

基于关键路径前瞻的工作流调度算法

孙 婷, 肖创柏, 张雅琴, 徐秀杰
(北京工业大学信息学部, 北京 100124)

摘 要: 为了进一步提高工作流调度的效率,对异构环境下的工作流调度算法进行研究,提出了一种基于关键路径前瞻算法(critical path lookahead algorithm,CPLA)的工作流调度算法.该算法在任务确定优先级阶段考虑了入口任务到当前任务的最长路径,以及当前任务到出口任务的最长路径;在资源选择阶段考虑了关键路径任务和非关键路径任务对调度结果的影响.使用随机生成的和真实世界的2种不同类型的有向无环图(directed acyclic graph,DAG)工作流来评估CPLA的性能,实验结果表明:CPLA能够有效地缩短调度长度,算法的效率、加速比、调度长度比等性能有所提高,并且算法的时间复杂度低于目前调度效果最好的Lookahead算法.

关键词: 静态工作流调度; 关键路径; 非关键路径; 加速比
中图分类号: TP 301.6; TP 338.8 **文献标志码:** A **文章编号:** 0254-0037(2018)08-1136-09
doi: 10.11936/bjutxb2017050046

Workflow Scheduling Algorithm Based on Critical Path Lookahead

SUN Ting, XIAO Chuangbai, ZHANG Yaqin, XU Xiujie
(Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China)

Abstract: To further improve the efficiency of workflow scheduling, a workflow scheduling algorithm in heterogeneous environment was studied, and a workflow scheduling algorithm based on critical path lookahead algorithm (CPLA) was proposed. In the task priority stage, the longest path of the entry task to the current task was considered, and the longest path of the current task to exit task was also taken into consideration. In the resource selection stage, the impact of tasks of the critical path and tasks of the non critical path on the scheduling results were considered. Two different types of DAGs which were randomly generated and from real-world were used to evaluate the performance of CPLA. The experimental results show that the CPLA can effectively shorten the scheduling length. The performances of this algorithm, such as efficiency, speedup, scheduling length ratio of algorithm are improved, and the time complexity of the algorithm is lower than that of the lookahead algorithm.

Key words: static workflow scheduling; critical path; non critical path; speedup

随着科技信息的发展,很多领域的科学和工程计算问题都被抽象为工作流模型^[1].这些工作流模型一般由一组有前后依赖关系的任务节点组成,可以抽象为有向无环图(directed acyclic graph,DAG).将给定工作流任务作为对象,若干处理器作为完成任务的资源,在满足一定的约束条件下,对任务和处理器进行分配并安排先后次序,将所需的资源进行最优调度,称为工作流调度问题.调度问题在物流、航空、生物、医疗等各领域都有广泛的应用.工作流调度问题已经被证明是多项式复杂程度的非

收稿日期: 2017-05-23
基金项目: 北京市自然科学基金资助项目(4162007);国家自然科学基金资助项目(61501008)
作者简介: 孙 婷(1990—),女,博士研究生,主要从事分布式计算、资源调度方面的研究,E-mail:sunting07@emails.bjut.edu.cn

确定性 (non-deterministic polynomial, NP) 完全问题, 即不能在多项式时间里找到问题的最优解^[2].

随着科学工作流问题的计算需求不断增长, 分布式异构资源的工作流调度被广泛研究^[3-13]. 分布式异构资源工作流调度问题根据采用方式不同, 可分为基于启发式 (heuristic-based) 和基于引导随机搜索 (guided-random-search-based) 两大类, 前者还可以细分为 3 类, 分别为列表调度启发式 (list-scheduling heuristics)、聚类启发式 (clustering heuristics)、基于复制的启发式 (duplication-based heuristics). 这些算法都分为任务优先级确定和资源选择 2 个阶段. 在列表调度启发式^[3-5]中, 任务依据每个任务被赋予的优先级进行调度, 每个被选择的任务将映射到能够满足预先定义目标函数的处理器上. 例如, 将任务映射到完成时间最早的处理器上进行调度. 此外, 列表调度算法又分为静态调度和动态调度. 在静态调度算法中, 任务在调度之前优先级就已确定, 因此, 在实际调度过程中优先级不发生改变; 相反, 动态算法^[6]在每一次调度之后, 会重新计算未调度任务的优先级, 然后用新计算的优先级顺序重新进行调度. 聚类启发式^[7-8]在调度过程初始阶段, 按照算法规则将任务分成不同的簇, 称为任务簇, 然后按照不同的任务簇分配处理器, 同一簇的任务在同一个处理器上调度. 聚类算法通常是针对无限数量的处理器, 因此, 不能被实际应用. 基于复制的启发式算法^[9]假设有无限的处理器可供分配, 通过父任务的重复调度来减少处理器间的通信时间, 进一步减少调度长度. 随机搜索技术中最流行并且应用最广泛的一种算法是遗传算法 (genetic algorithm, GA)^[10], 虽然遗传算法调度质量很高, 但是它的执行时间远长于其他类算法.

由于列表调度启发式算法简单高效, 该算法已被广泛研究, 其中最著名的算法是 Topcuoglu 等^[3]在 2002 年提出的异构分布式环境下的最早完成时间优先 (heterogeneous earliest-finish-time, HEFT) 调度算法. HEFT 算法已成功在 Askalon 异构分布式系统中应用, 同时 Topcuoglu 等^[3]还提出了基于关键路径任务映射到一个处理器的 (critical-path-on-a-processor, CPOP) 算法. 该算法定义了 DAG 工作流的关键路径, 在处理器选择阶段, 分别考虑关键路径任务和非关键路径任务, 但是 CPOP 算法的调度效果并没有 HEFT 算法的好; 因此, HEFT 算法被广泛应用. 2010 年, Bittencourt 等^[14]对 HEFT 算法进行改进, 提出了前瞻 (lookahead) 算法. 该算法的调度

决策不止依赖于任务本身, 还依赖于任务的执行结果. Lookahead 算法在调度前先获得子任务的调度信息, 然后再综合考虑当前任务和子任务的调度, 对这个任务做出最优的资源分配决策, 但是这增加了算法的时间复杂度. 2014 年, Arabnejad 等^[15]针对异构分布式系统下的工作流调度进一步提出了预测最早完成时间 (predict earliest finish time, PEFT) 算法. PEFT 算法使用任务的乐观花费表 (optimistic cost table, OCT) 中的平均值作为任务的优先级, OCT 是求子任务到出口任务的最长路径, 即 PEFT 算法在任务确定优先级阶段, 考虑了任务本身的调度决策对后继任务的影响, 在处理器选择阶段又考虑了任务的前驱任务对任务调度的影响, 从 DAG 的结构来说, 考虑得十分全面. PEFT 算法的时间复杂度与 HEFT 算法一样, 但是在 DAG 的并行度比较高的情况下, 大量实验表明 PEFT 算法的调度效果并不理想. 除了异构分布式环境中的应用, 启发式算法还应用到云环境中, Guo 等^[16]在云环境下对工作流用资源聚类的方法进行调度. Mohanapriya 等^[17]针对云环境下的多目标调度问题给出了调度算法的分类.

本文基于现有研究, 对异构分布式环境下的工作流调度问题, 提出了基于关键路径前瞻算法 (critical-path lookahead algorithm, CPLA) 的工作流调度算法. 该算法在任务确定优先级阶段考虑了入口任务到当前任务的最长路径, 以及当前任务到出口任务的最长路径; 在处理器选择阶段, 对于非关键路径任务, 考虑直接前驱任务的影响, 找到前驱任务到当前任务的最短路径. 对于关键路径任务, 不仅考虑了任务前驱到当前任务的最短路径, 还考虑到其子任务到出口任务的最长路径, 使当前任务的决策有一定的预见性. 综合来看, CPLA 充分利用了关键路径上子任务对整个调度的影响, 在性能与 Lookahead 算法相近的基础上降低了时间复杂度.

1 DAG 调度模型

1.1 问题描述及模型建立

工作流模型可以用一个 DAG 来表示, 记 DAG 为 $G = \{V, E\}$, 任务的集合表示为 $V = \{n_1, n_2, \dots, n_n\}$, 每一个 n_i 代表一个计算任务, $|V| = n$ 表示工作流中所有任务节点的个数为 n ; E 为有向边的集合, 表示任务间的前后依赖关系, 对于任意的边 $(n_i, n_j) \in E$, 任务 n_i 执行后才执行任务 n_j . 在 DAG 中没有前继的任务称为入口任务, 记作 n_{entry} , 没有后继的

任务称为出口任务,记作 n_{exit} . 处理器的集合记为 $P = \{p_1, p_2, \dots, p_q\}$, $|P| = q$ 表示有 q 个处理器,每一个任务 n_i 都要在处理器 p_j 上执行且存在执行时间 ω_{ij} , 每一个任务 n_i 的平均执行时间为

$$\bar{\omega}_i = \sum_{j=1}^q \omega_{ij}/q \quad (1)$$

任务间的通信数据使用一个 $n \times n$ 大小的矩阵 D 来表示, d_{ij} 表示任务 n_i 和任务 n_j 之间所需要通信的数据总和. 处理器之间传送数据的速度使用一个 $q \times q$ 大小的矩阵 B 来存储. 处理器间通信的启动时间存储在一个 q 维的矢量 L 中. 任务的通信传递时间使用一个 $n \times n$ 大小的矩阵 C 来表示, c_{ik} 表示在处理器 p_m 上的任务 n_i 到处理器 p_n 上的任务 n_k 的传递时间,公式为

$$c_{ik} = L_m + d_{ik}/b_{mn} \quad (2)$$

式中: L_m 为处理器 p_m 的启动时间; b_{mn} 为处理器 p_m 到处理器 p_n 间的传递速度. $\bar{c}_{ik}$ 表示在处理器 p_m 上的任务 n_i 到处理器 p_n 上的任务 n_k 的平均传递时间,公式为

$$\bar{c}_{ik} = \bar{L} + d_{ik}/\bar{B} \quad (3)$$

式中: $\bar{L}$ 为处理器的平均启动时间; $\bar{B}$ 为处理器间的平均传递速度. 若任务 n_i 与任务 n_k 在同一处理器上进行传递,则传递时间为 0.

1.2 任务调度中的基本定义

下面给出 workflow 调度中常用的基本定义.

定义 1 DAG 中从入口任务到出口任务最长的路径称为 DAG 的关键路径 (critical path, CP), 关键路径长度值为 $|L_{\text{CP}}|$. 若 DAG 中所有任务的执行时间最小, 那这样的关键路径也有最小的执行时间, 记这样的关键路径为 $\text{CP}_{\min}$.

定义 2 关键路径上的任务称为关键路径任务.

定义 3 定义 $\text{pred}(n_i)$ 为 DAG 中任务 n_i 的前继任务集合, $\text{succ}(n_i)$ 为 DAG 中任务 n_i 的后继任务集合.

定义 4 任务 n_i 在处理器 p_j 上的最早开始时间记为 T_{ES} , 定义为

$$T_{\text{ES}}(n_i, p_j) = \max \left\{ T_j, \max_{n_m \in \text{pred}(n_i)} (T_{\text{AF}}(n_m) + c_{m,i}) \right\} \quad (4)$$

式中: $\text{pred}(n_i)$ 表示任务 n_i 的前继任务集合; T_j 表示任务在处理器 p_j 上可以开始的最早时间; $T_{\text{AF}}(n_m)$ 表示任务 n_i 的前继任务 n_m 的实际完成时间; $c_{m,i}$ 表示任务 n_m 到任务 n_i 之间的传递时间, 内层 $\max$ 表示任务 n_i 的所有前继任务 n_m 到达处理器

p_j 的时间, 对于入口任务 n_{entry} , 到达任何处理器的时间为 0, 即 $T_{\text{ES}}(n_{\text{entry}}, p_j) = 0$.

定义 5 任务 n_i 在处理器 p_j 上的最早完成时间记为 T_{EF} , 定义为

$$T_{\text{EF}}(n_i, p_j) = T_{\text{ES}}(n_i, p_j) + w_{i,j} \quad (5)$$

式中 $w_{i,j}$ 表示任务 n_i 在处理器 p_j 上的执行时间.

2 DAG 调度的 CPLA

2.1 CPLA 的具体描述

为提高 DAG 调度的效率, CPLA 的调度过程也分 2 个阶段.

1) 任务优先级确定阶段

任务优先级确定阶段, 顾名思义就是按照一定的规则赋予每个任务优先级, 然后按照优先级大小对任务进行排序, 形成一个任务队列, 最后按照这个队列中的顺序选择任务, 并传送给第 2 阶段处理. 队列的构建有 2 种方式: 一种是静态队列, 另一种是动态队列. 与静态队列相比, 动态队列使用入口任务来初始化, 每次只存放当前时刻的就绪任务.

CPLA 采用动态列队, 由入口任务到出口任务的最长路径来确定任务的优先级. DAG 中任务 n_i 的向上排序值 r_u 为从任务 n_i 到出口任务的多条路径中平均执行时间和平均通信传递时间总和的最大值, 其中也包括这个任务本身的执行时间平均值. 一个任务 n_i 的向上权值 r_u 为

$$r_u(n_i) = \bar{\omega}_i + \max_{n_j \in \text{succ}(n_i)} \{r_u(n_j) + \bar{c}_{ij}\} \quad (6)$$

式中: $\text{succ}(n_i)$ 为 DAG 中任务 n_i 的后继任务集合; $\bar{c}_{ij}$ 为任务 n_i 与任务 n_j 之间的平均传递时间, 对于出口任务 n_{exit} , $r_u(n_{\text{exit}}) = 0$.

DAG 中向下权值 r_d 为入口任务到当前任务 n_i 的多条路径中平均执行时间和平均通信传递时间总和的最大值, 其中不包括这个任务 n_i 本身的执行时间平均值. 一个任务 n_i 的向下权值 r_d 为

$$r_d(n_i) = \max_{n_j \in \text{pred}(n_i)} \{r_d(n_j) + \bar{\omega}_j + \bar{c}_{j,i}\} \quad (7)$$

式中: $\text{pred}(n_i)$ 为任务 n_i 的直接前驱任务集合; $\bar{\omega}_j$ 为任务 n_j 在各个处理器上的平均执行时间; $\bar{c}_{j,i}$ 为任务 n_j 与任务 n_i 之间的平均传递时间. 对于入口任务 n_{entry} , $r_d(n_{\text{entry}}) = 0$.

CPLA 采用的是由入口任务到出口任务的最长路径来确定任务的优先级, 即计算每个任务的优先级 l , 其中 $l = r_u + r_d$. 在任意给定时刻, 队列里只包含所有的就绪任务. 每次调度选择就绪任务中优先

级最高的任务,也就是 l 值最大的任务.

2) 处理器的选择阶段

在处理器选择阶段,任务选择处理器的条件就是选择完成时间最小的处理器. 其中时间包括任务在处理器上运行的计算时间和任务与任务之间传递数据的通信时间,处理器内部的通信时间一般可以忽略. 首先,计算任务在各个处理器上的执行时间,然后从中找到执行时间最小的处理器分配给该任务执行,重复这个过程直到所有的任务都调度完成. 由于同一个处理器上执行的任务会出现一定的空隙,为了能够充分利用时间空隙,提高资源的利用率,一般采用插入策略,插入的位置是在同一个处理器上 2 个已经调度的任务之间,也可能是在处理器可以开始执行任务到该处理器上第 1 个任务开始执行之间,并且满足任务能够在这段空闲时间内完成调度,同时还应该维持优先约束.

CPLA 针对关键路径任务和非关键路径任务采用 2 种不同的策略:首先判断任务是否是关键路径任务,如果是关键路径任务,则尝试在每个处理器上运行这个任务,选择其子任务中最早完成时间最小的处理器;如果不是关键路径任务,则选择任务本身最早完成时间最小的处理器进行调度. CPLA 使任务的决策有一定的预见性,又降低了算法的时间复杂度,图 1 是算法的主要流程.

CPLA 的步骤如下.

- 1: 由式(1)(2)计算 DAG 中每个任务的平均执行时间和每 2 个任务间的平均通信时间
- 2: 从出口任务开始,根据式(6)计算所有任务的 r_u 值
- 3: 从入口任务开始,根据式(7)计算所有任务的 r_d 值
- 4: 以 $l = r_u + r_d$ 的值为任务的优先级,初始化任务队列 M
- 5: 以入口任务优先级的值为关键路径长度 L_{CP} ,并初始化关键路径任务集 $M_{CP} = \{n_{entry}\}$
- 6: $n_k < -n_{entry}$
- 7: if n_k 不是出口任务
- 8: 找到任务的后继任务中优先级 l 等于 L_{CP} 的任务 n_j
- 9: 更新关键路径任务的集合, $M_{CP} = M_{CP} \cup \{n_j\}$
- 10: end if
- 11: while 队列 M 里还有未调度的任务
- 12: 从任务列表中选择优先级最高的任务 n_i
- 13: if n_i 在 M_{CP} 中
- 14: 求任务 n_i 的直接后继任务集合 P_r
- 15: 对于处理器集合 P 中的每一个处理器 p_j

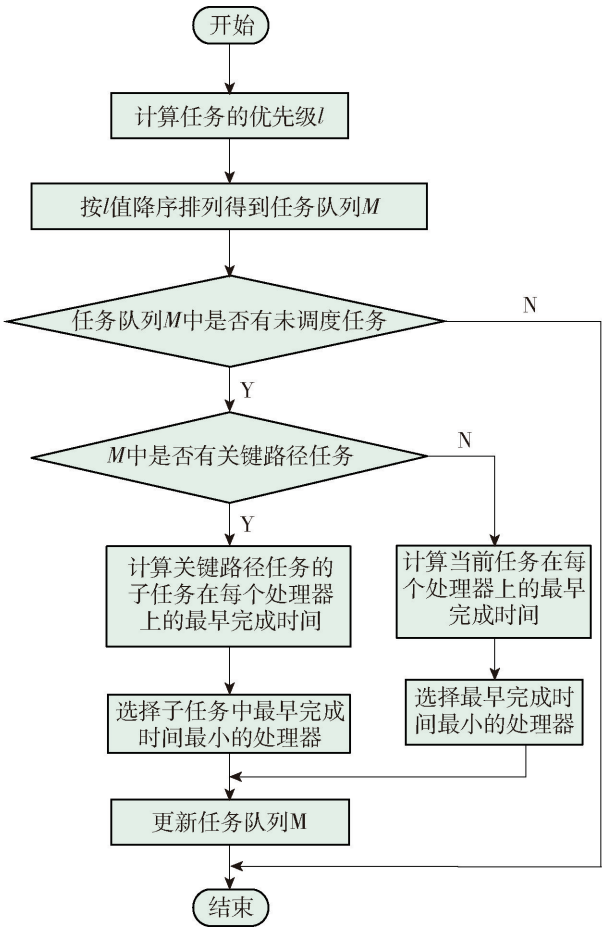


图 1 CPLA 流程

Fig. 1 CPLA flow

- 16: 使用基于插入的调度策略,根据式(4)(5)计算任务 n_i 的 T_{ef}
- 17: if P_r 中还有未调度的子任务
- 18: 从 P_r 中选出优先级最高的任务 n_m
- 19: 对于处理器集合 P 中的每一个处理器 p_j
- 20: 使用基于插入的调度策略计算任务 n_m 的 T_{EF}
- 21: end if
- 22: end if
- 23: 将任务 n_i 安排到满足 n_m 的 T_{EF} 值最小的处理器 p_x 上
- 24: else
- 25: 对于任务 n_i ,根据式(4)(5)选择最早完成时间 T_{EF} 最小的处理器 p_j
- 26: 删除任务 n_i ,判断 n_i 的后继任务是否成为就绪任务,更新任务队列
- 27: end while

在 CPLA 中,任务确定优先级阶段考虑了入口

任务到当前任务的最长路径,以及当前任务到出口任务的最长路径.在处理器选择阶段,对于非关键路径任务,考虑直接前驱任务的影响,找到前驱任务到任务的最短路径;对于关键路径任务,不仅考虑了任务前驱到任务本身的最短路径,还考虑了其子任务到出口任务的最长路径,使任务的决策有一定的预见性,更有利于后面其子任务的调度.

2.2 算法的时间复杂度

对于 DAG,任务数为 n ,处理器总数为 q ,任务的平均子任务个数为 c ,DAG 中关键路径任务的平均子任务数为 $m(m < c)$.若任务是关键路径任务,算法的复杂度增加为子任务数与处理器个数的乘积,因此,对于关键路径任务,算法的时间复杂度增加的因子为 $m \times q$.CPLA 采用最早完成时间策略,DAG 中有 n 个任务,那么边的个数为 $n(n-1)$,最早完成时间算法的复杂度为 $O(n(n-1) \times q)$,即 $O(n^2q)$,因此,CPLA 的时间复杂度为 $O(mn^2q^2)$.HEFT 算法的时间复杂度为 $O(n^2q)$,Lookahead 算法的时间复杂度比 HEFT 算法多了一个因子 $c \times q$,复杂度为 $O(cn^2q^2)$,其中 c 是平均的子任务数,CPLA 的时间复杂度为 $O(mn^2q^2)$,因为 $m < c$,所以 CPLA 的时间复杂度比 Lookahead 算法的时间复杂度低.CPLA 从理论上来说结合了 HEFT 算法和 Lookahead 算法的中心思想,其时间复杂度又恰在两者之间,在调度质量和复杂度之间找到了一个平衡.

3 实验结果与分析

下面给出 DAG 任务调度算法中常用的比较度量.

定义 6 DAG 中所有任务实际完成时间的最大值称为调度长度,表示为

$$M = \max_{n_i \in V} T_{AF}(n_i) \quad (8)$$

式中 $T_{AF}(n_i)$ 表示任务 n_i 的实际完成时间.

一般来说,一个 DAG 工作流的调度长度越小,算法的调度质量越好.

定义 7 每个 DAG 的调度长度与所有关键路径任务在各个处理器上执行时间的最小值之和的比值称为算法的调度长度比(schedule length ratio, SLR),即

$$R_{SL} = \frac{M}{\sum_{n_i \in CP_{\min}} \min_{p_j \in P} |w_{i,j}|} \quad (9)$$

式中分母表示所有任务的执行时间最小的关键路径 $CP_{\min}$ 上的任务在各个处理器上的执行时间最小

值之和.因为分母是下界,所以任何算法的 SLR 值一般都不会小于 1,算法的 SLR 值越低,性能越好.

定义 8 DAG 中所有任务在同一处理器上顺次执行总执行时间的最小值与算法调度长度的比值称为加速比(speedup).对于 $n_i \in V, p_j \in P$,加速比的计算公式为

$$S = \frac{\min_{p_j \in P} \left\{ \sum_{n_i \in V} w_{i,j} \right\}}{M} \quad (10)$$

式中:分子表示所有任务在每一个处理器上顺次执行的总执行时间最小的处理器所对应的执行时间;分母表示调度长度.加速比一般体现出任务的并行度,比值越大,并行度越高.另外,加速比与处理器的数量有很大关系,计算资源越多,算法调度长度越短,加速比越大.但是对于计算资源大于 DAG 中最大可并行任务数量的情况下,再增加处理器对缩短算法调度长度是无效的.

定义 9 加速比与处理器数量的比值定义为效率(efficiency).假设处理器数量为 q ,效率的计算公式为

$$E = \frac{S}{q} \quad (11)$$

一般算法的效率越大,表明该调度算法的性能越好.

3.1 随机 DAG 的实验调度分析

使用一个随机 DAG 生成器^[18],生成器对于 DAG 的结构有 5 种参数:1) n 表示 DAG 的任务数.2) fat 表示影响 DAG 高度和宽度的参数,一般定义 DAG 的高度(层数)是均值为 $\text{fat}/\sqrt{n}$ 的均匀分布,每层的宽度(任务数)是均值为 $\text{fat}/\sqrt{n}$ 的均匀分布. fat 值越大,DAG 图的并行度越大.3) density 表示 DAG 中不同层之间边的密度,即优先关系的多少.4) regularity 表示 DAG 中每层任务总数的差别.5) jump 表示 DAG 中任务间通信跨越的最大层数,至少大于等于 1 才表示 2 个任务之间有优先约束. R_{cc} 表示 DAG 中所有边的平均通信时间之和与所有任务节点的平均执行时间的比值,称为通信计算比. β 表示 DAG 中任务节点在处理器上的执行时间的范围比, β 值越大,表示任务在不同处理器上的计算速度差别越大. q 表示处理器的个数.

在每次实验中,设置不同的参数值,得到不同的 DAG,参数设置为

$n = [30, 40, 50]$;

fat = [0.3, 0.7];
density = [0.2, 0.8];
regularity = [0.2, 0.8];
jump = [1, 2];
 $R_{cc} = [1, 2]$;
 $q = [4, 8, 12]$.

在这些参数下一共产生 96 个随机 DAG 进行实验. 用 PEFT、HEFT、Lookahead、CPOP 四种调度算法与 CPLA 进行对比实验.

图 2 和图 3 分别展示的是处理器个数为 3, R_{cc} 为 1 和 2 的情况下, 5 种算法的平均调度长度比和平均加速比. 由图 2 可以看出, CPLA 的平均调度长度比最小, CPOP 算法的平均调度长度比最大.

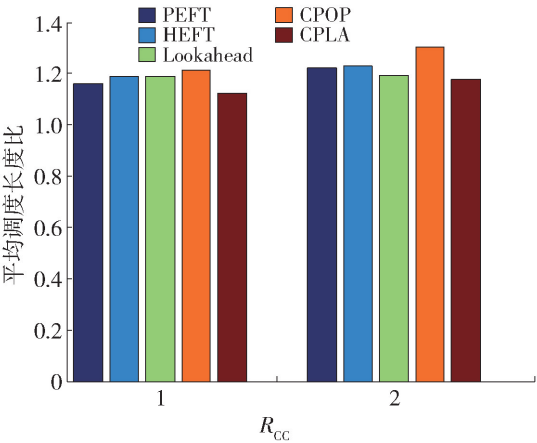


图 2 不同 R_{cc} 下算法的平均调度长度比

Fig. 2 Average scheduling length ratio of the algorithm under different R_{cc}

由图 3 可以看出, CPLA 的平均加速比最大, 性能最好, Lookahead 算法平均加速比次之, HEFT 算法和 PEFT 算法结果相似, CPOP 算法结果最差.

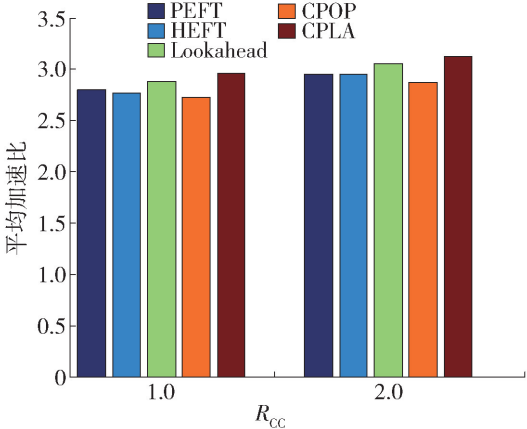


图 3 不同 R_{cc} 下算法的平均加速比

Fig. 3 Average speedup of the algorithm under different R_{cc}

图 4 表示 5 种算法分别在 4、8、12 个处理器下的效率. 可以看出, CPLA 和 Lookahead 算法的效率最高, HEFT 算法和 PEFT 算法的效率接近, CPOP 算法的效率最低.

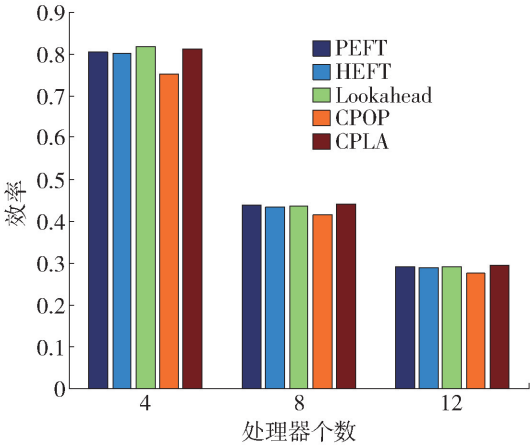


图 4 不同处理器数下算法的效率

Fig. 4 Efficiency of the algorithm under different processor numbers

图 5 表示 5 种算法在任务数量为 30、40、50 的 DAG 下的平均调度长度比. 可以看出, 在任务数为 30 和 40 的时候, CPLA 的 $\bar{R}_{sl}$ 最接近 1, 在任务数为 50 的时候, Lookahead 算法的 $\bar{R}_{sl}$ 最小, 但是在 3 种情况下, CPLA 的 $\bar{R}_{sl}$ 都比 CPOP 算法的小.

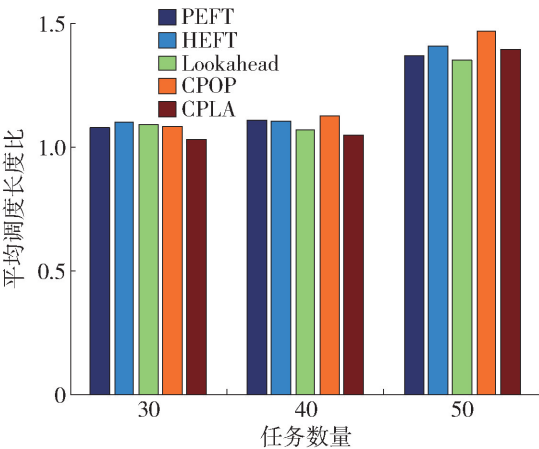


图 5 不同任务数下算法的平均调度长度比

Fig. 5 Average scheduling length ratio of the algorithm under different tasks

3.2 真实 DAG 的实验调度分析

除了随机产生的 DAG, 本文还采用真实的 DAG 来进行实验调度分析. 真实的 DAG 是根据真实的问题产生的图结构, 一些知名的真实的 DAG 应用有: Montage (天文学)^[19]、Epigenomics (生物学)^[20]、LIGO (引力波物理学)、CyberShake (地震波

物理学)等. 其中 Montage workflow 结构如图 6 所示.

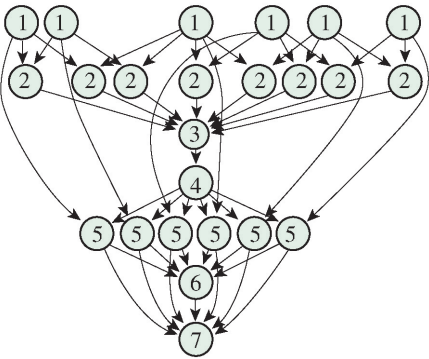


图 6 简单的 Montage workflow 结构
Fig. 6 Simple Montage workflow structure

分别用 25 个任务数的 Montage、24 个任务数和 100 个任务数的 Epigenomics 进行调度试验.

对于 25 个任务数的 Montage 任务图,设置 $R_{cc} = [0.5, 1, 2]$. 图 7 表示 5 种算法在处理器个数为 2 的计算环境下得到的平均调度长度. 图 8 表示 5 种算法在处理器个数为 5 的计算环境下得到的平均调度长度. 从 2 个图可以看出,CPLA 在不同的 DAG 下得到的平均调度长度都比 CPOP 算法的平均调度长度小. 图 7 中在 R_{cc} 为 1 的情况下,CPLA 的平均调度长度小于其他算法.

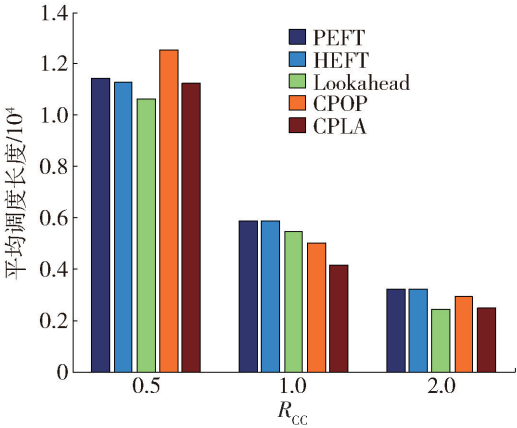


图 7 处理器个数为 2,不同 R_{cc} 下算法的平均调度长度
Fig. 7 Average makespan under different R_{cc} with two processors

在 24 个任务数的 Epigenomics 任务图下,图 9 表示 5 种算法在处理器个数为 5 的环境下得到的平均调度长度. 可以看出,在 3 种通信计算比下,CPLA 比 CPOP 算法的平均调度长度小.

在 100 个任务数的 Epigenomics 任务图下,图 10 表示 5 种算法在处理器个数为 10 的环境下得到的平均调度长度. 可以看出,CPLA 的平均调度长度比

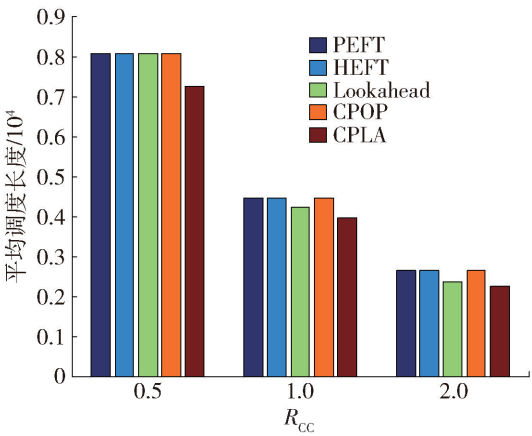


图 8 Montage 任务图下,处理器个数为 5,不同 R_{cc} 下算法的平均调度长度
Fig. 8 Average makespan under different R_{cc} with five processors in Montage graph

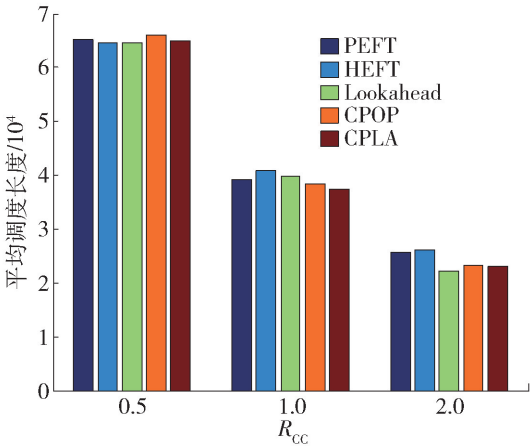


图 9 Epigenomics 任务图下,处理器个数为 5,不同 R_{cc} 下算法的平均调度长度
Fig. 9 Average makespan under different R_{cc} with five processors in Epigenomics graph

CPOP 算法的小,而且随着 R_{cc} 增大,CPLA 的平均调度长度小于 HEFT 算法和 PEFT 算法. 图 11 表示 5 种算法在处理器个数为 20 的环境下得到的平均调度长度. 可以看出,CPLA 的平均调度长度比 CPOP 算法的小,且随着 R_{cc} 增大,CPLA 的平均调度长度小于 HEFT 算法和 PEFT 算法.

对于任务数为 100 的 Epigenomics 任务图,设置 $R_{cc} = 1$,图 12 表示 5 种算法在处理器个数分别为 5、10、15、20 的情况下得到的效率. 可以看出,在 R_{cc} 比较大的情况下,CPLA 的效率和 Lookahead 算法的效率接近,而且 CPLA 的效率高于 HEFT 算法和 PEFT 算法.

由实验可以看出,CPLA 在任务决策中的预见

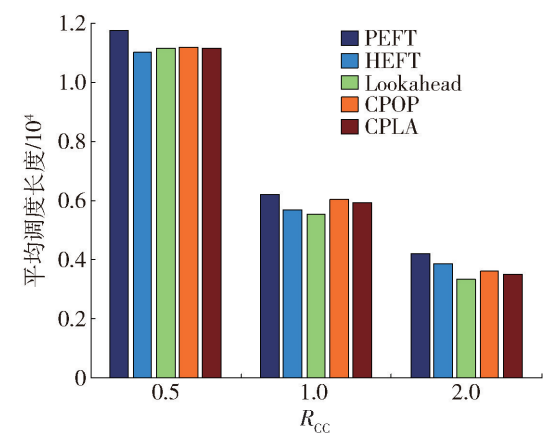


图 10 处理器个数为 10,不同 R_{cc} 下算法的平均调度长度

Fig. 10 Average makespan under different R_{cc} with ten processors

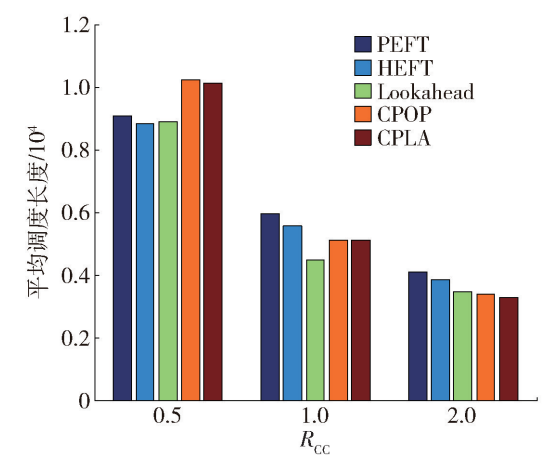


图 11 处理器个数为 20,不同 R_{cc} 下算法的平均调度长度

Fig. 11 Average makespan under different R_{cc} with twenty processors

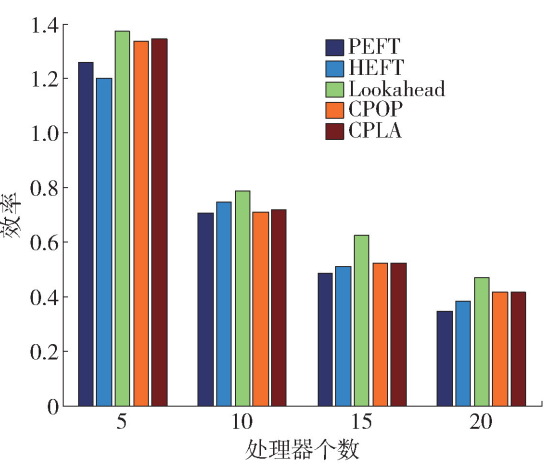


图 12 不同处理器数下 5 种算法的效率

Fig. 12 Efficiency of five algorithms under different processor numbers

性提高了算法的效率,算法采取关键路径分类任务降低了算法的时间复杂度,提高了算法的平均加速比。CPLA 的效率、调度长度、平均加速比优于其他算法。

4 结论

- 1) 本文提出了基于关键路径前瞻的工作流调度算法,在关键路径任务进行处理器分配的过程中考虑分配结果对后继任务的影响;对于非关键路径的任务,考虑直接前驱任务的影响,找到前驱任务到当前任务的最短路径,采用最早完成时间策略来选择处理器。
- 2) 与现有的算法相比, CPLA 的时间复杂度比 Lookahead 算法的时间复杂度低;与其他 DAG 调度算法相比, CPLA 的加速比、效率比较高; CPLA 在大多数情况下与其他算法相比调度长度比较小。
- 3) 如果将 CPLA 应用到多工作流调度中,是否会影响算法的调度长度,能否在调度过程中更合理地利用已调度任务间的间隙还有待进一步的研究。

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] GARY M R, JOHNSON D S. Computers and intractability: a guide to the theory of NP-completeness[D]. New York: WH Freeman and Company, 1979.
- [3] 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.
- [4] ILAVARASAN E, THAMBIDURAI P. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments [J]. Journal of Computer Sciences, 2007, 3(2): 94-103.
- [5] ILAVARASAN E, THAMBIDURAI P, MAHILMANNAN R. High performance task scheduling algorithm for heterogeneous computing system [C] // International Conference on Algorithms and Architectures for Parallel Processing. Berlin: Springer, 2005: 193-203.
- [6] KWOK Y, AHMAD I. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors [J]. IEEE Transactions on Parallel and Distributed Systems, 1996, 7(5): 506-521.
- [7] BOERES C, FILHO J V, REBELLO V E F. A cluster

- based strategy for scheduling task on heterogeneous processors[C]//Proceedings 16th Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2004. Piscataway: IEEE, 2004: 214-221.
- [8] CIROU B, JEANNOT E. Triplet: a clustering scheduling algorithm for heterogeneous systems [C] // Proceedings International Conference on Parallel Processing Workshops. Piscataway: IEEE, 2001: 231-236.
- [9] PARK G, SHIRAZI B, MARQUIS J. A new approach for duplication based scheduling for distributed-memory systems [C] // Proceedings 11th International Parallel Processing Symposium. Piscataway: IEEE, 1997: 157-166.
- [10] WANG L, SIEGEL H J, ROYCHOWDHURY V P. A genetic algorithm-based approach for task matching and scheduling in heterogeneous computing environments[C] // Proceedings of Heterogeneous Computing Workshop. Piscataway: IEEE, 1996: 72-85.
- [11] 田国忠, 肖创柏, 谢军奇. 有期限约束的多 DAG 共享资源的调度及公平费用优化方法[J]. 计算机学报, 2014, 37(7): 1607-1619.
- TIAN G Z, XIAO C B, XIE J Q. Scheduling and fair cost-optimizing methods for concurrent multiple DAGs with deadline sharing resources[J]. Chinese Journal of Computers, 2014, 37(7): 1607-1619. (in Chinese)
- [12] 田国忠. 多 DAG 共享资源调度的若干问题研究[D]. 北京: 北京工业大学, 2014.
- TIAN G Z. Research several problems of scheduling multiple DAGs sharing resources [D]. Beijing: Beijing University of Technology, 2014. (in Chinese)
- [13] ARABNEJAD H, BARBOSA J G. Performance evaluation of list based scheduling on heterogeneous systems[C]//Proceedings of European Conference on Parallel Processing. Berlin: Springer, 2011: 440-449.
- [14] BITTENCOUNT L F, SAKELLARIOU R, MADERIRA E R M. DAG scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm [C] // 2010 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). Piscataway: IEEE, 2010: 27-34.
- [15] ARABNEJAD H, BARBOSA J G. List scheduling algorithm for heterogeneous systems by an optimistic cost table[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(3): 682-694.
- [16] GUO F, YU L, TIAN S, et al. A workflow task scheduling algorithm based on the resources' fuzzy clustering in cloud computing environment [J]. International Journal of Communication Systems, 2015, 28(6): 1053-1067.
- [17] MOHANAPRIYA N, KOUSALYA G, BALAKRISHNAN P. Cloud workflow scheduling algorithms: a survey[J]. International Journal of Advanced Engineer Technology, 2016, 7(3): 88-195.
- [18] SUTER F. DAG generation program[R/OL]. [2010-10-19]. <http://www.loria.fr/suter/dags.html>.
- [19] Spitzer Science Center. MONTAGE: an astronomical image mosaic engine [R/OL]. [2013-03-31]. <http://montage.ipac.caltech.edu/>.
- [20] USC Molecular Genomics Core. USC epigenome center [R/OL]. [2013-05-16]. <http://epigenome.usc.edu/>.

(责任编辑 梁 洁)