计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2019,28(3):118−125 [doi: 10.15888/j.cnki.csa.006807]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于 HEFT 和 CPOP 的相关任务表调度算法①

刘林东[1,2], 邬依林[1]

[1](广东第二师范学院 计算机科学系, 广州 510303)
[2](华南理工大学 计算机系统研究所, 广州 510006)
通讯作者: 邬依林, E-mail: hongox@163.com

摘　要: DAG 任务调度是当前研究的热点, DAG 任务模型中任务的调度顺序一方面会影响用户服务满意质量, 另一方面也会影响云服务资源的利用率, 高效的任务调度算法能够使多核处理器的资源分配和并行计算能力更强. 表调度算法 HEFT 算法以及 CPOP 算法在相关任务调度中存在效率较低等问题. 本文基于 HEFT 算法和 CPOP 算法, 提出了一种相关任务调度模型和相关任务调度算法 IHEFT 算法, 对任务排序和任务调度两个方面进行改进. 任务排序阶段, 以任务的方差以及平均通信代价作为排序的依据; 任务调度阶段, 对满足任务复制条件的结点进行任务复制. 实验证明, IHEFT 算法在任务调度跨度、任务调度平均等待时间以及平均 *Slack* 值方面均优于 HEFT 算法和 CPOP 算法.

关键词: 处理器内核; DAG; 任务调度; 跨度; 平均等待时间

引用格式: 刘林东,邬依林.基于 HEFT 和 CPOP 的相关任务表调度算法.计算机系统应用,2019,28(3):118–125. http://www.c-s-a.org.cn/1003-3254/6807.html

List Scheduling Algorithm of Dependent Tasks Based on HEFT and CPOP

LIU Lin-Dong[1,2], WU Yi-Lin[1]

[1](Department of Computer Science, Guangdong University of Education, Guangzhou 510303, China)
[2](Research Institute of Computer Systems, South China University of Technology, Guangzhou 510006, China)

Abstract: DAG task scheduling is the current hot topic. In task model of DAG, the order of task scheduling affect the service satisfaction of users on one hand, and also affect utilization rate of cloud service resources on the other hand. High efficient task scheduling algorithm may strengthen the resources distribution of the multi-core and the parallel computing ability. HEFT algorithm and CPOP algorithm are of lower efficiency in related task scheduling. Based on HEFT algorithm and CPOP algorithm, a dependent task scheduling model and task scheduling algorithm IHEFT (Improvement Heterogeneous Earliest Finish Time) algorithm are proposed in this study. The IHEFT algorithm mainly optimizes two aspects: task ordering and task scheduling. The variance of task scheduling cost on every processor core and the average communication overhead are the basis of task ordering. In the stage of task scheduling, task duplication of some nodes in DAG with some conditions can make full use of heterogeneous processor resources and shorten the completion time of task set. Experiment results show that the IHEFT algorithm performs more performance than the HEFT algorithm and the CPOP algorithm in terms of the task scheduling *Makespan*, the average waiting time and the average value of *Slack*.

Key words: processor core; Directed Acyclic Graph (DAG); task scheduling; makespan; average waiting time

① 基金项目: 国家自然科学基金 (61772205); 广东省科技计划项目 (2014B010110004, 2016A010106007, 2016B090927010); 广东第二师范学院网络工程重点学科 (ZD2017004); 广东第二师范学院计算机实验教学示范中心 (SY2016014)
Foundation item: National Natural Science Foundation of China (61772205); Science and Technology Program of Guangdong Province (2014B010110004, 2016A010106007, 2016B090927010); Key Discipline Fund for Network Engineering of Guangdong University of Education (ZD2017004); Computer Experimental Teaching Demonstration Center Fund of Guangdong University of Education (SY2016014)
收稿时间: 2018-09-12; 修改时间: 2018-10-08; 采用时间: 2018-10-12; csa 在线出版时间: 2019-02-22

异构多核环境下的任务调度问题属于 NP-难问题[1–4], 即任务调度问题只能得到近似最优解, 因此异构多核环境下的任务调度问题对处理器系统的性能和效率提高具有非常重要的作用. 多核处理器任务调度的目标[5]就是按照某种策略将每个任务合理分配到各个处理器内核上并行有序地执行, 从而减少并行运行任务间的通信开销和等待时间, 使任务的执行时间最短.

异构多核处理器平台下 DAG 任务调度算法主要包括表调度算法[6,7]、聚类调度算法以及基于复制的调度算法等, 具体包括 DSC、DCP、CPFD、TDS、HEFT、CPOP 和 PEFT 等算法. 为了尽可以实现任务调度的目标, 需要从任务调度的排序以及任务调度两个环节对算法进行优化和改进. 文中利用任务的方差以及平均通信开销对任务进行排序, 将关键路径[8]、关键结点以及任务复制技术[9,10]应用于任务调度过程中, 从而使任务调度的效率更接近任务调度的目标. 在任务排序阶段, 以任务计算代价的二次方差以及平均通信开销作为任务排序的依据, 相比传统 HEFT 算法直接用计算开销作为排序的依据更科学、效率更高; 在任务调度阶段, 对满足任务复制条件的节点进行任务复制, 可以避免部分任务之间的通信开销, 提高资源的利用率, 降低任务集的计算时间.

1 相关任务调度及 DAG 任务模型

1.1 相关任务调度

根据被调度任务之间的相关性, 可以将任务调度分为独立任务调度和相关任务[11,12](依赖任务) 调度两种类型. 异构多核处理器平台下的任务调度可以分为静态任务调度和动态任务调度两种[13]. 静态任务调度[14]是指满足硬件资源的负载均衡和通信开支的前提下将任务分配给固定的处理器内核; 而动态任务调度[15]则指根据资源利用率的情况可以将任务分配到任何处理器内核上. 相关任务在异构多核处理器平台下的调度问题, 主要任务是将子任务映射到多个处理器内核上, 达到任务完成时间最小化等调度目标, 因此在任务调度的过程中, 不仅要考虑异构处理器内核计算能力的差异, 同时还要考虑处理器内核之间的数据通信等问题, 因此, 相关任务的调度问题均属于 NP 完全问题[16].

DAG 任务调度算法主要有 DSC[17]、DCP[18]、CPFD[19]、TDS[20]、HEFT[21,22]、CPOP[21,22]和 PEFT[23]等算法. 异构多核处理器平台下 DAG 任务调度算法主要包括三类, 分别为表调度算法、聚类调度算法以及基于复制的调度算法.

表调度算法[24]是最经典的 DAG 调度算法. 表调度算法包括静态调度算法和动态调度算法两种. 静态调度算法的基本思想: 根据 DAG 中的所有任务之间的关系, 结合某种策略决定任务之间的优先级并构造一个调度列表; 然后为具有最高优先级的任务分配能够使它最快完成的资源并开始调度执行, 循环上述步骤直到 DAG 中所有任务均被调度. 动态表调度算法在执行完每一个任务后根据当前处理器内核的情况以及任务之间的依赖关系重新计算任务的优先级, 静态调度算法的不重新计算优先级, 而动态调度算法需要重新计算优先级. 常见的表调度算法包括 Topcuoglu 等人提出的 HEFT (Heterogeneous Earliest Finish Time) 算法[25,26]以及 CPOP 算法.

1.2 DAG 任务模型

通常使用 DAG (Directed Acyclic Graph) 任务图来描述相关任务之间的关系. 相关任务的特点是任务的所有前驱节点执行完成后, 后续任务才能开始执行. DAG 表示为 $G=(V, E, ET, C)$, 其中 V 表示 n 个任务结点 t_1, t_2,⋯, t_n 的集合, 每一个结点 v_i 表示 DAG 中的一个任务, $|V|=n$; E 表示 k 条有向边 e_1, e_2,⋯, e_k 的集合, 每条有向边 $c_{i,j}\in E$, 表示任务结点 t_i, t_j 之间存在先后执行的关系, 即任务 t_j 必须在任务 t_i 执行完成之后才能开始执行, 每一条边的权值表示任务和任务之间的通信时间; ET 表示任务 t_i 在处理器内核 c_j 上的估算计算时间, 在 DAG 中, 通常用平均计算时间作为任务节点的权值, 任务 t_i 的平均执行时间为 $\left[\sum_{j=1}^{m} ET_{i,j}\right]/m$; C 是任务之间通信开销的集合, 在 DAG 中, 将任务的通信开销作为边的权值.

定义 1. $pred(t_i)$ 表示在给定的 DAG 中, 任务 t_i 直接前驱任务组成的集合. 如果 $pred(t_i)=\varnothing$, 那么称该任务为入口任务, 用 t_{entry} 表示. $succ(t_i)$ 表示在给定的 DAG 中, 任务 t_i 直接后继任务组成的集合. 如果 $succ(t_i)=\varnothing$, 那么称该任务为出口任务, 表示为 t_{exit}. 如果在一个 DAG 中, 有多个入口任务或出口任务, 那么增加一个通信时间和计算时间为 0 的虚拟入口或出口任务.

定义 2. 矩阵 ET, 是一个 $n*m$ 的计算时间矩阵, 其中 n 表示任务数, m 表示处理机内核数量, $ET(t_i, c_j)$ 表示在处理机内核 c_j 上执行任务 t_i 的估算时间.

定义 3. 关键路径 (Critical Path, *CP*), 在 DAG 中, 一条从入口节点 t_{entry} 到出口节点 t_{exit} 的最长路径称为关键路径.

定义 4. 跨度 (*Makespan*) 或调度长度表示某一个确定的调度算法对任务集 T 调度后, 任务集 T 中最后一个任务的完成时间, 如式 (1) 所示. 其中$AFT(t_i)$表示任务 t_i 实际完成时间, *Makespan* 是所有出口任务中实际完成时间的最大值.

$$Makespan = \max\{AFT(t_i)\} \tag{1}$$

定义 5. 最早开始时间$EST(t_i,c_j)$是指任务 t_i 在处理器内核 c_j 上的最早开始时间, $EST(t_i,c_j)$取$AFT(t_i)$和$EFT(c_j)$两个值中的较大值, 如式 (2) 所示.

$$EST(t_i,c_j) = \begin{cases} AT(t_i) \\ EFT(c_j) \end{cases} \tag{2}$$

定义 6. 最早完成时间$EFT(t_i,c_j)$是指任务 t_i 在处理器内核 c_j 上的最早完成时间, 其值等于任务 t_i 在处理器内核 c_j 上的最早开始时间, 加上 t_i 在处理机内核 c_j 上的计算时间, 如式 (3) 所示.

$$EFT(t_i,c_j) = EST(t_i,c_j) + ET(t_i,c_j) \tag{3}$$

定义 7. 平均等待时间 *AWT*, 任务集 T 中的所有任务在调度过程中, 设任务 t_i 的开始执行时间为 ST_i, 任务到达时间 AT_i, 则$AWT = \left[\sum_{j=1}^{n}(ST_j - AT_j)\right]/n$.

定义 8. C 表示通信开销的集合. $c_{i,j}$ 表示任务 t_i 和 t_j 之间的通信开销, 通常用平均通信代价$c_{i,j}$来表示任务 t_i 和 t_j 之间的通信开销, 任务 t_i 和 t_j 之间的平均通信开销如式 (4) 所示. 其中$\overline{L}$表示所有处理机内核的平均延迟, $\overline{B}$表示处理机内核相连的平均带宽, $data_{i,j}$ 表示 t_i 发送给任务 t_j 的数据总量. 当 t_i 和 t_j 被分配到同一个处理机内核时 $c_{i,j}$=0, 这是因为相比处理机内核之间的通信, 处理机内核内的通信可以被忽略.

$$\overline{L} + \frac{data_{i,j}}{\overline{B}} \tag{4}$$

定义 9. *Slack*. *Slack* 是一个关于任务调度算法鲁棒性的量度, 其反映的是一个算法调度产生的任务处理时间的不确定程度. *Slack* 的定义如式 (5) 所示. 其中 M 表示 DAG 的 *Makespan*, n 表示任务数量, b_{level}表示任务t_i到出口节点最长路径的长度, t_{level}表示从入口节点到任务t_i (不包含任务t_i) 最长路径的长度.

$$Slack = \left[\sum_{t_i \in T} M - b_{level}(t_i) - t_{level}(t_i)\right]/n \tag{5}$$

2 任务调度模型及算法

2.1 DAG 任务模型

针对传统的表调度算法中存在不能准确反应 DAG 任务图中的内在联系、任务与处理器内核之间调度效率不高以及负载不均衡等问题, 本文提出了一种的相关任务调度模型及相关任务调度算法 IHEFT 算法. 相关任务调度模型如图 1 所示, 任务集 T 中的任务被调度到处理器内核集 C 中包括分两个步骤: 任务排序和任务调度. 任务排序阶段, 主要根据任务的执行时间方差以及平均通信开销作为排序的依据; 任务调度阶段, 判断当前任务结点的前驱结点是否为关键结点, 如果为关键结点且满足任务复制条件, 则进行任务复制, 否则按 HEFT 调度算法对任务进行调度.

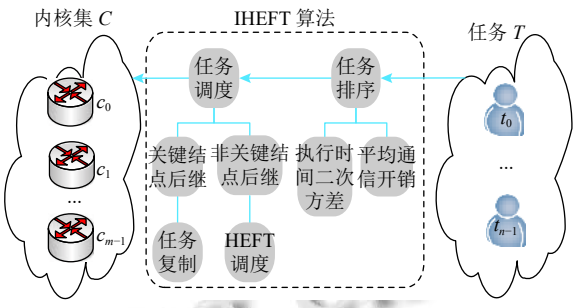


图 1 相关任务表调度模型

2.2 调度算法

相关任务调度 IHEFT 算法如算法 1 所示. 算法的主要思想为: 在任务排序阶段, 用任务在各个处理器内核上的执行时间方差以及通信成本作为参数对 DAG 任务进行排序, 任务执行时间的方差以及通信成本越大, 任务在不同的处理器内核上的执行时间差异就越大, 给任务赋予较高的优先级, 相反, 赋予较小的优先级, 任务 t_i 的方差及通信成本δ_i计算如式 (7) 所示. 在任务调度阶段, 首先获取 DAG 任务图的关键路径, 关键路径上的节点任务优先级最高, 非关键路径上的任务按δ_i降序排列; 然后利用任务复制技术优化任务调度过程, 若当前任务 t_i 的前驱节点 t_j 为关键节点任务, 且复制关键节点任务 t_j 到被调度的处理器内核 c_m 可以缩短当前任务的最早完成时间, $EFT(t_i,c_m)'$定义如式 (6) 所示, 则复制关键节点任务 t_j 到当前处理器内核 c_m; 若

不满足任务复制条件, 即$EFT(t_i,c_m)' \geqslant EFT(t_i,c_m)$, 则按 HEFT 算法将任务调度到使任务完成时间最早的处理器内核. 通过任务复制可以减少任务间通信开销、提高处理器内核利用率等目的.

$$EFT(t_i,c_m)' < EFT(t_i,c_m) \tag{6}$$

$$\delta_i = \sum_{k=1,n-1\geqslant i\geqslant 0}^{k\leqslant m} (ET_{t_i,c_k} - \overline{ET})^2/m + c_{i,j} \tag{7}$$

3 任务调度过程分析

以 1 个 DAG 任务图为例, 分析 IHEFT 相关任务调度算法在异构多核处理器内核上的调度过程.

算法 1. IHEFT 算法

1. 从 DAG 任务图中读取任务数到 n;
2. 从 DAG 任务图中读取计算节点的数量到 m;
3. 读取 DAG 任务图中任务之间的通信开销, 并保存在 $c[n][n]$中;
4. 计算所有任务的二次方差δ_i值;
5. 按降序顺序对所有任务的δ_i值进行排序;
6. 从出口任务开始, 对 DAG 任务进行遍历, 计算所有任务的 $rank_u$ 值;
7. 入口任务开始, 对 DAG 任务图进行遍历, 计算所有任务的 $rank_d$ 值;
8. 计算 DAG 中每个任务的优先级, 计算方法为 $priority[t_i]=rank_u[i]+rank_d[i]$;
9. 通过 $priority[t_i]$找出 DAG 任务图的关键路径 CP;
10. while 任务集非空{
11. 从 DAG 任务图中找出优先级最高的任务 t_i;
13. if 当前任务节点 t_i 的前驱节点 t_j 是关键路径节点, 即满足式 (5-11) 条件;
12. 找到相应的计算节点 f_m 调度任务;
14. 将任务 t_j 复制到计算节点 f_m 上, 并在此计算节点上执行;
15. 保存 $ST[i]$, $FT[i]$, $Avail[m]$的值;
16. else
17. 调度任务 t_i 到计算节点 f_m 上;
18. 保存 $ST[i]$, $FT[i]$, $Avail[m]$
19. $s[i][m]=1$
20. 计算所有任务的跨度 $Makespan$ 值;
21. }
22. 计算任务集中所有的任务的平均等待时间 AWT 以及 $Slack$ 值

3.1 DAG 初始化

图 2 表示一个 DAG 任务模型, 共有 10 个任务, 带箭头的边表示任务之间的通信边, 边上的数字表示两个任务之间的通信代价 $c_{i,j}$; 节点 t_1 为入口节点, 节点 t_{10} 为出口节点. 表 1 表示任务在不同处理器内核上的计算时间 $ET(t_i, c_j)$, 表 1 中包括 3 个处理器内核 c_0~c_2, 10 个任务 t_1~t_{10}.

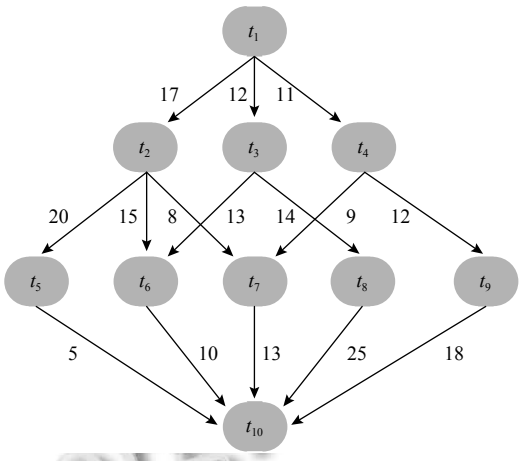


图 2 DAG 任务模型

表 1 任务集在内核集上的调度时间

任务	c_0	c_1	c_2
t_1	14	16	9
t_2	13	19	18
t_3	11	13	19
t_4	13	8	7
t_5	12	13	10
t_6	13	16	9
t_7	7	15	11
t_8	5	11	14
t_9	18	12	20
t_{10}	21	7	16

表 2 依赖关系矩阵

任务	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
t_1	0	17	12	11	0	0	0	0	0	0
t_2	0	0	0	0	20	15	8	0	0	0
t_3	0	0	0	0	0	13	0	14	0	0
t_4	0	0	0	0	0	0	9	0	12	0
t_5	0	0	0	0	0	0	0	0	0	5
t_6	0	0	0	0	0	0	0	0	0	10
t_7	0	0	0	0	0	0	0	0	0	13
t_8	0	0	0	0	0	0	0	0	0	25
t_9	0	0	0	0	0	0	0	0	0	18
t_{10}	0	0	0	0	0	0	0	0	0	0

3.2 依赖关系矩阵

根据 DAG 任务模型, 生成任务之间对应的依赖关系矩阵, 如表 2 所示. 依赖关系矩阵的定义为:

(1) 若结点 i 为结点 j 的前驱节点, 则矩阵元素 $d_{i,j}=c_{i,j}$;

(2) 若结点 i 和结点 j 之间不存在依赖关系, 则矩阵元素 $d_{i,j}=0$.

3.3 任务排序

计算每个任务 t_i 的 $rank_u$、$rank_d$、$rank_u+rank_d$ 值以及计算代价的二次方差、通信开销值δ_i, 并降序进行排序, 表 3 所示为 4 核环境下各参数的计算及排序情况.

表 3 IHEFT 算法的 $rank_u$、$rank_d$ 值

任务	$rank_u$	$rank_d$	$rank_u+rank_d$	δ_i
t_1	173.5	15.0	188.5	118.8
t_2	118.7	50.2	168.9	73.7
t_3	117.2	42.0	159.2	98.1
t_4	146.0	36.0	182.0	82.0
t_5	118.2	39.2	157.4	40.1
t_6	109.4	43.2	152.7	51.8
t_7	84.2	75.5	159.8	57.2
t_8	80.8	78.8	159.5	72.6
t_9	96.5	85.2	181.8	63.1
t_{10}	58.5	116.2	174.8	33.6

3.4 任务调度

(1) 计算关键路径

计算每个任务的 $rank_u$ 值以及 $rank_d$ 值, 在此基础上找出 DAG 任务模型中的关键路径, 任务集 t_1~t_{10} 的关键路径为: t_1, t_3, t_8, t_{10}.

(2) 任务复制

根据生成的任务调度序列 t_1, t_3, t_4, t_2, t_8, t_9, t_7, t_6, t_5, t_{10}, 对每个被调度的任务检查是否满足任务复制条件 $EFT(t_i,c_m)' < EFT(t_i,c_m)$, 即任务被复制到另一个处理器内核上, 要满足复制前驱任务后当前任务的完成时间要小于不复制前驱任务的完成时间. 在任务 t_4 和 t_2 被调度时, 满足任务复制条件, 分别将 t_4 和 t_2 的前驱任务 t_1 复制到处理器内核 c_1 和 c_0 上, 如图 3(a) 所示.

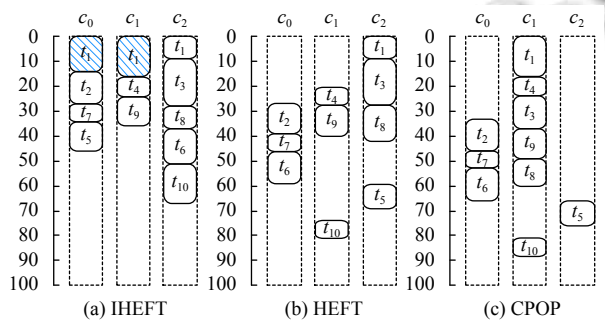


图 3 三种算法任务调度对比

(3) 任务调度

任务集经过调度之后, 任务与处理器内核的对应调度关系如图 3(a) 所示, 图 3(b)、图 3(c) 分别表示同一任务集通过 HEFT 算法、CPOP 算法调度的对应关系, 很明显, HEFT、CPOP 算法任务调度的总长度 *Makespan* 值比 IHEFT 算法要大.

4 仿真实验与结果分析

4.1 实验目的

为了验证本章提出的 IHEFT 算法, 在相同的实验条件下对本章所提出的算法 IHEFT 与现有的调度算法 HEFT 和 CPOP 算法进行性能比较, 主要比较任务跨度 *Makespan* 和任务平均等待时间 *AWT*.

4.2 模拟环境

基于 SimGrid 提供的模拟器工具包, 构建一个异构多核处理器的仿真环境. 其中:

(1) 处理器内核之间通过高速网络互连;

(2) 每个处理器内核可同时进行任务执行和与其它处理器内核通信, 而不需要争用;

(3) 任务在处理器内核上执行是非抢占的;

(4) 处理器内核之间是异构的, 即同一个任务在不同处理器内核上执行存在差异性.

实验中使用的计算机配置为: Intel Core i5-3210M@2.5 GHz 双核笔机本处理机, 8 GB 内存. 本次实验采用的处理器内核分别取 3、4 核.

4.3 测试数据集

IHEFT 算法输入的数据包括 DAG 任务图 (包括任务集、处理器内核集、任务间通信成本)、任务调度计算时间矩阵; 算法的输出为任务调度执行时间 *Makespan* 以及任务平均等待时间 *AWT*. IHEFT 算法任务调度测试分两种情形. 第一种情况: 任务集中包括 10 个任务, 处理器内核集中包括 3 个处理器内核, DAG 任务图的数量分别为 1~10; 第一种情况: 任务集中包括 10 个任务, 处理器内核集中包括 4 个处理器内核, DAG 任务图的数量分别为 1~10, 所有任务的到达时间为 0.

4.4 实验结果分析

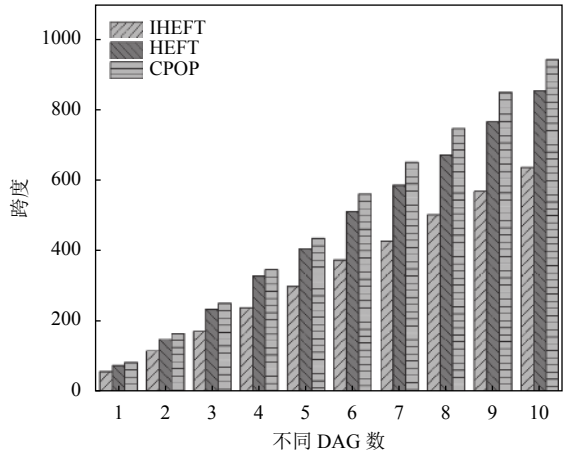
利用三种算法分别对 10 个 DAG 任务图进行调度, 表 4 为三种算法在 3 核、4 核情况下 *Slack* 值的对比情况.

(1) 3 个处理器内核

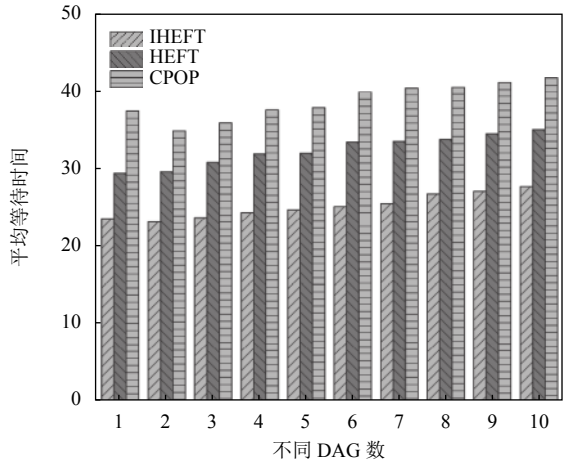
分别采用 3 种算法对 DAG 任务集进行调度, 在处理器内核的数量为 3 的情况下, 针对不同数量 DAG 任务模型进行调度, 得到所有任务的跨度 (调度长度) *Makespan*、平均等待时间 *AWT* 的值以及平均 *Slack* 值, 如图 4(a)、图 4(b)、图 5 所示.

表 4 IHEFT、HEFT、CPOP 算法的 *Slack* 值

DAG	3 核			4 核		
	IHEFT	HEFT	CPOP	IHEFT	HEFT	CPOP
1	53	77	83	58	76	83
2	66	71	84	55	71	81
3	73	83	85	63	82	82
4	87	96	93	63	90	91
5	64	78	85	57	75	85
6	82	102	109	68	98	121
7	55	75	87	50	73	87
8	71	81	91	71	81	91
9	70	89	97	63	89	97
10	83	85	92	64	84	91



(a) 任务跨度对比



(b) 任务平均等待时间对比

图 4 不同 DAG 的任务跨度对比和任务平均等待时间对比

图 4(a) 的纵坐标为算法所获得的调度跨度 *Makespan* 值, 横坐标为 DAG 数量; 图 4(b) 的纵坐标为算法所获得的调度跨度平均等待时间 *AWT* 值, 横坐标为 DAG 数量. 从实验数据得出, 在任务调度的跨度方面, IHEFT 算法比 HEFT 算法最低降低了 10%, 最高降低了 15.7%; IHEFT 算法比 CPOP 最低降低了 14.3%, 最高降低了 21.2%; 在任务平均等待时间方面, IHEFT 算法比 HEFT 算法最低降低了 3.3%, 最高降低了 17.5%, IHEFT 算法比 CPOP 算法最低降低了 14.6%, 最高降低了 28.0%. 图 5 中纵坐标为算法所获得的平均 *Slack* 值, 横坐标为 DAG 数量, 从图 5 可以看出, 随着 DAG 数量的的增加, IHEFT、HEFT 和 CPOP 算法的 *Slack* 值变化不明显, 但是横向比较可以看出 IHEFT 相对 HEFT 和 CPOP 算法有更低的 *Slack* 值, IHEFT 算法的 *Slack* 值最高比 HEFT 算法要减少 31.2%, 比 CPOP 算法要减少 36.1%.

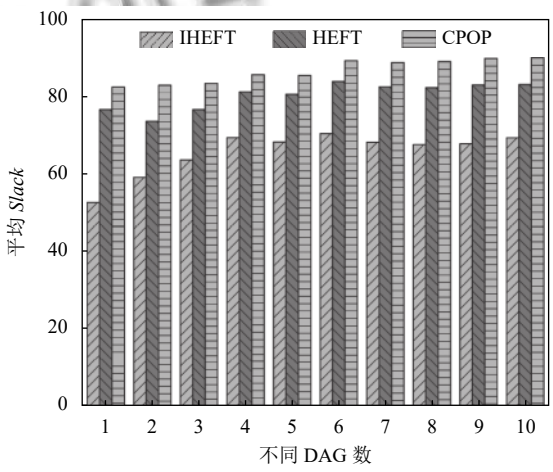


图 5 不同 DAG 的任务 *Slack* 对比

(2) 4 个处理器内核

分别采用 3 种算法对 DAG 任务集进行调度, 在处理器内核的数量为 4 的情况下, 针对不同数量 DAG 任务模型进行调度, 得到所有任务的跨度 *makespan*、平均等待时间 *AWT* 的值以及平均 *Slack* 值, 如图 6(a)、图 6(b)、图 7 所示.

图 6(a) 的纵坐标为算法所获得的调度跨度 *makespan* 值, 横坐标为 DAG 数量; 图 6(b) 的纵坐标为算法所获得的调度跨度平均等待时间 *AWT* 值, 横坐标为 DAG 数量. 从实验数据得出, 在任务调度的跨度方面, IHEFT 算法比 HEFT 算法最低降低了 22.2%, 最高降低了 27.1%; IHEFT 算法比 CPOP 最低降低了 29.1%, 最高降低了 34.4%; 在任务平均等待时间方面, IHEFT 算法比 HEFT 算法最低降低了 19.9%, 最高降低了 24.8%, IHEFT 算法比 CPOP 算法最低降低了 33.6%, 最高降低了 37.0%. 图 7 中纵坐标为算法所获得的平均 *Slack* 值, 横坐标为 DAG 数量, 从图 7 可以看出, 随

着 DAG 数量的的增加, IHEFT、HEFT 和 CPOP 算法的 *Slack* 值变化不明显, 但是横向比较可以看出 IHEFT 相对 HEFT 和 CPOP 算法有更低的 *Slack* 值, IHEFT 算法的 *Slack* 值最高比 HEFT算法要减少 26.6%, 比 CPOP 算法要减少 34.3%.

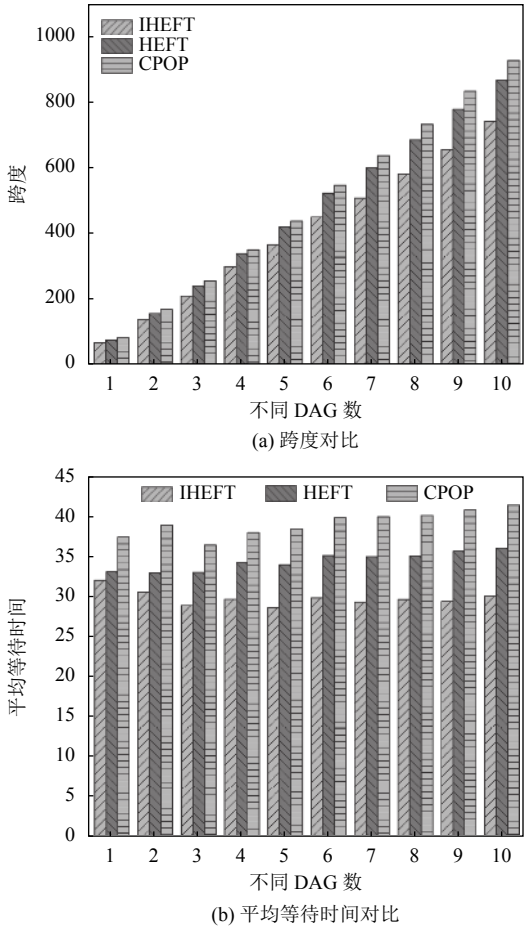


图 6 不同 DAG 的任务跨度对比和任务平均等待时间对比

通过两组实验分析得出, IHEFT 算法在任务调度跨度以及任务调度平均等待时间两方面均优于 HEFT 算法和 CPOP 算法. 在任务调度跨度方面最小分别降低了 10% 和 14.3%, 在任务调度平均等待时间方面最小分别降低了 3.3% 和 14.6%. IHEFT 的任务排序方面采用计算代价的二次方差作为排序依据更能体现任务的优先级, 而在任务调度的过程中对满足条件的任务进行任务复制可以有效地减少任务调度的跨度、任务调度平均等待时间以及平均 *Slack* 值.

5 结论与展望

文中介绍了相关任务调度及 DAG 任务模型的基本概念, 对相关任务调度算法 HEFT 算法和 CPOP 算法进行了介绍. 由于 HEFT 算法和 CPOP 算法在相关任务调度中存在一些缺陷, 基于 HEFT 算法和 CPOP 算法, 对相关任务调度中任务排序和任务调度两个方面进行改进, 提出了一种相关任务调度模型和相关任务调度算法 IHEFT 算法. 通过实验证明, IHEFT 算法在任务调度跨度、任务调度平均等待时间以及平均 *Slack* 方面均优于 HEFT 算法和 CPOP 算法.

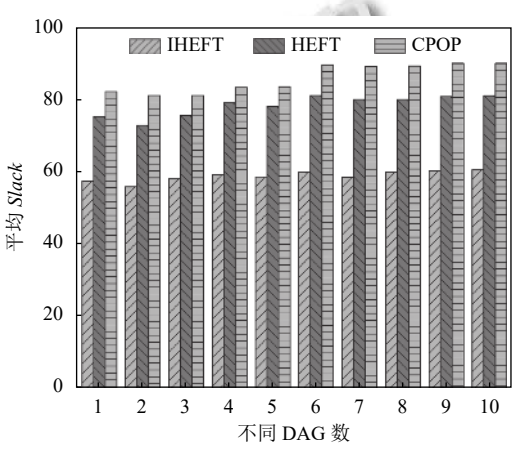


图 7 不同 DAG 的任务 *Slack* 对比

参考文献

1 蒋少丙. 基于遗传算法的多核处理器任务执行策略的优化研究[硕士学位论文]. 北京: 华北电力大学, 2016.
2 石祥龙. 基于异构多核处理器的静态任务调度算法研究[硕士学位论文]. 南京: 南京邮电大学, 2015.
3 房婷. 异构分布式环境下任务调度问题的研究[硕士学位论文]. 大连: 大连理工大学, 2014.
4 Li Y, Niu JW, Zhang J, *et al*. An optimized RM algorithm by task affinity on multi-core processor. Proceedings of the 2016 IEEE 22nd International Conference on Parallel and Distributed Systems. Wuhan, China. 2016. 286–293.
5 李召妮, 雷丽晖, 李永明. 多处理器任务调度算法 TDS 的建模与验证. 计算机科学, 2012, 39(11): 301–304. [doi: 10.3969/j.issn.1002-137X.2012.11.073]
6 张黎明, 张向利. 一种改进的实时任务调度算法. 桂林电子科技大学学报, 2014, 34(6): 460–463. [doi: 10.3969/j.issn.1673-808X.2014.06.007]
7 Atef A, Hagras T, Mahdy YB, *et al*. Lower-bound complexity and high performance mechanism for scheduling dependent-tasks on heterogeneous grids. Proceedings of 2018 International Conference on Innovative Trends in Computer Engineering. Aswan, Egypt. 2018. 1–7.

8 刘少伟, 任开军, 邓科峰, 等. 云平台上基于关键路径截取的有向无环图应用调度算法. 国防科技大学学报, 2017, 39(3): 97–104.
9 李静梅, 尤晓非, 韩启龙. 基于任务复制的多关键路径任务调度算法. 计算机工程与设计, 2014, 35(5): 1639–1645. [doi: 10.3969/j.issn.1000-7024.2014.05.028]
10 潘志舟. 基于任务复制和插入的分布式任务调度算法研究[硕士学位论文]. 武汉: 华中科技大学, 2015.
11 Graham RL, Lawler EL, Lenstra JK, *et al*. Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics, 1979, 5: 287–326. [doi: 10.1016/S0167-5060(08)70356-X]
12 田国忠, 肖创柏. 分布式系统下的 DAG 任务调度研究综述. 计算机工程与科学, 2015, 37(5): 882–894. [doi: 10.3969/j.issn.1007-130X.2015.05.005]
13 Pathan S, Voudouris P, Stenstr?m P. Scheduling parallel real-time recurrent tasks on multicore platforms. IEEE Transactions on Parallel and Distributed Systems, 2018, 29(4): 915–928. [doi: 10.1109/TPDS.2017.2777449]
14 Fechner B, H?nig U, Keller J, *et al*. Fault-tolerant static scheduling for grids. Proceedings of 2008 IEEE International Symposium on Parallel and Distributed Processing. Miami, FL, USA. 2008. 1–6.
15 Agullo E, Beaumont O, Eyraud-Dubois L, *et al*. Are static schedules so bad? A case study on cholesky factorization. Proceedings of 2016 IEEE International Parallel and Distributed Processing Symposium. Chicago, IL, USA. 2016. 1021–1030.
16 Ruan YL, Liu G, Li QH, *et al*. An efficient scheduling algorithm for dependent tasks. Proceedings of the 4th International Conference on Computer and Information Technology. Wuhan, China. 2004. 456–461.
17 Yang T, Gerasoulis A. DSC: Scheduling parallel tasks on an unbounded number of processors. IEEE Transactions on Parallel and Distributed Systems, 1994, 5(9): 951–967. [doi: 10.1109/71.308533]
18 Kwok YK, Ahmad I. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors. IEEE Transactions on Parallel and Distributed Systems, 1996, 7(5): 506–521. [doi: 10.1109/71.503776]
19 Ahmad I, Kwok YK. On exploiting task duplication in parallel program scheduling. IEEE Transactions on Parallel and Distributed Systems, 1998, 9(9): 872–892. [doi: 10.1109/71.722221]
20 Darbha S, Agrawal DP. Optimal scheduling algorithm for distributed-memory machines. IEEE Transactions on Parallel and Distributed Systems, 1998, 9(1): 87–95. [doi: 10.1109/71.655248]
21 Topcuoglu H, Hariri S, Wu MY. Task scheduling algorithms for heterogeneous processors. Proceedings of the 8th Heterogeneous Computing Workshop. San Juan, Puerto Rico. 1999. 3–14.
22 Topcuoglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3): 260–274. [doi: 10.1109/71.993206]
23 Arabnejad H, Barbosa JG. List scheduling algorithm for heterogeneous systems by an optimistic cost table. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(3): 682–694. [doi: 10.1109/TPDS.2013.57]
24 曹仕杰. 带通信开销的多 DAG 调度算法研究[硕士学位论文]. 大连: 大连理工大学, 2017.
25 田国忠. 多 DAG 共享资源调度的若干问题研究[博士学位论文]. 北京: 北京工业大学, 2013.
26 葛维春, 叶波. 负载均衡优先的改进优先级表调度算法. 沈阳工业大学学报, 2017, 39(3): 241–247.