ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*, Vol.21, No.12, December 2010, pp.3211−3219
doi: 10.3724/SP.J.1001.2010.03764

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

# 独立多处理机任务静态调度问题的近似算法*

黄金贵[1+], 李荣珩[2]

[1](湖南师范大学 计算机教学部,湖南 长沙 410081)

[2](湖南师范大学 数学与计算机学院,湖南 长沙 410081)

## Approximation Algorithm for Scheduling Independent Multiprocessor Jobs

HUANG Jin-Gui[1+], LI Rong-Heng[2]

[1](Department of Computer Teaching, Hu'nan Normal University, Changsha 410081, China)

[2](College of Mathematics and Computer, Hu'nan Normal University, Changsha 410081, China)

+ Corresponding author: E-mail: hjg@hunnu.edu.cn

**Huang JG, Li RH. Approximation algorithm for scheduling independent multiprocessor jobs. *Journal of Software*, 2010,21(12):3211−3219.** http://www.jos.org.cn/1000-9825/3764.htm

**Abstract**: This paper studies the multiprocessor job scheduling problem, and describes the $m$ processors system, and analyze the algorithm for the problem of the offline version, both $P_m|fix|C_{\max}$ of the scheduling problem with arbitrary process time jobs, and $P_m|fix, p=1|C_{\max}$ of the scheduling problem with unit processing time jobs. Severalvery simple and practical polynomial time approximation algorithm are constructed, a $(\sqrt{2m}+1)$-approximation algorithm for the problem $P_m|fix, p=1|C_{\max}$ and a $2\sqrt{m}$-approximation algorithm for the problem $P_m|fix|C_{\max}$, by usingthe Split Scheduling (SS), the First Fit (FF) and the Large Wide First (LWF) technique. The results are better than any seen in the literature at present.

**Key words**: multiprocessor job scheduling; approximation algorithm; approximation ratio; NP-hard problem

摘 要: 研究独立多处理机任务静态调度问题 $P_m|fix|C_{\max}$,即在 $m$ 个处理机系统中调度 $n$ 个多处理机任务,每个任务指派到所需一组处理机上不可剥夺地执行.该问题应用广泛但早已证明为 NP 难问题,而且也不存在常数近似算法.分析了问题 $P_m|fix|C_{\max}$ 和其中所有任务都是单位处理机时间的特殊情形 $P_m|fix,p=1|C_{\max}$ 的调度,并利用实例划分(split scheduling,简称 SS)、首次满足优先(first fit,简称 FF)和最大宽度优先(large wide first,简称 LWF)等方法,构造了问题 $P_m|fix,p=1|C_{\max}$ 的 $\sqrt{2m}+1$ 近似算法和问题 $P_m|fix|C_{\max}$ 的 $2\sqrt{m}$ 近似算法,优于目前已有文献的最好结果.

关键词: 多处理机任务调度;近似算法;近似比;NP 难问题

中图法分类号: TP316 文献标识码: A

一个调度问题通常是给定一组具有相互依赖关系的任务和一组处理机,目标是指派任务到处理机以满足优化指标,如最小最大完工时间等等[1,2].根据任务在执行过程中是否可被中断,可以将调度模型分为剥夺式调

* Supported by the National Natural Science Foundation of China under Grant Nos.60872039, 10771060 (国家自然科学基金)
Received 2009-06-26; Accepted 2009-11-04

度模式和非剥夺式调度模型.如果任务之间没有依赖关系,则称之为独立的.在经典的调度模型中,每个任务仅需要一个处理机就能够执行完成,然而,随着大规模并行机和网络并行计算技术的发展,新的理论调度模型被提出,以适应这种并行体系结构.多处理机任务调度模型[3,4]就是其中的一种.在这种模型中,一个任务往往需要一个或多个处理机同时执行才能完成,这样的任务被称为多处理机任务.

关于多处理机任务调度问题,无论是在理论研究还是实际应用上,都引起了越来越广泛的兴趣和关注.近年来,关于其可行性和近似算法已有大量的研究成果.对于多处理机调度模型及其变体,研究热点不断涌现.Jansen[5]等人用图着色问题研究了剥夺式多处理机任务调度问题的近似可解性,还研究了可塑性(malleable)多处理机任务的调度[6,7],即任务执行时间是指派给该任务的处理机数目的函数.Johannes[8],Ye 和 Zhang[9]研究了同等处理机系统中的多处理机任务调度问题,每个任务需要一定数目的处理机,与具体的处理机无关.

本文仍然关注最基本的多处理机任务调度模型,具有如下特征:(1) 每个任务对处理机的需求是一个固定的处理机子集,且每个处理机的功能和性能可能是不相同的;(2) 任务在执行过程中是不可剥夺的;(3) 每个任务是独立的,即没有优先依赖关系;(4) 一组任务在调度之前全部到达,需要的信息全部已知,即为离线静态调度.该问题可以用标准的三段式记为 $P_m|fix|C_{\max}$[10,11].其中,$P_m$ 表示 $m$ 个处理机的系统,$fix$ 表示执行每个任务需要拥有的一组固定的处理机,$C_{\max}$ 表示调度目标是使得时间跨度(makespan)最小.每一个任务都只能在它所需要的所有处理机都空闲的情况下才能被调度执行,而且在执行期内,它所占用的任一处理机不能被剥夺、中断或执行其他任务.

Hoogeveen 等人证明:当 $m\geq 3$ 时,该问题是强 NP 难问题的,即除非 $P=NP$,不存在完全多项式时间近似调度算法(FPTA)[12,13],文献[14]还证明,不存在常数近似算法除非 $NP=co\text{-}NP$.为了寻求一种可行的近似解,人们首先关注的是小数目处理机的特殊情形.当 $m=3$ 时,Dell'Olmo 等人为 $P_3|fix|C_{\max}$ 构造了多项式时间的近似比为 5/4 的近似调度算法[15],后被 Goemans[16]改进为 7/6,Chen 和 Huang 又进一步改进为 9/8[17].当 $m=4$ 时,Huang 等人于 2007 年给出了 3/2-近似算法[18];2009 年,Huang 等人又通过构造一个最优规则调度得到了该问题的 4/3-近似算法[19].但是,这些算法都很难推广到一般的 $m$ 处理机系统中.

对于 $m>2$ 的更为一般的问题 $P_m|fix|C_{\max}$,Amoura 等人[20]于 1998 年对于固定的 $m$,利用大枚举和大规模的线性规划方法给出了近似比为 $1+\varepsilon$的多项式时间算法;后来由 Chen 和 Miranda[21]给出了一些改进.看来,该问题似乎已被彻底解决.但遗憾的是,这些多项式时间算法都是伪多项式的,即只是针对 $n$ 而言是多项式,而对于 $m$ 和 $1/\varepsilon$来说却是超指数的复杂度.例如,即使 $m=3,\varepsilon=0.2$, 时间就可以达到 $O(n^{50})$,这显然是不实用的.因此,寻求更为实用的近似算法的工作还远未终止.Chen 和 Lee[22]给出近似比为 $m/2$ 的线性时间近似调度算法.Bampis 等人[23]利用 Split-Round 技术,给出了问题 $P_m|fix,p=1|C_{\max}$(每个任务具有 1 个单位处理时间)的 $2\sqrt{m}$ -近似算法和问题 $P_m|fix|C_{\max}$ 的 $3\sqrt{m}$ -近似算法.

本文研究了 $m$ 个处理机系统的多处理机任务调度模型 $P_m|fix|C_{\max}$ 和 $P_m|fix,p=1|C_{\max}$,利用部分调度技术和启发式调度方法,构造了问题 $P_m|fix,p_j=1|C_{max}$ 的($\sqrt{2m}$ +1)-近似算法和问题 $P_m|fix|C_{\max}$ 的 $2\sqrt{m}$ -近似算法,优于 Bampis 等人[23]给出的最好结果.

## 1 问题描述

$P_m|fix|C_{\max}$ 问题的多处理机系统定义为 $m$ 个处理机的集合 $P=\{1,2,3,\ldots,m\}$,所需要该系统调度执行的多处理机任务集合为 $J=\{1,2,3,\ldots,n\}$.其中,每个任务 $j(1\leq j\leq n)$具有二元属性$(fix_j,p_j)$.$fix_j\subseteq P$ 是执行任务 $j$ 需要拥有的一组处理机,通常称为处理机模式或处理机类型.每个任务 $j$ 需要的处理机个数记为$|fix_j|$,称为任务的"宽度".$p_j$ 是这组处理机同时处理该任务所需要的时间量,称为时间长度.我们的目标就是构造一个调度,以非剥夺方式调度该系统中的 $m$ 个处理机,以最短的时间跨度完成所有给定的多处理机任务,即 $C_{\max}$ 要尽可能地小.

问题 $P_m|fix,p=1|C_{\max}$ 是问题 $P_m|fix|C_{\max}$ 的一种特殊情形.设 $P_m|fix|C_{\max}$ 问题的任务实例 $J$ 的最优调度为 $Opt$,其时间跨度记为 $Opt(J)$.对于任务实例 $J$,若每个处理机 $i(i=1,2,3,\ldots,m)$所需要的总执行时间为 $L_i(J)$,平均执行时间为 $L_0(J)$,则有

$$Opt(J)\geq\max\{L_i(J)|i=0,1,2,\ldots,m\} \tag{1}$$

## 2 基本算法

本节主要介绍实例划分调度(split schedule,简称 SS)、首次满足优先(first fit,简称 FF)和最大宽度优先(large wide first,简称 LWF)这 3 种基本算法.

### 2.1 划分调度SS

首先给出问题 $P_m|fix|C_{\max}$ 的任务实例 $J=\{1,2,3,\ldots,n\}$的一个划分 *Split*

$$J_1\underline{\Delta}\{j\in J:|fix_j|>k\},\ J_2\underline{\Delta}\{j\in J:|fix_j|\leq k\} \tag{2}$$

其中,$k(1\leq k\leq m)$称为划分参数.

划分调度 SS 就是利用任务实例 $J$ 的 *Split* 划分,然后,对两个子实例 $J_1$ 和 $J_2$ 分别采用相应的调度算法独立调度,则 SS 调度的时间跨度为 $C_{\max}(J)=C_{\max}(J_1)+C_{\max}(J_2)$,且有 $\max\{Opt(J_1),Opt(J_2)\}\leq Opt(J)\leq Opt(J_1)+Opt(J_1)$.

引理 **1**. 对于任务实例 $J$ 由 *split* 划分的子实例 $J_1$,不论怎样调度,其时间跨度为 $C_{\max}(J_1)\leq(m/k)Opt(J)$.

证明:对于子实例 $J_1$,无论怎样调度,其时间跨度 $C_{\max}(J_1)\leq\sum_{j\in J_1}p_j$ .又当 $j\in J_1$ 时,$|fix_j|>k$,所以,

$$C_{\max}(J_1)\leq\sum_{j\in J_1}p_j\leq\frac{1}{k}\sum_{j\in J_1}p_j\mid fix_j\mid=\frac{m}{k}\left(\frac{1}{m}\sum_{j\in J_1}p_j\mid fix_j\mid\right)\leq(m/k)Opt(J).$$

引理得证. □

### 2.2 首次满足优先算法FF

FF 算法就是调度任务队列中第 1 个能被当前空闲处理机集合所能处理的任务.记当前空闲和忙碌处理机集合分别为 $P_{idle}$ 和 $P_{busy}$,它们都是处理机集合 $P$ 的子集.用一个 $m$ 元数组 $T[m]$记录各个处理机需要处理的时间,全程计时器 $t$ 记录从调度开始到全部任务完成的每个调度时刻,则对$\forall i\in\{1,2,3,\ldots,m\}$,当 $i\in P_{idle}$ 时,$T[i]=0$;当 $i\in P_{busy}$ 时,$T[i]>0$.

FF 算法基于如下 3 种基本操作:

- 初始化空闲处理机集合 $P_{idle}$ 和忙碌处理机集合 $P_{busy}$,即 $P_{idle}=P;P_{busy}=\varnothing$.初始化数组 $T[m]$和计时器 $t$,即 $T[m]=0;t=0$;
- 为任务 $j$ 分配处理机集合 $fix_j$,即 $P_{idle}=P_{idle}-fix_j;P_{busy}=P_{busy}\cup fix_j$;记录任务 $j$ 的调度时刻 $t$;对每一个 $i\in fix_j$ 更新数组 $T[i]=p_j$;
- 回收最早将要完成任务的处理机集合.首先求出忙碌处理机的最小处理时间 $t_{\min}$,再更新计时器、空闲处理机集合、忙碌处理机集合及处理时间数组.即 $t=t+t_{\min}$;对每一个 $i\in P_{busy},T[i]=T[i]-t_{\min}$;若 $T[i]=0$,则 $P_{idle}=P_{idle}\cup\{i\};P_{busy}=P_{busy}-\{i\}$.

引理 **2**. 设 $J$ 是问题 $P_m|fix|C_{\max}$ 的任务实例 $J$,且每个任务 $j$ 的处理机模式都不大于一个常数 $k$,即$|fix_j|\leq k$,其中,$k$ 为整数且 $1\leq k\leq m$,则调度算法 FF 是该实例的 $k$-近似算法,即 $C_{\max}(J)\leq kOpt(J)$.

证明:考察实例 $J$ 的 FF 调度图(如图 1 所示),假设按上述调度最后完成的任务是 $s_q\in J$,且$|fix(s_q)|\leq k$.我们现在重点关注任务 $s_q$ 所需要的一组处理机 $fix(s_q)$,分析它们从开始到最后结束时的空闲情况.

通过观察图 1 可以发现,存在一组任务$\{s_1,s_2,\ldots,s_q\}\subseteq J$,它们无间隙地依次接连完成,即 $s_1$ 完成后 $s_2$ 执行,$s_2$ 完成后 $s_3$ 执行,直到 $s_q$ 完成.事实上,若在任务组$\{s_1,s_2,\ldots,s_q\}$执行中存在间隙,即在时间$(t,t+p)$内 $fix(s_q)$中的所有处理机都处于空闲,则根据 *FF* 调度算法,此时的任务 $s_q$ 一定是在 $t$ 时刻被调度,即说明这种情况是不可能出现的.

因此,在 FF 调度的整个执行过程中的任何时刻,$fix(s_q)$中的这组处理机 $s_1,s_2,\ldots,s_q$ 至少有一个不空闲.调度的时间跨度 $C_{\max}$ 和该组处理机的平均工作量 $L_0'$ 分别满足:

$$C_{\max}(J)=\sum_{i=1}^{q}p_{s_i},\ L_0'\geq\frac{1}{|fix(s_q)|}\sum_{i=1}^{q}p_{s_i}\ .$$

考虑到引理条件$|fix(s_q)|\le k$,并结合公式(1)可知, $C_{\max}(J)\le kL'_0\le k\max\{L_i\mid i\in fix(s_q)\}\le kOpt(J)$.

引理得证. □

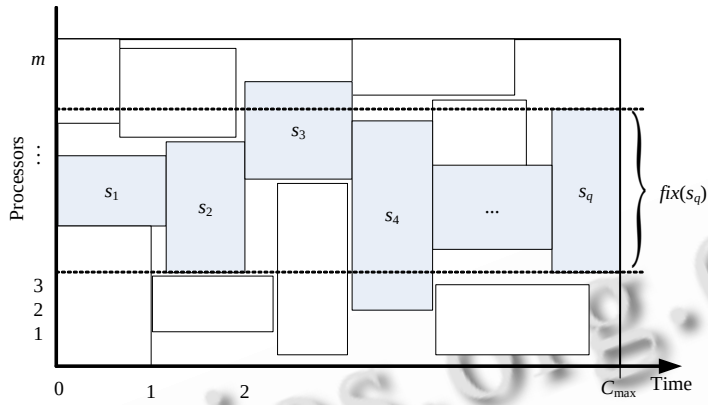



Fig.1 Makespan of the *FF* schedule

图 1 *FF* 调度的时间跨度

### 2.3 最大宽度优先算法LWF

最大宽度优先算法 LWF(largest wide first)的基本思路是:在当前空闲处理机集合所能满足的任务中,优先调度具有最大宽度(即所需要的处理机个数最多)的任务.记调度队列为 $J_Q$,并定义如下两种操作:

(1) 初始化调度队列 $J_Q$,即将任务集 $J$ 中的所有任务按 $fix$ 宽度从大到小排序,并加入到调度队列 $J_Q$ 中,即 $J_Q=Sort(J)=\{fix_1\ge fix_2\ge\ldots\ge fix_n\}$;

(2) $Fetch(J_Q)$:从调度队列中查找第 1 个能满足当前空闲集合 $P_{idle}$ 的任务.若找到任务 $j$,则从调度队列 $J_Q$ 中移走任务 $j$,并返回任务 $j$ 的处理机模式 $fix_j$ 和需要的处理时间 $p_j$;若未找到满足的任务,则返回 0.

## 3 任务为单位处理时间的情形

当每个任务都是 1 个单位处理时间时,问题表示为 $P_m|fix,p_j=1|C_{\max}$,它是问题 $P_m|fix|C_{\max}$ 的一种特例.下面综合应用 LWF 算法和 FF 算法给出问题 $P_m|fix,p_j=1|C_{\max}$ 的近似算法 LWF-FF.

由于所有任务都是单位处理时间,所以 FF 算法得到一些简化.首先,不必记录忙碌处理机的当前需要处理时间,时间可以按 1 个时间单位来处理.回收当前完成任务的处理机集合时,这些忙碌的处理机一定同时完成,且计时器只需加 1 即可.

算法 LWF-FF 描述如下:

Input:处理机系统 $P=\{1,2,3,\ldots,m\}$,问题 $P_m|fix,p_j=1|C_{\max}$ 的任务实例 $J=\{1,2,3,\ldots,n\}$;

Output:调度时间表 $S=\{t_1,t_2,\ldots,t_n\}$,对应每个任务的开始时间.

```
1. Initialization() {
     J_Q=Sort(J)={fix_1≥fix_2≥…≥fix_n};          //调度队列初始化
     P_idle=P; P_busy=∅;                          //空闲处理机集合和忙碌处理机集合初始化
     t=0;}                                        //记录当前时刻的计时器
2. While (J_Q≠∅) do {
     j=Fetch(J_Q);                                //取第 1 个满足当前空闲处理机集合的任务
     If (j=0) {t=t+1; P_idle=P_idle∪P_busy; P_busy=∅;}   //当前任务不存在,回收所有空闲处理机
     Else {t_j=t; P_idle=P_idle−fix_j; P_busy=P_busy∪fix_j;}   //j>0 时,为合适任务 j 分配处理机
   }
```

3. $S=\{t_1,t_2,\ldots,t_n\}$; End.

算法 LWF-FF 在最坏情况下的时间复杂度为 $O(n\log(n)+n^2m)=O(n^2m)$,其中,$n$ 为任务数,$m$ 为系统的处理机数.由于各任务的处理时间都是 1 个单位,所以调度在整个执行过程中可以分成若干个单位时间区间,每段区间内都恰好包含一个或几个任务的全部执行期.下面给出一个例子加以说明.

例 1:设 6-处理机系统 $P=\{1,2,3,4,5,6\}$,即 $m=6$,$P_6|fix,p_j=1|C_{\max}$ 问题的一个任务实例是 $J=\{(12345),(2356),(1246),(156),(135),(246),(235),(145),(23),(34),(46),(25),(46),(13),(3)\}$,共有 15 个任务,每个任务的处理机时间都是1个单位,实例中列出了各任务的处理机模式并按宽度从大到小排序.则按调度算法LWF-FF进行调度的过程和结果如表 1 和图 2 所示.

**Table 1** LWF-FF schedule and optimal schedule on the instance of Example 1

表 **1** 例 1 中实例的 LWF-FF 调度和最优调度

| Schedule time | LWF-FF schedule | Optimal schedule |
|---|---|---|
| 0 | (12345) | (12345) |
| 1 | (2356) | (2356) |
| 2 | (1246), (3) | (1246), (3) |
| 3 | (156), (23) | (156), (34) |
| 4 | (135), (246) | (135), (246) |
| 5 | (235), (46) | (235), (46) |
| 6 | (145) | (145), (23) |
| 7 | (34), (25) | (25), (46), (13) |
| 8 | (46), (13) | |

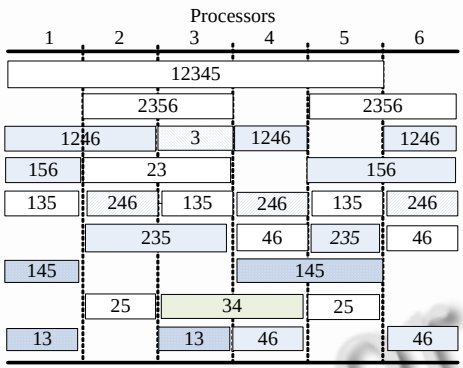



Fig.2 An example of the LWF-FF schedule for the $P_m|fix,p_j=1|C_{\max}$

图 2 LWF-FF 调度的例子

对于该实例,15 个任务按 LWF-FF 调度的调度时间依次列表为 $S=\{0,1,2,3,4,4,5,6,3,7,5,7,8,8,2\}$,调度的时间跨度为 $C_{\max}=9$,而其最优调度(见表 1)的时间跨度为 $Opt(J)=8$.所以,此时 LWF-FF 调度的近似比为 $Ratio=9/8$.

引理 **3**. 对于 $P_m|fix,p_j=1|C_{\max}$ 问题的任务实例 $J$,若每个任务 $j$ 的处理机模式都不小于一个常数 $k$,即 $|fix_j|\geq k$,其中,$k$ 为整数且 $1\leq k\leq m$,则调度算法 LWF-FF 是该实例的$(m/(2k)+1)$近似算法,即近似比

$$Ratio=C_{\max}(J)/Opt(J)\leq m/(2k)+1.$$

证明:根据 LWF-FF 调度算法可知,在调度过程中的每个单位时间内,至少有一个任务恰好覆盖整个单位时间区间.因为$|fix_j|\geq k$,所以至多有$\lfloor m/k\rfloor$个任务在一个单位时间区间内.可以将这些单位时间区间$[t,t+1]$分为两类:(1) 单位时间区间$[t,t+1]$内有且只有一个任务;(2) 单位时间区间$[t,t+1]$内至少有两个任务.

由于 LWF-FF 调度中每个单位时间区间可以调整次序而不会影响调度的时间跨度,为了便于分析方便,通过交换单位时间区间段的位置将原调度分成两个部分:(1) 每个单位时间区间内有且只有一个任务,记该部分的所有任务集合为 $J_1$;(2) 每个单位时间区间内至少有两个任务,记该部分的所有任务集合为 $J_2$;显然,$C_{\max}(J)=C_{\max}(J_1)+C_{\max}(J_2)$.

分别考虑这两部分的调度.在第 1 部分中,显然,任意两个任务之间都是不可并行的,否则,这两个任务应该同时执行且划分到第 2 部分中.因此,第 1 部分的时间跨度 $C_{\max}(J_1)\le Opt(J_1)\le Opt(J)$.

在第 2 部分中,每个单位时间区间内都至少有两个任务,且每个任务 $j$ 有 $|fix_j|>k$,所以从每个处理机的平均执行时间上考虑,结合公式(1),容易推导 LWF-FF 算法在任务子实例 $J_2$ 上的时间跨度为

$$C_{\max}(J_2)\le \frac{1}{2k}\sum_{j\in J_2}1\cdot|fix_j|\le \frac{m}{2k}\cdot\frac{1}{m}\sum_{j\in J_2}|fix_j|\le \frac{m}{2k}\cdot Opt(J_2)\le \frac{m}{2k}\cdot Opt(J).$$

综合这两部分,即得 $C_{\max}(J)=C_{\max}(J_1)+C_{\max}(J_2)\le (m/(2k)+1)\cdot Opt(J)$.引理得证. □

定理 **1**. 调度算法 LWF-FF 是问题 $P_m|fix,p_j=1|C_{\max}$ 的 $\sqrt{2m}+1$ 近似算法.

证明:因为 LWF-FF 的时间复杂度为 $O(n^2m)$,即为多项式时间近似算法,所以下面只需证明近似比 $Ratio=\sqrt{2m}+1$.设任务集合 $J$ 是问题 $P_m|fix,p_j=1|C_{\max}$ 的任意实例,用 *Split* 划分并在公式(2)中取 $k=\sqrt{m/2}$ 得到两个子实例 $J_1$ 和 $J_2$,可以证明,当用 LWF-FF 算法调度任务实例 $J$ 和分别调度两个子实例 $J_1$ 和 $J_2$ 时的时间跨度满足关系

$$C_{\max}(J)\le C_{\max}(J_1)+C_{\max}(J_2).$$

事实上,按照 LWF-FF 算法调度任务实例 $J$ 时,初始调度队列即为 $J_Q=\{J_1,J_2\}$,即队列的前面部分是 $J_1$,后面部分是 $J_2$.所以在调度时,满足当前空闲处理机集合的任务中,一定是优先 $J_1$ 中的任务;若当前 $J_1$ 中的任务都不满足当前空闲处理机集合时,也会调度 $J_2$ 中合适的任务,但这不影响 $J_1$ 中任务的调度.所以,调度队列前面部分 $J_1$ 的时间跨度和 LWF-FF 单独调度 $J_1$ 时的时间跨度 $C_{\max}(J_1)$相等;而调度队列后面部分 $J_2$ 的时间跨度小于或等于 LWF-FF 单独调度 $J_2$ 时的时间跨度 $C_{\max}(J_2)$,因为 $J_2$ 中可能有一部分任务在前面部分被调度.所以有上述关系公式成立.

这说明,我们只需要得出 LWF-FF 算法单独调度 $J_1$ 和 $J_2$ 时的近似比即可.对于 $J_1$ 应用引理 2,并将 $k=\sqrt{m/2}$ 代入,得到 $C_{\max}(J_1)\le\left(\frac{m}{2k}+1\right)\cdot Opt(J_1)=(\sqrt{m/2}+1)\cdot Opt(J_1)$.同理,对于 $J_2$ 应用引理 3,并将 $k=\sqrt{m/2}$ 代入,得到 $C_{\max}(J_2)\le k\cdot Opt(J_2)=\sqrt{m/2}\cdot Opt(J_2)$.于是,

$$C_{\max}(J)\le C_{\max}(J_1)+C_{\max}(J_2)\le(\sqrt{m/2}+1)\cdot Opt(J_1)+\sqrt{m/2}\cdot Opt(J_2)\le(\sqrt{2m}+1)Opt(J).$$

定理 1 得证. □

## 4 任务为任意处理时间的情形

结合 SS 调度和 FF 算法给出任务为任意处理时间时问题 $P_m|fix|C_{\max}$ 的近似算法 SS-FF,描述如下:

Input:问题 $P_m|fix|C_{\max}$ 的任务实例 $J=\{1,2,3,\ldots,n\}$;

Output:任务实例 $J$ 的调度序列 $S=\{t_1,t_2,\ldots,t_n\}$.

1 将任务实例 $J$ 按公式(2)划分,其中,划分参数 $k$ 取为 $\sqrt{m}$,得到两个子实例 $J_1$ 和 $J_2$;

2 按任意顺序调度子任务实例 $J_1$;

3 用 *FF* 算法调度子任务实例 $J_2$;

```
{J_Q=J_2; P_idle=P; P_busy=∅; T[m]=0; t=C_max(J_1);        //调度初始化
  While (J_Q≠∅) do {
    j=Fetch(J_Q);          //当调度队列不空时,取第 1 个满足当前空闲处理机集合的任务
    If (j=0) {             //当前合适任务不存在,回收最早将要完成的所有任务的所有处理机
      求出忙碌处理机的最小处理时间 t_min;
      t=t+t_min;
      对每一个 i∈P_busy,T[i]=T[i]−t_min; if T[i]=0 {P_idle=P_idle∪{i}; P_busy=P_busy−{i};}
    } else {               //为合适任务 j 分配处理机
      t_j=t;                                  //记录任务 j 的调度时刻
      P_idle=P_idle−fix_j; P_busy=P_busy∪fix_j;
```

对每一个 $i\in fix_j, T[i]=p_j;\}$ //更新数组

}

4 $S=\{t_1,t_2,\ldots,t_n\}$; End.

定理 **2**. 算法 SS-FF 是问题 $P_m|fix|C_{\max}$ 的 $2\sqrt{m}$ 近似算法.

证明:容易得知,算法 SS-FF 的时间复杂度为 $O(n^2m)$,所以为多项式时间算法.按照算法 SS-FF 得到实例 $J$ 的调度,其时间跨度 $C_{\max}(J)=C_{\max}(J_1)+C_{\max}(J_2)$,根据引理 1 和引理 2(其中,$k=\sqrt{m}$ ),可以得到

$$C_{\max}(J)=C_{\max}(J_1)+C_{\max}(J_2)\le(m/\sqrt{m})Opt(J)+\sqrt{m}Opt(J)=2\sqrt{m}Opt(J).$$

定理 2 得证. □

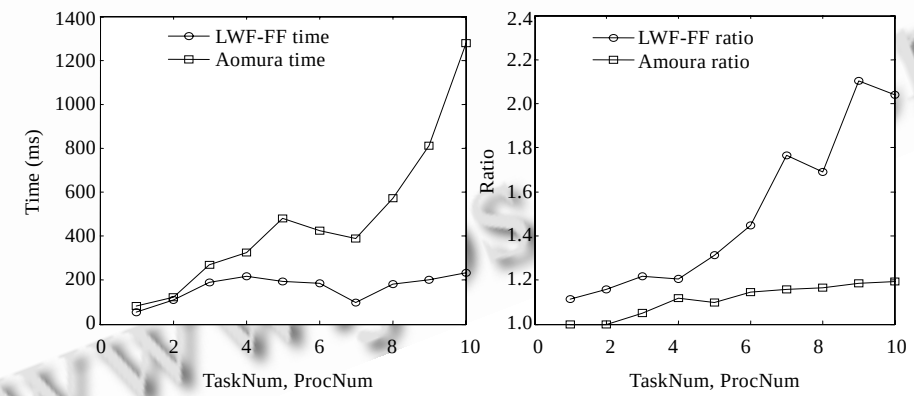
## 5 实验验证与分析

本文实验环境是:CPU T7250/2.0GHz, RAM1G,操作系统 Windows Vista.采用 VC++6.0 和 Matlab 7.0 联合编程.实例任务数 TaskNum 分别选取 500,600,700,800,900 和 1 000 个,处理机个数 *ProcNum* 分别选取 50,60,70,80,90 和 100 个.为了方便,只对 $p$=1 的情况进行验证.任务模式采用随机数产生,并将模式宽度限制在 *ProcNum*/2 以内,以便任务之间有足够多的可并行性.最优调度的时间跨度(Makespan)下界 *OptLB* 由 $\max\{L_i(J)|i=0,1,2,\ldots,m\}$确定.

为了比较,实验针对 Amoura 算法[20]、Chen[22]算法、Bampis[23]算法和本文的 LWF-FF 算法进行.由于原文献只提供理论证明而没有提供实验数据,所以所有数据将在本实验环境中产生.在同样的环境下,用不同算法调度随机产生的相同实例,记录算法花费时间 Time(单位:s)和调度时间跨度.近似比 *Ratio* 由公式 *Ratio=Makespan*/*OptLB* 求出.

(1) 与 Amoura 算法的比较分析

Amoura 算法是(1+$\varepsilon$)近似算法,取$\varepsilon$=0.2,即近似比 $Ratio\le 1.2$.本文 LWF-FF 算法在算法执行时间上进行比较,实验结果如图 3 所示.




(a) Compared on the algorithm execution time (b) Compared on the algorithm approximation ratio

(a) 算法执行时间比较 (b) 算法近似比的比较

Fig.3 Comparative analysis between LWF-FF algorithm and Amoura algorithm

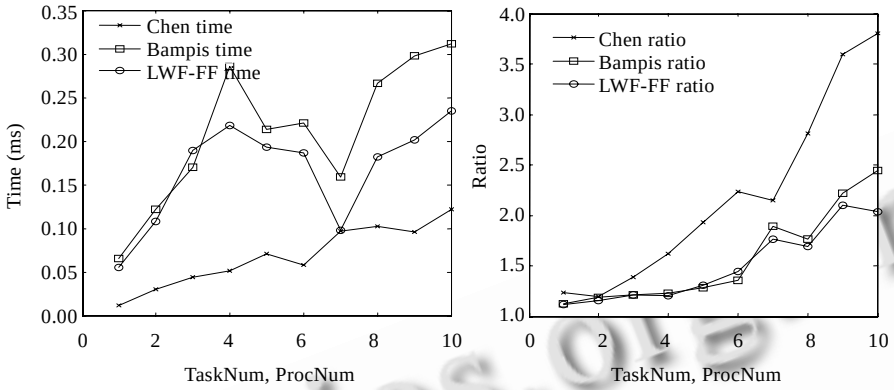
图 3 LWF-FF 算法与 Amoura 算法比较分析

实验结果表明,Amoura 算法保持了很好的近似比性能,但其花费的时间随着任务数目和处理机数目的增加而急剧增大,理论上的多项式时间与实际情况是有差距的.而 LWF-FF 算法花费时间相对较少,且随着任务数目和处理机数目的增加,时间增长缓慢,但其近似比会增大.

(2) Chen 算法、Bampis 算法与 LWF-FF 算法的比较分析

Chen 算法是线性时间算法,Bampis 算法与 LWF-FF 算法都是启发式算法.对于相同的任务实例,3 种算法在

时间开销和近似性能上的实验结果如图 4 所示.




(a) Compared on the algorithm execution time (b) Compared on the algorithm approximation ratio

(a) 算法执行时间比较 (b) 算法近似比性能比较

Fig.4 LWF-FF algorithm's analysis compare to Chen and Bampis algorithm

图 4 LWF-FF 算法与 Chen 算法、Bampis 算法的比较分析

实验结果表明,在算法时间开销方面,Chen 算法较好,LWF-FF 算法和 Bampis 算法时间开销要大,但 LWF-FF 算法略好于 Bampis 算法.在算法近似比性能方面,LWF-FF 算法和 Bampis 算法均好于 Chen 算法,在大多数情况下,LWF-FF 算法也优于 Bampis 算法.

## 6 结 语

多处理机任务调度问题由于本身的难度,寻求其实用算法就变得相当艰难.本文研究了基本的多处理机任务调度问题 $P_m|fix|C_{\max}$ 及其特例 $P_m|fix,p=1|C_{\max}$.利用 SS,FF 和 LWF 等基本算法进行复合,构造了问题 $P_m|fix,p=1|C_{\max}$ 的($\sqrt{2m}$ +1)近似算法和问题 $Pm|fix|C_{\max}$ 的 $2\sqrt{m}$ 近似算法,优于目前已有文献的最好结果.值得说明的是,在引理 1 和引理 2 的结论下,SS-FF 调度的近似比不会优于本文定理 2 的结果.进一步的改进可以关注任务子实例 $J_1(fix_j>k)$和 $J_2(fix_j\leq k)$的调度算法上.

**References**:

[1] He K, Zhao Y, Huang WQ. A clustering and scheduling algorithm based on task duplication. Chinese Journal of Computers, 2008, 31(5):733−740 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2008.00733]

[2] Wu Q, Bian JN, Xue HX. Scheduling with resource allocation for system-level synthesis. Journal of Software, 2007,18(2):220−228 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/220.htm [doi: 10.1360/jos180220]

[3] Blazewicz J, Drabowski M, Weglarz J. Scheduling multiprocessor tasks to minimize schedule length. IEEE Trans. on Computing, 1986,35(5):389−393. [doi: 10.1109/TC.1986.1676781]

[4] Drozdowski M. Scheduling multiprocessor tasks—An overview. European Journal of Operational Research, 1996,94(2):215−230. [doi: 10.1016/0377-2217(96)00123-3]

[5] Jansen K, Porkolab L. Preemptive scheduling with dedicated processors: Applications of fractional graph coloring. Journal of Scheduling, 2004,7:35−48. [doi: 10.1023/B:JOSH.0000013054.30334.b9]

[6] Jansen K. Scheduling malleable parallel tasks: An asymptotic fully polynomial time approximation scheme. Algorithmica, 2004,39: 59−81. [doi: 10.1007/s00453-003-1078-6]

[7] Jansen K, Porkolab L. Linear-Time approximation schemes for scheduling malleable parallel tasks. Algorithmica, 2002,32: 507−520. [doi: 10.1007/s00453-001-0085-8]

[8] Johannes B. Scheduling parallel jobs to minimize the makespan. Journal of Scheduling, 2006,9:433−452. [doi: 10.1007/s10951-006-8497-6]

[9] Ye DS, Zhang GC. On-Line scheduling of parallel jobs in a list. Journal of Scheduling, 2007,10:407−413. [doi: 10.1007/s10951-007-0032-x]

[10] Graham RL, Lawler EL, Lenstra JK, Kan AHG R. 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]

[11] Veltman B, Lageweg BJ, Lenstra JK. Multiprocessor scheduling with communication delays. Parallel Computing, 1990,16(2-3): 173−182. [doi: 10.1016/0167-8191(90)90056-F]

[12] Hoogeveen JA, Van de Velde SL, Veltman B. Complexity of scheduling multiprocessor tasks with prespecified processor allocations. Discrete Applied Mathematics, 1994,55:259−272. [doi: 10.1016/0166-218X(94)90012-4]

[13] Hall LA. Approximation algorithms for scheduling. In: Hochbaum DS, ed. Proc. of the Approximation Algorithms for NP-hard Problems. PWS Publishing Company, 1997. 1−45.

[14] Huang JG, Chen JE, Chen SQ. Parallel-Job scheduling on cluster computing systems. Chinese Journal of Computers, 2004,27(6): 765−771 (in Chinese with English abstract).

[15] Dell'Olmo P, Speranza MG, Tuza ZS. Efficiency and effectiveness of normal schedules on three dedicated processors. Discrete Mathematics, 1997,164:67−79. [doi: 10.1016/S0012-365X(97)84781-4]

[16] Goemans MX. An approximation algorithm for scheduling on three dedicated machines. Discrete Applied Mathematics, 1995,61: 49−59. [doi: 10.1016/0166-218X(94)00160-F]

[17] Chen JE, Huang JG. Semi-Normal schedulings: Improvement on Goemans. In: Eades P, Takaoka T, eds. Proc. of the Algorithms and Computation (ISAAC 2001). LNCS 2223, Berlin: Springer-Verlag, 2001. 48−60.

[18] Huang JG, Chen JE, Chen SQ. A simple linear time approximation algorithm for multiprocessor job scheduling on four processors. Journal of Combinatorial Optimization, 2007,13:33−45. [doi: 10.1007/s10878-006-9011-y]

[19] Huang JG, Li RH. An optimal algorithm for normal scheduling on $P_4|fix|C_{\max}$. Chinese Journal of Computers, 2009,32(8): 1631−1636 (in Chinese with English abstract).

[20] Amoura AK, Bampis E, Kenyon C, Manoussakis Y. Scheduling independent multiprocessor tasks. Algorithmica, 2002,32:247−261. [doi: 10.1007/s00453-001-0076-9]

[21] Chen JE, Miranda A. A polynomial time approximation scheme for general multiprocessor job scheduling. In: Proc. of the 31st Annual ACM Symp. on Theory of Computing (STOC'99). ACM Press, 1999. 418−427.

[22] Chen JE, Lee CY. General multiprocessor tasks scheduling. Naval Research Logistics, 1999,46:57−74.

[23] Bampis E, Caramia M, Fiala J, Fishkin AV, Iovanella A. Scheduling of independent dedicated multiprocessor tasks. In: Bose P, Morin P, eds. Proc. of the ISAAC 2002. LNCS 2518, Berlin: Springer-Verlag, 2002. 175−194. [doi: 10.1002/(SICI)1520-6750(199902)46:1<57::AID-NAV4>3.0.CO;2-H]

## 附中文参考文献:

[1] 何琨,赵勇,黄文奇.基于任务复制的分簇与调度算法.计算机学报,2008,31(5):733−740.

[2] 吴强,边计年,薛宏熙.系统级综合中结合资源分配的调度算法.软件学报,2007,18(2):220−228. http://www.jos.org.cn/1000-9825/18/220.htm [doi: 10.1360/jos180220]

[14] 黄金贵,陈建二,陈松乔.网络集群计算系统中的并行任务调度.计算机学报,2004,27(6):765−771.

[19] 黄金贵,李荣珩.$P_4|fix|C_{\max}$ 问题的最优规则调度算法.计算机学报,2009,32(8):1631−1636.

黄金贵(1964－),男,湖南临澧人,博士,教授,主要研究领域为网络优化计算,参数计算与近似算法,资源管理与调度,软件过程技术.

李荣珩(1963－),男,博士,教授,主要研究领域为组合优化及算法理论.