基于动态关键路径的云工作流调度算法*

陶 勇[1]，沈济南[1,2†]

(1. 湖北民族学院 信息工程学院，湖北 恩施 445000；2. 华中科技大学 计算机科学与技术学院，武汉 430074)

摘 要：为了提高资源行为动态异构的云环境中工作流任务的调度效率，提出了一种基于动态关键路径的工作流调度算法 CWS-DCP。将工作流任务结构定义为有向无循环图 DAG 模型，改进了传统关键路径的一次性搜索模式，结合云资源可用性动态可变的特征，以动态自适应方式搜索关键路径，并确定关键任务；在关键任务调度后，局部 DAG 的关键路径搜索根据资源可用性再次迭代更新，从而动态决策任务与资源间的调度方案。通过仿真实验，构建了三种不同类型的工作流结构作为测试数据源，并与其他六种同类型的启发式和元启发式算法进行了性能比较。实验结果表明，在资源可用性动态改变和工作流规模不断增大的情况下，CWS-DCP 算法在多数工作流结构中均能得到执行跨度更好的调度方案和更少的调度开销。

关键词：云计算；工作流调度；关键路径；执行跨度

中图分类号：TP301.6 **文献标志码**：A **文章编号**：1001-3695(2018)05-1500-06

doi:10.3969/j.issn.1001-3695.2018.05.048

Cloud workflow scheduling algorithm based on dynamic critical path

Tao Yong[1], Shen Jinan[1,2†]

(1. *School of Information & Engineering, Hubei University for Nationalities, Enshi Hubei* 445000, *China*; 2. *School of Computer Science & Technology, Huazhong University of Science & Technology, Wuhan* 430074, *China*)

Abstract: For improving the scheduling efficiency of workflow tasks in cloud environment with dynamic and heterogeneous behaviors of resources, this paper proposed a cloud workflow scheduling algorithm CWS-DCP based on dynamic critical path. This algorithm defined workflow tasks' structure as directed acyclic graph (DAG) model and improved the one-time search pattern of the traditional critical path. Combined with the characteristics of dynamic and variability of cloud resources, the algorithm dynamically and self-adaptively searched the critical path and determined the critical task of DAG. Meanwhile, after scheduling the critical task, the searching of critical path in the partial DAG would be iteratively updated again according to resources' availability, which could dynamically determine the mapping between tasks and resources. Through the simulation experiments, this paper constructed three different types of workflow structures as the testing data source, and compared the performance with other six same types of heuristics and meta-heuristics. Experimental results show that under the condition of dynamic and changing resource availability and continuously increasing workflow sizes, CWS-DCP can generate a better scheduling scheme of execution makespan and less scheduling overhead for most of the workflow types.

Key words: cloud computing; workflow scheduling; critical path; execution makespan

0 引言

大多数大规模科学应用通常表现为复杂的科学工作流形式，包含了一个通过数据依赖性连接而成的有序任务集合[1]。工作流调度系统即使用具体的调度策略实现可用云资源与工作流任务间的映射，并满足用户需求[2]。传统的工作流调度环境多为静态的，即资源可用性在保持不变的情况下进行工作流任务的调度。作为新兴的计算模式，云计算可以按即付即用和按需提供的方式提供各种互联网资源[3]，因此，云资源的计算性能和可用状态是动态变化的，该环境下的工作流调度将更为复杂。本文设计一种动态自适应的工作流调度算法，通过寻找动态的关键路径，从而动态地以更低的调度开销降低整个工作流的执行跨度。

目前，云工作流调度问题已经成为云计算领域中的研究热点。相关研究中，文献[4]提出的 HEFT 是一种异构最快完成时间工作流调度算法，旨在通过赋予任务不同的优先级，最小化工作流的总调度时间；文献[5]提出基于遗传算法 GA 的调度算法，将染色体表示为任务资源映射方案，同时表示任务执行序列，算法根据最优化目标对个体进行进化，并基于约束目标进化种群，但算法的进化代数过多，可能无法找到可行解；文献[6]提出一种基于动态规划的云工作流调度算法，有效解决了资源价格变化环境中任务调度的开销优化问题；文献[7]提出 fussy-PSO 算法尝试使用粒子群优化 PSO 算法实现工作流调度时执行跨度与执行代价的均衡。以上算法在进行任务映射时，其针对的对象多为静态行为的资源，没有考虑资源使用本身的动态性，可能导致任务调度的开销过大。

基于关键路径的启发式算法是解决工作流依赖任务间调度问题的有效手段，其目标是寻找任务 DAG 中所有执行路径

收稿日期：2017-01-17；**修回日期**：2017-03-30 **基金项目**：国家自然科学基金资助项目(61662022)；湖北省自然科学基金资助项目(2016CFB371)

作者简介：陶勇(1973-)，男，讲师，硕士，主要研究方向为云计算；沈济南(1980-)，男(通信作者)，副教授，硕士，主要研究方向为云计算、云安全(shenjinan@163.com).

中的最长路径(关键路径),从而得到整个工作流的最小执行跨度[8]。显然,在单个任务被调度后,关键路径会发生动态变化,而传统的动态关键路径调度算法仅仅实现了同质环境中的任务与资源映射问题,即调度方案仅在任务 DAG 中计算一次。由于云计算环境是动态异构环境,本文的目标即是扩展传统的动态关键路径以适应于云工作流的动态调度环境。

1　工作流调度问题

通常,工作流应用以有向无循环图(directed acyclic graph, DAG)表示,图的节点表示工作流任务,图的边表示任务间的依赖性;节点权重表示任务计算复杂性,边的权重则表示任务间的通信数据量。因此,工作流调度问题通常可视为一种 DAG 调度问题,为 NP 完全问题。

令 $W(T,E)$ 表示云工作流,T 表示工作流任务集,$T=\{T_1,T_2,\cdots,T_x,\cdots,T_y,T_n\}$;$E$ 表示任务间依赖关系集,$E=\{\langle T_a,T_b\rangle,\cdots,\langle T_x,T_y\rangle\}$,其中,$T_x$ 为 T_y 的父任务。$R=\{R_1,R_2,\cdots,R_x,\cdots,R_y,R_m\}$ 表示云可用资源集,工作流的入口任务不存在父任务,出口任务不存在子任务;同时,只有任务的所有父任务完成后,该任务才能执行。在调度发生的任意时间,其所有父任务已经完成的任务称为就绪任务(ready task)。工作流调度问题即是寻找任务与云资源间的映射方案,使得工作流执行跨度 makespan 或调度长度达到最小化。

2　CWS-DCP 算法设计

在工作流任务 DAG 中,单个任务开始时间的下限与上限可分别表示为绝对最早开始时间 AEST 和绝对最晚开始时间 ALST。传统的动态关键路径算法中,由于关键任务的延迟会影响任务 DAG 的整体执行时间,所以处于关键路径上的任务拥有相同的 AEST 和 ALST。处于关键路径上的第一个任务首先被调度,直至所有任务被调度完成为止。然而,以上的调度方法仅适用于调度资源不受限且不考虑计算和通信时间的环境,云计算提供的计算、存储、带宽资源拥有不同的能力和可用性,是异构动态环境,因此,本文作了如下几点改进:

a)对于单个任务,其初始 AEST 和 ALST 由为该任务提供最小执行时间的资源计算得到,总体目标是降低每次通过的关键路径的长度;

b)为了实现关键路径上的任务映射,所有可用的云资源均被考虑进来,由于在异构云环境中资源的可用性会发生变化,而通信与计算时间会受此影响;

c)当任务映射至资源时,其执行时间和父任务间的数据传输时间进行实时更新,这会改变后续任务的 AEST 和 ALST。

为了便于 CWS-DCP 算法的描述,表 1 给出了本文相关的符号及其含义说明。

表 1　符号说明

参数	配置
$AET(t)$	任务 t 的绝对执行时间
$ADTT(t)$	任务 t 的绝对数据传输时间
$AEST(t,R)$	资源 R 上任务 t 的绝对最早开始时间
$ALST(t,R)$	资源 R 上任务 t 的绝对最晚开始时间
$C_{t,tk}(R_t,R_{tk})$	R_t 上任务 t 与 R_{tk} 上任务 t_k 间的数据传输时间
$PC(R)$	资源 R 的处理能力
$BW(R)$	与资源 R 连接的网络带宽能力
DCPL	工作流动态关键路径的长度

2.1　AEST 和 ALST 计算

CWS-DCP 算法中,任务的开始时间直到被映射至资源上才被确定下来。此时,引入任务的绝对执行时间 AET 和绝对数据传输时间 ADTT 两个属性,其中,AET 表示任务的最小执行时间,ADTT 表示在当前部署状态下传输任务输出数据的最小时间。初始状态下,AET 和 ADTT 的计算方法为

$$AET(t)=\frac{task\_size(t)}{\max_{k\in resourceList}\{PC(R_k)\}} \tag{1}$$

$$ADTT(t)=\frac{task\_output\_size(t)}{\max_{k\in resourceList}\{BW(R_k)\}} \tag{2}$$

其中:$PC(R_k)$ 表示资源 R_k 的处理能力;$BW(R_k)$ 表示资源 R_k 的传输能力(带宽)。

当任务 t 调度至资源时,$AET(t)$ 和 $ADTT(t)$ 根据式(1)(2)进行更新。因此,资源 R 上任务 t 的 AEST 可表示为 $AEST(t,R)$,以递归形式表示为

$$AEST(t,R)=\max_{1\leqslant k\leqslant p}\{AEST(t_k,R_{tk})+AET(t_k)+C_{t,tk}(R_t,R_{tk})\} \tag{3}$$

其中:t 拥有 p 个父任务,t_k 表示第 k 个父任务,且 a)如果 t 为入口任务,则 $AEST(t,R)=0$;b)如果 $R_t=R_{tk}$,则 $C_{t,tk}(R_t,R_{tk})=0$;c)如果 t 和 t_k 未被调度,则 $C_{t,tk}(R_t,R_{tk})=ADTT(t_k)$。

此时,如果两个任务被调度至相同资源,则其通信时间等于 0,如果子任务仍未被调度,则其通信时间等于父任务的 ADTT。利用以上定义,AEST 可以通过从入口任务开始以宽度优先方式遍历任务 DAG 的方式计算 AEST。计算所有任务的 AEST 后,可以计算动态关键路径长度 DCPL,即部分已映射工作流的调度长度(时间),定义为

$$DCPL=\max_{1\leqslant i\leqslant n}\{AEST(t_i,R_{t_i})+AET(t_i)\} \tag{4}$$

其中:n 表示工作流的任务总数量。

计算 DCPL 后,ALST 可以通过反向宽度优先方式遍历任务 DAG 的方式计算得到。因此,资源 R 上任务 t 的 ALST 可表示为 $ALST(t,R)$,以递归形式表示为

$$ALST(t,R)=\min_{1\leqslant k\leqslant c}\{ALST(t_k,R_{tk})-AET(t)-C_{t,tk}(R_t,R_{tk})\} \tag{5}$$

其中:t 拥有 c 个子任务,t_k 表示第 k 个子任务,且 a)如果 t 为出口任务,则 $ALST(t,R)=DCPL-AET(t)$;b)如果 $R_t=R_{tk}$,则 $C_{t,tk}(R_t,R_{tk})=0$;c)如果 t 和 t_k 未被调度,则 $C_{t,tk}(R_t,R_{tk})=ADTT(t_k)$。

2.2　任务与资源选择

任务调度过程中,任务 DAG 的关键路径决定了部分调度工作流的调度长度,因此,必须赋予关键路径上的任务优先级。然而,随着调度过程的进行,关键路径是动态变化的,即由于资源行为的动态变化可能导致在当前步骤中关键路径上的任务在下一步骤中可能不是关键路径上的任务,所以在云计算环境中,工作流的关键路径是动态可变的。

动态关键路径上的任务有着相同的开始时间上限和下限,即相同的 AEST 和 ALSP。因此,如果任务的 AEST 和 ALST 是相同的,则动态关键路径的任务也在关键路径上,称为关键任务。为了降低每一步中的 DCPL,调度过程中选择的任务需要处于关键路径上,且不存在未被映射的父任务,即选择的为拥有最低 AEST 的关键路径。

确定关键任务后,需要为任务选择合适资源,此时选择的资源是为任务提供最小执行时间的资源。该过程可以通过遍

历所有可用资源，寻找在相同资源上关键子任务的最小开始时间的任务，该子任务即为关键任务所有子任务中拥有最小的 AEST 与 ALST 之差的任务。最后，关键任务被映射至能够提供最早开始时间的资源上。

2.3 算法步骤

以下伪代码给出了 CWS-DCP 算法的具体执行过程。

```
Input: Workflow W(T,E), TashDependencyList, resource set R.
Output: Mappling strategy.
for all t ∈ T of workflow W
    compute AET and ADTT for t
end for
for all t ∈ T of workflow W
    compute AEST for r running BFS
end for
compute DCPL
for all t ∈ T of workflow W
    compute ALST for t running BFS following reverse task dependency
end for
while all tasks in T are not completed do
    TaskList←get unscheduled ready tasks for workflow W
    schedule tash(TaskList,R)
    update TaskDependencyList
end while

Schedule task
Input: TaskList, resource set R.
while TaskList is not empty do
    T_ct←get critical task from all tasks of TaskList
    T_ctc←get critical chile task from all child task of T_ct
    r←get a resource from R that can provide earliest start time for both T_ct and T_ctc
    schedule T_ct on r
    update status of r
    for all t ∈ T of workflow W
        compute AEST for t running BFS
    end for
    cmpute DCPL
    for all t ∈ T of workflow W
        compute ALST for t running BFS following reverse task dependency
    end for
end while
```

算法说明：首先，CWS-DCP 算法计算所有任务的初始 AET、ADTT、AEST 和 ALST；然后选择 AEST 与 ALST 之差最小的任务，并通过选择拥有最小 AEST 的任务切断与前驱任务的连接。根据之前的讨论，处于动态关键路径 DCP 上的该任务即为关键任务；关键任务的关键子任务也通过这种方式确定。接着，算法计算关键任务在所有可用资源上的开始时间（同时考虑了其所有父任务的完成时间），选择为任务 t_{ct} 和其关键子任务提供了最早开始时间的资源作为目标资源。选择合适资源 R 后，算法计算开始时间 AEST(t_{ct},R) 和在该资源上任务 t_{ct} 的持续时间 AET(t_{ct})，并更新任务 t_{ct} 的实际开始时间和执行时间。其他任务的 AEST 和 ALST 在每次调度的结束进行更新，以决定下一个关键任务。该过程进行至工作流中所有任务被调度时结束。

2.4 算例说明

为了进一步说明 CWS-DCP 算法的思想，图 1 给出了以 CWS-DCP 算法进行工作流任务调度的算例过程。样本工作流由五个任务组成，表示为 $\{T_0,T_1,T_2,T_3,T_4\}$，各任务拥有不同的执行时间和数据传输需求。图 1(a)给出了每个任务的大小和输出数据的大小，分别表示为 MI（百万指令数，million instruction）和 GB，任务被调度至两个云资源 R_1 和 R_2，其处理能力表示为 PC，传输能力表示为 BW（带宽），如图 1 所示。

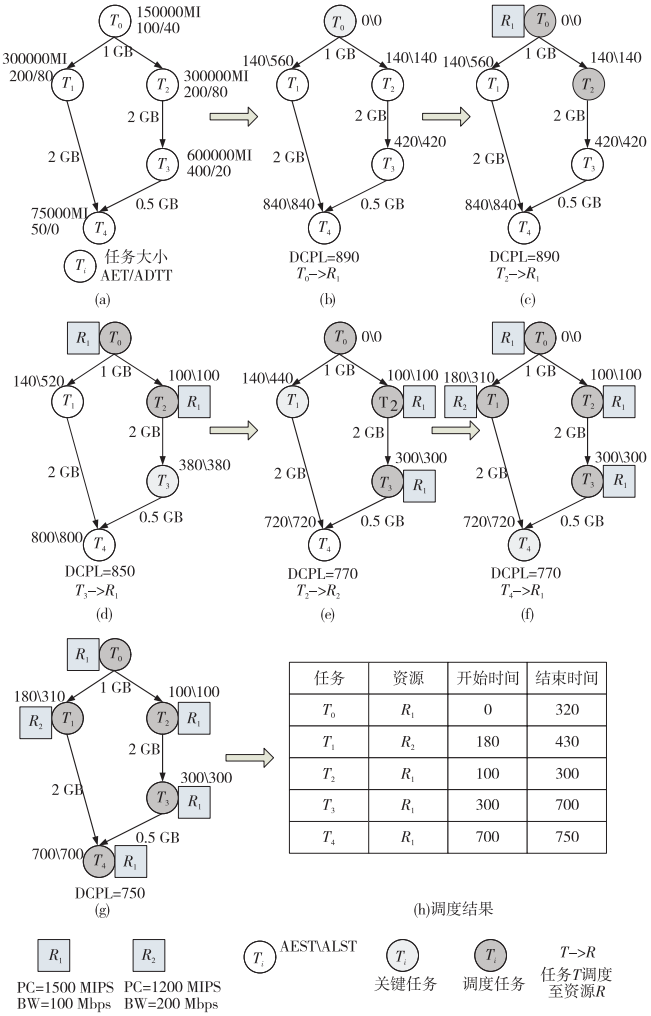

图1 算例说明

算例说明：图 1(a)计算每个任务的 AET 和 ADTT，然后，使用以上得到的值计算所有任务的 AEST 和 ALST，如图 1(b)所示。由于 T_0、T_2、T_3 和 T_4 拥有相同的 AEST 和 ALST，则这四个任务均在关键路径上，且 T_0 作为最高任务。所以，选择 T_0 作为关键任务并被映射至 R_1，由于 R_1 带给 T_0 最小开始时间。该步骤完成后，可以计算此时工作流的调度长度 DCPL 为 890。类似地，图 1(c)中，选择 T_2 作为关键任务并被映射至 R_1。由于 T_0 和 T_2 均被映射至 R_1，T_0 的数据传输时间此时为 0，所有任务的 AEST 和 ALST 均发生改变，工作流的调度长度 DCPL 变为 850，如图 1(d)所示。在下一步，T_3 被映射至 R_1，由于 T_2 的数据传输时间为 0，所以 DCPL 减少至 770。

当前，T_4 是唯一处于关键路径上的任务，如图 1(e)所示。然而，T_4 的子任务之一 T_1 仍然没有被映射，因此，选择 T_1 作为关键任务。由于 T_2 和 T_3 已经被映射至 R_1，R_1 上 T_1 的开始时间为 700，另外在 R_2 上其开始时间和结束时间分别为 180 和 430，T_1 被映射至 R_2。最后，当 T_4 映射至 R_1 时，如图 1(g)所示，所有任务已被映射，工作流的调度长度 DCPL 无法进一步改进，得到最终 DCPL 为 750。由 CWS-DCP 得到的最终调度方案如图 1(h)所示。

3　性能评估

为了评估 CWS-DCP 算法的性能,利用 CloudSim[9] 构建云工作流调度环境,并建立了不同类型的工作流模型与同类型算法进行性能比较。

3.1　工作流模型

为了模拟产生不同格式、不同权重的工作流模型,实验中将以下参数作为工作流模型的输入参数:

a)N 表示工作流任务的总数量。

b)α 表示形状参数,即任务总量与宽度(即同一层次上节点的最大数量)的比例,因此,宽度 $W=\text{int}[N/\alpha]$。

c)工作流类型。提供三种不同类型的工作流:并行工作流、fork-join 工作流和随机工作流。

(a)并行工作流。并行工作流中,一组任务形成一个拥有单入口任务和出口任务的任务串,多个任务串形成一个工作流,因此,单个任务的执行仅取决于某个任务,而任务串的串首任务取决于入口任务,出口任务取决于所有串尾任务。并行工作流的层次数为

$$\text{层次数}=\text{int}[\frac{N-2}{W}] \tag{6}$$

(b)Fork-join 工作流。Fork-join 工作流中,任务先分支再连接,因此,只拥有一个入口任务和出口任务,在每个层次上的任务数量取决于任务总数和层次宽度 W。Fork-join 工作流的层次数为

$$\text{层次数}=\text{int}[\frac{N}{W+1}] \tag{7}$$

(c)随机工作流。随机工作流中,单个任务与其他任务的依赖性与父任务数量即是工作流 DAG 中节点的入度,均是随机产生的。此时,任务依赖性和入度定义为

$$\max\ \text{indegree}(T_i)=\text{int}[\frac{W}{2}] \tag{8}$$

$$\min\ \text{indegree}(T_i)=1 \tag{9}$$

$$\text{parent}(T_i)=\{T_x \mid T_x\in[T_0,\cdots,T_{i-1}]\}\quad \text{if } T_i \text{ is not a root task} \tag{10}$$

$$x \text{ is a random number and } 0\leqslant x\leqslant \text{int}[W/2] \tag{11}$$

$$\text{parent}(T_i)=\{\varnothing\}\quad \text{if } T_i \text{ is a root task} \tag{12}$$

图 2 给出以上三种工作流模型的示意图,其中 $N=10$,$\alpha=5$。仿真中,利用百万指令数 MI 表示任务长度,兆字节数 MB 表示任务输出数据量大小。

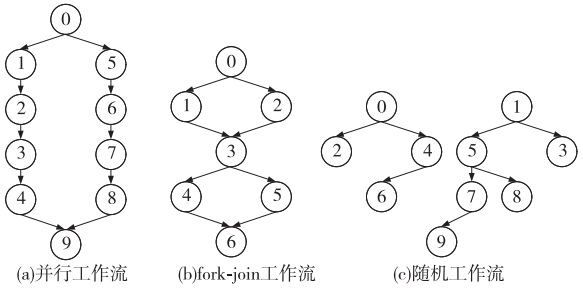

图2　三种工作流结构

3.2　资源模型

工作流任务的执行环境模拟为异构资源环境,即设置不同处理能力的异构云资源供任务调度。参考文献[10],设置八种不同地域不同能力的云资源,具体参数如表 2 所示。其中,资源的处理能力以每秒百万指令数 MIPS 和每秒兆位带宽 Mbps 度量。

表2　资源配置

资源名称	地点	节点数量	PE/MIPS	平均负荷
RAL	UK	41	1 140	0.9
NorduGrid	Norway	17	1 176	0.9
NIKHEF	Netherlands	18	1 166	0.9
Milano	Italy	7	1 000	0.5
Torino	Italy	4	1 330	0.5
Catania	Italy	5	1 200	0.6
Padova	Italy	13	1 000	0.4
Bologna	Italy	20	1 140	0.8

3.3　比较算法

a)Myopic 算法[11]。将未映射的任意就绪任务调度至能最早完成任务的资源上,直到所有任务被调度。

b)Min-min 算法[12]。基于任务在资源上的期望完成时间 ECT 最小为任务分配优先级,将任务分组并迭代调度,每次迭代中,寻找最小 ECT 中使得总时间达到最小的调度方案。

c)Max-min 算法[13]。与 min-min 类似,区别在于算法为拥有最长执行时间 ECT 的任务分配优先级。

d)HEFT 算法[4]。异构最早完成时间算法,给予拥有更高等级的工作流任务更高优先级,该等级使用每个任务的平均执行时间和两个连续任务在资源上的平均通信时间计算得到。

e)GRASP 算法[14]。贪婪随机自适应搜索算法,利用贪婪思想搜索整个解空间(工作流任务与所有可用资源),将最优解保留至最终解中,搜索过程直到达到最大迭代次数为止。

f)GA[5]。算法利用遗传进化的思想,通过对解空间的快速搜索和对个体的适应度评价,不同于 GRASP 的随机搜索方式,可以在更短时间内得到较优解。

按算法分类,Myopic、min-min、max-min 和 HEFT 均属于启发式算法,而 GRASP 和 GA 则属于元启发式算法,以上算法均是分布式计算环境中常用的工作流调度算法。

3.4　实验参数

a)工作流类型为并行工作流、fork-join 工作流和随机工作流。通过配置三种不同类型的工作流拓扑结构,可以观察和评估算法在搜索关键路径时与任务结构的关系。

b)$N=\{50,100,200,300\}$。通过在工作流中配置不同数量的任务,可以观察算法性能与工作流任务间的关系。

c)$\alpha=\{10\}$。该参数值随机选取,由于工作流的结构宽度 $W=\text{int}[N/\alpha]$,所以该值主要影响拓扑结构中同一层次上的任务最大数量。

为了使实验结果更加具有普遍性,设置每个工作流任务的大小均匀分布于区间[100000 MI,500000 MI],任务的输出数据量大小均匀分布于区间[1 G,5 G]。同时,对于 GRASP 算法,调度迭代次数为 600 次。对于 GA,参数参考原始配置,如表 3 所示。

表3　GA 参数

参数	配置	参数	配置
种群大小	60	适应度函数	工作流执行跨度 makespan
交叉概率	0.7	选择策略	轮盘赌策略
对换变异概率	0.5	迭代次数	300
替换变异概率	0.8	初始个体	随机生成

3.5　实验结果分析

为了评估算法在工作流执行跨度上的性能,本文设计了两

种不同的实验场景,第一种场景考虑为静态场景,即云资源的可用性和负荷保持为静态不变,根据不同的调度算法映射任务至资源并执行调度;第二种场景考虑为现实场景,即云资源的可用性和负荷为动态改变的,此时,仿真期间每个资源的瞬时负荷(占用 PE 数量)服从高斯分布。

1)静态场景中的执行跨度 makespan

图 3 给出了算法在三种工作流结构中任务数分别为 50、100、200 和 300 时的工作流执行跨度。对于随机工作流(图 3(c)),CWS-DCP 的调度 makespan 比 HEFT 低 13% 左右,而 HEFT 则优于 Myopic、min-min 和 max-min,主要是由于随机工作流中任意任务与出口任务间拥有多条路径,而 CWS-DCP 通过为任务动态分配优先权可以形成更优的解。由于 GRASP 和 GA 搜索了最优解的全部空间,比较 CWS-DCP 节省了 20% ~ 30% 的 makespan。

对于 fork-join 工作流(图 3(b)),启发式算法与元启发式算法性能差别较大。任务选择过程中,启发式算法没有考虑映射子任务的影响,因此,所有启发式算法与 CWS-DCP 拥有类似的结果。然而,在 fork-join 工作流中,连接交叉点任务取决于上层所有分支非依赖任务的输出。如果连接点任务分配至与其他资源带宽较低的资源,数据传输时间的增加会影响工作流执行跨度。而元启发式算法 GRASP 和 GA 不仅考虑了父分支任务对映射的影响,而且考虑了子分支任务的影响,因此,比较 CWS-DCP,性能提升了 40% ~50% 。

对于并行工作流(图 3(a)),其执行跨度随着工作流大小的变化表现出相对较慢的指数级增长,原因在于,不同于 fork-join 工作流,在调度每一步中,并行工作流中未被映射的就绪任务的数量恒等于 W,且只要其父任务完成,该任务即转变成就绪任务。因此,当可用资源量少于未被映射任务量时,任务等待被调度的时间将导致执行跨度的增加。同时,并行工作流中,CWS-DCP 和 GA 的结果优于其他算法,makespan 至少低于其他算法 20% 左右。此时,GRASP 的执行跨度高于 CWS-DCP,主要是由于任务映射候选解的数量会随着工作流大小的增长呈指数级增加。

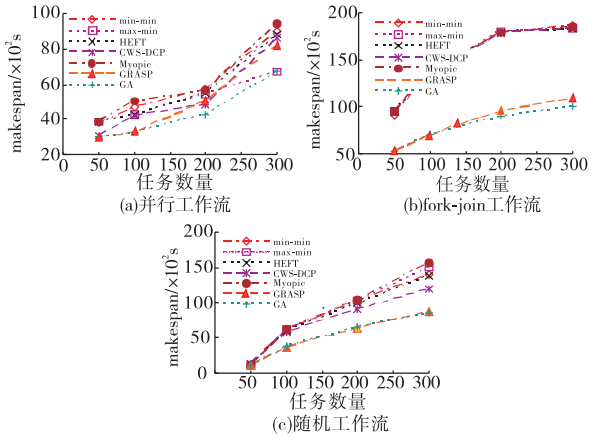

图3 静态场景中的makespans

2)动态场景中的执行跨度 makespan

由于动态环境中资源可用性是动态变化的,在确定周期中资源可用性信息需要连续不断更新,且任务需要根据资源可用性重新映射。此时,需要比较 CWS-DCP 与其他启发式调度算法和元启发式调度算法在静态场景下的结果。

图 4 显示了动态场景下的结果,其资源可用性的更新周期为 50 s。此时,资源可用处理元素 PE 数量和资源上可以开始执行的任务数量会随着资源负荷动态变化。对于 GRASP 和 GA 算法而言,如果资源负荷较重且不可用,被映射至该资源的任务需要等待执行,该等待时间会相应影响其他依赖任务的开始时间,并增加工作流执行跨度 makespan,GRASP 和 GA 的较差性能也可以反映出这一结果。而且,启发式算法的结果高于元启发式算法 30% 左右。在所有启发式算法中,CWS-DCP 可以比其他算法节省 6% 左右 makespan,主要是由于在 CWS-DCP 中,在负载较重的资源上等待执行的关键路径上的任务会被重新调度至拥有可用 PE 的资源上,这会降低关键路径长度,即工作流执行跨度 makespan 会降低。同时可以看出,对于相同类型的工作流,动态场景中的启发式算法性能优于静态场景的性能,主要原因是动态场景中的资源负荷和资源可用性是周期性更新的,这表明启发式算法可以更加适应于资源的可用性变化,产生调度方案的更优解。

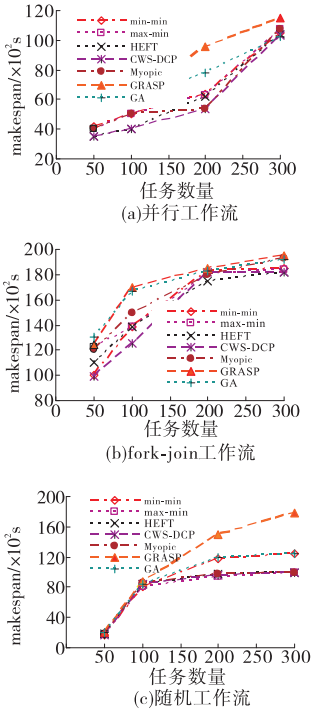

图4 动态场景中的makespans

3)算法调度时间比较

图 5 给出了不同类型工作流下算法的调度时间比较结果。调度时间即为调度开销,即调度算法产生调度解的运行时间。为了表述方便,单个任务的平均调度时间(单位:ms)得到的单个调度方案如表 4 所示。从表 4 可以看出,为产生一个调度方案,Myopic、min-min、max-min 和 HEFT 需要接近 1 ms 产生调度解,而 CWS-DCP 需要 16 ~ 17 ms,且不会随着工作流类型发生变化,主要是由于该算法的任务选择与工作流结构是无关的。

GRASP 的调度时间不仅会随着工作流任务的增加呈指数级增长,而且与工作流结构也密切相关。在每次迭代中,GRASP 为每个未被映射的就绪任务建立约束候选资源列表 RCL,然后随机为任务选择资源映射。当任务增加时,RCL 会呈指数级增长,从而导致调度时间的增加,而 RCL 本身的大小又依赖于工作流结构的不同。例如,如果工作流由 300 个任务组成,并行和 fork-join 工作流结构在每个层次包括 30 个任务,

随机工作流结构层次与每个层次上的任务数量均是随机的。因此在每一步骤中,并行工作流拥有 30 个就绪任务,fork-join 工作流拥有最多 30 个就绪任务,随机工作流每个层次上的平均就绪任务数量低于 30。因此,随机工作流的调度时间是最低的,并行工作流的调度时间是最高的。同时,GA 的调度时间不会因为工作流类型的变化而变化,由于 GA 始终执行相同数量的遗传操作,与工作流类型无关。但是解空间中每个个体的大小等于工作流任务的数量,所以调度时间会随着工作流大小的增加而增加。

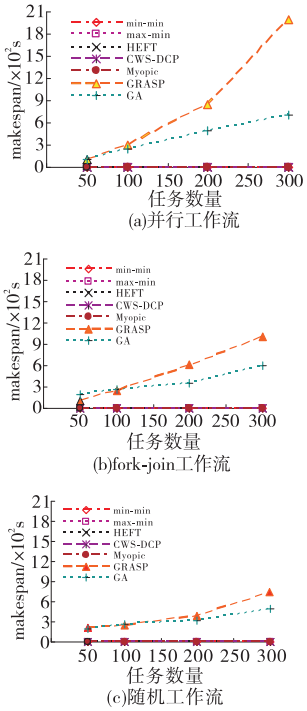

图5　算法调度时间

表 4　单个任务的平均调度时间　　/ms

算法	随机工作流	fork-join 工作流	并行工作流
Myopic	1	1	1
min-min	1	1	1
max-min	1	1	1
HEFT	1	1	1
CWS-DCP	17	16	16
GRASP	1 180	2 840	5 720
GA	1 940	1 780	1 750

综上所述,结合图 3,很明显在启发式算法中,静态场景中 CWS-DCP 的性能可以提升 20%,尤其对于随机工作流和并行工作流,且与工作流类型无关。在随机工作流和 fork-join 工作流中,GRASP 和 GA 的性能优于 CWS-DCP,但是其调度时间也更高。对于任务数为 300 的并行工作流,CWS-DCP 需要 6 s 将任务映射至资源,而 GRASP 和 GA 分别需要 580 s 和 2 076 s。

在动态场景中,启发式算法可以自适应于资源的动态特征并避免性能降低。但是,元启发式算法在静态场景中且由于确定间隔内映射资源的不可用而表现更差。同时在动态场景中,无论工作流类型和大小的不同,CWS-DCP 的性能均优于其他算法。

4　结束语

云资源的动态行为特征导致云工作流调度不同于传统静态分布式计算环境中的调度问题。针对该问题,提出了一种动态自适应工作流调度算法 CWS-DCP,算法通过不断迭代计算工作流任务 DAG 中的关键路径,为关键路径上的任务分配优先级的方式,有效实现了工作流任务与云资源间的映射调度。实验结果表明,在资源可用性动态改变的情况下,CWS-DCP 算法在多数工作流结构中均能得到更好的调度方案,且与工作流规模无关。进一步的研究将关注多 QoS 参数下的工作流调度问题,如考虑任务与服务之间映射时的资源可靠性问题或服务资源执行任务时的能效问题,设计多目标最优化算法实现多 QoS 指标的同步优化,并以 Pareto 最优评估多目标优化性能。

参考文献:

[1] Liu Li, Zhang Miao, Lin Yuqing, *et al*. A survey on workflow management and scheduling in cloud computing[C]//Proc of the 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Washington DC: IEEE Computer Society, 2014: 837-846.

[2] Vöckler J S, Juve G, Deelman E, *et al*. Experiences using cloud computing for a scientific workflow application[C]//Proc of the 2nd International Workshop on Scientific Cloud Computing. New York: ACM Press, 2011: 15-24.

[3] Cusumano M. Cloud computing and SaaS as new computing platform [J]. Communication of the ACM, 2011, 53(4): 27-29.

[4] Topcuoglu H, Hariri S, Wu Minyou. Performance-effective and low-complexity task scheduling for heterogeneous computing[J]. IEEE Trans on Parallel and Distributed Systems, 2002, 13(3): 260-274.

[5] Huang Jie. The workflow task scheduling algorithm based on the GA model in the cloud computing environment[J]. Journal of Software, 2014, 9(4): 873-880.

[6] 郑敏,曹健,姚艳. 面向价格动态变化的云工作流调度算法[J]. 计算机集成制造系统,2013,19(8):1849-1858.

[7] Ritu G, Singh A K. Multi-objective workflow grid scheduling using ε-fuzzy dominance sort based discrete particle swarm optimization[J]. Journal of Supercomputing, 2014, 68(2): 709-732.

[8] Wu Zhangjun, Liu Xiao, Ni Zhiwei, *et al*. A market-oriented hierarchical scheduling strategy in cloud workflow systems[J]. Journal of Supercomputing, 2013, 63(1): 256-293.

[9] Cacheiros R N, Ranjan R, Beloglazov A, *et al*. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms[J]. Software: Practice and Expe-rience, 2011, 41(1): 23-50.

[10] Venugopal S, Buyya R. A set coverage-based mapping heuristic for scheduling distributed data-intensive applications on global grids [C]//Proc of the 7th IEEE/ACM International Conference on Grid Computing. Washington DC: IEEE Computer Society, 2006: 238-245.

[11] Wieczorek M, Prodan R, Fahringer T. Scheduling of scientific workflows in the ASKALON grid enviornment[J]. ACM SIGMOD Record, 2005, 34(3): 56-62.

[12] Maheswaran M, Ali S, Siegel H J, *et al*. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems[C]//Proc of the 8th Heterogeneous Computing Workshop. Washington DC: IEEE Computer Society, 1999: 30-44.

[13] Mandal A, Kennedy K, Koelbel C, *et al*. Scheduling strategies for mapping application workflows onto the grid[C]//Proc of the 14th IEEE International Symposium on High Performance Distributed Computing. 2005: 125-134.

[14] Blythe J, Jain S, Deelman E, *et al*. Task scheduling strategies for workflow-based applications in grids[C]//Proc of IEEE International Symposium on Cluster Computing and the Grid. 2005: 759-767.