doi:10.11887/j.cn.201703016　　http://journal.nudt.edu.cn

云平台上基于关键路径截取的有向无环图应用调度算法*

刘少伟[1],任开军[2],邓科峰[2],宋君强[2]
(1. 国防科技大学 计算机学院,湖南 长沙 410073;
2. 国防科技大学 海洋科学与工程研究院,湖南 长沙 410073)

摘　要:针对云平台上有向无环图科学应用执行容易产生虚拟机资源过剩、资源使用率低及费用虚高的问题,给出一种基于关键路径截取的有向无环图应用调度算法。该算法采取关键路径截取技术,循环找出最晚完成的未分配任务,从该任务出发,在所有未分配任务构成的图中找出最大连通子图,并计算该子图的关键路径,然后将关键路径上的任务集调度到性能匹配的虚拟机上执行;同时通过任务回填技术充分利用虚拟机的空闲时间槽,提高资源使用率。实验结果表明,在云计算平台上,该算法不仅能够在截止时间内完成有向无环图科学应用,而且可以提高资源使用率,有效减少完成该应用所需整体费用。

关键词:云计算平台;关键路径;虚拟机;有向无环图;资源配置
中图分类号:TP393　　**文献标志码**:A　　**文章编号**:1001-2486(2017)03-097-08

Directed acyclic graph application scheduling strategy based on critical path cut on cloud platform

LIU Shaowei[1], *REN Kaijun*[2], *DENG Kefeng*[2], *SONG Junqiang*[2]
(1. College of Computer, National University of Defense Technology, Changsha 410073, China;
2. Academy of Ocean Science and Engineering, National University of Defense Technology, Changsha 410073, China)

Abstract: To address the problems that the resource is surplus, the resource utilization rate is low and the cost is unreasonably high for virtual machines in the scientific application of DAG(directed acyclic graph), a novel DAG scientific workflow scheduling algorithm based on CPC(critical path cut) was proposed. In the algorithm, the CPC technology was adopted to circularly find the unallocated task which is finished at last; the biggest connected subgraph was found from the graph constructed by the whole unallocated tasks; the critical path of this subgraph was calculated and the task set on the critical path was scheduled to the performance-matched virtual machine to execute. Meanwhile, the isolated tasks were used to fill in the idle slots of the virtual machines, such that the resource utilization could be improved. Experimental results demonstrate that, the proposed CPC algorithm can effectively reduce the execution cost of the scientific workflows while satisfying the deadline constraint in mean time.

Key words: cloud computing platform; critical path; virtual machine; directed acyclic graph; resource allocation

在许多科学研究领域,例如高能物理学、生物信息学、大气科学等,科学计算过程往往由成千上万个子任务聚合而成,而且任务之间存在严格的依赖关系。这些子任务含有依赖关系的大规模科学应用可以抽象为大规模有向无环图(Directed Acyclic Graph, DAG)科学应用。

这些应用,如天文学应用 Montage[1]、天体物理学应用激光干涉引力波观察台(Laser Interferometer Gravitational wave Observatory, LIGO)[1],通常需要在复杂的分布式计算机系统上执行,例如超级计算机、分布式集群系统以及网格系统等。针对这种应用,目前已经有很多成熟的算法,例如文献[2]提出了一种服务计数算法(Server Count Bound, SCB)启发式算法,在服从截止时间约束的同时,寻找最小资源分配方案以减少用户费用,可以在集群上最小化资源使用。文献[3]在费用约束下通过数据还原及重新利用技术提高 DAG 的执行效率,侧重于对 DAG 中节点的重复利用以减少计算开销。文献[4]则主要针对 DAG 应用,通过分析应用的同步完成特征,提出了三种针对性算法。

但是,上述算法主要运行平台为高性能集群或超级计算机,构造这样的高性能计算平台往往代价异常昂贵,对其访问一般也需要复杂耗时的

* **收稿日期**:2016-02-14
基金项目:国家自然科学基金资助项目(61572510);国家公益行业专项计划资助项目(GYHY201306003)
作者简介:刘少伟(1987—),男,陕西渭南人,博士研究生,E-mail:liushaowei@nudt.edu.cn;
宋君强(通信作者),男,研究员,硕士,博士生导师,E-mail:junqiang@nudt.edu.cn

申请过程。

近年来,云计算的兴起为大规模 DAG 科学应用的高性价比执行提供了潜力。云计算技术是一种共享基础架构的方法,它通过虚拟化技术将计算资源和存储资源虚拟成一个资源池,以虚拟机(Virtual Machine, VM)的形式向用户提供资源。这种资源提供方式降低了供应商运行成本,同时用户可以根据自己需求合理配置自己所需要的资源。因此,云计算吸引着越来越多的用户将大规模 DAG 科学应用迁移到云平台上执行。但同时,云计算的资源供应模式和收费模式为大规模 DAG 科学应用的运行带来了新的挑战。

Byun 等针对云计算环境提出了分层的负载均衡时间调度(Partitioned Balanced Time Scheduling, PBTS)算法[5],PBTS 算法将工作流截止期划分为多个时间段,并利用 BTS 算法[6]为每个时间段计算 DAG 应用需要的最少资源量,但算法针对的是同构资源模型。针对云计算平台虚拟机资源的获取与任务调度完全由用户负责的特点,文献[7]提出了一种云平台上基于截止时间分发的偏序关键路径调度(IaaS Cloud Partial Critical Paths with Deadline Distribution, IC-PCPD2)算法,首先初始化关键路径上任务的截止时间,然后采用递归的方法依次求取其他路径上任务的截止时间,然后循环求取偏序关键路径 PCP,在截止时间约束下将 PCP 放置到最便宜的虚拟机实例上,直到 DAG 中所有任务分配完毕。由于是按路径分配任务,该算法容易造成更多的虚拟机空闲碎片,降低资源使用率,相对地增加了用户费用。

本文以最小化完成 DAG 所需费用为目标,通过分析大规模 DAG 科学应用中任务的依赖关系,提出一种云平台上基于关键路径截取(Critical Path Cut, CPC)的 DAG 调度算法。

1 模型定义及问题分析

1.1 相关模型

1.1.1 应用模型

针对大规模 DAG 科学应用,使用符号 $\boldsymbol{W}$ 进行指代,$\boldsymbol{W}=\{\boldsymbol{V},\boldsymbol{E}\}$。其中,$\boldsymbol{V}=\{t_i \mid i=1,2,\cdots,|V|\}$ 是顶点的集合,表示 DAG 应用中任务集合;$\boldsymbol{E}=\{e_k \mid k=1,2,\cdots,|E|\}$ 为边的集合,表示任务间的控制或数据依赖关系。单个任务进一步由五元组描述:$t_i=\{seq,\omega,\lambda,EST,LST\}$。其中,$seq(t_i)$ 表示任务的串行执行时间;$\omega(t_i)$ 表示任务的实际执行时间;$\lambda\in[0,1]$ 为并行系数,$\omega(t_i)=seq(t_i)\cdot\lambda$;$EST(t_i)$ 表示任务的最早开始时间;$LST(t_i)$ 表示任务的最晚开始时间。

边表示为 $e_k=(t_i,t_j)$,其中 t_i 为 e_k 的起点,t_j 为 e_k 的终点,t_j 只有在 t_i 执行完毕之后才可以开始执行;$\omega(e_k)$ 为权重,表示两个任务之间的通信开销,当任务 t_i 和 t_j 调度到同一个虚拟机上时,通信开销忽略不计。

无任何前驱的任务为入口任务,无任何后继的任务为出口任务。入口任务和出口任务具有唯一性,如果 DAG 包括多个入口任务或出口任务,可以将所有入口任务和出口任务连接到一个零成本的虚拟入口任务 t_{start} 或虚拟出口任务 t_{finish},对系统性能并无影响。

1.1.2 资源模型

假设云计算平台提供 N 种不同类型的虚拟机,表示为 $vmType_i=\{cpuNum, memory, storage, price\}, i=1,2,\cdots,N$。不同类型的 VM,参数值各不相同,其中 *cpuNum* 表示 CPU 数量,*memory* 表示内存大小,*storage* 表示硬盘大小,*price* 表示此种类型的虚拟机单个时间周期费用。

用户通过租赁云计算平台的虚拟机构建虚拟机集群执行科学工作流,虚拟机集群表示为 $VMC=\{vm_i \mid i=1,2,\cdots\}$,$vm_i=\{id, vmType, sTime, aTime, rTime, \boldsymbol{scheTL}\}$,其中 *id* 表示虚拟机编号,*vmType* 表示虚拟机类型,*sTime* 表示虚拟机启动时间,*aTime* 表示虚拟机生存时间,*rTime* 表示虚拟机当前时间周期的剩余时间,***scheTL*** 表示调度到虚拟机的任务集合。

考虑现代虚拟化技术下虚拟机的启动时间和关闭时间已经达到秒级,因此假设虚拟机启动时间和关闭时间可以忽略不计。

1.2 符号定义

本节以图 1 的简单 DAG 为示例,对本文用到的参数、符号进行说明。

图中 $t_1 \sim t_9$ 为 DAG 中的任务集合,t_{start} 和 t_{finish} 为新增的开始任务和结束任务,箭头表示任务间依赖关系。为了便于说明和理解,本示例中将通信时间略去。假设云平台上有两种类型的虚拟机,一种只有 1 个 VCPU,另一种类型虚拟机拥有 2 个 VCPU。如图 1 所示,括号内、外分别是任务在双核和单核虚拟机实例上的运行时间,单位为 min。虚拟机实例需要按小时申请。假设整个 DAG 的截止时间(deadline)为 120 min。

1.2.1 任务并行性

任务并行性包括三个方面:多任务并行、单任

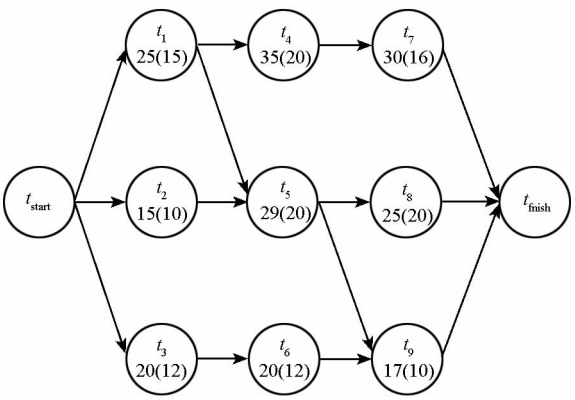


图 1　简单 DAG 示例

Fig. 1　Simple DAG example

务多核并行和单任务多虚拟机并行。

多任务并行指不同任务在多个虚拟机上并发执行,如图 1 中 t_1 和 t_2 可以在不同虚拟机上并发执行;单任务多核并行指单个任务可以在拥有多个 VCPU 上的虚拟机上并行执行,如图 1 中 t_1 在单核虚拟机上执行时间为 25 min,在双核虚拟机上执行时间为 15 min;单任务多虚拟机并行指单个任务可以在多个虚拟机上通过并行执行减少任务执行时间。

1.2.2　关键路径

将 DAG 图中从开始任务 t_{start} 到结束任务 t_{finish} 所需时间最长的一条路径定义为关键路径。DAG 应用的关键路径决定了其最短执行时间。计算关键路径首先需要对所有任务的 *EST* 和 *LST* 进行赋值,其中最早开始时间 *EST* 计算方式如下:

$$EST(t_i)=\begin{cases}0, & t_i=t_{start}\\ \max\{EST(t_j)+\omega(t_j)+\omega(e_k)\}, & e_k=(t_j,t_i)\in E\end{cases}$$

最晚开始时间 *LST* 计算方式如下:

$$LST(t_i)=\begin{cases}EST(t_i), & t_i=t_{finish}\\ \min\{LST(t_j)-\omega(t_j)-\omega(e_k)\}, & e_k=(t_i,t_j)\in E\end{cases}$$

关键路径计算方式分以下 6 步:①计算所有任务的 *EST*;②计算所有任务的 *LST*;③计算所有边的 *EST*;④计算所有边的 *LST*;⑤所有 *LST* 等于 *EST* 的边构成关键路径。

需要注意的是,如果任务的实际执行时间不固定,那么关键路径也可能发生变化。如图 1 所示 DAG,如果所有任务都在单核虚拟机上执行,那么关键路径为 $\{t_{start},\ t_1,\ t_4,t_7,t_{finish}\}$;如果所有任务都在双核虚拟机上执行,关键路径则为 $\{t_{start},\ t_1,\ t_5,t_8,t_{finish}\}$。

1.2.3　截止时间

用户一般会指定一个截止时间 δ,其取值需要在合理范围之内,本文采取以下取值方法。

假设所有任务的实际执行时间 $\omega(t_i)$ 为其在配置最高的虚拟机实例上的执行时间,此时完成关键路径上任务所需时间为 δ_{short},即最快完成时间;假设所有任务的实际执行时间 $\omega(t_i)$ 为其在配置最低的虚拟机实例上的执行时间,此时完成关键路径上任务所需时间为 δ_{long},即最慢完成时间。

可以看出,如果 $\delta<\delta_{short}$,则无论如何调度,DAG 应用都无法在截止时间 δ 约束下完成;如果 $\delta_{short}\leqslant\delta\leqslant\delta_{long}$,为了保证按时完成 DAG 应用,肯定需要配置高性能的虚拟机实例;如果 $\delta>\delta_{long}$,所生成的虚拟机集群一般都是最低配置的虚拟机实例,以实现费用最小化。

1.3　问题分析

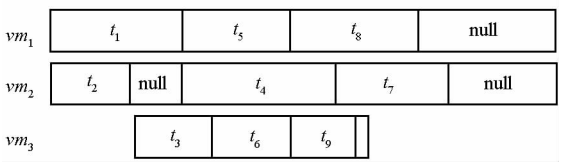
DAG 应用在云计算平台执行时,调度算法需要解决以下几个方面所面临的问题:

1)配置虚拟机集群。既不能启动大量 VM,导致相当一部分 VM 空转,造成资源浪费,也不能启动少量 VM,导致无法按时完成 DAG 科学应用。调度算法必须合理规划好每个 VM 的类型、启动时间和生存时间,在满足整个 DAG 应用截止时间约束的情况下最大化每个虚拟机的成本效益。

2)任务调度。调度父任务到低性能虚拟机上时,可能引起后续子任务长时间等待,导致 DAG 应用无法在截止时间 δ 内完成;调度任务到到高性能虚拟机上执行,由于任务很快执行完毕,当前虚拟机因为等待其他虚拟机上的父任务完成而处于空转状态,则会造成资源浪费,间接增加用户费用。

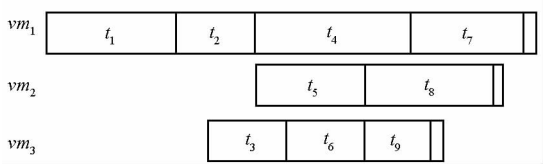
3)减少空闲时间槽,提高资源使用率。当前大多数云计算平台采用基于时间周期计费的方式,如 Amazon EC2 虚拟机实例以小时为单位计费,不足 1 小时按 1 小时计算。在此情况下,调度算法必须有效地将任务整合在一起,使得租赁的虚拟机能够在整数时间周期内充分利用。

4)以图 1 所给 DAG 为例。IC-PCPD2 算法调度方案如图 2(a)所示。IC-PCPD2 总共需要 5(cpu · hour)的资源,可以看出 IC-PCPD2 算法会产生部分虚拟机碎片。实际上如果按照图 2(b)所示的新方案调度,只需要 4(cpu · hour)的资源,而且能够在截止时间内完成 DAG 应用所有任务,费用可以节省 20%。因此,为了进一步提高虚拟机空闲时间槽的利用,减少执行开销,本文提出了基于关键路径截取的 DAG 调度算法,即 CPC 算法。



(a) IC-PCPD2 算法

(a) IC-PCPD2 algorithm



(b) 新解决方案

(b) New solution

图 2 两种调度方案

Fig. 2 Two kinds of scheduling schemes

2 算法描述

CPC 算法的目标是在截止时间条件约束下,以费用最小化为目标,完成 DAG 应用。

CPC 算法吸取了 IC-PCPD2 路径调度的思想,但略有不同。在路径调度方法中,首先找出最迟完成的未分配任务,并从该任务出发,找出未分配任务构成的最大连通子图,求取该子图的关键路径,并进行合理调度;同时增加了任务回填方法,通过任务回填技术将单任务填充到已有虚拟机空闲时间槽,提高对虚拟机资源的使用率。

2.1 CPC 调度算法

算法 1 为基于 CPC 的 DAG 调度算法。其中算法输入为 DAG 应用节点 $\boldsymbol{V}$ 和边 $\boldsymbol{E}$ 的情况、所有边的权重 $\omega(e_k)$、用户给出的截止时间 δ;算法输出为 CPC 算法生成的虚拟机集群及任务到具体虚拟机的调度方案。

算法主要分以下几步。第一步(语句 1 ~ 13):在截止时间约束下,通过计算关键路径长度确定第一个虚拟机实例 vm_1 的类型;然后将关键路径上的任务调度到 vm_1 上,同时更新 vm_1 各项参数,并将 vm_1 加入到 ***vmList*** 中。其中,语句 2 ~4计算关键路径总权重,语句 5 ~ 12 设置 vm_1 的相关信息。第二步(语句 14):将未分配的任务加入到 ***remainTaskList*** 列表。第三步(语句 15 ~ 23):首先采取任务回填方法,尝试将未分配的单个任务调度到已存在的 ***vmList*** 中;然后使用路径调度方法,在剩余未分配的任务构成的子图中,计算出一条关键路径,新增虚拟机实例来完成任务。

算法 1 CPC 算法

Alg. 1 CPC algorithm

Name: CPC: Critical path cut algorithm
Input: $\boldsymbol{W}=\{\boldsymbol{V}, \boldsymbol{E}\}$; every $\omega(e_k)$; δ(deadline)
Output: ***vmList*** $=\{vm_k \mid k=1,2,\cdots\}$; $vm_k=\{id, vmType, sTime, aTime, rTime,$ ***scheTL***$\}$

```
1  FOR (from vmType_1 to vmType_N)
2      FOR(each t_i ∈ V) ω(t_i) = seq(t_i) * λ(vmType_i)
3      {kpEL, kpTL} = getCriticalPath(V, E)
4      exeTime = Σ_{t_i∈kpTL} ω(t_i) + Σ_{e_k∈kpEL} ω(e_k)
5      IF(exeTime < δ)
6        New first VM: vm_1;
7            vm_1. vmType = curType; //vm type of
             first VM
8            vm_1. scheTL = kpTL; //task scheduled
             on vm_1
9            update vm_1's sTime, aTime and rTime;
10       add vm_1 to vmList;
11       break;
12     END IF
13 END FOR
14 update remainTaskList;
15 While(exist(t_i) in remainTaskList)
16   FOR(each t_i ∈ remainTaskList)
17     FOR(each vm_k ∈ vmList)
18       placeTaskOnExistVM( ); //schedule single
         task
19     END FOR
20   END FOR
21   placeTaskOnNewVM( );      //schedule path
22   update remainTaskList;
23 END While
```

2.2 任务回填算法

CPC 算法第 18 步算法 placeTaskOnExistVM 尝试将单个任务放到已经申请的虚拟机空闲时间槽中,以提高资源利用率,如算法 2 所示。

该算法尝试将 t_i 调度到 vm_k 上。语句 1 计算 t_i 在 vm_k 上的实际执行时间。语句 3 ~ 8 对 t_i 在 vm_k 的 number(vm_k. ***scheTL***) +1 个可能的插入位置进行逐一尝试。

其中 canPlace(t_i,j) 计算是否能够将 t_i 放在 vm_k 的任务列表 ***scheTL*** 的第 j 个位置,并返回一个布尔值。特别地,只有当以下两个条件同时满足时,canPlace(t_i,j) 才返回 true,调度成功:① t_i 插入到 ***scheTL*** 的第 j 个位置后,vm_k 上 ***scheTL*** 中位于 t_i 前面的任务不包括 t_i 的子任务,位于 t_i 后面的任务不包括 t_i 的父任务,保证任务间存在的依赖关系; ② t_i 插入到 ***scheTL*** 的第 j 个位置时,计算所有任务的 *EST* 和 *LST*,保证 max{*LST*(t_i) +

$\omega(t_i)\}$的值小于截止时间 δ。

算法 2　任务回填算法

Alg. 2　Task backfill algorithm

Name: placeTaskOnExistVM
Input: t_i; vm_k; $\boldsymbol{W}=(\boldsymbol{V},\boldsymbol{E})$
Output: flag;

```
1  ω(t_i) = seq(t_i) * λ(vm_k. vmType);
2  IF(vm_k. rTime > ω(t_i))
3    FOR(j = 0; j ≤ number(vm_k. scheTL); j + +)
4      IF(canPlace(t_i, j))
5        Insert(vm_k. scheTL, j, t_i);
6        return true;
7      END IF
8    END FOR
9    return false;
10 ELSE return false;
```

2.3　路径调度算法

当单个任务无法回填到已有虚拟机的空闲时间槽时,CPC 算法启动路径调度算法 placeTaskOnNewVM。该方法在剩余未分配的任务构成的子图中,计算出一条关键路径,并新增虚拟机实例来完成这条关键路径上的任务,如算法 3 所示。

算法 3　路径调度算法

Alg. 3　Path scheduling algorithm

Name: placeTaskOnNewVM
Input: ***remainTaskList***; ***vmList***; $\boldsymbol{W}=(\boldsymbol{V},\boldsymbol{E})$; δ(deadline);
Output: empty

```
1  FOR(from vmType_1 to vmType_N)
2    FOR(each t_i ∈ remainTaskList) ω(t_i) = seq(t_i) *
     λ(currentVmType);
3    calculate all tasks' EST;
4    calculate all tasks' LST
5    FOR(each t_i ∈ remainTaskList)
6      FinishTime(t_i) = LST(t_i) + ω(t_i);
7      IF(FinishTime(t_max) < FinishTime(t_i)) t_max = t_i;
8    END FOR
9    IF(FinishTime(t_max) < δ)
10     childGraph = getChildGraph(t_max, remainTaskList, E);
11     childCriticalPath = getCriticalPath(childGraph);
12     new VM: vm_k;
13       vm_k. vmType = curType;    //vm type
14       update vm_k's sTime, aTime, rTime
15       vm_k. scheTL = childCriticalPath;    //task
         scheduled on vm_i
16     add vm_k to vmList;
17     break;
18   END IF
19 END FOR
```

算法中语句 2 ~ 4 计算出所有任务的 *EST* 和 *LST* 值;语句 5 ~ 8 找出 ***remainTaskList*** 完成时间最晚的任务 t_{max},FinishTime(t_i)表示任务执行完毕的时间点;语句 10,从 t_{max} 出发,在 ***remainTaskList*** 中找到最大连通子图;语句 11,计算该连通子图的关键路径,同时计算出该关键路径的开始时间和结束时间;语句 12 ~ 16,启动一个新的 VM 以执行连通子图关键路径上的任务,并将该 VM 加入 ***vmList***。

2.4　简单示例

以图 1 所给 DAG 为例,CPC 算法具体执行过程如图 3 所示。

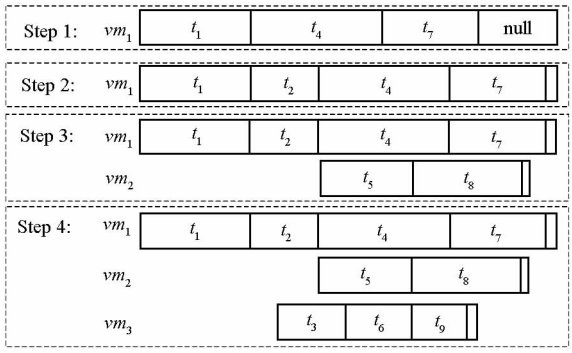


图 3　CPC 算法示例

Fig. 3　Example for CPC algorithm

第一步:找到 DAG 中最晚完成的任务 t_7,并从 t_7 出发找到最大连通子图,求得关键路径 $\{t_{start},t_1,t_4,t_7,t_{finish}\}$,然后根据关键路径长度申请虚拟机 vm_1,虚拟机类型为单核,生存时间为 120 min。第二步:尝试将其余任务放入 vm_1,发现 t_2 符合条件,放置成功。第三步:从未分配的任务中找到最晚完成的任务 t_8,然后从 t_8 出发找到最大连通子图,并求得关键路径 $\{t_5,t_8\}$,再根据关键路径长度申请虚拟机 vm_2,虚拟机类型为单核,生存时间为 60 min;尝试将其余任务放置到 vm_2,剩余空间不够,适配失败。第四步:找到未分配任务中最晚完成的任务 t_9,并从 t_9 出发找到最大连通子图,求得关键路径 $\{t_3,t_6,t_9\}$,然后根据关键路径长度申请虚拟机 vm_3,虚拟机类型为单核,生存时间为 60 min;任务分配完毕,算法结束。

可以看出,CPC 算法能尽量减少虚拟机的空闲时间碎片,最可能提高资源使用率,减少用户使用费用。

2.5　时间复杂度

如算法 1 所示,CPC 算法中语句 1 对可选的虚拟机类型进行遍历,由于可选类型一般不超过 10,所以复杂度为 O(1);语句 3 计算关键路径,复杂度

为 $O(n^2)$,因此语句 1 ~ 13 复杂度为 $O(n^2)$。第 18 句任务回填中需要计算所有任务的 *EST* 和 *LST*,计算复杂度为 $O(q \cdot n)$,其中 q 是每个 VM 上任务的平均数量;第 21 句任务调度需要计算剩余任务的关键路径,复杂度为 $O(k^2)$;那么 15 ~ 23 句的复杂度为 $O(k) \cdot O(p) \cdot O(q \cdot n) + O(k^2) = O(k \cdot p \cdot q \cdot n) + O(k^2)$,其中 p 为已存在的 VM 数量,k 为剩余任务数量,那么有 $k = n - p \cdot q$,所以 $O(k \cdot p \cdot q \cdot n) + O(k^2) = O(k \cdot (n-k) \cdot n) + O(k^2) = O(n^2k - nk^2) + O(k^2)$,其中 k 平均值为 n/2,所以 15 ~ 23 句复杂度为 $O(n^3)$。

综上,CPC 算法整体时间复杂度为 $O(n^3)$。

3 算法验证

3.1 测试用例

采用真实的 DAG 科学应用 Montage 科学工作流来对算法进行验证。Montage 科学工作流是 NASA/IPAC 开发的一个天文学应用,该应用能够利用 FITS 格式的图像生成自定义的天空拼接图。

3.2 云平台仿真

采用云仿真工具 CloudSim[8] 对云计算平台进行模拟,主要使用 VirtualMachine 类模拟虚拟机,使用 VMCharacteristics 类对虚拟机特征进行描述,并在 VMScheduler 类中实现不同的调度策略。

计费模式采用 Amazon EC2 的计费模式,即虚拟机实例按时间周期计费,单 VCPU 虚拟机实例当前价格取 0.16 $/h。实验使用的四种通用类型虚拟机如表 1 所示。

表 1 费用模式

Tab. 1 Cost model

VM 类型	性能指标	价格/($/h)
type 1	1 VCPU,1 G 内存,50 G 硬盘空间	0.16
type 2	2 VCPU,2 G 内存,100 G 硬盘空间	0.32
type 3	4 VCPU,8 G 内存,200 G 硬盘空间	0.64
type 4	8 VCPU,16 G 内存,500 G 硬盘空间	1.28

3.3 对比算法

采用混合最早完成时间(Heterogeneous Earliest Finish Time, HEFT),IC-PCP 和 IC-PCPD2 算法作为对比算法。

HEFT 算法:根据任务的平均执行时间和通信时间计算出一个 HEFT 值,在任务分配时,选择具有最大 HEFT 值的任务并将其调度到完成时间最小的虚拟机上。

IC-PCP 和 IC-PCPD2 算法:首先初始化关键路径上任务的截止时间,然后采用递归的方法依次求取其他任务的截止时间,之后依次求取偏序关键路径 PCP,在截止时间约束下将 PCP 路径上任务放置到可将其完成的最便宜的虚拟机上执行,其中 IC-PCP 算法每次尝试将该路径上的任务放置到虚拟机实例上,而 IC-PCPD2 算法则是计算偏序关键路径上的任务的子截止时间,并对任务的子截止时间进行设定,任务的分配则根据其子截止时间进行合理调度。IC-PCP 和 IC-PCPD2 算法复杂度为 $O(n^2)$。

3.4 实验结果

本文主要对比两个实验参数:总体费用和资源使用率。

总体费用指完成同一个科学工作流所需的整体费用,即费用 $= \sum_{i=1}^{N} price_{vm_i} \cdot aTime_{vm_i}$,其中 $price_{vm_i}$ 表示 vm_i 单位时间的收费标准,$aTime_{vm_i}$ 表示 vm_i 的生存时间。资源使用率指 CPU 的实际使用占所申请的整体 CPU 资源的比例,即:

$$\text{资源使用率} = 1 - \frac{\sum_{j=1}^{N} rTime_{vm_j} \cdot cpuNum_{vm_j}}{\sum_{i=1}^{N} aTime_{vm_i} \cdot cpuNum_{vm_i}}$$

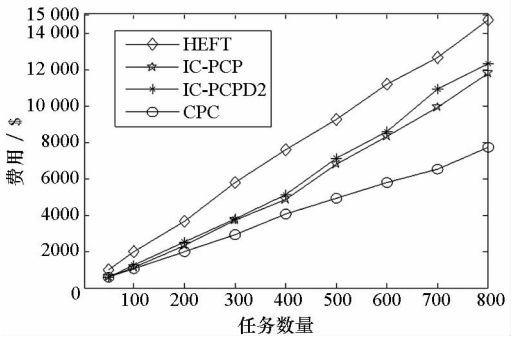
式中:$rTime_{vm_j}$ 表示 vm_j 的剩余时间,即空闲时间。

3.4.1 任务数量变化对结果的影响

实验中,虚拟机实例类型采用表 1 所给,虚拟机实例时间周期长度为 60 min,截止时间取 $\delta \in [\delta_{\text{short}}, \delta_{\text{long}}]$,任务数量取值范围为{50, 100, 200, 300, 400, 500, 600, 700, 800},任务并行比例 η 服从[0.1, 0.9]的均匀分布。实验结果如图 4 所示。

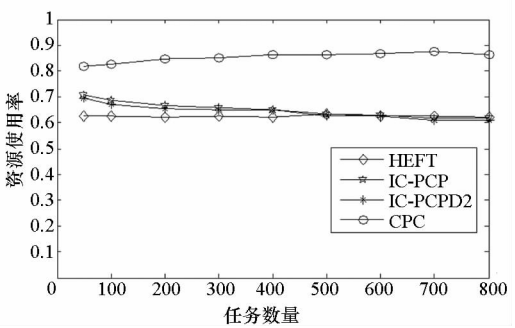
从图 4(a)可以看出,随着 Montage 任务数量的增加,HEFT, IC-PCP, IC-PCPD2 和 CPC 算法完成 DAG 应用所需费用几乎线性增加。由于时间周期为较长的 60 min,IC-PCP 略优,比 IC-PCPD2 费用减少约 2%;而 CPC 算法比 IC-PCP 算法费用减少了约 3% ~ 29%。从图 4(b)可以看出,CPC 算法的资源使用率在 80% 左右,而 IC-PCP 和 IC-PCPD2 算法的资源使用率为 57% ~ 70%。

这是因为 CPC 算法不仅采用关键路径调度,对大堆任务进行合理调度,同时还采用灵活的任务回填方法,尝试将单个任务放置到虚拟机时间空闲槽,成功率高。因此 CPC 算法对虚拟机实例的空闲时间槽利用率高,从而提高了资源使用率。而且任务数量越大,所需虚拟机实例越多,这种优势就越明显。



(a) 费用与任务数量的变化曲线

(a) Total cost with different task number



(b) 资源使用率与任务数量的变化曲线

(b) Resource utilization with different task number

图 4　任务数量变化对结果影响

Fig. 4　Results with different task number

3.4.2　DAG 截止时间变化对结果的影响

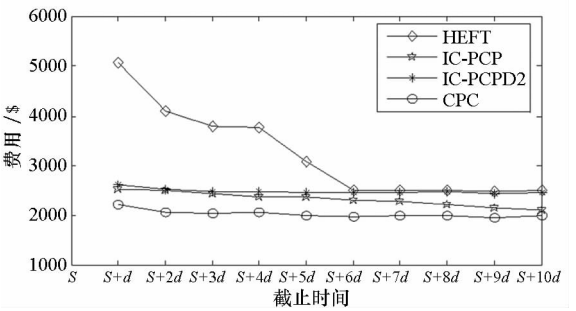
实验中,虚拟机实例类型采用表 1 所给,任务数量选取 200,任务并行比例 η 服从[0.1,0.9]的均匀分布,虚拟机实例时间周期长度为60 min。实验结果如图 5 所示,图中,$S=\delta_{\text{short}}$,$d=(\delta_{\text{long}}-\delta_{\text{short}})/9$ 表示将[δ_{short}, δ_{long}]划分为 9 等份,那么 $\delta_{\text{long}}=S+9d$。

从图 5(a)可以看出,随着截止时间 δ 取值的增加,HEFT 所需费用越小,而 IC-PCP,IC-PCPD2 和 CPC 算法变化不大,这是因为后三种算法已经最大化利用了费用优化空间。图 5(b)反映了 CPC 算法的资源使用率相比 IC-PCP、IC-PCPD2 和 HEFT 算法的较优。

这是因为 IC-PCP、IC-PCPD2 和 CPC 算法都采用了路径调度策略,分别计算了偏序关键路径和关键路径,完工时间与路径长度相关,充分利用了优化空间,受截止时间影响较小。即使截止时间变长,对费用和资源使用率的影响也有限。

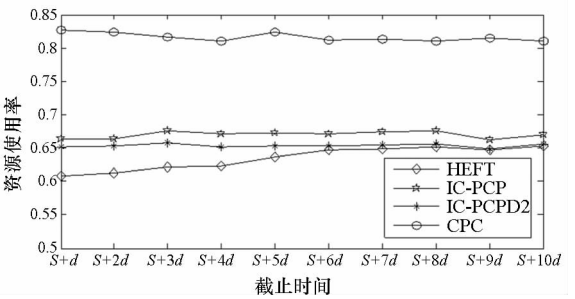
3.4.3　时间周期变化对结果的影响

实验中,虚拟机实例类型采用表 1 所给,截止时间 $\delta\in[\delta_{\text{short}}, \delta_{\text{long}}]$,任务数量选取 200,任务并行比例 η 服从[0.1,0.9]的均匀分布,虚拟机实例时间周期长度选取{60,50,40,30,20,10,5},



(a) 费用与截止时间的变化曲线

(a) Total cost with different deadlines



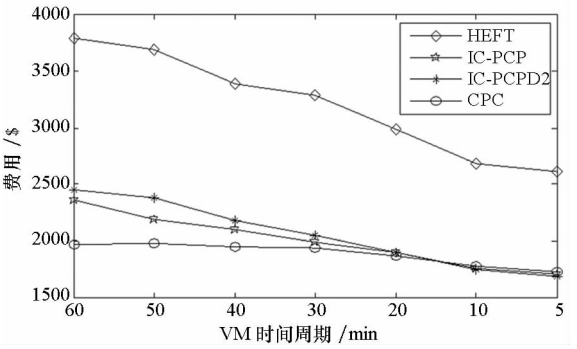
(b) 资源使用率与截止时间的变化曲线

(b) Resource utilization with different deadlines

图 5　截止时间 δ 对结果的影响

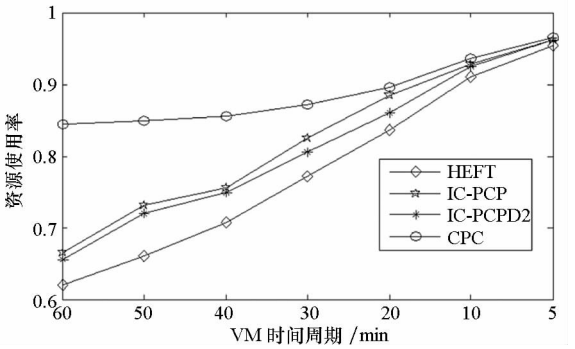
Fig. 5　Results with different deadline δ

单位为 min。实验结果如图 6 所示。



(a) 费用与 VM 时间周期的变化曲线

(a) Total cost with different VM time intervals



(b) 资源使用率与 VM 时间周期的变化曲线

(b) Resource utilization with different VM time intervals

图 6　VM 时间周期长度对结果的影响

Fig. 6　Results with different time intervals of VM

从图 6(a)可以看出,随着虚拟机时间周期长度的减少,HEFT、IC-PCP、IC-PCPD2 和 CPC 算法的费用也不断减少。时间周期为 30 ~ 60 min 时,CPC 算法比 IC-PCP 算法所需费用少约 4% ~ 13%,但时间周期为 5 ~ 20 min 时,两者费用相当。从图 6(b)看出,四种算法在 VM 时间周期减少的过程中,资源使用率不断提高;时间周期越短,CPC 算法的优势就越小。

这是因为时间周期越短,虚拟机实际空转时间变短,资源利用率就越高,因此所需的花费越少。

4 结论

本文研究云计算平台上截止时间约束下 DAG 科学应用的费用优化问题,根据云计算平台资源特性,设计了基于关键路径截取的 DAG 调度算法。该算法通过分析任务间依赖关系,采取关键路径截取技术将任务聚合在匹配的虚拟机上执行,通过路径调度完成任务集到资源的映射,同时通过任务回填方法将单个任务调度到已分配虚拟机的空闲时间槽上执行,提高了虚拟机的资源利用率。最后,利用真实科学工作流 Montage 对 CPC 算法进行了验证。结果表明,CPC 算法可以根据 DAG 应用内在特点,配置合适的虚拟机集群资源,并对任务到虚拟机进行合理调度,不仅能够在截止时间内完成 DAG 应用,而且可以有效减少执行费用。

参考文献(References)

[1] Bharathi S, Chervenak A, Deelman E, et al. Characterization of scientific workflows[C]//Proceedings of Third Workshop on Workflows in Support of Large-Scale Science, 2008: 1 - 10.

[2] Moens H, Handekyn K, De Turck F. Cost-aware scheduling of deadline-constrained task workflows in public cloud environments[C]//Proceedings of IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), 2013: 68 - 75.

[3] Ma Y L, Shi M Y, Wei J. Cost and accuracy aware scientific workflow retrieval based on distance measure[J]. Information Sciences, 2015, 314: 1 - 13.

[4] 刘灿灿, 张卫民, 骆志刚. 基于逆向分层的工作流时间 - 费用优化方法[J]. 国防科技大学学报, 2013, 35(3): 61 - 66.
LIU Cancan, ZHANG Weimin, LUO Zhigang. Time and cost trade-off heuristics for workflow scheduling based on bottom level[J]. Journal of National University of Defense Technology, 2013, 35(3): 61 - 66. (in Chinese)

[5] Byun E K, Kee Y S, Kim J S, et al. Cost optimized provisioning of elastic resources for application workflows[J]. Future Generation Computer Systems, 2011, 27(8): 1011 - 1026.

[6] Byun E K, Kee Y S, Kim J S, et al. BTS: resource capacity estimate for time-targeted science workflows[J]. Journal of Parallel and Distributed Computing, 2011, 71(6): 848 - 862.

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

[8] Calheiros 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 Experience, 2011, 41(1): 23 - 50.