文章编号:1001 - 9081(2006)09 - 2217 - 05

单调速率任务分配算法利用率的界限分析

王 涛,刘大昕
(哈尔滨工程大学 计算机科学与技术学院, 黑龙江 哈尔滨 150001)
(wt730620@126.com)

摘 要:通过对单调速率任务分配算法调度策略和可调度条件的分析,在多处理器周期任务抢占调度模型基础上,细致刻画了任务分配算法如何分配任务的行为。依据 Liu 和 Layland 定理,给出多处理器下任务分配算法的最小 RM 利用率界的定理。仿真结果表明,分配算法的利用率界是不同特征任务集选择不同分配算法进行任务划分的关键,通过对任务集总利用率与算法利用率界的比较,判断使用该算法对任务集是否可以产生可行分配。

关键词:速率单调; 任务分配; 可调度性; 利用率界
中图分类号: TP316 **文献标识码**:A

Analysis of utilization bound for rate monotonic tasks assignment algorithms

WANG Tao, LIU Da-xin
(*Department of Computer Science and Technology, Harbin Engineering University, Harbin Heilongjiang* 150001, *China*)

Abstract: The scheduling strategies and schedulability conditions of RM(Rate Monotonic) tasks allocation algorithms were analyzed, and the actions that how the tasks allocation algorithms allocate the task to processors were depicted in detail based on the model of preemptive period task in multiprocessor system. According to the theory of Liu & Layland, the theory of the minimum utilization bound for task allocation algorithm was presented. The results of simulation show that the utilization bound of allocation algorithm is the key principle for differently characterized task sets to select different algorithms to allocate tasks. Through the comparison between the total utilization of the task set and the utilization bound of the algorithm, it can be judged whether the algorithm will bring out feasible allocation of the task set.

Key words: RM(Rate Monotonic); task allocation; schedulability; utilization bound

0 引言

多处理器环境下任务调度算法比单处理器系统复杂。多处理器系统中调度算法不仅需要对任务集调度排序还要决定哪些任务需要使用哪些处理器进行调度。大型周期固定优先级任务集在多处理器系统上的可抢占调度问题被认为是难以计算的,是 NP 困难问题[1]。因此,研究集中在开发适合的启发式算法,这些启发式算法与理想的最优算法相比,要求不仅能够高效的实现还要能够尽量少地使用有限数量的额外处理器。

解决多处理器系统划分方案下的任务调度问题基本思想是在单处理器上使用 RM 调度算法[2],并利用多处理器下任务集的可调度条件。开发新型实时任务分配算法成为实时界研究的一个热点问题。RM 算法的可调度判定条件是扩展 RM 调度范围的核心理论基础,不仅在单处理器系统在多处理器环境下,RM 调度的可调度条件仍然起着举足轻重的作用。文献[3]详细分析归纳了至今为止单处理器环境下大部分已被证明的 RM 算法可调度性判定条件,并讨论了考虑时间开销和优先级逆转情况下的 RM 算法可调度性判定条件。迄今为止,国内在多处理器任务分配算法利用率界分析和性能评价方面的工作较为少见。针对上述情况,基于算法利用率界分析,本文对几种主要单调速率任务分配算法进行了理论和实验分析,目的是用于指导对不同任务集如何选择分配算法,从而提供一种提高任务集可调度利用率的有效途径。

1 任务调度策略和模型

1.1 任务调度策略

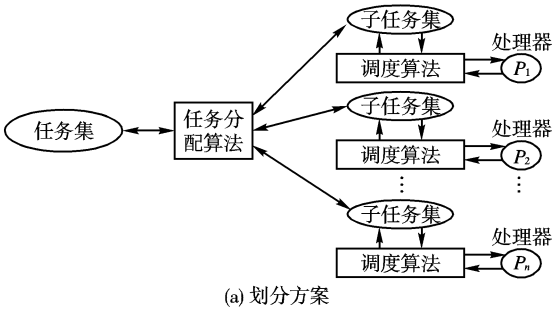
多处理器环境的任务调度存在两种策略,全局方案和划分方案[4]。在全局调度方案策略中,实时任务的每一次出现都在不同的处理器上执行,所有处理器上只运行同一种调度算法。任务在未执行完之前可以被抢占并且可以在不同的处理器间迁移,同时假定多处理器间共享内存的开销非常低,这种方案的主要目标就是为多处理器系统产生一个能够满足它们各自期限的任务分配。相反,在划分调度方案中,一个任务的所有出现都在同一处理器上执行,全部任务由任务分配算法预先划分到处理器; 每一个处理器可以运行不同或者相同的单处理器任务调度算法,这种划分方法主要用于任务集参数已知的静态优先级调度,对任务集进行脱机分配,它充分利用了“分治算法”的设计思想,将难解的问题分解为 n 个子问题,分而治之,逐一突破。

划分方案与全局方案相比,具有几个优势。首先,由于多处理器调度的开销仅仅在于给处理器分配任务,并且这个操作在任务第一次执行前仅仅执行一次,所以划分调度的复杂性更低; 第二,一旦给处理器分配任务的操作完成,剩下的工

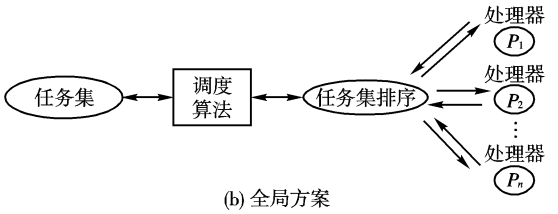
收稿日期:2006 - 03 - 10; **修订日期**:2006 - 05 - 23

作者简介:王涛(1973 -),男,博士研究生,主要研究方向:实时系统调度理论; 刘大昕(1941 -),男,教授,主要研究方向:数据库与知识库、入侵检测、安全数据库、工作流管理系统.

作就可以由单处理器调度算法完成。图 1 是两种策略的示意图,(a)为划分方案,任务由分配算法分配给处理器,每个处理器运行单处理器调度算法,被划分的子任务集与处理器绑定不再变化;(b)为全局方案,调度算法对任务调度排序,任务在执行过程中可在不同处理器间迁移。



(a) 划分方案



(b) 全局方案

图 1 划分方案和全局方案

划分方案的性能由两个因素决定:给处理器分配任务的任务分配算法和每个处理器上决定任务执行顺序的任务调度算法。对一个给定的调度算法,一个最优的任务分配算法可以用最少数量的处理器为每一个处理器找到一个可行的调度。但是,为固定优先级调度算法(包括静态优先级 RM 算法和动态优先级 EDF 算法)找到一个最优任务分配问题却是一个 NP 困难问题[5]。在一个多处理器系统上分配实时任务的任何一个实际调度算法的计算复杂性和算法的性能总是相互矛盾的。早期的研究主要集中在性能保证和最坏情况界方面对算法的组合分析。度量一个任务分配算法 A 的性能渐进性界,使用公式 $\Re_A = \lim_{N_{opt}\to\infty} N/N_{OPt}$,其中 N 为启发式分配算法所需处理器数量,N_{opt} 为最优分配算法所需处理器数量。但 $\Re_A$ 只是用于比较不同任务分配算法的性能,代表的是算法与最优算法相比较的性能界。对任务在算法下的可调度性测试并不实用,因为最优算法所需的处理器数在多项式时间内无法获得。

1.2 调度模型

考虑一组周期任务在同等多处理器上的可抢占调度问题,多处理器和任务集描述如下:

设 $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$ 为含有 n 个周期任务的任务集合,$\tau_i = (C_i, T_i)$ 且 $i = 1, \cdots, n$。其中 τ_i 表示相互独立的单个任务(假设任务间没有任何依赖关系和资源约束),其执行时间和周期分别为 C_i 和 T_i($T_i > 0$),相对期限为 D_i,本文仅考虑任务周期和相对期限相等并且任务执行时间小于等于周期的情况,即 $T_i = D_i$ 且 $0 \leqslant C_i \leqslant T_i$。任务 τ_i 的利用率为 $u_i = C_i/T_i$,任务集的总利用率为 $U = \sum_{i=1}^{n} C_i/T_i$。设 α 为任务最大可达利用率,$\alpha = \max u_i$ 且 $i = 1, \cdots, n$。显然 $\alpha \in (0,1]$,仿真实验中将根据不同的 α 取值比较算法的性能。

$P = \{P_1, P_2, \cdots, P_m\}$ 为含有 m 个具有相同处理能力的同等处理器集,η_m 为第 m 个处理器 P_m 的利用率,即分配给该处理器的任务的总利用率 U_m。假设处理器间无任何共享资源同时假定每个处理上运行 RM 调度算法,即如果 $T_i < T_j$,则 τ_i 比 τ_j 具有更高的优先级,优先在该处理器上执行。另外假定任务调度是以可抢占的方式进行并且中断后可恢复执行。

2 多处理器任务分配算法

周期任务的处理器分配问题不仅要依靠分配算法自身同时也需要依靠算法使用的任务集可调度条件。从 Liu 和 Layland 提出基于任务利用率的单处理器可调度性判定定理以来,可调度条件一直在不断被扩展和提高。多处理器系统的可调度条件正是基于单处理器系统的基础建立的。本节讨论多处理器划分调度下的任务集可调度性判定条件、算法及复杂性。

2.1 多处理器 RM 可调度性判定条件

Liu 和 Layland 的可调度条件是基于最坏情况(Worst-Case,WC)对任务集总利用率的界定(简称 LLWC),在多处理器环境下依然可以作为 RM 算法在单个处理上处理所分配的任务可调度性的判定条件,因此在此有必要重申 LLWC 条件[2]。

定理 1(LLWC 条件) 给定周期任务集 τ,如果总利用率满足 $U \leqslant n(2^{1/n} - 1)$,则任务集在 RM 算法下是可调度的。当 $n \to \infty$ 时,处理器利用率 $f(n) = n(2^{1/n} - 1) \to \ln 2$。

为分析 LLWC 条件,选择了两组各包含不同数量任务的任务集,第一组中每个任务集包含任务数量在[100,1 000]之间均匀分布,称为大型任务组;第二组中每个任务集包含任务数量在[5,95]之间均匀分布,称为小型任务组;试验表明小型任务组利用率随任务数递减的速度较快,大型任务组利用率随任务数的增加,利用率趋于平缓,向 $\ln 2 \approx 0.69$ 逼近。图 2 表明函数 $f(n) = n(2^{1/n} - 1)$ 为连续严格单调递减函数,当 $n \to \infty$ 时,处理器利用率 $f(n)$ 将最终稳定在 0.69 附近。LLWC 条件是充分但非必要条件,可能存在任务集能够满足任务的各自期限,但不满足该条件的情况。Burchad 等人针对具有简单周期(具有周期任务的系统具有简单周期是指,对于系统中的任意一对任务 T_i 和 T_k,当相应周期 $p_i < p_k$ 时,p_k 是 p_i 的整数倍)和周期接近的任务集提出更高的处理器利用率上界。这里需要指出,下面将要引出的定理并不能很好的应用于任务利用率较高的任务集合。由于该条件是面向任务周期的,因此简称为面向周期(Period Oriented,PO)条件[5]。

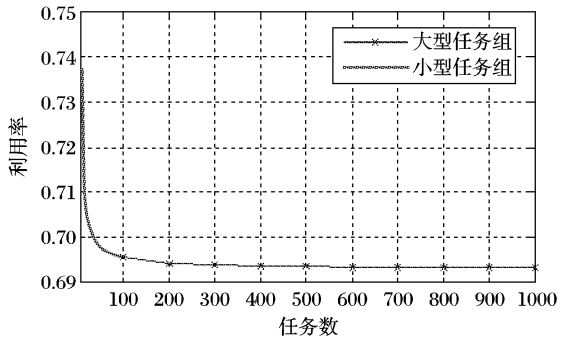


图 2 利用率随 n 变化关系

定理 2(PO 条件) 对于给定周期任务集 $\tau = \{\tau_1, \tau_2,$

…,τ_n},定义:

$$\gamma = \max_{1\leqslant i\leqslant n} D_i - \min_{1\leqslant i\leqslant n} D_i$$

其中:$D_i = \log_2 T_i - \lfloor \log_2 T_i \rfloor$,　$i = 1,\cdots,n$

1)若 $\gamma \leqslant 1 - 1/n$ 并且总利用率满足 $U \leqslant (n-1)(2^{\gamma/(n-1)} - 1) + 2^{1-\gamma} - 1$,则任务集在该处理器下用 RM 算法是可调度的;

2)若 $\gamma > 1 - 1/n$ 并且总利用率满足 $U \leqslant n(2^{1/n} - 1)$,则任务集在该处理器下用 RM 算法是可调度的。

以上两个条件都是紧密的。PO 条件中的 γ 直接的含义是度量系统中任务偏离简单周期的距离。为便于算法实现,推论 1[5] 给出简化的 PO 条件将作为简单周期和近简单周期任务集在分配给处理器时的可调度性判定标准。

推论 1　对于给定任务集 $\tau = \{\tau_1,\tau_2,\cdots,\tau_n\}$ 和 PO 条件中定义的 γ,若总利用率 $U \leqslant \max\{\ln 2, 1 - \gamma\ln 2\}$,则任务集在该处理器下用 RM 算法是可调度的。

为更精确的刻画 RM 算法的行为特征,Lehoczky 等人对随机产生的周期任务集从任务细分利用率概率分布的角度作了随机分析,给出一种 基于累积处理器时间需求的 RM 可调度充分必要条件[6]。该条件表明,对随机周期任务集,随着任务数量的增加,任务执行时间变得不重要,任务利用率会根据任务周期趋于一个常数。

定理 3(NS 条件)　设给定周期任务集 $\tau = \{\tau_1,\tau_2,\cdots,\tau_n\}$ 的周期满足 $T_1 \leqslant T_2 \leqslant \cdots \leqslant T_n$。用 $W_i(t) = \sum_{j=1}^{i} C_j \cdot \lceil t/T_j \rceil$ 表示[0,t]时间间隔任务集前 i 个任务对处理器的累积需求,定义 $L_i(t) = W_i(t)/t$,$L_i = \min_{0<t\leqslant T_i} L_i(t)$,$L = \max_{1<i\leqslant n} L_i$。则

1)第 i 个任务 τ_i 能被可行调度,当且仅当 $L_i \leqslant 1$;

2)任务集 $\tau = \{\tau_1,\tau_2,\cdots,\tau_n\}$ 用 RM 算法可调度当且仅当 $L \leqslant 1$。

2.2　RM 任务划分算法

几乎目前所有的单处理器任务分配方案都是基于文献[2]提出的 RM 可调度充分条件,因而,现存 RM 任务划分方案与启发式装箱方法存在不同。以前工作并没有注意到任务集以下特征,即当某个处理器上的任务知识简单的周期任务时,任务的可调度利用率会比普通任务高。为此文献[7]给出了如下定理。

定理 4　任务系统 τ 中具有独立可抢占的周期任务,并且任务的相对期限等于各自周期。如任务集可划分为 h 个包含简单周期任务的不相邻子集 $S_1,S_2,\cdots,S_h$,并且 h 个子集中任务的总利用率 $U(S_i)$ 且 $i = 1,2,\cdots,h$。满足以下不等式:

$$(1 + U(S_1))(1 + U(S_2))\cdots(1 + U(S_h)) \leqslant 2$$

则该任务系统按照 RM 算法是可调度的。

从定理 4 可知,任务集按速率单调算法在某处理器上调度,若将任务划分为少量仅包含简单周期任务的若干子集,可以显著提高可调度利用率,因此一种多处理器分配算法首先将任务集按照条件划分为多个子集,只要子集总利用率不大于分配给它的处理器利用率上限,那么没给自己都是可调度的,如果有多个子集分配给同一个处理器,那么他们的可调度利用率就是关于子集数的函数,与任务数无关。基于上述思想, Burchad 等人提出了定理 2 的 PO 条件和 RMST 及 RMGT 算法[5],并且证明其性能界分别为 $\Re_{RMST} \leqslant 1/(1-\alpha)$($\alpha$ 为任务最大可达利用率)和 $\Re_{RMGT} = 7/4$,计算时间复杂性都是 $O(n\log n)$。下面给出两种算法的伪代码描述。

算法.RMST	算法.RMGT
Input: Task set $\tau = \{\tau_1,\tau_2,\ldots,\tau_n\}$ and a set of processors $\{P_1,\ldots,P_m\}$	Input: Task set $\tau = \{\tau_1,\tau_2,\ldots,\tau_n\}$ and a set of processors $\{P_1,\ldots,P_m\}$
Output: j; number of processors required.	Output: j; number of processors required.
1. Obtain $D_i := \log_2 T_i - \lfloor \log_2 T_i \rfloor$ $\log_2 T_i$ for all tasks and sort the task set such that $0 \leq D_i \leq \ldots \leq D_n < 1$	1. Partition the task set in two groups: $G_1 := \{\tau_i \mid u_i \leq 1/3\}$; $G_2 := \{\tau_i \mid u_i > 1/3\}$;
2. $i := 1$; $j := 1$; /* $i = i^{th}$ task, $j = j^{th}$ processor */	2. Allocate tasks from the first group using RMST.
3. $D := D_i$; $\gamma_j := 0$; $U_j := u_i$; $i := i + 1$;	3. Allocate tasks from the second group as follows:
4. while ($i \leq n$) do	4. $i := 1$; $U_m := u_i$;
5. $\gamma_j := D_i - D$; /* γ_j value is updated */	5. $i := i + 1$;
6. if ($U_j + u_i \leq max\{\ln 2, 1 - \gamma_j \ln 2\}$ then	6. Use heuristic First-Fit to find a processor k that executes only one task (e.g, task τ_w) and satisfies the following conditions: $\lfloor T_w/T_i \rfloor (T_i - C_i) \geq C_w$ or $T_w \geq \lceil T_w/T_i \rceil C_i + C_w$ if $T_i < T_w$; $\lfloor T_i/T_w \rfloor (T_w - C_w) \geq C_i$ or $T_i \geq \lceil T_i/T_w \rceil C_w + C_i$ if $T_i \geq T_w$
7. $U_j := U_j + u_i$; /* Task τ_i to processor P_j */	7. if such processor exists then
8. else	8. $U_k := U_k + u_i$; /* Allocate task τ_i to processor P_k and set it as fully utilized
9. $j := j + 1$; $D := D_i$; $\gamma_j := 0$;	9. else
10. $U_j := U_j + u_i$; /* Task τ_i to processor P_{j+1} */	10. $U_l := u_i$; /* Allocate task τ_i to an idle processor with index l. */
11. $i := i + 1$;	11. goto 5
12. return(j);	12. Terminate when all tasks from group G_2 are allocated.
13. end	13. return(j) /* j number of processors required in sets G_1 and G_2*/
	14. end

RMGT 算法不依赖于一个任务的利用率,它首先将任务集按照利用率划分为两个子集。利用率小于等于 1/3 的任务使用 RMST 算法处理,利用率大于 1/3 的大型任务使用最先适合方式,分配给最多拥有一个按 RMST 算法分配的任务的处理器。

3　算法利用率界分析

根据装箱问题的启发设计一个任务分配算法并不十分困难,困难的是分析算法在最坏情况下的性能,尤其困难的是找到性能紧密的界。分析算法的利用率界,目的就是将任务集总利用率与各算法的利用率比较,判断待分配的任务集在给定处理器集上使用该算法是否能产生可行的分配,也就是判断什么类型的任务集该使用哪种算法进行划分。因此设计实现一个任务分配算法后算法分析最重要的一个部分就是看能否找到算法紧密利用率界。单处理器下基于 LLWC 的利用率界近似为 0.69,在多处理器下,基于 LLWC 的界变得复杂,原因是不仅要考虑任务数还要考虑到任务利用率和处理器数量对多处理器 RM 分配算法的影响,不仅要可以进行快速的可

调度性测试还要量化分析某个参数对可调度性的影响。下面给出多处理器系统基于 LLWC 的最小 RM 利用率的界的证明。在此之前,需要引入一个参数 β_{LLWC}。

定义 1 一个使用 RM 算法调度的处理器上适合的任务利用率为 α 的最大任务数,用 β_{LLWC} 表示。

引理 1 $\beta_{LLWC} = \left\lfloor \frac{1}{\log_2(\alpha+1)} \right\rfloor$。

证明 β_{LLWC} 为一个任务数,根据定义 1 和定理 1 的 *LLWC* 条件有 $\beta_{LLWC}\alpha \leqslant \beta_{LLWC}(2^{1/\beta_{LLWC}} - 1)$,解该不等式,$\beta_{LLWC} \leqslant \log_2(\alpha+1)$。由于 β_{LLWC} 为任务数,不可能是小数,故有 $\beta_{LLWC} \leqslant \left\lfloor \frac{1}{\log_2(\alpha+1)} \right\rfloor$。根据定义 1,$\beta_{LLWC}+1$ 个任务定会使利用率超过 *LLWC*,故有 $(\beta_{LLWC}+1)\alpha > (\beta_{LLWC}+1)(2^{1/(\beta_{LLWC}+1)} - 1)$,解该不等式有 $\beta_{LLWC} > 1/\log_2(\alpha+1) - 1$。由于 β_{LLWC} 为任务数,不可能是小数,故 $\beta_{LLWC} \geqslant \left\lfloor \frac{1}{\log_2(\alpha+1)} \right\rfloor$,综上引理 1 得证。

基于 *LLWC*,同时考虑到系统处理器数 m,任务数 n 以及任务利用率 α,文献[8]给出多处理器任何合理分配算法最坏情况利用率界,定理如下:

定理 5 设 RA(Reasonabe Allocation) 为任意合理分配算法,若 $n > m\beta_{LLWC}$,则算法利用率

$$U_{LLWC-RA}(n,m,\alpha) \geqslant m_\alpha U_\alpha + m_b U_b - (m-1)\alpha$$

其中:

$$m_\alpha = n + m - 1 - \left\lfloor \frac{n+m-1}{m} \right\rfloor m$$

$$m_b = m - m_\alpha$$

$$U_\alpha = \left\lceil \frac{n+m-1}{m} \right\rceil \left(2^{1/\left\lceil \frac{n+m-1}{m} \right\rceil} - 1\right)$$

$$U_b = \left\lfloor \frac{n+m-1}{m} \right\rfloor \left(2^{1/\left\lfloor \frac{n+m-1}{m} \right\rfloor} - 1\right)$$

定理 5 给出任何一种合理分配算法利用率下限。该定理是个复杂的算法利用率界的分析结果。直观意义是,在 $n-1$ 个任务近似公平的分配给 m 个处理器后,m_a 个处理器分到 $\lceil (n-1)/m \rceil$ 个任务,$m_b = (m - m_a)$ 个处理器分到 $\lfloor (n-1)/m \rfloor$ 个任务,U_a 是 m_a 个处理器中每一个接受一个以上任务的处理的利用率界,U_b 是 m_b 个处理器中每一个接受一个以上任务的处理的利用率界。

由于 *LLWC* 是在任务周期和利用率最坏情形下得出的[2],所以仅是单处理器 RM 可调度的充分不必要条件,因此定理 5 的结果也是最坏情况下的可调度条件。同时也给我们提出了一个问题,多处理器 RM 可调度条件可能不是紧密条件,可能会存在更高的多处理器最大最小利用率界。理论上,找到紧密的多处理器利用率界应该使用 RM 可调度充分必要条件,但是这个多处理器 RM 利用率紧密界的推导将是极其复杂的,至今仍无研究成果公布。

Dhall 和 Liu 在提出 RMFF 和 RMNF 算法的同时给出了 RM 最先适合算法在 $m > 3$ 时,如果任务集不能被 RMFF 算法在 $m-1$ 个处理器下可行调度,那么任务集利用率大于 $U > m/(1+2^{1/3})$。文献[9]中基于 LLWC 条件设计了 RMFF 和 RMBF 算法,并指出,如果 m 个任务在 $m-1$ 个处理器上用 RMFF 算法不能产生可行调度,则任务集利用率 $U > m/(1+2^{1/2})$。文献[10]中分析了在处理器数 $m \geqslant 2$ 时 RM 最先适合划分算法的最坏情况最小利用率界为 $m(2^{1/2}-1) < U_{\min} \leqslant (m+1)/(1+2^{1/(m+1)})$。

4 仿真结果

4.1 处理器集和任务集描述

仿真试验设计使用 30 个处理器,并假定对任务集而言每个处理器上运行相同速率单调度调算法,处理器处理能力相同。同时为保证任务集样本可划分,限制最大任务集含 250 个任务,所有任务集中大多数任务的利用率均匀分布在 30/250 = 0.12 以内,然后为满足不同最大可达利用率分组测试的需要,对每组任务集设置若干机动调整任务已保证任务集在一定的小范围内均匀达到每组最大可达利用率的要求,但在分组中,不同算法严格按照对同一任务集的测试,以保证实验结果能够公平反映各划分算法的性能。

任务集根据最大可达利用率分为三组,$\alpha = 0.3$,$\alpha = 0.6$,$\alpha = 0.9$。每组中在[25,250]中每隔 25 取 1 个值,产生 10 个大小不同的任务集。每个任务集中任务周期取值范围为 $1 \leqslant T_i \leqslant 1000$,在[1,1000]产生 250 个均匀分布的值,任务执行时间根据 $C_i = T_i * u_i$ 计算,u_i 为任务利用率。为保证多数任务集任务利用率小于 0.12,先以 0.12/250 比值为间隔,在0.12 以内产生250 个利用率 u_i。根据处理器需求调整前面几个利用率值均匀分布至最大可达利用率并刚好能够划分 250 个任务,计算出执行时间并不在变化。从后面每次减少 25 个任务,在每组中产生 10 个任务集。三组不同最大可达利用率的任务集共产生 30 个大小不同的任务集。实验使用 XML 文件格式作为任务集载体,所有算法运行环境均为 Windows2000 advanced server,PIV1.8,512M RAM。

4.2 性能指标

由于最优划分所需处理器数量无法计算,所以使用任务集总利用率 U 作为划分所需处理器数量的下界,即划分任务集最少需要 U 个处理器。一个性能指标定义为实际使用处理器数量与被划分任务集总利用率的比值,

$$\rho = \frac{M}{U}$$

其中 M 为试验所需实际处理器数,U 为任务集总利用率。

4.3 实验结果及分析

实验数据表明,随着各算法所需处理器数随任务数增加而增加,RMGT 和 RMST 算法表现出优异的性能,同样大小任务集下所需处理器最少,两算法性能表现非常接近,尤其在 $\alpha = 0.3$ 组的任务划分表现出同样的性能,符合两算法的理论依据,因为 RMGT 算法在 $\alpha \leqslant 0.3$ 时使用 RMST 划分任务集。在 $\alpha = 0.6$ 时,RMGT 性能有所下降,因为算法需要使用最先适合的方式处理利用率大于 0.3 的任务而需要额外的处理器。$\alpha = 0.9$ 时,RMST 算法性能明显下降,证明小任务速率单调算法适合于简单周期较低利用率任务划分,不适合高利用率任务集。另外三种 RMBF、RMNF、RMFF 算法始终表现了相近的性能,符合使用同样可调度条件的理论依据。

图 3 直观描述了三组任务集与处理器关系的变化曲线,从图 3(a)中 RMBF、RMNF、RMFF 三种算法性能表现相同,曲线重合;RMGT 和 RMST 算法重合,表现出相同性能。随着利用率提高,各算法出现差异,表明在对相同任务集由于算法

各有针对,所以会出现差别。图3(b)清楚表明 RMNF 算法使用了最多数量的处理器,从而算法性能最低。图 3(c)出现部分重合的曲线,表明算法对高利用率任务集合性能会下降趋势,尤其 RMST 算法在任务集小于 125 以前出现了曲线高于其他另外三种算法的现象,原因是 RMST 算法不适合高利用率较小的任务集划分。

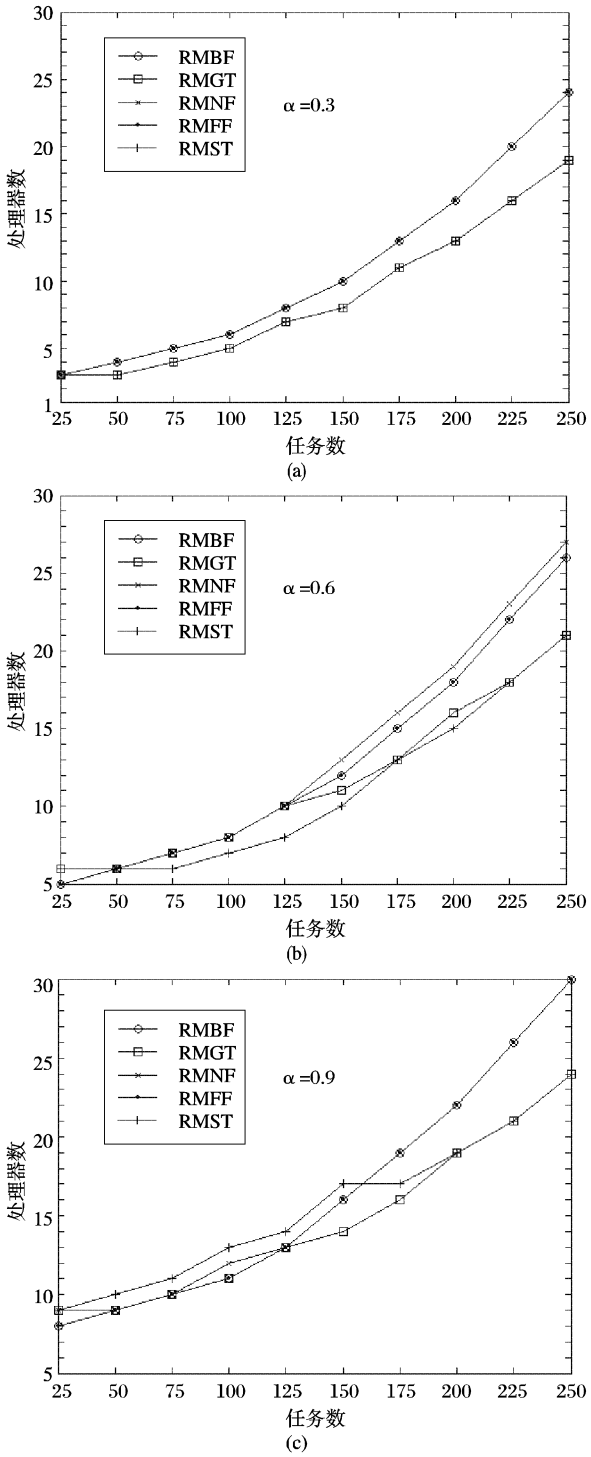


图3　任务数与处理器数关系曲线

表 1　仿真结果

α	ρ_{RMBF}	ρ_{RMGT}	ρ_{RMNF}	ρ_{RMFF}	ρ_{RMST}
α=0.3	[1.41,1.66]	[1.13,1.47]	[1.41,1.66]	[1.41,1.66]	[1.13,1.47]
α=0.6	[1.40,1.54]	[1.16,1.70]	[1.42,1.54]	[1.40,1.54]	[1.15,1.54]
α=0.9	[1.20,1.41]	[1.13,1.35]	[1.28,1.41]	[1.20,1.41]	[1.13,1.48]

表 1 是各算法的最终性能指标,数据表明各算法使用处理器数与任务集总利用率成正比。理论上 ρ 值应该随最大利用率增加而增加,但受个体任务在任务集中出现的位置不同和额外的上下文切换开销等因素影响,实验结果并非与理论一致,不过总体 ρ 值还是没有出现很大的波动,在正常范围之内,在一定程度上也反映出各算法性能较稳定,与理论上的 $\Re_A = \lim_{N_{opt}\to\infty} N/N_{OPt}$ 比较,由于任务集中多数任务采用较小利用率,各算法 ρ 值分别低于最大利用率界 $\Re_A$。

5　结语

通过研究和实验分析总结出以下几点认识:(1)多处理器任务分配算法渐近性能界 $\Re_A = \lim_{N_{opt}\to\infty} N/N_{OPt}$ 用来度量算法性能好坏,目的是使用尽可能少的处理器可行的分配任务集,因此设计一种任务分配算法,要尽可能降低 $\Re_A$ 的值,尽可能计算紧密的 $\Re_A$;(2)多处理器可调度判定条件,用于分配算法本身在进行任务划分时的判定依据,是算法的核心,准确合理的可调度条件公式,可以提高算法对任务集划分的效率,提高任务可调度率;(3)分配算法的利用率界分析是分配算法分析的主要核心内容,用于对不同特征任务集选择不同分配算法进行任务划分(通过对任务集总利用率与算法利用率界的比较,判断使用该算法对任务集是否可以产生可行分配)。未来工作将包括考虑资源约束条件下的多处理器任务调度问题;第二,考虑开始时间抖动问题,提出降低抖动发生的方法;第三,多处理器联机任务分配相关算法研究。

参考文献:

[1] LEUNG JYT, WHITEHEAD J. On the complexity of fixed-priority scheduling of periodic, real-time tasks[J]. Performance Evaluation, 1982, 2(4): 237 – 250.

[2] LIU CL, LAYLAND JW. Scheduling algorithms for multiprogramming in a hard-real-time environment[J]. Journal of ACM, 1973, 20(1): 174 – 189.

[3] 王永吉,陈秋萍. 单调速率及其扩展算法的可调度性判定[J]. 软件学报 2004, 15(6): 799 – 814.

[4] OH Y, SON SH. Allocating fixed-priority periodic tasks on multiprocessor systems[J]. Real-Time Systems, 1995, 9(3): 207 – 239.

[5] BURCHARD A, LIEBEHERR J, OH Y, *et al*. New strategies for assigning real-time tasks to multiprocessor systems[J]. IEEE Transactions on Computers, 1995, 44(12).

[6] LEHOCZKY JP, SHA L, DING Y. The rate-monotonic scheduling algorithm: exact characterization and average behavior[A]. IEEE Real-Time Systems Symposium[C], 1989. 166 – 171.

[7] KUO TW, MOK AK. Load adjustment in adaptive real time systems [A]. Proceedings of IEEE Real Time Systems Symposium[C], 1991.

[8] LÖPEZ JM, DÍAZ JL, GARCÍA DF. Minimum and maximum utilization bounds for multiprocessor rate monotonic scheduling[J]. IEEE Transactions on Parallel and Distributed Systems, 2004, 15(7): 642 – 653.

[9] OH Y, SON SH. Preemptive scheduling of periodic tasks on multiprocessor: dynamic algorithms and their performance[R]. Technical Report No. CS-93-26. University Of Virginia. Dept. of Computer Science, 1993.

[10] DONG-IK OH, BAKER TP. Utilization bounds for n-processor rate monotone scheduling with stable processor assignment[J]. Real Time Systems Journal, 1998, 15(2): 183 – 193.