基于云科学工作流调度的代价与能效优化算法*

魏秀然[1], 王 峰[2]
(1. 河南农业大学 信息与管理科学学院, 郑州 450046; 2. 华北水利水电大学 软件学院, 郑州 450045)

摘 要: 为了降低云环境中科学工作流调度的执行代价与数据中心能耗,提出了一种基于能效感知的工作流调度代价最优化算法(CWCO-EA)。算法在满足截止时间约束下,以最小化工作流执行代价与降低能耗为目标,将工作流的任务调度划分为四步执行:通过代价效用的概念设计虚拟机选择策略,实现了子 makespan 约束下的任务与最优虚拟机间的映射;通过串行与并行任务合并策略,同步降低了工作流的执行代价与能耗;通过空闲虚拟机重用机制,改善了租用虚拟机的利用率,进一步提高了能效;通过任务松驰策略实现了租用虚拟机的能力回收,节省了能耗。通过四种科学工作流的仿真实验结果表明,CWCO-EA 算法比较同类型算法,在满足截止时间的同时,可以同步降低工作流的执行代价与执行能耗。

关键词: 云计算; 科学工作流; 代价最优化; 能耗; 截止时间
中图分类号: TP301.6 **文献标志码**: A **文章编号**: 1001-3695(2018)07-2001-05
doi:10.3969/j.issn.1001-3695.2018.07.018

Cost and energy-efficiency optimization algorithm based on cloud scientific workflow scheduling

Wei Xiuran[1], Wang Feng[2]
(1. *College of Information & Management Science, Henan Agricultural University, Zhengzhou* 450046, *China*; 2. *College of Software, North China University of Water Resources & Electric Power, Zhengzhou* 450045, *China*)

Abstract: In order to reduce the execution cost and energy consumption of data center in scientific workflow scheduling under cloud environment, this paper presented a workflow scheduling cost optimization algorithm based on energy-efficiency aware (CWCO-EA). Under satisfying deadline constraint, the proposed algorithm defined minimizing the workflow execution cost and reducing energy consumption as the objectives, and divided the task scheduling of workflow into four steps. First, it used the virtual machine (VM) selection strategy applying the concept of cost utility to map tasks to their optimal VM types by the sub-makespan constraint. Second, it employed merging strategies of the sequence tasks and parallel tasks to reduce execution cost and energy consumption of workflow. Then, it used the idle VM reuse mechanism to improve the utility of leased VMs and further improve the energy-efficiency. Finally, it utilized the scheme of slack time reclamation to save energy of leased VM. Through the simulation experiments of four real-world scientific workflow, the results show that, compared with the same types of related well-known algorithms, under meeting the deadline constraint, CWCO-EA can synchronously reduce the workflow execution cost and energy consumption.

Key words: cloud computing; scientific workflow; cost optimization; energy consumption; deadline

工作流通常由完成特定实际应用的有序任务集合组成,而科学工作流已被证实是并行和分布式系统中建模科学计算问题的最有效手段[1]。随着科学计算系统复杂度的提高,科学工作流越来越体现为大数据形式,包括数据密集型、通信密集型及计算密集型应用,这些应用任务的执行时间也越来越长[2]。传统集群与网格系统代价高昂,且资源扩展性较差,而云计算系统则以灵活的即付即用模式为科学工作流的调度与执行提供更有力的手段[3]。云计算是以经济驱动的大规模分布式计算模式,其资源和服务以按需申请的方式通过互联网提交到外部用户使用。由于其商业化的使用和定价模式,使用代价问题成为科学工作流调度面临的一大难题。同时,虽然云服务提供商以不同定价提供了不同类型的虚拟机(VM)实例,但对于调度系统而言仍然很难找到截止时间约束下的工作流最优映射策略。

除此之外,能耗问题是云环境中科学工作流调度的另一大难题。据报道,Amazon 云数据中心的 50% 预算用于能源消耗及服务器的冷却,其总能耗则占世界总能耗的 0.5%[4]。除了能耗成本,数据中心带来的大量的 CO_2 排放也导致了如温室效应等环境问题[5]。可以看出,高能耗将是环境与经济因素的双重问题,在进行科学工作流调度时,应尽可能降低能耗。本文的主要工作将重点关注于如何在满足截止时间约束下,设计同步优化工作流执行代价和执行能耗的调度算法。

1 相关工作

文献[6]提出一种异构最快完成时间调度算法(HEFT),通过赋予任务不同的优先级,实现最小化任务调度时间。文献[7]提出一种基于预算约束的异构最快完成时间算法(BHEFT),该算法是经典工作流调度算法 HEFT 的扩展,考虑了任务调度时的最优预算约束问题。文献[8]提出一种异构预算约束调度算法(HBCS),通过定义代价因子调整可用预算与最低廉价格可能性的比例,实现调度优化。HEFT、BHEFT 和 HBCS 算法均只进行了单目标优化,且都没有考虑数据中心能耗问题。而文献[9]提出的加强能效调度算法 EES 虽然考虑了能耗,但没有考虑执行代价。

多目标调度方面,文献[10]提出的加强 HEFT 算法(EHEFT)实现了 makespan 与能耗双目标最优化;文献[11]中的 NSPSO 算法和文献[12]中的 fussy-PSO 算法均尝试使用粒

收稿日期: 2017-03-22; **修回日期**: 2017-04-20 **基金项目**: 河南省重点科技攻关项目(152102210112);河南省教育厅科学技术研究重点项目(13A520713)

作者简介:魏秀然(1975-),女,河南郑州人,实验师,硕士,主要研究方向为软件技术、人工智能及其应用(weixiuran1975@21cn.com);王峰(1970-),男,河南汤阴人,副教授,硕士,主要研究方向为图像处理、软件技术.

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。文献[13]提出多目标异构最快完成时间算法(MO-HEFT),旨在通过 Pareto 线性启发式算法对 HEFT 进行扩展,实现多目标优化。然而,与本文工作相比,以上多目标调度算法均忽略了工作流任务特点及空闲 VM 的利用,且对能耗考虑较少。国内相关云工作流调度研究[14~16]中则对于能耗优化问题考虑较少。

不同于以上工作,本文将以同步降低工作流执行代价与数据中心能耗为目标,设计能效感知的工作流调度代价优化算法,使得工作流任务执行在满足截止时间约束的情况下,能够进一步降低工作流执行代价,并减少云数据中心的能耗。

2 系统模型

表 1 给出了本文算法描述过程中应用的符号及其含义说明。

表 1 符号说明

符号	参数含义	符号	参数含义
t_i	工作流中的任务 $i(i=1,2,\cdots,n$	c_k	vm_k 的单位时间代价
I_i	任务 t_i 的输入数据总量	v_k^j	vm_k 的第 j 级电压
O_i	任务 t_i 的输出数据总量	f_k^j	v_k^j 的 CPU 频率
w_i	任务 t_i 的负载量	P_k^j	vm_k 在第 j 级的功耗
$\mathrm{pre}(t_i)$	任务 t_i 的前驱任务集	E_k^j	vm_k 的运行能耗
$\mathrm{succ}(t_i)$	任务 t_i 的后继任务集	E_{idle}	空闲时间能耗
t_{entry}	工作流的入口任务	T_M	工作流执行跨度 makespan
t_{exit}	工作流的出口任务	T_D	工作流截止时间约束
vm_k	类型为 k 的 VM	$T(t_i,vm_k)$	任务 t_i 在 vm_k 上的执行时间
$p_k^{\max}$	vm_k 的最大计算能力	$T_{\mathrm{sub}}(t_i)$	任务 t_i 的子 makespan
$p_k^{\min}$	vm_k 的最小计算能力	$u(t_i,vm_k)$	任务 t_i 在 vm_k 上的代价效用
p_k^j	vm_k 在第 j 级的计算能力		

2.1 工作流模型

以有向无循环图 DAG 表示存在任务优先关系的工作流应用模型,表示为 $D(T,E)$,其中,$T=\{t_1,t_2,\cdots,t_n\}$ 表示 n 个任务的集合,E 表示图的有向边集合,即任务间的依赖关系。例如,边 $e(i,j)$ 表示任务 t_i 必须在任务 t_j 开始之前完成执行,此时,任务 t_i 称为 t_j 的直接前驱,任务 t_j 称为 t_i 的直接后继。令 $\mathrm{pre}(t_i)$ 表示任务 t_i 的前驱任务集合,$\mathrm{succ}(t_i)$ 表示任务 t_i 的后继任务集合,O_i 表示任务 t_i 完成后产生的输出数据。DAG 中不存在前驱任务的任务称为入口任务,不存在后继任务的任务称为出口任务,同时,假设 DAG 仅拥有一个入口任务 t_{entry} 和出口任务 t_{exit}。定义 T_M 表示整个工作流的执行时间 makespan,T_D 为用户定义的工作流执行的截止时间约束,表示对工作流执行时间的限制。

2.2 云资源模型

假设云数据中心提供 K 种类型 VM,表示为 $VM=\{vm_1,vm_2,\cdots,vm_k,\cdots,vm_K\}$,VM 使用以单位小时作为定价模型,如 Amazon EC2 提供四种类型的 VM 实例,包括 small(S)、medium(M)、large(L)和 extra-large(XL)。一个 VM 类型 vm_k 拥有以下几个特征属性:最大计算能力 $p_k^{\max}$,单位为每秒百万指令数(million instruction per second,MIPS),单位小时使用代价 c_k 和带宽 B。假设云数据中心中的所有 VM 拥有相同的通信带宽,且所有 VM 按照其最大计算能力进行升序排列,即 $p_1^{\max}<p_2^{\max}<\cdots<p_k^{\max}<\cdots<p_K^{\max}$,那么与计算能力排序一致,其代价排序为 $c_1<c_2<\cdots<c_k<\cdots<c_K$。假设 VM 的使用代价与性能满足式(1)。

$$\frac{c_k}{p_k^{\max}}<\frac{c_{k+1}}{p_{k+1}^{\max}} \tag{1}$$

式(1)表明计算性能越高,其单位性能代价也越高。因此,在不同的截止时间约束下,算法需要将工作流任务映射至不同类型的 VM 上执行,并且,单个任务的执行时间越长,其执行代价也越小。本文在工作流调度过程中,在 VM 上应用 DVFS 技术,以 p_k^j 表示 vm_k 在第 j 级 CPU 频率上的计算能力,其变化范围为 $[p_k^{\min},p_k^{\max}]$。

2.3 能量模型

工作流应用在云资源上执行过程中的能耗主要包括静态能耗 E_{static} 和动态能耗 E_{dynamic}。通常,动态能耗占总能耗较大比例,而静态能耗相对固定,本文暂时不考虑静态能耗部分。VM vm_k 在第 j 级电压上的动态功耗可表示为

$$P_k^j=\lambda_k\times(v_k^j)^2\times f_k^j \tag{2}$$

其中:λ_k 为常量因子,与 VM 类型与计算能力相关;v_k^j 表示 vm_k 的第 j 级供电电压;f_k^j 表示与 v_k^j 对应的 CPU 频率。通常,f_k^j 与 v_k^j 与 p_k^j 成正比,因此,对应的取值范围分别为 $[f_k^{\min},f_k^{\max}]$ 和 $[v_k^{\min},v_k^{\max}]$,则时间 t_{runtime} 内的能耗可表示为

$$E_k^j=P_k^j\times t_{\mathrm{runtime}}=\lambda_k\times(v_k^j)^2\times f_k^j\times t_{\mathrm{runtime}} \tag{3}$$

能耗为功耗与时间的乘积。在这种情况下,任务的执行时间与 CPU 频率是成反比的,而相应的能耗与电压的平方是成正比的,因此,降低电压可以有效降低能耗。例如,如果电压降低至 70%,则能耗相应会降低 50%。由于 CPU 的空闲时间中其供电电压与频率无法归零,所以可将其降至最低电压状态,以节省能耗。综上,空闲时间的能耗可表示为

$$E_{\mathrm{idle}}=\lambda_k\times(v_k^{\min})^2\times f_k^{\min}\times t_{\mathrm{idletime}} \tag{4}$$

那么,一个 VM 实例的总能耗可表示为

$$E_{\mathrm{total}}=E_k^j+E_{\mathrm{idle}} \tag{5}$$

3 CWCO-EA 算法设计

3.1 虚拟机选择

为了最小化工作流任务的执行代价,本节首先设计一种 VM 选择算法,实现每个任务至最优 VM 间的映射。引入 sub-makespan 表示单个任务的执行时间,其概念与工作流的 makespan 类似。显然,如果每个任务的执行时间不大于分配的 sub-makespan,则整个工作流也可以确保在其截止时间内完成。首先,将每个任务映射至拥有最大计算能力的类型为 K 的 VM 上,那么任务 t_i 的最快执行时间可表示为

$$T(t_i,vm_K)=\frac{w_i}{p_K^{\max}}+\frac{I_i}{B} \tag{6}$$

其中:$T(t_i,vm_K)$ 表示任务 t_i 在 vm_K 上的执行时间,包括数据传输时间和有效的任务执行时间。同时,任务 t_i 的最早开始时间 $\mathrm{EST}(t_i)$ 和最早完成时间 $\mathrm{EFT}(t_i)$ 可表示为

$$\mathrm{EFT}(t_i)=\mathrm{EST}(t_i)+T(t_i,vm_K) \tag{7}$$

$$\mathrm{EST}(t_i)=\begin{cases}0 & \text{if}\quad t_i=t_{\mathrm{entry}}\\ \max\limits_{t_l\in\mathrm{pre}(t_i)}\mathrm{EFT}(t_l) & \text{otherwise}\end{cases} \tag{8}$$

工作流的最小 makespan 即为出口任务 t_{exit} 的最早完成时间,即 $\min T_M=\mathrm{EFT}(t_{\mathrm{exit}})$。假设用户定义的截止时间 T_D 不小于最小 makespan,即 $T_D\geqslant\min T_M$。可以定义任务 t_i 的 sub-makespan 为 $T_{\mathrm{sub}}(t_i)$,并表示为其最快执行时间的正比例形式:

$$T_{\mathrm{sub}}(t_i)=T(t_i,vm_K)\times T_D/\min T_M \tag{9}$$

定义代价效用表示单位代价内完成的任务负载量,则任务 t_i 在 vm_k 上执行的代价可表示为

$$u(t_i,vm_k)=\frac{w_i}{\lceil T(t_i,vm_k)\rceil\times c_k} \tag{10}$$

其中:分母部分表示任务 t_i 在 vm_k 上执行总代价,即 $c(t_i,vm_k)$。由式(10)可知,代价效用越高,任务执行代价越低。由于每个任务均有 sub-makespan 约束,需要为每个任务选择相应的 VM,以确保在其 sub-makespan 内完成。任务 t_i 的效用集合为包含所有代价效用的降序排列,表示为 $U(t_i)=\{u(t_i,vm_{k(1)}),u(t_i,vm_{k(2)}),\cdots,u(t_i,vm_{k(j)}),\cdots,u(t_i,vm_{k(K)})\}$,因此,$U(t_i)$ 中的第一个代价效用 $u(t_i,vm_{k(1)})$ 即为任务 t_i 调度的最优 VM $vm_{k(1)}$,且将任务 t_i 的 $vm_{k(1)}$ 标记为 vm_k^{opt}。此时,将所有工作流任务划分为以下两类:

a)对于任务 t_i,如果 $T(t_i,vm_k^{\mathrm{opt}})\leqslant T_{\mathrm{sub}}(t_i)$,则表明任务 t_i 可以在满足 sub-makespan 的情况下以最优代价效用完成。因此,可将任务 t_i 映射至 vm_k^{opt},表示为 $M:t_i\rightarrow vm_k^{\mathrm{opt}}$,并更新任务 t_i 的 sub-makespan 为 $T_{\mathrm{sub}}(t_i)=T(t_i,vm_k^{\mathrm{opt}})$。

b)对于任务 t_i,如果 $T(t_i, vm_k^{opt}) > T_{sub}(t_i)$,则将任务添加至任务集 TASK 并按算法 1 进行处理。

算法 1　虚拟机选择算法

a)按照最优代价效用对任务集 TASK 进行降序排列。

b)选择 TASK 中拥有最大代价效用的第一个任务。

c)令 $T_{sub}(t_i) = T(t_i, vm_k^{opt})$,并重新计算工作流 makespan T_W,并作如下处理:

(a)如果 $T_M \leqslant T_D$,则表明增大任务 t_i 的 sub-makespan 没有影响截止时间,那么可令 $T_{sub}(t_i) = T(t_i, vm_k^{opt})$ 并将任务 t_i 映射至 vm_k^{opt},即 $M: t_i \rightarrow vm_k^{opt}$,然后,从 TASK 中除移任务 t_i;

(b)如果 $T_M > T_D$,从任务 t_i 的效用集 $U(t_i)$ 中选择次优代价效用作为最优目标(即如果当前 VM 实例类型为 $k(j)$,则 $vm_k^{opt} = vm_{k(j+1)}$),并按照任务 t_i 的新的代价效用对 TASK 重新排序。

重复以上步骤直到 TASK 集合为空。可以看出,步骤 c)中更新每个任务的 sub-makespan 并不会影响其他任务的 sub-makespan,每个任务均可以找到最优 VM 目标。

算法 1 的时间复杂度分析。算法 1 中,对所有任务的代价效用进行排序的时间复杂度为 $O(nK \log K)$,由于仅需要进行插入操作,步骤(b)中重新排序的最差时间复杂度为 $O(n)$,重新计算 makespan 的时间复杂度为 $O(n+e)$,其中,e 表示工作流 DAG 中边的数量,而步骤 c)可能进行 kn 次迭代,因此,算法 1 的最差时间复杂度为 $O(kn^2(n+e))$。

3.2　串行任务合并

假设两个任务 t_i 和 t_{i+1} 分别为映射至类型为 k^i 和 k^{i+1} 的 VM 的串行任务,任务 t_{i+1} 为任务 t_i 的前驱,那么两个任务的执行时间分别为 $T(t_i, vm_{ki})$ 和 $T(t_i, vm_{ki+1})$,执行代价分别为 $c(t_i, vm_{ki})$ 和 $c(t_i, vm_{ki+1})$。如果将两个任务映射至 vm_k 上,其执行时间可表示为

$$T(t_{i,i+1}, vm_k) = \frac{w_i + w_{i+1}}{p_k^{max}} + \frac{I_i}{B} \tag{11}$$

那么,两个任务的执行代价可表示为

$$c(t_{i,i+1}, vm_k) = \lceil T(t_{i,i+1}, vm_k) \rceil \times c_k \tag{12}$$

由式(11)可知,由于两个任务在相同 VM 上执行,所以可以忽略数据传输时间 I_{i+1}/B。但是,必须确保总能够找到一个 vm_k,使得在时间 $(T(t_i, vm_{ki}) + T(t_i, vm_{ki+1}))$ 内可以完成任务 t_i 和 t_{i+1},即至少 $k = \max\{k^i, k^{i+1}\}$。尽管可能存在多个 VM 可以在满足时间约束的情况下完成两个任务,但其代价将大于 vm_k 上执行的代价(见式(1))。串行任务的合并步骤如算法 2 所示。

算法 2　串行任务合并算法

a)将所有任务放入任务集合 TASK。

b)对于 TASK 中的任务 t_i,如果仅有一个后继且该后继仅有一个前驱,或者仅有一个前驱且该前驱仅有一个后继,则选择在时间 $(T(t_i, vm_{ki}) + T(t_i, vm_{ki+1}))$ 内刚好能够完成任务 t_i 和 t_{i+1} 的 vm_k,并作如下处理:

(a)如果 $c(t_{i,i+1}, vm_k) \leqslant c(t_i, vm_{ki}) + c(t_{i+1}, vm_{ki+1})$,则合并任务 t_i 和 t_{i+1} 为新任务 t_i',并将 t_i' 映射至 vm_k,令 $T_{sub}(t_i') = T(t_{i,i+1}, vm_k)$,同时更新新任务的前驱、后继及负载量;

(b)如果 $c(t_{i,i+1}, vm_k) > c(t_i, vm_{ki}) + c(t_{i+1}, vm_{ki+1})$,则无须进行任务合并,直接将任务 t_i 移出 TASK。

图 1 表示三个串行任务的合并过程,且每次仅能进行两个任务的合并。串行任务合并的优势在于:a)合并任务均为串行,不会影响任务间的依赖关系;b)可以降低任务执行代价;c)即使 $c(t_{i,i+1}, vm_k) = c(t_i, vm_{ki}) + c(t_{i+1}, vm_{ki+1})$,也可合并任务,以消除数据传输成本。

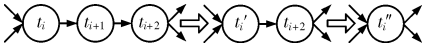

图 1　串行任务合并

算法 2 的时间复杂度分析。为合并任务寻找最优 VM 的时间复杂度为 $O(K)$,如果所有任务均为串行任务,则算法 2 的最差时间复杂度为 $O(Kn)$。

3.3　并行任务合并

并行任务拥有相同的开始执行时间和不同的完成时间。以图 2 为例,第一个工作流中拥有三个并行任务 t_i、t_{i+1} 和 t_{i+2},假设所有任务映射至相同 VM 上,三个任务的执行时间分别为 20、30 和 60 min。由于为并行任务,而 t_{i+2} 的执行时间最长,所以任务 t_{i+3} 在任务 t_{i+2} 完成前无法开始执行。然而,如果将任务 t_i 和 t_{i+1} 的并行关系转换为图 2 第二个工作流中描述的串行关系,两个任务的总执行时间将小于 60 min,因此,合并任务 t_i 和 t_{i+1} 不会影响工作流的 makespan,且可以降低执行任务的 VM 数量,达到降低执行代价和能耗的目标。

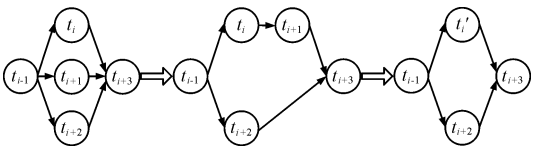

图 2　并行任务合并

本文的并行任务合并中,仅考虑拥有相同前驱和后继的两个并行任务的情况。假设一个并行任务组 g_g 包含两个以上的并行任务,且 $t_i, t_{i+1} \in g_g$。令 $\max T_{sub}$ 表示 g_g 中所有任务的最大 sub-makespan,表示为

$$\max T_{sub} = \max_{t_i \in g_g} T_{sub}(t_i) \tag{13}$$

如果两个合并任务的执行时间没有超过最大 sub-makespan $\max T_{sub}$,则合并不会影响其后继的开始执行时间,即可以进行任务合并。

假设并行任务 t_i 和 t_{i+1} 分别映射至类型为 k^i 和 k^{i+1} 的 VM,两个任务的执行代价分别为 $c(t_i, vm_{ki})$ 和 $c(t_i, vm_{ki+1})$。如果将两个并行任务映射至 vm_k,则执行时间和执行代价可由式(11)和(12)表示。合并算法按算法 3 进行。

算法 3　并行任务合并算法

a)给定工作流 DAG,寻找 DAG 中的所有并行任务组 $G = \{g_1, g_2, \cdots, g_g, \cdots, g_G\}$。

b)对于每个 g_g,结合式(6),根据最小执行时间对任务作升序排列。

c)选择拥有最小执行时间的两个任务,并寻找刚好满足时间约束 $T(t_{i,i+1}, vm_k) < \max T_{sub}$ 的 vm_k,如果无法找到该 VM,则停止合并并选择下一任务组;否则,执行以下步骤:

(a)如果 $c(t_{i,i+1}, vm_k) \leqslant c(t_i, vm_{ki}) + c(t_{i+1}, vm_{ki+1})$,则合并任务 t_i 和 t_{i+1} 为新任务 t_i',并将 t_i' 映射至 vm_k,令 $T_{sub}(t_i') = T(t_{i,i+1}, vm_k)$,同时更新新任务的前驱、后继及负载量,并对 g_g 中的任务按 t_i' 的最小执行时间重新排序。

(b)如果 $c(t_{i,i+1}, vm_k) > c(t_i, vm_{ki}) + c(t_{i+1}, vm_{ki+1})$,则停止合并并选择下一任务组。

重复步骤 c)直到集合 G 为空。并行任务合并过程如图 3 所示,其优势在于:不会影响后继的执行时间;可以降低任务执行代价;可以节省数据传输过程带来的能耗。

算法 3 的时间复杂度分析。步骤 a)中,当工作流不存在并行任务时,寻找并行任务集的最差时间复杂度为 $O(n^2)$;步骤 b)中对任务排序的最差时间复杂度为 $O(n \log n)$;当所有任务均为并行任务时,步骤 c)的最差时间复杂度为 $O(n^2 \log n)$,在集合中选择两个任务的时间复杂度为 $O(n \log n)$,步骤(a)中重新排序的时间复杂度为 $O(n)$。综上,并行任务合并的时间复杂度为 $O(n^2 \log n)$。

3.4　虚拟机重用

由于 VM 按小时支付费用,即如果任务 t_i 的执行时间为 $T_{sub}(t_i) = 61$ min,仍会按 2 h 付费,所以 VM 将会出现 59 min 的空闲时间,这会导致用户任务的代价浪费。VM 重用的目标是重用租用 VM 的空闲时间,以执行当前的其他调度任务。令 $T_{idle}(vm_l)$ 表示调度任务 $t_{i\text{-next}}$ 到来时 VM vm_l 的空闲时间,如果该空闲 VM 可被任务 $t_{i\text{-next}}$ 重用,则需要满足以下两个条件之一:

条件 1　空闲 VM 可在分配的 sub-makespan 中完成任务 $t_{i\text{-next}}$,表示为

$$T(t_{i\text{-next}},vm_l)\leqslant T_{\text{sub}}(t_{i\text{-next}})\leqslant T_{\text{sub}}(vm_l) \tag{14}$$

条件 2 在不超过 sub-makespan 的情况下,使任务 $t_{i\text{-next}}$ 对 vm_l 的租用时间为整数,表示为

$$\begin{cases}\lceil T(t_{i\text{-next}},vm_l)-T_{\text{idle}}(vm_l)\rceil\times c_l\leqslant\lceil T(t_{i\text{-next}},vm_k)\rceil\times c_k\\ T(t_{i\text{-next}},vm_l)\leqslant T_{\text{sub}}(t_{i\text{-next}})\end{cases} \tag{15}$$

图 3 给出两种情况下的 VM 重用过程。

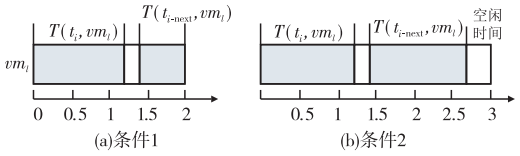

图 3 VM 重用

令空闲 VM 集为 $IV=\{vm_1^{IV},\cdots,vm_l^{IV},\cdots,vm_L^{IV}\}$,当前的调度任务集为 $ST=\{t_1^{ST},\cdots,t_i^{ST},\cdots,t_n^{ST}\}$,空闲 VM 的重用步骤如算法 4 所示。

算法 4 空闲 VM 重用算法

a)从调度任务集 ST 中随机选择一个任务 t_i^{ST}。

b)如果重用空闲 VM vm_l^{IV},则条件 1 或 2 计算任务的最小执行代价 $c(t_i^{ST},vm_l^{IV})$。显然,如果任务 t_i 重用满足式(14)的空闲 VM vm_l^{IV},则代价为 0,那么,存在任务 t_i^{ST} 的一个代价集合 $C(t_i^{ST})=\{c(t_i^{ST},vm_1^{IV}),\cdots,c(t_i^{ST},vm_l^{IV}),\cdots,c(t_i^{ST},vm_L^{IV})\}$,可以从该集合中得到最小代价 $\text{min}c(t_i^{ST},vm_l^{IV})$。

(a)如果 $\text{min}c(t_i^{ST},vm_l^{IV})\leqslant\lceil T(t_i^{ST},vm_k)\rfloor c_k$,则表明 VM 重用后比初始映射策略代价更少,任务 t_i 可重用满足条件 1 或 2 的 vm_l^{IV},并将任务 t_i^{ST} 映射至 *VM* vm_l^{IV},然后分别从 *ST* 和 *IV* 中移除任务 t_i^{ST} 和 vm_l^{IV},并更新任务 t_i^{ST} 的 sub-makespan 为 $T_{\text{sub}}(t_i^{ST})=T(t_i^{ST},vm_l^{IV})$;

(b)如果 $\text{min}\ c(t_i^{ST},vm_l^{IV})>\lceil T(t_i^{ST},vm_k)\rfloor c_k$,则应用初始映射策略 $M(t_i^{ST})=vm_k$,并从 *ST* 中移除 t_i^{ST}。

重复以上步骤直到 *ST* 或 *IV* 为空。显然,若条件 1 满足,则算法可以降低能耗。

算法 4 中,计算代价集合的时间复杂度为 $O(L)$,因此,VM 重用算法的时间复杂度为 $O(nL)$,其中,L 表示工作流调度过程中空闲 VM 的最大数量。

3.5 任务松驰

由于工作流中的非关键任务拥有较多的松驰时间(slack time),为了降低能耗,本节基于 DVFS 技术设计任务松驰算法来降低能耗,主要是通过降低租用 VM 的电压与频率的方式减少 VM 能耗。任务 t_i 的最迟开始时间 $\text{LST}(t_i)$ 可表示为

$$\text{LST}(t_i)=\begin{cases}T_M-T(t_{\text{entry}},vm_k) & \text{if } t_i=t_{\text{entry}}\\ \min\limits_{t_l\in\text{succ}(t_i)}\text{LST}(t_l)-T(t_l,vm_k) & \text{otherwise}\end{cases} \tag{16}$$

任务 t_i 的松驰时间可表示为

$$T_{\text{slacktime}}(t_i)=\text{LST}(t_i)-\text{EST}(t_i) \tag{17}$$

当任务 t_i 的松驰时间大于 0 时,即 $T_{\text{slacktime}}(t_i)>0$,该任务为非关键任务。令 *NT* 表示非关键任务集合,为了降低 VM 的执行频率,按照算法 5 实现非关键任务的执行时间松驰化。

算法 5 任务松驰算法

a)按照式(16)和(17)计算每个任务的松驰时间,将所有非关键任务添加至 *NT*,并标志所有关键任务为"CRITICAL",表示该任务没有松驰时间可以回收。

b)选择前驱均为 CRITICAL 的任务 t_i,计算最长路径 T_{path},即表示以任务 t_i 为起点至 CRITICAL 任务为终点,路径上所有任务的执行时间总和。

c)计算最大执行时间 $T'_{\text{sub}}(t_i)$。根据算法 4,如果任务 t_i 租用的 VM 被任务 $t_{i\text{-next}}$ 重用,则任务 t_i 的最迟完成时间不会超过任务 $t_{i\text{-next}}$ 的开始时间;如果该 VM 未被重用,则最迟完成时间应小于租用时间,此时按以下步骤处理:

(a)如果 VM 被重用,修改任务 t_i 的 sub-makespan 为

$$T'_{\text{sub}}(t_i)=\min\{T_{\text{sub}}(t_i)\times\frac{T_{\text{path}}+T_{\text{slacktime}}(t_i)}{T_{\text{path}}},\text{EST}(t_{i\text{-next}})\} \tag{18}$$

参见图 4(a)示例;

(b)如果 VM 未被重用,修改任务 t_i 的 sub-makespan 为

$$T'_{\text{sub}}(t_i)=\min\{T_{\text{sub}}(t_i)\times\frac{T_{\text{path}}+T_{\text{slacktime}}(t_i)}{T_{\text{path}}},\lceil T(t_i,vm_k)\rceil\} \tag{19}$$

参见图 4(b)示例。

d)修改任务 t_i 的 VM 频率为

$$f_k^i=f_k^{\max}\times T_{\text{sub}}(t_i)/T'_{\text{sub}}(t_i) \tag{20}$$

表明可以以正比于 $T_{\text{sub}}(t_i)$ 和 $T'_{\text{sub}}(t_i)$ 的方式降低 CPU 频率。标志任务 t_i 为"CRITICAL"并将其从 *NT* 中移除。

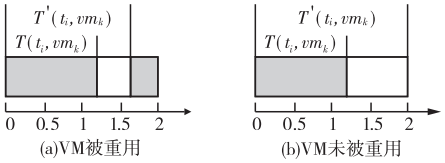

图 4 任务松驰两种情况

算法 5 中,计算所有任务的松驰时间的最差时间复杂度为 $O(n^2)$,寻找最长路径的最差时间复杂度为 $O(n^2)$。因此,算法 5 的时间复杂度为 $O(n^2)$。

3.6 CWCO-EA 算法流程

CWCO-EA 算法由以上五个子算法组成,包括 VM 选择、任务合并(串行、并行)、空闲 VM 重用及任务松驰。算法的实现目标是在云环境中进行科学工作流调度时,在满足截止时间约束的情况下,实现工作流调度代价的最小化及能耗的降低。算法流程如图 5 所示,各子算法的功能与时间复杂度如表 2 所示。

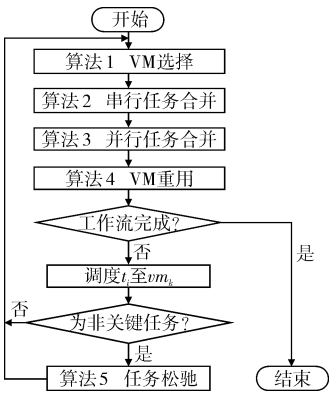

图 5 CWCO-EA 算法流程

表 2 CWCO-EA 的子算法性能

性能指标	算法 1	算法 2	算法 3	算法 4	算法 5
降低代价	—	√	√	√	—
降低能耗	—	√	√	√	√
时间复杂度	$O(kn^2(n+e))$	$O(Kn)$	$O(n^2\log n)$	$O(Ln)$	$O(n^2)$

4 仿真实验

4.1 实验配置

为了评估工作流调度算法的性能,本章利用 WorkflowSim[17] 实现对云环境的仿真模拟,并选取四种不同科学领域中的工作流结构作为测试源,包括 epigenomic、LIGO、montage 和 cyberShake,这四种工作流属于不同领域中大规模数据集的科学应用。工作流结构如图 6 所示,其详细的数据与计算特征描述可参考文献[18,19]。假设工作流中每个任务的负载量的取值范围为[1 000 MIPS,5 000 MIPS],其输出数据量取值范围为[100 MI,1 000 MI],工作流的截止时间取值范围为[5 h,10 h]。假设云数据中心拥有 10 个 VM,VM 间的带宽为 1 GB,其计算能力、单位时间的使用代价参考 Amazon EC2 的 VM 配置,如表 3 所示。计算能力与使用代价间的关系参考式(1),并假设 CPU 频率与计算能力成正比,以便于仿真实验中的性能度量。并且,每种 VM 的 CPU 最大频率和最小频率以 DVFS 技术实现,从而实现降频对能耗的降低。

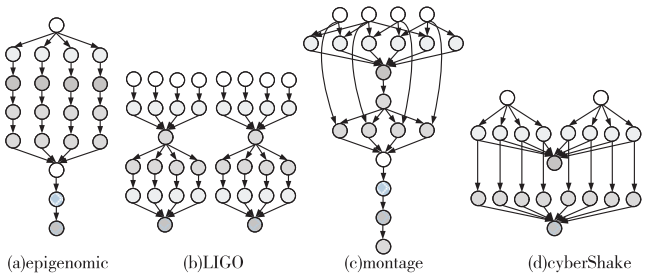

图 6　工作流结构

表 3　VM 配置

VM 类型	代价 /$/h	计算能力 /MIPS	CPU 频率级别/GHz	
			最大值	最小值
1	0.1	1 000	1	0.5
2	0.2	1 500	1.5	0.75
3	0.32	2 000	2	1
4	0.46	2 500	2.5	1.25
5	0.58	3 000	3	1.5
6	0.73	3 500	3.5	1.75
7	0.9	4 000	4	2
8	1.05	4 500	4.5	2.25
9	1.21	5 000	5	2.75
10	1.4	5 500	5.5	2.75

实验目标重点关注算法在截止时间约束下工作流执行代价与执行能耗的情况。选取异构最早完成时间算法(HEFT)[6]和多目标异构最早完成时间算法(MOHEFT)[13]作为执行代价性能比较的基准算法,选取加强能效调度算法(EES)[9]和加强 HEFT 算法(EHEFT)[10]作为执行能耗性能比较的基准算法。

4.2　实验结果分析

实验 1 比较了 CEAS、HEFT 和 MOHEFT 三种算法在工作流执行代价方面的性能。HEFT 是以最小化工作流执行 makespan 为目标的线性启发式算法,HEFT 在每一步中选择拥有最大前向分级值的任务进行优先调度,从而实现最早完成时间的最小化。MOHEFT 是在 HEFT 基础上扩展得到的基于 Pareto 的多目标线性调度算法,比较 HEFT,其额外开销更小,而且 MOHEFT 可以同步优化 makespan 和执行代价。

图 7 为三种算法的执行代价情况。可以看出,三种算法的执行代价具有明显差异,CEAS 代价最小,HEFT 代价最高,MOHEFT 相对居中。图中相对平缓的曲线表明 HEFT 的性能与工作流截止时间约束并无明显依赖关系,这主要是由于云数据中心拥有多种类型 VM,且每种 VM 能够向外界提供不受限的 VM 实例。而且,HEFT 将任务分配至最小化最早完成时间的 VM 上,换言之,该 VM 是性能最高的 VM,并没有考虑工作流的执行代价,因此其代价最高,且不受截止时间影响。

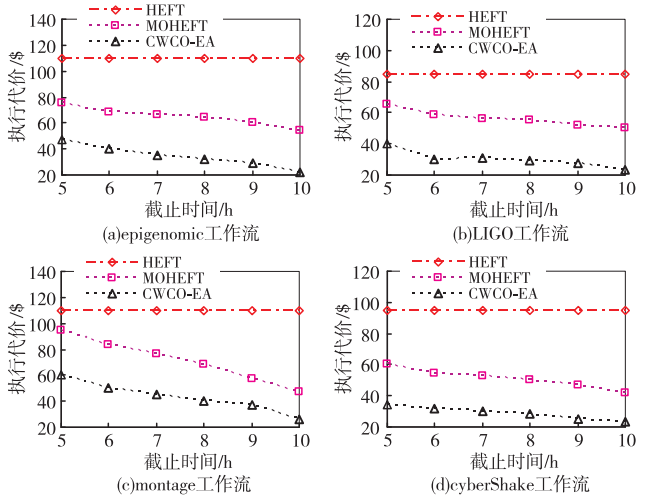

图 7　工作流执行代价

MOHEFT 选择的 VM 较 HEFT 更为合适,因此在 HEFT 基础上明显降低了代价;同时,MOHEFT 的代价会随着截止时间改变而变化,截止时间越大,代价越低。这是由于在截止时间变大时,MOHEFT 可以选择计算性能更低的 VM 类型执行任务,相应代价也更小。然而,MOHEFT 以小时为单位租用 VM,即在整个 makespan 中均占用 VM,这会导致空闲 VM 时的额外代价。CEAS 通过任务合并及 VM 重用有效降低了空闲 VM 的代价,其执行代价是最小的。

实验 2 比较了 EES、EHEFT 和 CEAS 三种算法在工作流执行能耗方面的性能。EES 是在 HEFT 的基础上,在满足服务等级协议 SLA 的同时实现的能效调度算法,其主要思想是通过利用松驰时间,实现对非关键路径上的任务的松驰调度,以降低能耗。EHEFT 则是实现 makespan 与能耗双目标最优化的调度算法,与 EES 比较,EHEFT 不仅可以利用非关键任务的松驰时间,而且可以关闭低效率处理器节省能耗。

图 8 为三种算法的执行能耗情况。可以看出,CEAS 能耗最少,EHEFT 次之,EES 能耗最高。EES 的能耗并不随着截止时间发生变化,这是由于 EES 以 HEFT 为基础,始终选择性能最高的 VM 执行任务导致的,没有考虑代价与能耗。EHEFT 能耗少于 EES,主要是由于 EHEFT 可以实现低效率处理器的关闭。同时,不同于 CEAS,EES 和 EHEFT 对 VM 的租用均在整个 makespan 中,尽管对空闲处理器的降频处理可以降低能耗,但能耗仍然较高;CEAS 则通过任务合并与 VM 重用,使得 VM 的空闲时间更少,能耗也更低。

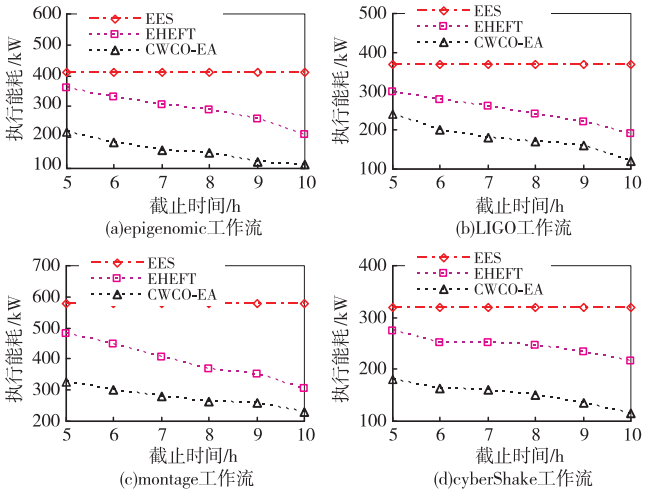

图 8　工作流执行能耗

5　结束语

为了优化云环境中截止时间约束下科学工作流调度的执行代价和执行能耗,提出了一种能效感知的工作流调度代价最优化算法(CWCO-EA)。该算法通过最优 VM 选择有效实现了任务与 VM 间的映射,通过串行和并行任务合并实现了执行代价与能耗的同步降低,通过 VM 重用和任务松驰提高了 VM 的使用效率,减少了 VM 的空闲时间;同时分析了各个子算法的时间复杂度。实验结果表明,CWCO-EA 算法与同类型算法相比可以在满足截止时间的同时进一步降低工作流执行代价,并减少云数据中心的能耗。

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. Piscataway, NJ: IEEE Press, 2014: 837-846.

[2] Kashlev A, Lu Shiyong. A system architecture for running big data workflows in the cloud [C]//Proc of IEEE International Conference on Services Computing. Washington DC: IEEE Computer Society, 2014: 51-58.

[3] Su Sen, Li Jian, Huang Qingjia, *et al*. Cost-efficient task scheduling for executing large programs in the cloud [J]. Parallel Computing, 2013, 39(4-5): 177-188.

[4] Shuja J, Madani S A, Bilal K, *et al*. Energy-efficient data centers [J]. Computing, 2012, 94(12): 973-994.　(下转第 2021 页)

[2] Jennings N R, Sycara K, Wooldridge M. A roadmap of agent research and development [J]. Autonomous Agents and Multi-agent System, 1998, 1(1): 7-38.
[3] Kühne U. Wie erklärt man mit unsichtbaren Händen? [M]//Modellierung und Simulation von Dynamiken mit Vielen Interagierenden AkteurenIProc. Bremen, Germany:Bremen University. 1997: 37-51.
[4] Hegselmann R, Krause U. Opinion dynamics and bounded confidence models, analysis, and simulation [J]. Artificial Societies and Social Simulation, 2002, 5(3): 1-33.
[5] Blondel V D, Hendrickx J M, Tsitsiklis J N. On Krause's multi-agent consensus model with state-dependent connectivity [J]. IEEE Trans on Automatic Control, 2009, 54(11): 2586-2597.
[6] Reynolds C W. Flocks, herds, and schools: a distributed behavioral model [J]. ACM SIGGRAPH Computer Graphics, 1987, 21(4): 25-34.
[7] Gazi V, Passino K M. Stability analysis of social foraging swarms [J]. IEEE Trans on Systems, Man, and Cybernetics, Part B: Cybernetics, 2004, 34(1): 539-557.
[8] 谢光强, 章云. 多智能体系统协调控制一致性问题研究综述 [J]. 计算机应用研究, 2011, 28(6): 2035-2039.
[9] Olfati-Saber R, Fax J A, Murray R M. Consensus and cooperation in networked multi-agent systems [J]. Proceedings of the IEEE, 2007, 95(1): 215-233.
[10] Ren Wei, Beard R W. Consensus seeking in multiagent systems under dynamically changing interaction topologies [J]. IEEE Trans on Automatic Control, 2005, 50(5): 655-661.
[11] Gazi V, Passino K M. Stabillity analysis of swarms [J]. IEEE Trans on Automatic Control, 2003, 48(4): 692-697.
[12] Gazi V. Stability of an asynchronous swarm with time-dependent communication links [J]. IEEE Trans on Systems, Man, and Cybernetics, Part B: Cybernetics, 2008, 38(1): 267-74.
[13] Tanner H G, Jadbabaie A, Pappas G J. Flocking in fixed and switching networks [J]. IEEE Trans on Automatic Control, 2007, 52(5): 863-868.
[14] Jadbabaie A, Lin Jie, Morse A S. Coordination of groups of mobile autonomous agents using nearest neighbor rules [J]. IEEE Trans on Automatic Control, 2003, 48(6): 988-1001.
[15] Olfati-Saber R, Murray R M. Consensus problems in networks of agents with switching topology and time-delays [J]. IEEE Trans on Automatic Control, 2004, 49(9): 1520-1532.
[16] Olfati-Saber R. Flocking for multi-agent dynamic systems: algorithms and theory [J]. IEEE Trans on Automatic Control, 2006, 51(3): 401-420.
[17] Lin J, Morse A S, Anderson B D O. The multi-agent rendezvous problem, part 1: the synchronous case [J]. SIAM Journal on Control and Optimization, 2008, 46(6): 1508-1513.
[18] Lin J, Morse A S, Anderson B D O. The multi-agent rendezvous problem, part II: the asynchronous case [J]. SIAM Journal on Control and Optimization, 2008, 46(6): 2120-2147.
[19] Hui Qing. Finite-time rendezvous algorithms for mobile autonomous agents [J]. IEEE Trans on Automatic Control, 2011, 56(1): 207-211.
[20] Cortés J, Martinez S, Bullo F. Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions [J]. IEEE Trans on Automatic Control, 2006, 51(8): 1289-1298.
[21] Li Zhongkui, Wen Guanghui, Duan Zhisheng, *et al.* Designing fully distributed consensus protocols for linear multi-agent systems with directed graphs [J]. IEEE Trans on Automatic Control, 2015, 60(4): 1152-1157.
[22] Thunberg J, Goncalves J, Hu Xiaoming. Local lyapunov functions for consensus in switching nonlinear systems [J]. IEEE Trans on Automatic Control, 2017, 62(12): 6466-6472.
[23] Zheng Yuanshi, Wang Long. Consensus of switched multiagent systems [J]. IEEE Trans on Circuits and Systems Ⅱ: Express Briefs, 2016, 63(3): 314-318.
[24] Gupta N, Chopra N. Confidentiality in distributed average information consensus [C]//Proc of the 55th Conference on Decision and Control. Piscataway, NJ:IEEE Press, 2016: 6709-6714.
[25] Wen Guanghui, Duan Zhisheng, Chen Guangrong, *et al.* Consensus tracking of multi-agent systems with Lipschitz-type node dynamics and switching topologies [J]. IEEE Trans on Circuits and Systems Ⅰ: Regular Papers, 2014, 61(2): 499-511.
[26] Su Housheng, Chen M Z Q, Wang Xiaofan, *et al.* Semiglobal observer-based leader-following consensus with input saturation [J]. IEEE Trans on Industrial Electronics, 2014, 61(6): 2842-2850.
[27] Wang Dong, Zhang Ning, Wang Jianliang, *et al.* A PD-like protocol with a time delay to average consensus control for multi-agent systems under an arbitrarily fast switching topology [J]. IEEE Trans on Cybernetics, 2017, 47(4): 898-907.
[28] Moreau L. Stability of multi-agent systems with time-dependent communication links [J]. IEEE Trans on Automatic Control, 2005, 50(2): 169-182.
[29] Horn R A, Johnson C R. Matrix analysis [M]. Cambridge: Cambridge University Press, 1987.
[30] Qin Jiahu, Yu Changbin, Gao Huijun. Coordination for linear multi-agent systems with dynamic interaction topology in the leader-following framework [J]. IEEE Trans on Industrial Electronics, 2014, 61(5): 2412-2422.

(上接第 2005 页)

[5] Cao Fei, Zhu M M. Energy-aware workflow job scheduling for green clouds [C]// Proc of IEEE International Conference on Green Computing and Communication and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing. Washington DC:IEEE Computer Society, 2013: 232-239.
[6] Topcuoglu H, Hariri S, Wu M Y. Performance-effective and low-complexity task scheduling for heterogeneous computing [J]. IEEE Trans on Parallel and Distributed Systems, 2012, 13(3): 260-274.
[7] Zheng Wei, Sakellariou R. Budget-deadline constrained workflow planning for admission control [J]. Journal of Grid Computing, 2013, 11(4): 633-651.
[8] Arabnejad H, Barbosa J G. A budget constrained scheduling algorithm for workflow applications [J]. Journal of Grid Computing, 2014, 12(4): 665-679.
[9] Huang Qingjia, Su Sen, Li Jian, *et al.* Enhanced energy-efficient scheduling for parallel applications in cloud [C]//Proc of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. Washington DC: IEEE Computer Society, 2012: 781-786.
[10] Thanavanich T, Uthayopas P. Efficient energy aware task scheduling for parallel workflow tasks on hybrids cloud environment [C]//Proc of International Computer Science and Engineering Conference. Piscataway, NJ:IEEE Press, 2013: 37-42.
[11] Padmaveni K, Aravindhar D J. Hybrid memetic and particle swarm optimization for multi objective scientific workflows in cloud [C]// Proc of IEEE International Conference on Cloud Computing in Emerging Markets. Washington DC: IEEE Computer Society, 2016: 66-72.
[12] Garg R, 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.
[13] Zhang Fan, Cao Junwei, Hwang K, *et al.* Ordinal optimized scheduling of scientific workflows in elastic compute clouds [C]//Proc of IEEE the 3rd International Conference on Cloud Computing Technology and Science. Washington DC:IEEE Computer Society, 2011: 9-17.
[14] 景维鹏, 吴智博, 刘宏伟, 等. 多 DAG 工作流在云计算环境下的可靠性调度方法[J]. 西安电子科技大学学报, 2016, 43(2): 83-88.
[15] 王岩, 汪晋宽, 王翠荣, 等. QoS 约束的云工作流调度算法 [J]. 东北大学学报: 自然科学版, 2014, 35(7): 939-943.
[16] 杨玉丽, 彭新光, 黄名选, 等. 基于离散粒子群优化的云工作流调度 [J]. 计算机应用研究, 2014, 31(12): 3677-3681.
[17] Chen Weiwei, Deelman E. WorkflowSim: a toolkit for simulating scientific workflows in distributed environments [C]//Proc of the 8th International Conference on E-Science. Washington DC:IEEE Computer Society, 2012: 1-8.
[18] Rodriguez M A, Buyya R. Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds [J]. IEEE Trans on Cloud Computing, 2014, 2(2): 222-235.
[19] Pietri I, Malawski M, Juve G, *et al.* Energy-constrained provisioning for scientific workflow ensembles [C]//Proc of International Conference on Cloud and Green Computing. Washington DC:IEEE Computer Society, 2013: 34-41.