高利用率集合 Sporadic 实时任务调度方法研究

黄姝娟，肖 锋*，曹子建

(西安工业大学计算机与工程学院 西安 710021)

【摘要】该文提出一种基于最少迁移度和分割度的任务调度方法。该方法将各个实时周期任务分比例执行在不同处理器核上，并规定任务调度时的优先顺序，然后根据相应的实时调度流程对实时周期任务进行调度。并与已有的高利用率集合调度的准划分调度算法 EDF-os、EDF-fm 进行对比。结果表明该方法在保证系统利用率的同时，减少了任务分割和迁移的数量和不必要的任务切换开销。

关 键 词 嵌入式系统； 多核； 调度算法； 调度模型； 实时任务

中图分类号 TP319 文献标志码 A doi:10.12178/1001-0548.2020228

Research on Scheduling Method of High Utilization Rate Sets for Sporadic Real-Time Tasks

HUANG Shu-Juan, XIAO Feng*, and CAO Zi-jian

(School of Computer Science and Engineering, Xi'an Technological University Xi'an 710021)

Abstract This paper proposes a new scheduling algorithm which can reduce the unnecessary migration and context switching overhead. In this method, each real-time cycle task is proportionally executed on different processor cores, and the priority of task scheduling is specified, and then the real-time cycle tasks are scheduled according to the corresponding real-time scheduling process. By comparing with EDF-os and EDF-fm, which have been considered as set scheduling algorithms of high utilization rate, the experiments show this method not only can ensure high utilization rate but also reduce the times of migrating tasks and context switching overhead.

Key words embedded system; multi-core; scheduling algorithm; scheduling model; real-time tasks

当前嵌入式多核平台对具有严格时间限制的复杂应用提供了强有力的计算执行能力[1]。然而随着嵌入式系统复杂性的攀升，实时调度也更具挑战性[2]。在嵌入式多核平台下，大部分算法基于 Partitioned[3] 或 Global[4-5] 调度方法，近年来 Semi-partitioned[6-7] 调度方法逐渐获得大家的重视。该类调度算法吸取了前两者的优点，即在 Partitioned 调度算法的基础上允许少部分任务迁移，被建议用在支持暗含时限的软实时 Sporadic 任务系统中[8]。此后，还被应用于混合关键系统中[9-12]。然而，无论哪种应用，该类调度算法始终要求在一定的时限延迟的基础上进行讨论，而且任务迁移仅在 job 的边界上进行，否则系统开销太大。但随着当前多核能力的提高，嵌入式系统的集成度越来越高，导致高利用率的任务也越来越集中，系统运行的负荷增大，因此 Semi-partitioned 调度算法划分方案的精确性以及减少迁移和系统运行开销就成为研究热点。

目前针对高利率集合的 semi-partitioned 调度算法，如 EDF-fm[13] (earliest-deadline- first-based fixed and migrating) 和 EDF-os[14](earliest-deadline-first-based optimal semi-partitioned)，都存在迁移次数较多和上下文切换开销较大的问题。本文在这两种调度算法的基础上，提出一种基于最少迁移度和分割

收稿日期：2020－05－18；修回日期：2021－04－13

基金项目：陕西省科技厅自然科学基础研究计划 (No.2020JM-565)；民用飞机专项科研项目 (No. MJ-2015-D-066)；新型网络与检测控制国家地方联合工程实验室基金 (GSYSJ2017004)

作者简介：黄姝娟 (1975－)，女，博士，副教授，主要从事嵌入式与分布式计算方面的研究.

通信作者：肖锋，E-mail：xffriends@163.com

度 (earliest-deadline-first-based migrating and splitting tasks least, EDF-MSTL) 的调度方法，旨在提高系统效率的同时，减少分割任务的数量和不必要的迁移和任务切换开销。

1　Sporadic 实时系统任务调度模型

假设一个实时系统τ由 n 个周期性任务组成，记为$\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$。每个周期任务都包含 4 个参数。即$\tau_i(r_i,e_i,p_i,d_i)(1\leqslant i\leqslant n)$，其中$r_i$表示发布时刻 (release time)，即处理器核响应的时刻，e_i表示任务T_i最坏情况下的执行时间 (worst-case execution time, WCET），p_i是τ_i的周期时间，d_i表示时限 (deadline)，$e_i\leqslant d_i\leqslant p_i$。若$d_i=p_i$，称该系统为隐含时限系统 (implicit deadlines)，将其任务称为隐含时限任务。如果$d_i<p_i$，称该系统为包含时限系统 (constrained deadline system)，将其任务称为包含时限任务。如果两者没有强制性的约束条件则称为任意时限系统 (arbitrary deadline system)。用$\tau_i(r_i,e_i,p_i,d_i)$表示包含或任意时限系统，$\tau_i(r_i,e_i,p_i)$表示暗含时限系统。τ_i的第$j(j\geqslant 1)$个 job，用$J_{i,j}$表示。一个任务的第一个 job 可以在任何时刻被发布。将$J_{i,j}$的发布时间记为$r_{i,j}$，将其绝对时限$d_{i,j}$定义为$r_{i,j}+d_i$，$J_{i,j}$的执行时间记为$e_{i,j}$。

Sporadic 任务模型指n个重复发生的任务$\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$，每个$\tau_i$具有 3 个参数特征：$e_i(e_i\geqslant 0)$指示 WCET；周期$p_i>e_i$，指示一个$\tau_i$连续两次 job 之间的最小间隔。时限$d_i\geqslant e_i$，指示$\tau_i$的每个 job 在它发布之后到其完成时的最大时间间隔。τ_i是顺序执行的，在同一个时间只有一个 job 在执行。Sporadic 任务模型与周期模型不同之处在于两个连续 job 发布的时间间隔不固定，其中最小的时间间隔成为该任务的周期。周期型任务模型可以视为 Sporadic 任务模型的一个特殊情况，即该任务中连续 job 发布是由固定p_i时间单元分割的。本文着重讨论隐含时限的周期任务和 Sporadic 任务都存在的系统。

多核模型：$P=\{P_1,P_2,\cdots,P_M\}$为含有 M 个具有相同处理能力的处理器集合。在某个时间段内，分配到某个处理器P_m上的任务τ_i的第j次 job 被激活，那么在该处理器上执行该任务的第j次 job 的时间片段记为$T_{i,j,m}$。

由以上可知，实时系统中的周期任务具有 4 个重要属性：发布时间r_i、时限d_i、最坏执行时间e_i和周期p_i。

2　相关定义、定理和结论

定义 1　对于一个隐含时限的实时系统τ，如果该系统的所有任务都从 0 时刻发布，那么该任务可以表示为$\tau_i(e_i,p_i)$。该任务的利用率因子u_i定义为$U(\tau_i)=u_i=\dfrac{e_i}{p_i}$，该系统任务的总利用率$U(\tau)\overset{\text{def}}{=}\sum_{\tau_i\in\tau}U(\tau_i)$。

定义 2　对于任一周期任务τ_i，如果$U(\tau_i)\geqslant\dfrac{1}{2}$，则称该周期任务为高利用率任务集合，用 S^H 表示；如果$U(\tau_i)<\dfrac{1}{2}$，则称该周期任务为低利用率任务集合，用 S^L 表示。显然，任何一个周期任务要么属于 S^H 要么属于 S^L[15]。

定义 3　在一个具有 n 个实时周期任务$\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$的系统中，当所有实时周期任务都属于$S^H$，即$U(\tau_i)\geqslant\dfrac{1}{2}$，$i\in\{1,2,\cdots,n\}$，则称该系统为高利用率任务系统，即$S^H$系统。

定义 4　假定某个处理器核P_m，已经分配了实时周期任务$\{\tau_1,\tau_2,\cdots,\tau_k\}$且$\sum_{i=1}^{k}U(\tau_i)\leqslant 1$，则定义该处理器核所能继续分配任务的剩余利用率因子为$U_{\text{res}}=1-\sum_{i=1}^{k}U(\tau_i)$。

定理 1　对于高利用率任务系统，如果该系统所有任务利用率因子之和为处理器数量 M，则该系统可以被调度的必要条件是一定存在需要迁移的任务。

证明：假设一个S^H系统中有 n 个实时周期任务$\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$，该系统有 M 个处理器，且系统任务利用率总和$U(\tau)=\sum_{\tau_i\in\tau}U(\tau_i)=M$。因为该系统为高利用率系统，其中任何一个任务$\tau_i$的利用率因子均满足$1>U(\tau_i)>\dfrac{1}{2}$，那么会出现该任务和任何一个任务的利用率因子之和均大于 1 的情况，这种情况下，如果该任务能够被系统调度，则该任务只能独享或和多个任务共享一个处理器，否则会出现丢失时限现象。如果独享一个处理器，就会出现其他任务的利用率之和大于 M–1 的情况，即$U(\tau)=\sum_{\tau_j\in\tau(j\neq i)}U(\tau_j)>M-1$，使得其他任务组成的系统成为无法调度的系统。因此，该系统如果能够被调度，就必须让该任务和其他任务共享多个处理器，即为迁移任务。

定义 5 如果存在高利用率系统S^H，在某种调度算法 S 下，定义该系统迁移度migrat_λ_s为所有任务需要迁移的次数之和与所有任务利用率因子总和的比值。迁移度越小说明需要迁移的次数也越少，系统迁移开销也就越少。定义该系统任务的分割度split_λ_s为被分割的任务数量与系统任务总数量的比值。任务分割度越少，说明需要被迁移的任务数量越少，系统的执行效率就越好。

3 高利用率任务集合

将具有 6 个暗含时限的实时周期任务$\tau_1(4,6)$、$\tau_2(2,3)$、$\tau_3(5,6)$、$\tau_4(2,3)$、$\tau_5(1,2)$、$\tau_6(2,3)$分配到 4 个处理器上，从 0 时刻发布，按照 GEDF 调度方法会得到调度图序列如图 1 所示。可以看出，τ_2和τ_4在 3 个处理器上来回迁移，τ_5也在两个处理器上迁移，τ_3和τ_6仍然有丢失时限发生。可以看出这种调度迁移开销太大且仍有时限丢失现象。而 semi-partitioned 算法将调度过程分为两个阶段：离线分配阶段和执行阶段。在分配阶段只允许少部分任务为迁移任务，其他任务不允许迁移。EDF-fm 算法、EDF-os 算法和 EDF-MSTL 算法区别在于离线分配阶段：EDF-fm 算法类似深度优先分配，即将一个处理器份额全部占用完之后再分配下一个处理器，如图 2 所示，所以任务最多在两个处理器上迁移。EDF-os 算法类似广度优先遍历，先将任务按照利用率从高到低降序排列，然后将和处理器数量相同的任务依次分配到处理器上作为非迁移任务，再从第一个处理器上依次分配该处理器的剩余份额，如图 3 所示。

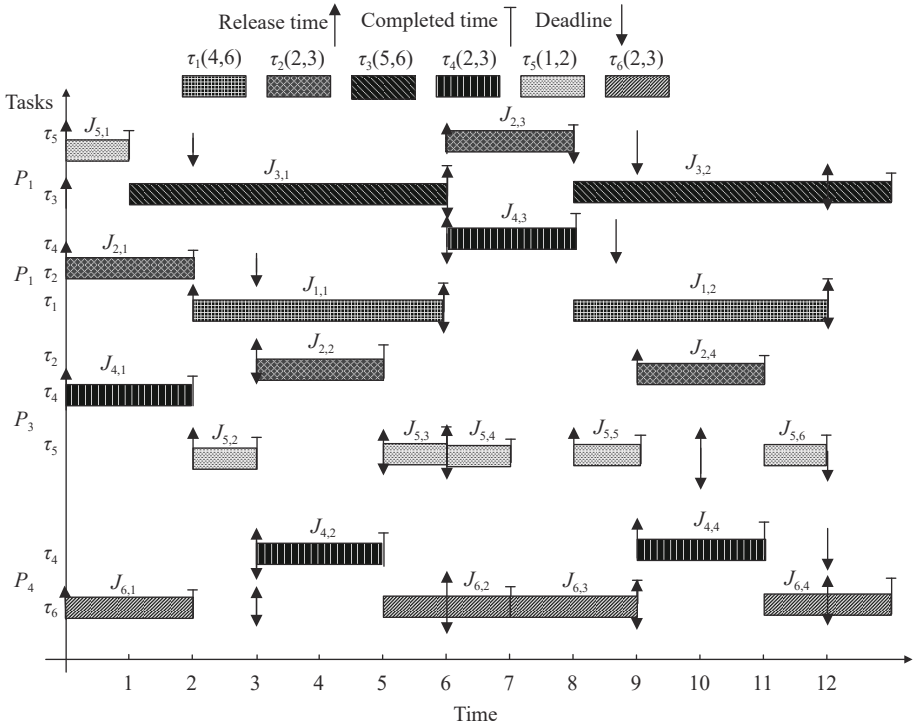

图 1 GEDF 算法执行 12 个时间片结果

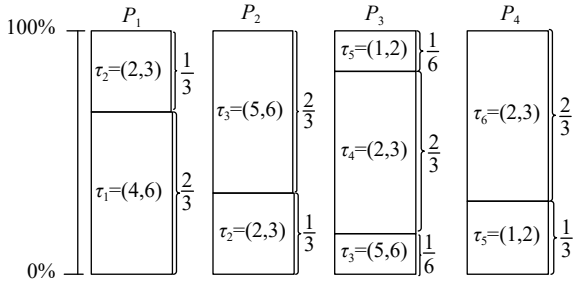

图 2 EDF-fm 算法任务分配的份额

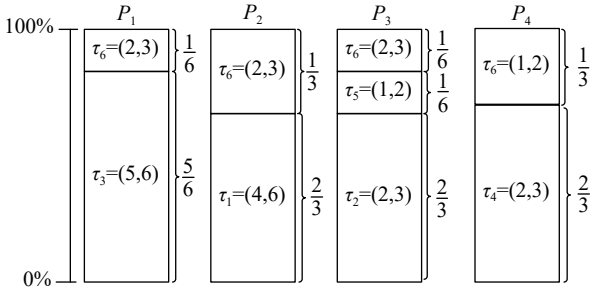

图 3 EDF-os 算法任务分配的份额

这两种算法中的迁移任务都会按照一定的比例将不同数量的 job 分配到指定的处理器上，比例计算方法为：

$$f_{i,j}=\frac{s_{i,j}}{u_i} \tag{1}$$

式中，$f_{i,j}$为第 i 个任务分配到第 j 个处理上 job 数量的比例；$s_{i,j}$为该第 i 个任务在第 j 个处理器上获得的份额。本例中 EDF-fm 算法分配给任务对应的份额矩阵为：

$$\boldsymbol{s}=\begin{bmatrix} \frac{2}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & \frac{2}{3} & \frac{1}{6} & 0 \\ 0 & 0 & \frac{2}{3} & 0 \\ 0 & 0 & \frac{1}{6} & \frac{1}{3} \\ 0 & 0 & 0 & \frac{2}{3} \end{bmatrix}$$

EDF-fm 算法任务对应的 job 比例矩阵为：

$$\boldsymbol{f}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{4}{5} & \frac{1}{5} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{3} & \frac{2}{3} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

从这里可以看到$\tau_2(2,3)$被分配到处理器 P_1 和 P_2 上，分配的 job 比例为 1∶1；那么调度时就会将奇数项 job 配到 P_1，偶数项 job 分配到 P_2。$\tau_3(5,6)$在 P_2 和 P_3 分配的 job 的比例为 4∶1，那么 5 个 job 中就有一个 job 被分配到 P_3 处理器上。$\tau_5(1,2)$在 P_3 和 P_4 处理器上 job 分配的比例为 1∶2。同理，可以得到 EDF-os 算法的分配份额和任务 job 的执行比例。

这两种算法在执行阶段，迁移任务要比非迁移任务优先级高，而当某处理器执行两个迁移任务时，该处理器为非第一次分配处理器的迁移任务优先级最高。

EDF-os 算法分配给任务对应的份额矩阵为：

$$\boldsymbol{s}=\begin{bmatrix} 0 & \frac{2}{3} & 0 & 0 \\ 0 & 0 & \frac{2}{3} & 0 \\ \frac{5}{6} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{2}{3} \\ 0 & 0 & \frac{1}{6} & \frac{1}{3} \\ \frac{1}{6} & \frac{1}{3} & \frac{1}{6} & 0 \end{bmatrix}$$

EDF-os 算法任务对应的 job 比例矩阵为：

$$\boldsymbol{f}=\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{1}{3} & \frac{2}{3} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & 0 \end{bmatrix}$$

EDF-fm 和 EDF-os 具体执行结果如图 4 和图 5 所示。

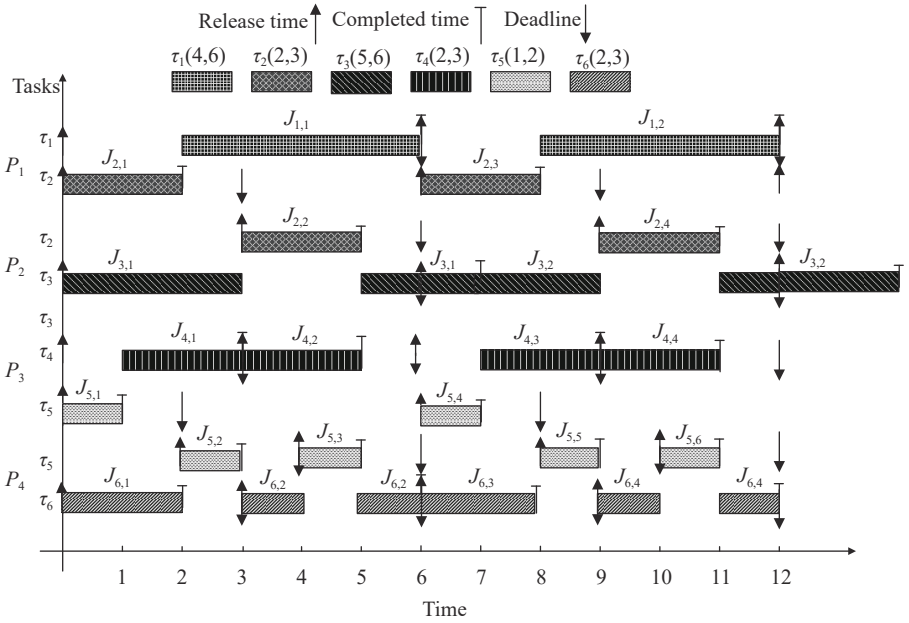

图 4　EDF-fm 算法执行 12 个时间片结果

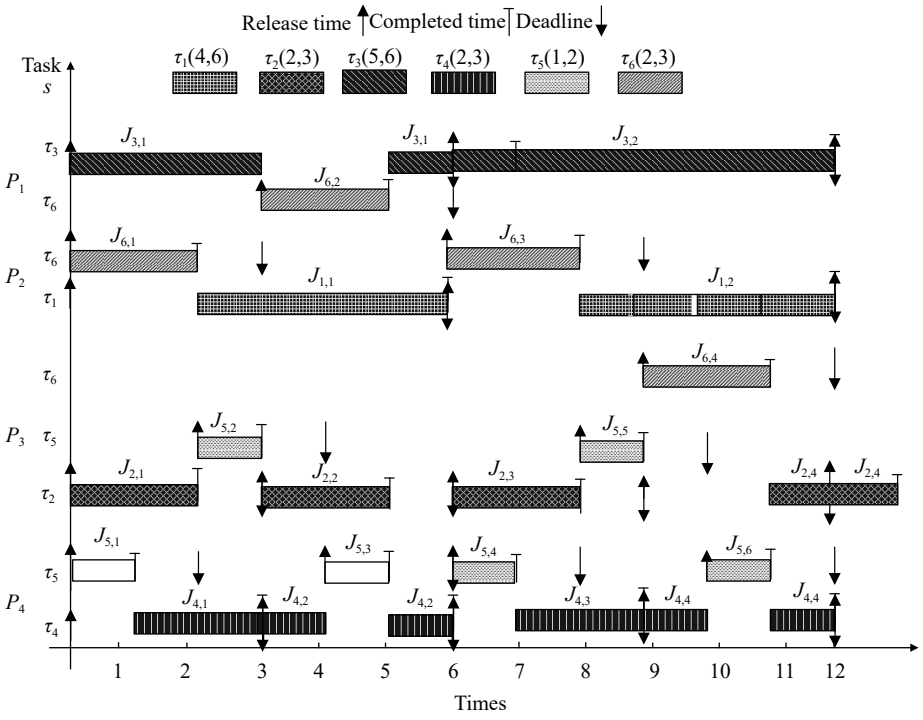

图 5 EDF-os 算法执行 12 个时间片结果

EDF-fm 算法被分割的任务为τ_2、τ_3、τ_5，这 3 个任务分别被迁移了 1 次，故该调度算法下的迁移度 $\text{migrat\_}\lambda_s=\frac{3}{4}=0.75$，任务分割度 $\text{split\_}\lambda_s=\frac{3}{6}=0.5$。

EDF-os 算法被分割的任务为τ_5、τ_6，这两个任务分别被迁移了 2 次和 1 次，故该调度算法下的迁移度 $\text{migrat\_}\lambda_s=\frac{3}{4}=0.75$，任务分割度 $\text{split\_}\lambda_s=\frac{2}{6}=\frac{1}{3}$。

EDF-MSTL 算法分配给任务对应的份额矩阵为：

$$\boldsymbol{s}=\begin{bmatrix} \frac{2}{3} & 0 & 0 & 0 \\ 0 & \frac{2}{3} & 0 & 0 \\ 0 & 0 & \frac{5}{6} & 0 \\ 0 & 0 & 0 & \frac{2}{3} \\ \frac{1}{3} & 0 & \frac{1}{6} & 0 \\ 0 & \frac{1}{3} & 0 & \frac{1}{3} \end{bmatrix}$$

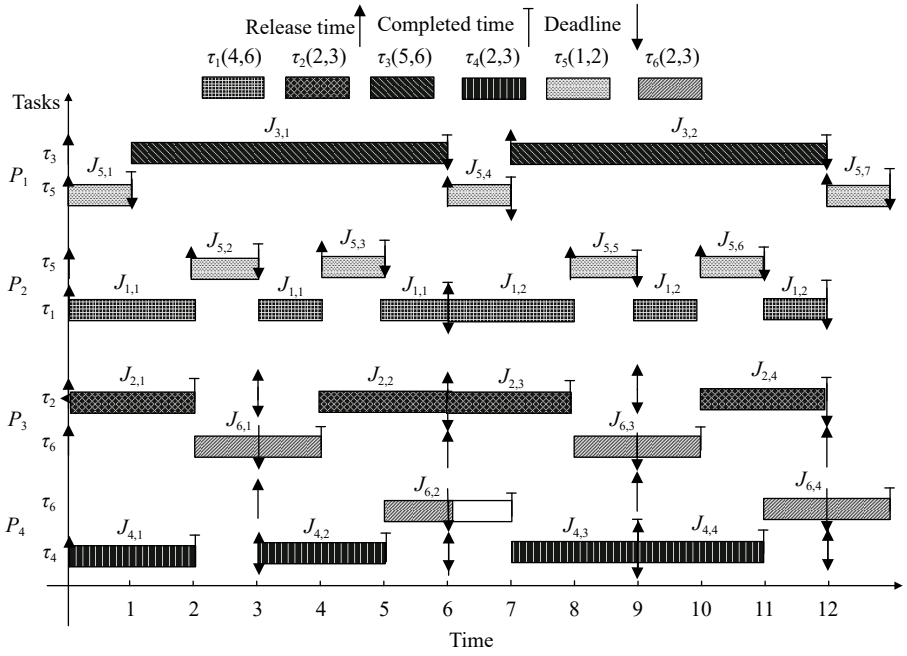

图 6 EDF-MSTL 算法执行 12 个时间片结果

EDF-MSTL 算法任务对应的 job 比例矩阵为：

$$f=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\\\frac{2}{3}&0&\frac{1}{3}&0\\0&\frac{1}{2}&0&\frac{1}{2}\end{bmatrix}$$

对应的该调度算法下的迁移度$\text{migrat}_\lambda_s=\frac{2}{4}=0.5$，任务分割度$\text{split}_\lambda_s=\frac{2}{6}=\frac{1}{3}$。系统总体的迁移度和任务分割度是最少的，调度结果如图 6 所示。

4　算法实现

EDF-MSTL 调度方法描述如下：

1) 首先将 n 个任务的利用率因子按照从大到小顺序排列放入集合S^H，m 个处理器顺序放入集合 P 中，初始化矩阵$\boldsymbol{s}_{n,n}$，$s_{i,i}=u_i$，其他值为 0。

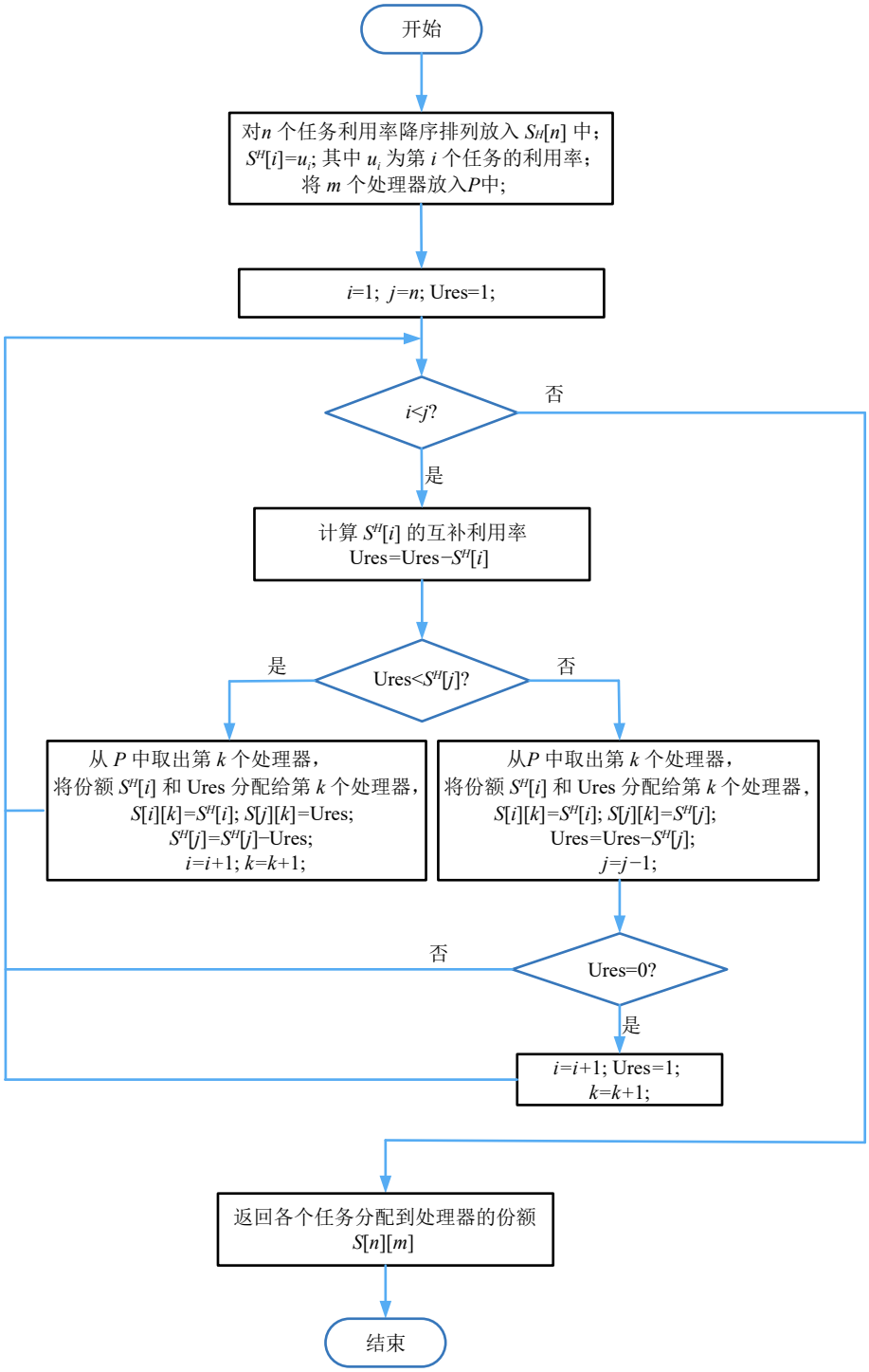

图 7　EDF-MSTL 计算任务份额的流程图

2) 找到具有最小利用率因子的任务u_n，将其拆分为两部分$u_n = u_i + u_j$，其中u_i与第一个具有最大利用率因子任务的利用率之和为 1，即$u_i + u_1 = 1$，设置$s_{n,k} = u_i$，$s_{n,n} = u_j$，并将这两个利用率因子从集合S^H中删除，如果u_j为 0，则也将其从S^H中删除，将处理器 k 从集合 P 中删除。

3) 在剩下的S^H集合中继续重复步骤 1) 的工作，直到处理器集合为空为止。得到$s_{n,m}$即为所分配份额。根据该份额通过式 (1) 计算$f_{n,m}$。利用$f_{n,m}$中的值，在调度执行过程中进行按比例分配相应的任务 job，可以得到相依的执行序列。具体计算任务份额的算法流程图如图 7 所示。

5 实验结果

本文测试的环境是在一个 Intel®Core™2 Quad Q8400 多核平台上。分别采用 EDF-fm，EDF-os，EDF-MSTL 这 3 种调度方法进行比较。测试方法随机产生 1 000 个任务集，每个任务集中产生 50 个参数不等的 Sporadic 软实时周期任务，所有周期任务都满足利用率大于 0.5，执行时间小于时限，时限小于或等于周期。在整个仿真实验过程中，为了描述算法之间的性能差异，采用多次模拟求平均值的方法得到某时间段内的系统吞吐率以及任务切换 job 数量所占总 job 数的比例，如表 1 所示。

表 1　3 种算法的系统利用率和丢失时限任务数所占总任务的比例

随机抽取的任务集	系统吞吐率			任务切换job数量占总job数的比例/%		
	EDF-fm 算法	EDF-os 算法	EDF-MSTL 算法	EDF-fm 算法	EDF-os 算法	EDF-MSTL 算法
1	0.899	0.921	0.942	12.21	12.20	11.62
2	0.878	0.903	0.948	12.41	12.39	11.85
3	0.881	0.912	0.941	12.33	12.31	11.80
4	0.892	0.917	0.940	12.25	12.26	11.69
5	0.883	0.913	0.939	12.31	12.30	11.79
6	0.865	0.901	0.937	12.68	12.66	11.88
7	0.887	0.916	0.943	12.28	12.25	11.77
8	0.903	0.921	0.944	12.03	12.03	11.60
9	0.894	0.918	0.941	12.22	12.21	11.65
10	0.879	0.910	0.939	12.38	12.35	11.83

另外，选择对应这 10 个任务集的平均任务迁移度和任务分割度两个方面进行性能对比。从表 1、图 8 和图 9 可以看出，EDF-算法和 EDF-MSTL 算法在系统吞吐率、任务切换 job 数、任务迁移度和任务分割度方面，后者算法明显占据优势。

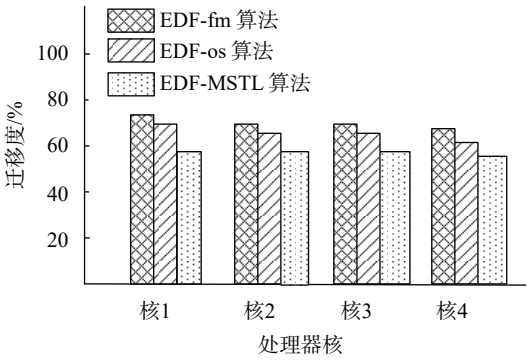

图 8　3 种算法任务迁移度

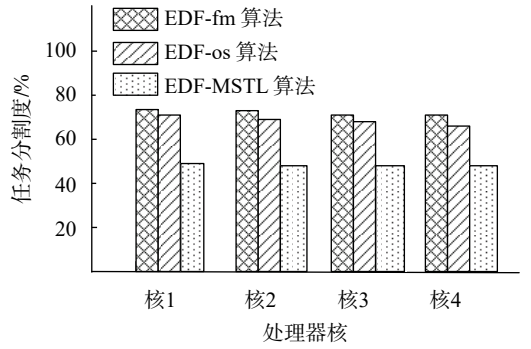

图 9　3 种算法的任务分割度

6 结 束 语

本文的目的是为了解决在嵌入式多核平台下，任务全部属于高利用率因子集合的软实时系统中的实时周期任务的调度问题。在 EDF-fm 和 EDF-os 算法的基础上，提供一种基于最少迁移度的多核实时调度方法，减少了现有技术中存在由于迁移带来的过多开销以及上下文切换次数，在任务量很大的情况下可以大大提高系统的整体效率。

参 考 文 献

[1] CLÁUDIO M, PATRICK M Y, LUÍS N, et al. Real-time semi-partitioned scheduling of fork-join tasks using work-stealing[J]. EURASIP Journal on Embedded Systems, 2017, 1(31): 1-14.

[2] MAGDICH A, KACEM Y H, MICKAEL K, et al. A design pattern-based approach for automatic choice of semi-partitioned and global scheduling algorithms[J]. Information and Software Technology, 2018, 97: 83-98.

[3] HAN Jian-jun, CAI Wen, ZHU Da-kai. Resource-aware partitioned scheduling for heterogeneous multicore real-time systems[C]//Proceedings of the 2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC). San Francisco: IEEE, 2018: 1-6.

[4] GANG Y, PELLIZZONI R, BAK S. Global real-time memory-centric scheduling for multicore systems[J]. IEEE Transactions on Computers, 2016, 65(9): 2739-2751.

[5] XIAO J, ALTMEYER S, PIMENTEL A D. Schedulability

analysis of global scheduling for multicore systems with shared caches[J]. IEEE Transions on Computers, 2020, DOI: 10.1109/TC.2020.2974224.

[6] VORONOV S, ANDERSON J H. An optimal semi-partitioned scheduler assuming arbitrary affinity masks[C]// Processing of 2018 IEEE Real-Time Systems Symposium (2018RTSS). [S.l.]: IEEE, 2018: 408-420.

[7] BONIFACI V, D 'ANGELO G, MARCHETTI-SPACCAMELA A. Algorithms for hierarchical and semi-partitioned parallel scheduling[C]//Processing of 2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). Orlando, USA: IEEE, 2017.

[8] ANDERSON J H, ERICKSON J P, DEVI U C, et al. Optimal semi-partitioned scheduling in soft real-time systems[J]. Journal of Signal Processing Systems, 2016, 84(1): 3-23.

[9] ZENG Li-ning, LEI Yuan, LI Yan-xing. A semi-partition algorithm for mixed-criticality tasks in multiprocessor platform[C]//Proceedings of the 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS). Beijing: IEEE, 2019: 694-697.

[10] RAMANATHAN S, EASWARAN A. Mixed-criticality scheduling on multiprocessors with service guarantees[C]// Processing of the 21st IEEE International Symposium on Real-Time Distributed Computing in Singapore. [S.l.]: IEEE, 2018: 29-31.

[11] XU H, BURNS A. A semi-partitioned model for mixed criticality systems[J]. Journal of Systems and Software, 2019, 150(4): 51-63.

[12] NAIR P P, DEVARAJ R, SEN A, et al. DES based modeling and fault diagnosis in safety-critical semi-partitioned real-time systems[J]. International Federation of Automatic Control, 2017, 1(50): 5029-5034.

[13] ANDERSON J H, ERICKSON J P, DEVI U C, et al. Optimal semi-partitioned scheduling in soft real-time systems[C]//Proceedings of the 2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications. Chongqing: IEEE, 2014: 1-10.

[14] SANATI B, CHENG A M K. Poster abstract: Online semi-partitioned multiprocessor scheduling of soft real-time periodic tasks for QOS optimization[C]//Proceedings of the 2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS). Vienna: IEEE, 2016: 1-14.

[15] 黄姝娟, 朱怡安, 李兵哲, 等. 基于利用率和负载均衡的多核实时调度算法研究[J]. 西北工业大学学报, 2012, 30(1): 117-123.
HUANG Shu-juan, ZHU Yi-an, LI Bing-zhe, et al. An efficient BUWBP (based on utilization and workload balance partition) algorithm for multiprocessor real-time system[J]. Journal of Northwestern Polytechnical University, 2012, 30(1): 117-123.

编　辑　叶　芳