现实不相符,因为在云端其资源总量、类别数量都比较多,不可能单一化;文献[14]采用deadline-assignment策略,将整个工作流的deadline约束到各个子任务中,存在着资源利用率不高的问题;文献[15-17]研究了关于云计算的资源动态分配与共享技术,主要集中

本文受国家自然科学基金项目(61472005,61201252),安徽省自然科学基金项目(1308085MF100)资助。
张佩云(1974—),女,博士,教授,CCF会员,主要研究方向为云计算、服务计算与智能信息处理等,E-mail:zpyustc@ustc.edu.cn 凤 麒(1989—),男,硕士生,主要研究方向为云计算。

在无用户差别情况下的资源动态分配策略，但云计算最终对象是不同的用户，而用户不同的行为习惯决定了不同的服务质量要求。此外，云计算环境下按小时收费的策略，使得用户面临着如何减少因剩余时间(partial hour)而额外花费的用户费用问题。例如，用户的某任务总共需要调度执行 1.2 小时，但云端服务提供商要按 2 小时收取费用，额外的 0.8 小时的收费给用户造成了不少损失，如果云端把这 0.8 小时另外租给其他用户，既可以避免用户的额外损失，又可以最大化云端服务提供商的自身利益。因此，需要设计合理的调度算法。

针对当前研究中存在的因剩余时间而导致的虚拟机资源利用效率不高、没有考虑到用户差别化等问题，从用户角度出发，针对虚拟机按小时收费模型，提出一种动态的双向调度算法。该算法在满足用户指定的截止时间的同时，又可以根据剩余时间实际情况动态地对虚拟机租用数量进行调节，从而进一步节约用户成本，满足用户灵活调度的个性化需求。

3 工作流及虚拟机映射

3.1 工作流描述

图 1 给出了一个基于 DAG 图的工作流实例。

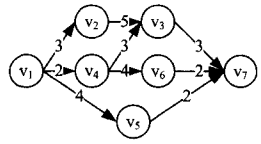

图 1 基于 DAG 图的工作流示例

图 1 所示工作流总共有 7 个节点，v_1 到 v_7 代表各个任务，箭头方向从父任务节点指向子任务节点，表示调度时节点之间的先后关系，箭头上的数字代表两个节点之间的通信时长(单位：min)。令 $W=\{v_1,\cdots,v_n\}$ 表示一个有 n 个任务(节点)的工作流，W 中除首、尾任务外，其它任务都有直接父任务和直接子任务。用集合 $E=\{e_{ij}\}$ 表示父子任务之间拥有的直接有向边，其中，$e_{ij}=(v_i,v_j)$，$1\leqslant i<j\leqslant n$。只有当某子任务的所有父任务都执行完成后，该子任务才可以开始执行。当子任务与其父任务在同一个虚拟机上执行时，数据传输时间忽略不计。用 $TT(v_i,v_j)$ 表示任务 v_i 和任务 v_j 之间的传输时间，用 D 表示工作流 W 的截止时间 deadline。

与双向调度算法有关的定义如下。

定义 1(工作流最迟结束时间 LFT，Latest Finish Time)
$LFT(v_{exit})=D$，$LFT(v_c)=\min LFT(v_i)-TT(v_i,v_c)-d^t_{v_i}$。

定义 1 中，v_{exit} 表示工作流的尾节点(该节点只是一个虚拟的最后节点，与其它节点之间并无通信量，本身也无计算量)，v_i 和 v_c 均为工作流中的节点，且 v_c 是 v_i 的子节点，$d^t_{v_i}$ 代表任务 v_i 在所分配的虚拟机 t 上的执行时间。

定义 2(最早开始时间 EST，Earliest Start Time)
$EST(v_{entry})=0$，$EST(v_i)=\max EST(v_p)+TT(v_p,v_i)+d^t_{v_p}$。

定义 2 中，v_{entry} 表示工作流的首节点，v_p 和 v_i 均为工作流中的节点，且 v_p 是 v_i 的父节点，$d^t_{v_p}$ 代表任务 v_p 在所分配的虚拟机 t 上的执行时间。

定义 3(最迟开始时间 LST，Latest Start Time)
$LST(v_i)=LFT(v_i)-d^t_{v_i}$。$d^t_{v_i}$ 的含义同定义 1。

定义 4(实际开始时间 AST，Actual Start Time)
$AST(v_i)$ 指任务 v_i 在所分配的虚拟机上实际的开始时间。

3.2 虚拟机类型映射

在云端有多种类型的虚拟机，每种都有不同的配置，主要表现为内核数、CPU 频率、存储空间大小、I/O 带宽的不同。表 1 给出了 Rackspace 公司的云平台虚拟机配置及收费。

表 1 Rackspace 公司云平台虚拟机配置及收费标准

虚拟机类型		配置	价格(美元/小时)
Normal types	Small(N_S)	1.7GB MEM. 1 EC2 CU	0.06
	Medium(N_M)	3.75GB MEM. 2 EC2 CU	0.15
	Large(N_L)	7.5GB MEM. 8 EC2 CU	0.36
	Extral Large(N_EL)	15GB MEM. 8 EC2 CU	0.9
High_memory types	Extra Large(M_EL)	17.1GB MEM 6.5EC2 CU	0.8
	Double Extra Large(M_DL)	34.2GB MEM 13 EC2 CU	1.9
	Quardrupe Extra Large(M_QEL)	68.4GB MEM 26 EC2 CU	4.3
High-CPU types	Medium(C_M)	1.7GB MEM 5 EC2 CU	0.145
	Extra Large(C_EL)	7 GB MEM 20 EC2 CU	0.87

由表 1 可知，云供应商所提供的虚拟机有多种类型且按小时进行收费，不同类型的虚拟机收费标准也不相同，不同云供应商收费也不同。表 1 中所分 Normal type、High-memory type、High-CPU type 3 种类型的虚拟机分别适用于普通任务、存储密集型任务、CPU 密集型任务。参照文献[12]，图 1 中任务执行时间及执行成本如表 2 所列。

表 2 图 1 中任务的执行时间以及执行成本

任务 / 虚拟机类型	v_1		v_2		v_3		v_4		v_5		v_6		v_7	
	Time (min)	Cost ($)	Time (min)	Cost ($)	Time (min)	Cost ($)	Time (min)	Cost ($)	Time (min)	Cost ($)	Time (min)	Cost ($)	Time (min)	Cost ($)
N_S	138	0.14	86	0.09	91	0.09	208	0.21	61	0.06	135	0.14	171	0.17
N_M	69	0.17	43	0.11	45	0.11	104	0.26	30	0.08	61	0.15	85	0.21
N_L	34	0.20	21	0.13	22	0.14	52	0.31	15	0.09	30	0.18	42	0.25
N_EL	17	0.26	10	0.15	11	0.16	26	0.39	7	0.11	15	0.23	21	0.32
M_EL	21	0.28	13	0.17	14	0.18	32	0.43	9	0.12	18	0.24	26	0.35
M_DEL	10	0.32	6	0.19	7	0.22	16	0.51	4	0.13	9	0.29	13	0.41
M_QEL	5	0.36	3	0.22	4	0.25	8	0.57	2	0.14	4	0.29	6	0.43
C_M	27	0.07	21	0.05	17	0.04	83	0.20	14	0.03	135	0.33	42	0.10
C_EL	6	0.09	5	0.07	5	0.06	20	0.29	3	0.04	32	0.46	10	0.15

表 2 给出各个任务在不同类型虚拟机上的执行时间。在双向调度算法实现前，首先需要解决虚拟机的映射问题，以保证执行成本最小，同时要满足工作流的 deadline 限制，该映射问题是一个混合整数线性规划问题。约定图 1 中的工作流在规定的 deadline 时刻完成，设 deadline 时长为 120min，采用 CPLEX 实现任务与虚拟机类型之间的映射，过程如下：在 eclipse中引入 CPLEX. jar 文件，通过调用 AddMinimize 添加优化目标(cost)，并通过 IRange 来定义约束(time)。映射结果如表 3 所列。

表 3　基于表 2 的任务与虚拟机类型映射

任务	v_1	v_2	v_3	v_4	v_5	v_6	v_7
虚拟机类型	C_M	C_M	C_M	C_EL	C_M	N_L	C_EL

由表 3 知，CPLEX 算法执行后，v_1 被映射到 C_M 类型虚拟机上，v_2 被映射到 C_M 类型的虚拟机上，v_7 被映射到 C_EL 类型的虚拟机。映射结果在第 4 节的双向调度算法中将被用到。

CPLEX 实现了把任务映射到高性价比的虚拟机类型上，但是具体是该类型的哪个虚拟机并没有给定。为合理调度到具体的高性价比虚拟机，本文设计了双向调度算法，该算法包括 Backward 和 Forward 两个子算法。首先，采用 Backward 算法把任务调度到相应的虚拟机的具体时间戳，即把所有任务的“最后通牒”时间段规定好。其次，由于在对应的虚拟机上，该任务的具体时间戳的前面必然有大量的空闲时间，为了进一步利用这些空闲时间，采用 Forward 算法根据任务的最早开始时间和最迟开始时间，尽最大可能地接近最早开始时间去调度该任务，节省调度时间和用户开支。

4　双向调度算法

本文设计的双向调度算法的具体思路为：

首先采用 Backward 算法从尾节点开始调度，在工作流的 deadline 时刻准时完成，同时保证每个任务都正好在最迟时刻完成。

其次采用 Forward 算法，根据虚拟机的空闲情况，动态调节任务的时间戳，根据每个任务的最早开始时间，从头节点开始，对每个任务进行向前调度，并将该任务标记为已分配任务。之后再动态调整后续任务的最早开始时间。调度时，在不违反任务之间相互依赖关系以及不超出工作流 deadline 时，尽可能地将同类型虚拟机上的任务合并在一个收费周期(为 60min)内，从而不需要重新租用额外的虚拟机时间，以尽可能地为用户节省费用。

4.1　Backward 算法

输入参数：初始工作流(包括所有任务及任务间的前后关系、任务之间的传输时间、任务通过 CPLEX 所映射到的虚拟机类型、工作流 deadline)。

输出参数：经过 Backward 算法后的工作流(包括所有任务通过 CPLEX 所映射的具体虚拟机的执行时间、任务在具体虚拟机上所分配的时间段)。

Step1　从工作流尾节点开始向前计算每个任务的 $LST(v_i)$ 以及 $LFT(v_i)$，按任务最迟完成时间递减排序，依次分配每个任务，被分配以后的任务标记为 assigned，否则标记为 unassigned。

Step2　如果任务 v_i 所有直接后继节点都分配在同一个虚拟机上，并且 v_i 所映射到的虚拟机类型与后继节点所分配的虚拟机类型相同，同时，在该虚拟机时间间隔$[LST(v_i)+TT(v_p,v_i),LFT(v_i)+TT(v_p,v_i)]$空闲时，把 v_i 分配到该虚拟机上并标记为 assigned；否则跳转到 Step3。

Step3　计算 v_i 所有直接后继节点，求出具有最小 $\min LFT(v_c)+TT(v_c,v_i)$ 的任务，如果任务 v_c 所租用的虚拟机与任务 v_i 所映射的虚拟机是同类型，同时在$[LST(v_i)+TT(v_i,v_c),LFT(v_i)+TT(v_p,v_i)]$空闲时，将 v_i 分配到该虚拟机上，并标记为 assigned；否则跳转到 Step4。

Step4　在所有同类已分配的虚拟机上寻找$[LST(v_i),LFT(v_i)]$是否存在空闲段，如果有，则把 v_i 分配到该时段，并标记为 assigned；否则跳转到 Step5。

Step5　由于该时间戳找不到可以分配的虚拟机，因此重新租用一个该类的虚拟机(即与 v_i 所映射的虚拟机同类型)，把 v_i 分配在时段$[LST(v_i),LFT(v_i)]$内。跳转到 Step1，继续分配 unassigned 的任务，直到所有任务都标记为 assigned。

Step6　返回经过 Backward 算法后的工作流(包括任务通过 CPLEX 所映射的具体虚拟机的执行时间、任务在具体虚拟机上所分配的时间段)。

针对图 1 中的工作流，执行 Backward 算法进行调度，最终结果如图 2 所示。

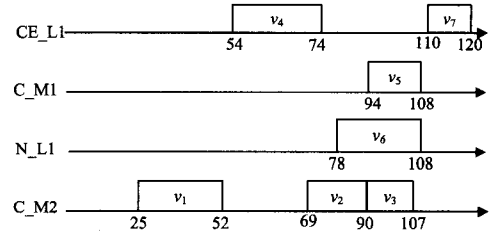

图 2　Backward 算法调度的最终结果

图 2 中，设用户规定的 deadline 为 120 min，计算每个任务最迟开始时间 $LST(v_i)$，可知，v_7-v_1 的 $LST(v_i)$ 分别为 110min、94min、78min、90min、69min、54min、25min。由表 3 可知，v_7 分配到 CE_L 类型的虚拟机上，该虚拟机执行 v_7 只需要 10min，由于 $LST(v_7)$ 为 110min，可把 CE_L1 时间戳[110,120]分配给 v_7。由表 3 可知，v_5 被分配到 C_M 类型虚拟机，v_5 与其直接后继节点所需类型虚拟机不同，因此需要租用 C_M1 虚拟机，同理，将[94,108]时间戳分配给 v_5。继续分配 v_6，同理需要租用 N_L1 虚拟机，将[78,108]时间段分配给 v_6。接着对 v_3 进行分配，由表 3 可知，v_3 被映射到了 C_M 类型虚拟机上，由于 v_5 已经租用了 C_M1 虚拟机但是在[93,110]时间段不空闲，那么只能重新租用新的 C_M 类型虚拟机 C_M2，把时间段[90,107]分配给 v_3。接着对 v_2 进行分配，由表 3 可知，v_2 被映射到了 C_M 类型虚拟机上，其最迟开始时间到最迟完成时间为[64,85]，v_2 的唯一直接后继节点 v_3 也是采用 C_M 类型虚拟机并且在[64,85]空闲，即把 v_2 分配在该虚拟机 C_M2，时间段为[69,90]，可以把最迟开始时间和最迟结束时间都向后延迟 5 分钟，因为原本其最迟开始时间到最迟结束时间为[64,85]，但是由于其孩子节点 v_3 与其在同一个虚拟机上执行，它们之间的传输时间即可省去，所以最迟开始时间和最迟结束时间都会向后推迟。继续分配 v_4，根据 CPLEX 映射关系，其被映射到 CE_L 类型虚拟机上，最迟开始时间到最迟完成时间为[54,74]。由于 v_4 的两个直接后

继节点 v_3 和 v_6 所使用的虚拟机的类型与 C_EL 不同，即只能查看是否有类型为 CE_L 的虚拟机在[54,74]上空闲，经计算发现 CE_L1 满足要求，将 CE_L1 在[54,74]时间段分配给 v_4。最后分配 v_1，其映射虚拟机类型是 C_M 类型，并且最迟开始时间到最迟完成时间为[25,52]，由于其所有直接后继节点不都是与它同类型，而且其直接后继节点最早的最迟开始时间是 54min，所属任务是 v_4，所以只能找所有 C_M 类型虚拟机在[25,52]是否空闲。由图 1 可知，任务 v_1 和 v_4 之间的传输时间为 2min，因此，要留下 2min 为 v_1 和 v_4 之间的传输时间，由于 C_M1 和 C_M2 虚拟机均满足分配条件，通过随机选择，把任务 v_1 分配给 C_M2。

4.2 综合影响因素指数

基于 Backward 算法结果，Forward 算法主要从 3 个方面考虑如何选择虚拟机以及时间戳的分配问题。

(1)ESTF(最早开始时间影响因素)

让 AST 与 EST 尽可能地接近，以便于后面任务有更多的时间戳去选择，$ESTF=(AST(v_i)-EST(v_i))/LST(v_i)$。在选择时间戳分配时，$ESTF$ 越小越好。

(2)WTF(浪费时间影响因素)

记任务 v_i 的开始时间为 $ST(v_i)$，结束时间为 $FT(v_i)$。租用虚拟机的时间为 HP(分钟)，$HP=60*i, i=1,\cdots,n$。HP_{left} 代表该虚拟机租用的起始时间，HP_{right} 代表该虚拟机租用的截止时间。当任务 v_i 分配到该虚拟机上时，该任务左右各有两种空闲时间戳(分别记为 $Slot_{left}$ 和 $Slot_{right}$)。左边的空闲时间戳 $Slot_{left}$ 的范围是[$Slot_{start}$, $ST(v_i)$]或者是[HP_{left}, $ST(v_i)$]，同理右边的空闲时间戳的范围是[$FT(v_i)$, $Slot_{end}$]或者是[$FT(v_i)$, HP_{right}]，任务 v_i 浪费的总时间戳 $Slot_{sum}=Slot_{left}+Slot_{right}$。浪费时间影响因素 WTF 示例如图 3 所示。

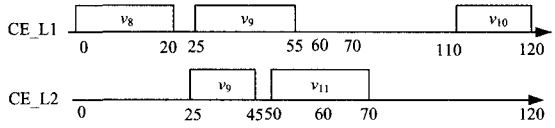

图 3 浪费时间影响因素实例

图 3 中，在 CE_L1 虚拟机上，v_9 左边空闲时间戳为[20,25]，而在 CE_L2 虚拟机上中，v_9 左边空闲时间戳为[0,25]。以 60 分钟为一个租赁周期，60 是分水岭，因此，在 CE_L1 虚拟机上 v_2 右边空闲时间戳为[55,60]，而在CE_L2虚拟机上为[45,50]。

(3)ACF(额外租用的费用影响因素)

$ACF=Addedhour(v_i,slot)*60/(d^t_{v_i}+TT(v_p,v_i))$

当任务 v_i 在被分配的虚拟机 t 上时，函数 $Addedhour(v_i,slot)$用于计算执行任务 v_i 所需要增加的额外虚拟机的小时数，$d^t_{v_i}$ 代表任务 v_i 在虚拟机 t 上的执行时间，$TT(v_p,v_i)$表示任务 v_p 和任务 v_i 之间的传输时间，v_p 是 v_i 的直接父节点。额外租用的费用影响因素 ACF 的示例如图 4 所示。

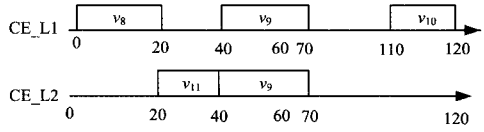

图 4 额外租用的费用影响因素实例

如果 v_9 被分配到图 4 的 CE_L1 虚拟机上，则不需要额外增加收费，因为 v_9 利用的是该虚拟机调度 v_8 和 v_{10} 之间的剩余时间。但是若将 v_9 分配到 CE_L2 虚拟机上，还需要增加一个小时的付费，因为在时间戳为[60,120]时，该虚拟机并未分配其它任务，所以仍需租用一个小时去执行 v_9，而在 CE_L1上即可省去此次费用。

定义 5(任务最优放置指数 $Index(v_i,slot)$) 上述 ESTF、WTF、ACF 3 个因素的结合，将决定任务分配到哪个空闲时间戳，如式(1)所示。

$$Index(v_i,slot)=ESTF*a+WTF*b+ACF*c \quad (1)$$

式中，a、b、c 是权重，范围为[0,1]，用户根据自己的偏好进行设定，本文 a、b、c 取值均为 1。如果用户对于整个工作流时间要求尽可能快地完成，在满足小于用户 $cost$ 的限制条件下，可以增大权重 a 的值。任务最优放置指数 $Index(v_i,slot)$值越小，表示可以提前知道任务浪费的时间戳越小，因此能减少因虚拟机时间剩余而浪费的额外费用。

定义 6(总费用 $Cost_{sum}$) 计算所有已分配虚拟机的总费用(向上取整)，如式(2)所示。

$$Cost_{sum}=\lfloor \sum_{i=1}^{n} d^t_{v_i}/60 \rfloor * price(t) \quad (2)$$

式(2)用于计算租用的 n 个虚拟机的总费用。$price(t)$代表第 t 种类型虚拟机每小时的收费标准，$d^t_{v_i}$ 表示任务 v_i 在所分配的 t 类虚拟机上执行需要的时间(单位:min)，$\lfloor d^t_{v_i}/60 \rfloor * price(t)$表示任务 v_i 在所分配的 t 类虚拟机上执行所需要的总费用。$\lfloor d^t_{v_i}/60 \rfloor$表示 $d^t_{v_i}$ 除以 60 并且向上取整，以求出该虚拟机总共使用多少小时，充分体现了云计算按小时收费的特点。

4.3 Forward 算法

输入参数:经过 Backward 算法后的工作流(包括所有任务及任务间的前后关系、任务之间的传输时间、任务通过 CPLEX 所映射的具体虚拟机的执行时间、任务在具体虚拟机上所分配的时间段、工作流 deadline 和 $price(t)$)、用户偏好 $(cost,time)$。/ * $price(t)$代表第 t 种类型虚拟机每小时的收费标准 * /。

输出参数:调整后的工作流调度信息 $FeasibleSet$(包括各任务分配到的具体虚拟机及任务在其上的调度时间)。

Step1(初始化) 由于是前向调度，先将工作流中的 v_1 放入集合 SET 中。

Step2 计算 SET 集合中所有任务 v_i 的 $EST(v_i)$。

Step3 取出 SET 集合任务，对 SET 中的任务：

Step4 分别计算它们在 Backward 算法中所分配到的虚拟机以及与该虚拟机同类的虚拟机的空闲 slot。对于每个空闲的 slot，尽可能越早分配任务 v_i 越好，计算该任务分配的开始时间与结束时间，并且计算并找到最小值 $Index(v_i,slot)$所在的 slot，把任务 v_i 分配到该 slot 上，并把该任务标记为已分配。按式(2)计算 $Cost_{sum}$，如果 $Cost_{sum}$ 小于用户规定 $cost$，则跳转到 Step5，否则跳转到 Step6。

Step5 把该次 Forward 调度结果加入到 $FeasibleSet$ 集合中。

Step6 更新 SET 集合，即把本次 EL 集合中的所有已分配好的任务移去，换成它们的孩子任务。如果 EL 集合不为空，跳转到 Step2，否则返回调整后的工作流调度信息 $FeasibleSet$。

在 Step1 中，由于要实现从 v_1 节点开始 Forward 调度，

因此将 v_1 先放入到集合 SET 中。以图 2 中工作流的 v_5 为例，采用以最省钱为调度目标的 Forward 算法调度，如图 5—图 8 所示。v_5 分配之前的虚拟机状态如图 5 所示。

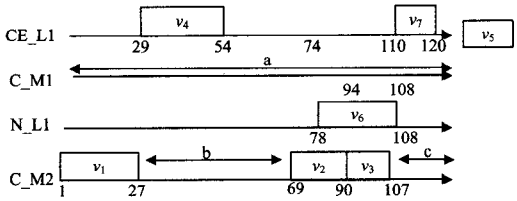

图 5　v_5 分配之前的虚拟机状态

图 5 中，v_5 放在图 5 的最右边，表示其还没有被分配虚拟机。由于 Backward 算法为每个任务设定了最迟开始时间，Forward 算法需要将 v_5 尽可能早地分配到某虚拟机上。由表 3 可知，v_5 被映射到 C_M 类型虚拟机上，又由图 5 可知，所有已经分配的 C_M 类型虚拟机有 C_M1 和 C_M2，这两个虚拟机上总共有 a、b、c 3 个空闲时间戳。由 Forward 算法可知，由于 c 违反了 v_5 的最迟开始时间限制，因此首先排除 c。v_5 将在 a 或者 b 中进行分配，排除 c 之后，v_5 在虚拟机上有两种可选择状态，如图 6 所示。

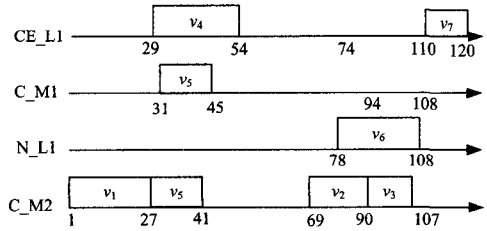

图 6　排除 c 之后 v_5 在虚拟机上的两种可选择状态

从图 6 可知，在排除时间戳 c 之后，只能从时间戳 a（见图 5 的 C_M1）或时间戳 b（见图 5 的 C_M2）中选择一个。由于是按时间调度的，因此，可通过计算时间戳 a 和时间戳 b 的综合影响因素指数的“任务最优放置指数”来确定，即计算时间戳 a 和时间戳 b 的任务最优放置指数 $Index(v_5, slot)$，计算公式见式(1)，此处，式(1)中 a、b、c 取值均为 1。

针对时间戳 a，计算 $Index(v_5, slot)$：$ESTF=0$，$WTF=(31+15)/60=0.76$，$ACF=60/14=4.2$，代入式(1)得 $Index(v_5, slot)=0+0.76+4.2=4.96$。针对时间戳 b，计算 $Index(v_5, slot)$：$ESTF=0$，$WTF=19/60=0.31$，$ACF=0$，得 $Index(v_5, slot)=0+0.31+0=0.31$。可知，$4.96>0.31$。由定义 5 可知，任务最优放置指数 $Index(v_i, slot)$ 值越小越好，所以 v_5 分配给了图 6 中的 C_M2。v_5 被分配后，虚拟机最终状态如图 7 所示。

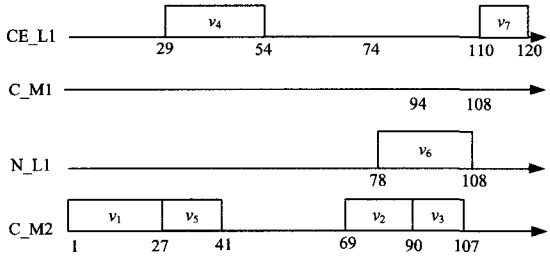

图 7　v_5 分配后虚拟机最终状态

从图 7 可知，由于 v_5 没有租用 C_M1 虚拟机而节省了费用，否则 v_5 将按一个周期（小时）付费。v_5 被分配在虚拟机 C_M2 上，通过和 v_1 共同租用 C_M2 的一个周期，达到了省钱的目的。同理，后面的任务继续使用 Forward 算法。基于双向调度算法，图 8 给出了虚拟机针对图 2 所示的整个工作流的最终调度结果。

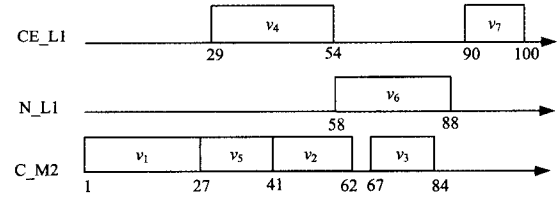

图 8　虚拟机调度整个工作流的最终结果

图 8 所示的整个工作流在虚拟机上调度的最终结果是所有调度方案中最节约的一种，在于对每个任务分配虚拟机时都求出综合影响因素指数，通过选出最小的空闲时间戳进行分配，以节省租用虚拟机的费用。

4.4　算法时间复杂度

Backward 算法中时间复杂度为 $O(N^2)$，其中 N 表示任务数。在 Forward 算法中，由于出现了空闲时间戳选择，假设总共有 H 个空闲时间戳需要去选择，则双向调度算法的总时间复杂度为 $O(N^2H)$。

5　实验仿真和结果分析

5.1　实验环境和平台

实验的硬件环境为 Intel Core 1.86GHz CPU，4GB 内存，320GB 硬盘，编程语言为 Java。软件环境为 Win7 操作系统，Eclipse 以及云计算仿真软件 Cloudsim 仿真器。Cloudsim 是由澳大利亚墨尔本大学的网络实验室研究发布的，对云计算开发和仿真具有很好的效果[8]。用 ILOG CPLEX 解决任务与虚拟机类型的匹配的最小线性二乘问题，将 ICPCP[9] 算法、BDA[6] 算法与本文算法进行比较。

5.2　实验分析

取工作流包含的任务数为 100，共有如表 2 所列的 9 种类型虚拟机。考虑到工作流的复杂度 OS[11] 参数（OS 是指整个工作流中实际 path 路径条数除以该工作流可以容纳的最大 path 条数），分别取 $OS=0.2$、$OS=0.3$、$OS=0.4$ 这 3 种情况。工作流 deadline 的生成方法：$D=D_{min}+(D_{max}-D_{min})*\theta$，其中 D_{min} 是工作流执行时间的最小值，D_{max} 是工作流执行时间的最大值，θ 为系数，取值范围为 0 到 1，以保证取到相对合理的 deadline，实验中取 $\theta=0.15$、$\theta=0.3$ 两种情况进行分析。工作流中的任务节点路径将根据 OS 取值随机产生[6]。当 θ 取 0.15 时，本文算法与其它算法在执行费用方面的对比结果如图 9 所示。

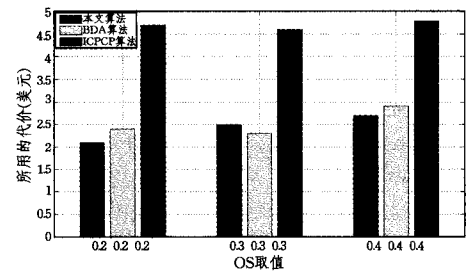

图 9　θ 取 0.15 时本文算法在执行费用上与其它算法的对比

从图 9 可以看出，当 θ 取 0.15 时，在 $OS=0.2$ 和 $OS=0.4$ 两种情况下，本文算法比 ICPCP 算法和 BDA 算法进一步

节约了成本，尤其与 ICPCP 算法相比，优势非常明显。

随着 θ 增大，本文算法的优势更加明显，如 θ 取 0.3 时，对比结果如图 10 所示。

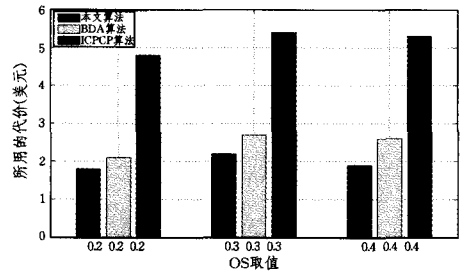

图 10 θ 取 0.3 时本文算法在执行费用上与其它算法的对比

由图 10 可知，随着 deadline 限制的逐步减小，本文算法与 BDA 以及 ICPCP 算法进行比较，平均降低 20%～60%的成本，体现出了本文方法节约成本的明显优势。

结束语 由于虚拟机按周期调度，而当任务执行时间小于虚拟机周期时，将会因产生剩余时间而出现虚拟机浪费成本问题。为减少当前云计算中工作流任务调度成本及等待时间，提出了一种双向调度算法。该算法包括了 Backward 以及 Forward 两部分调度，实现将任务调度到具体类型的虚拟机，该算法充分考虑到云计算环境下的虚拟机按小时收费特点以及任务之间传输时间限制，在调度具体虚拟机时，不仅要满足任务按照用户所规定的 deadline 完成，还要满足用户支付费用最小的要求。通过实验可以看出，本文算法比 ICPCP 以及 BDA 算法进一步节约了费用。

参 考 文 献

[1] Dastjerdi A V, Buyya R. An autonomous reliability-aware negotiation strategy for cloudcomputingenvironments [C] // 12th IEEE/ACM International Symposium on Cluster, Cloud andGrid Computing (CCGrid). IEEE, 2012: 284-291

[2] Abrishami S, Naghibzadeh M, Epema D. Deadline-constrained workflow scheduling algorithms for iaas clouds[J]. Future Generation Computer Systems, 2013, 29(1): 158-169

[3] Zhou A C, He B S. Transformation-Based Monetary Cost Optimizations for Workflows in the Cloud[J]. IEEE Transactions on Cloud Computing, 2014, 2(1): 85-98

[4] Rimal B P, Jukan A, Katsaros D, et al. Architectural requirements for cloud computing systems: an enterprise cloud approach[J]. J Grid Comput, 2011, 9(1): 3-26

[5] Singh S, Chana I. QoS-aware resource scheduling framework in cloud computing[J]. Journal of Supercomputing, , 2015, 71(1): 241-292

[6] Toosi A N, Calheiros R N, Thulasiram R K, et al. Resource Provisioning Policies to Increase IaaS Provider's Profit in a Federated Cloud[C]// Proceedings of the 13th IEEE International Conference on High Performance Computing and Communications (HPCC'11). IEEE Computer Society, 2011: 279-238

[8] Kwok Y K, Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors[J]. ACM Computing Surveys(CSUR), 1999, 31(4): 406-471

[8] Bajaj R, Agrawal D P. Improving scheduling of tasks in a heterogeneous environment[J]. IEEE Transactions on Parallel and Distributed Systems, 2004, 15(2): 107-118

[9] Poola D, Garg S K, Buyya R, et al. Robust Scheduling of Scientific Workflows with Deadline and Budget Constraints in Clouds [C] // 2014 IEEE 28th International Conference on Advanced Information Networking and Applications. 2014: 858-865

[10] Yuan Y, Li X, Wang Q, et al. Deadline division- based heuristic for cost optimization in workflow scheduling[J]. Information Sciences, 2009, 179(15): 2562-2575

[11] Pandey S, Karunamoorthy D, Buyya R. Workflow engine for clouds[M]// Cloud Computing. 2013: 321-344

[12] Byun E K, Kee Y S, Kim J S, et al. Cost Optimized Provisioning of Elastic Resources for Application Workflows[J]. Future Generation Computer System, 2011, 27(8): 1011-1026

[13] 苑迎春，李小平，王茜，等. 成本约束的网格工作流时间优化方法[J]. 计算机研究与发展，2009，46(2)：194-201

[14] 李强，郝沁汾. 云计算环境中虚拟机放置的自适应管理与多目标优化[J]. 软件学报，2011，34(12)：2253-2266

[15] 张伟哲，张宏莉，张迪. 云计算平台中多虚拟机内存协同优化策略研究[J]. 计算机学报，2011，34(12)：2266-2277

[16] 周景才，张沪寅，查文亮，等. 云计算环境下基于用户行为特征的资源分配策略[J]. 计算机研究与发展，2014，51(5)：1108-1119

[17] Cai Z C, Li X P, Chen L, et al. Bi-direction adjust heuristic for Workflow Scheduling in Clouds[C]// 19th IEEE International Conference on Parallel and Distributed System. 2013: 94-101

(上接第 420 页)

[4] 刘鹏. 云计算[M]. 北京：电子工业出版社，2010

[5] 陈康，郑纬民. 云计算：系统实例与研究现状[J]. 软件学报，2009，20(5)：1337-1348

[6] Lenk A, Klems M, Nimis J, et al. What's inside the Cloud? An Architectural Map of the Cloud Landscape[C]// Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing. 2009: 23-31

[7] Martin D, Burstein M, Hobbs J, et al. OWL-S: Semantic Markup for Web Services [OL]. http://www.w3.org/Submission/OWL-S/

[8] Clement L, Hately A, von Riegen A, et al. UDDI Version 3.0.2 [OL]. http://uddi.org/pubs/uddi-v3.0.2-20041019.htm

[9] Papazoglou M P, Traverso P, Dustdar S, et al. Service-oriented Computing: State of the Art and Research Challenges[J]. IEEE Computer, 2007, 40(11): 38-45

[10] Wang Pei, Jiang Chao-hui, Kang Zhi-qian. Research on ESB-based Enterprise Application Integration [C] // Second Asia-Pacific Conference on Information Processing. 2010

[11] Papazoglou M P. Web Services-Principles and Technology[M]. Prentice Hall, 2008

[12] Weske M, et al. Business Process Management[M]// Concepts, Languages, Architectures. Springer, 2007

[13] Bures M, Jelinek I. Framework for Easy and Effective Implementation of Adaptive Features in Web Portal[C]// International Conference on Computers and Advanced Technology in Education. Crete, Greece, 2008

[14] Peter D, Antoine B. Mule 2: A Developer's Guide[M]. Apress, 2008

[15] Mule[OL]. http://www.mulesoft.com/