一种分布式实时嵌入式系统的调度分析算法

张海涛[1,2]，艾云峰[2]

(1. 河南科技大学电子信息工程学院 河南 洛阳 471003; 2. 中国科学院自动化研究所 北京 海淀区 100080)

【摘要】针对现有的基于Petri网的调度分析算法存在的不足，提出了一种适合于分布式实时嵌入式系统的调度分析算法。该算法使用相对的触发域判定调度序列中的变迁是否可以调度；通过引入并行间隔，记录了调度序列中的并行变迁的相对执行时间；在计算调度长度时，使用并行间隔作为并行变迁的执行时间，从而得到了正确的调度分析结果。

关 键 词 嵌入式系统; 建模; Petri网; 调度
中图分类号 TP311 文献标识码 A

An Algorithm of Scheduling Analysis for Distributed Real-Time Embedded Systems

ZHANG Hai-tao[1,2], AI Yun-feng[2]

(1. Electronic Information Engineering College, Henan University of Science and Technology Luoyang Henan 471003)

2. Institute of Automation, Chinese Academy of Sciences Haidian Beijing 100080)

Abstract Aiming at the deficiencies of existing scheduling analysis algorithms, we present a scheduling analysis algorithm for distributed real-time embedded systems. The algorithm uses relative firing domain to decide whether the transitions of scheduling sequence are scheduled. The relative executing time of parallel transition is recorded by introducing parallel space. The correct results of scheduling analysis can be obtained by taking space as executing time of parallel transition.

Key words embedded system; modeling; Petri nets; scheduling

在设计复杂的分布式嵌入式系统时，使用形式语言建模是非常必要的。通过对形式模型的有效化和校验，可以尽早发现许多设计中的问题，避免在设计软硬件的最后阶段才发现问题所带来的经济和时间上的损失。Petri 网具有极强的建模能力，可以描述并行、同步、冲突、异步等各种事件[1]。目前，Petri 网已经被应用于嵌入式系统的建模、评估和分析[2-3]。

在嵌入式实时系统中，调度起着关键的作用。目前，大多数使用情况下，Petri 网建模调度的方法主要适用于单处理器[4-6]，但基于资源的 Petri 网(Resources Based Time Petri Nets，RBTPN)[7]通过在表示计算和通信任务的变迁上附着所占用的处理器或通信资源，以及相应的优先级，可以直观地建模分布式嵌入式系统的实时调度。

然而，由于分布式嵌入式系统的调度问题是一个 NP 完全问题，必须进行调度分析。而在 RBTPN 中，现有的适用于单处理器的调度分析算法存在着明显的问题，在其被应用到分布式嵌入式系统后，无法计算出正确的调度长度。本文详细分析了现有调度分析算法的不足，提出了新的适宜分布式实时嵌入式系统的调度分析算法。该调度分析算法通过引入并行间隔，不改变原有的状态类，可以得到正确的调度长度。

1 基于资源的时间Petri网

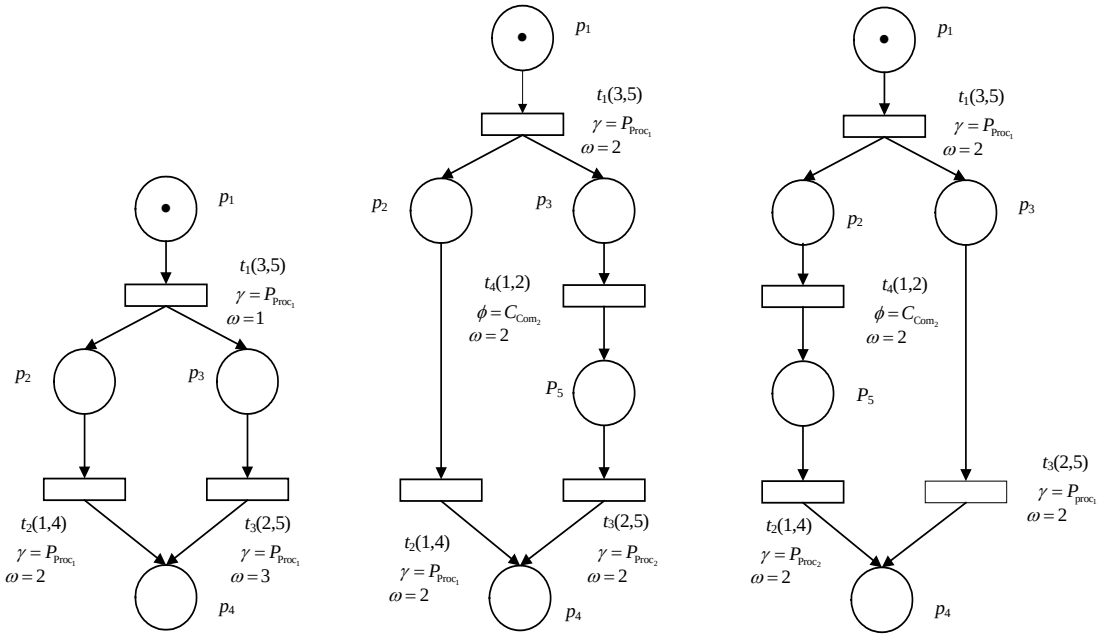
基于资源的时间Petri网是由多个元素描述的有向图 $G_{\mathrm{RBTPN}}=\{P,T,I,O,M_0,R_{\mathrm{SR}}\}$，其中 $P=\{p_1,p_2,\cdots,p_n\}$ 是位置的有限集合，$n>0$ 为位置的个数；$T=\{t_1,t_2,\cdots,t_m\}$ 是变迁的有限集合，$m>0$ 为变迁的个数，且 $P\bigcap T=\varnothing$；$I:P\times T\to N$ 是输入函数，定义了从 P 到 T 的有向弧的权的集合；$O:T\times P\to N$

收稿日期: 2006 – 05 – 09
基金项目: 国家自然科学基金资助项目(60334020)
作者简介: 张海涛(1972 –), 男, 博士, 主要从事智能控制和嵌入式系统方面的研究.

是输出函数，定义了从 T 到 P 的有向弧的权的集合；$M_0: P \to N$ 是最初的标识函数，定义了初始时刻、每个位置中托肯的数目；调度资源 R_{SR} 主要包括以下参数：

(1) $P_{\mathrm{proc}} = \{p_{\mathrm{proc}_1}, p_{\mathrm{proc}_2}, \cdots, p_{\mathrm{proc}_l}\}$ 是有限的处理器集合，$l > 0$ 是微处理器的个数。该资源可以抢先。(2) $C_{\mathrm{com}} = \{c_{\mathrm{com}_1}, c_{\mathrm{com}_2}, \cdots, c_{\mathrm{com}_t}\}$ 是有限的通信资源集合，$t > 0$ 是通信资源的个数。该资源不可抢先。(3) $\omega: T \to N$、$\gamma: T \to P_{\mathrm{proc}}$ 和 $\varphi: T \to C_{\mathrm{com}}$ 分别为优先级、处理器和通信资源分配函数。

此外，RBTPN 模型有以下定义：(1) 每个变迁 $t_i \in T$，具有相关联的静态时间间隔 $S_{\mathrm{STI}}(t_i) = (\alpha^s(t_i), \beta^s(t_i))$ 和动态的时间间隔 $(\alpha(t_i), \beta(t_i))$。这些都是相对的时间间隔，如果变迁 t_i 在 τ 时刻使能，那么 t_i 仅仅可以在间隔 $(\tau + \alpha_i^s, \tau + \beta_i^s)$ 或 $(\tau + \alpha_i, \tau + \beta_i)$ 触发。(2) RBTPN 的状态类记作 $C = (M, D)$，其中 $M: P \to N$ 是状态类的标识，且 M 是一个向量，表示每个位置所含的托肯数；D 是所有使能的变迁的触发域，表示这些变迁在标识 M 下可能的触发时间。(3)如果 $\forall p \in {}^* t_i : M(p) \geq I(p,t)$，$\forall p \in t_i^* : M(p) + O(t,p) \leq C$，那么变迁 $t_i \in T$ 在状态类 C 下称作使能的。如果 t_i 请求的资源没有被另一个具有较高优先级的使能的变迁所请求，那么 $t_i \in T$ 称作运行变迁；使能但没有正在进行的变迁称作就绪变迁。以上变迁的集合分别记作 $E_{\mathrm{enabled}}(C)$、$W_{\mathrm{working}}(C)$ 和 $H_{\mathrm{halting}}(C)$。



a. 单处理器的RBTPN模型　　b. 两个处理器的RBTPN模型1　　c. 两个处理器的RBTPN模型2

图1　一个RBTNP模型例子

图1a所示为一个单处理器系统的 RBTPN 模型。t_1、t_2和t_3 工作于处理器 p_{proc_1} 上，优先级分别为1、2和3。图1b和图1c所示为两个处理器系统的 RBTPN 模型。模型中有处理器资源 p_{proc_1}、p_{proc_2} 和通信资源。

2　调度分析

假定模型的状态类 $C_i = (M_i, D_i)$，从模型状态类的定义可知，$D_i(t) = (\alpha_i(t), \beta(t))$ 是相对的触发域，表示变迁相对时间基准的触发间隔。

对于某个调度序列 $t_1 t_2 \cdots t_n$，即 $C_0 t_1 C_1 t_2 \cdots t_n C_n$，其调度分析包括两个步骤：(1) 确定在某个状态类 C_i 下，变迁 t_{i+1} 是否是可以调度的；(2) 计算调度序列的调度长度，确定其是否满足任务的时限。

2.1　现有的算法

在状态类 $C_i = (M_i, D_i)$ 下，对于任意变迁 $t_j \in W_{\mathrm{working}}(C_i)$，其调度间隔记为 $(S_{\mathrm{SE}_i}, S_{\mathrm{SL}_i})$，该值为相对值，相对于进入该状态类的时间。对于某个触发序列 $(C_0 t_1 C_1 t_2 \cdots t_n C_n)$，可以从 C_0 开始，使用触发域 D_i 检验变迁 $t_{i+1} (0 \leq i \leq n-1)$ 是否是可以调度的，使用调度间隔 $(S_{\mathrm{SE}_i}, S_{\mathrm{SL}_i})$ 计算调度长度[8]。该算法主要包括两个步骤：

(1) 如果变迁 $t_{i+1} \in W_{\mathrm{working}}(C_i)$，且 $\alpha_i(t_{i+1}) \leq$

$\min\{\beta_i(t_j)$， $t_j \in W_{\text{working}}(C_i)\}$，那么变迁 t_{i+1} 在状态类 C_i 下是可以调度的。

(2) 某个变迁 t_{i+1} 触发后,计算其相应的调度间隔 $(S_{\text{SE}_{i+1}}, S_{\text{SL}_{i+1}}) = (\alpha_i(t_i), \min\{\beta_i(t_j): t_j \in W_{\text{working}}(C_i)\})$。

在调度序列中，如果有一个变迁不可调度，那么整个调度序列就是不可调度的。如果该调度序列是可调度的，那么调度时间 $t_{\text{sl}}(\rho) = \sum_{i=1}^{n}(S_{\text{SE}_i}, S_{\text{SL}_i})$。

上面的调度分析算法在处理单处理器时，可以得到完全正确的结果。然而，在分布式嵌入式系统中，由于多个处理器可以并行地执行任务，使多个变迁可以同时获得不同的资源而同时运行，从而导致该算法无法计算出正确的调度长度，也就无法得到正确的调度可行性结果。该算法的主要错误在于调度间隔 $(S_{\text{SE}_{i+1}}, S_{\text{SL}_{i+1}})$ 是相对于状态类 C_i 的，导致所计算出的调度长度总是大于实际的调度长度。

为此，另外一种调度分析算法[9]被提出。该算法主要是引入了状态类 C_i 的基准时间 $\tau_i(*) = (N_{NE_i}, N_{NL_i})$ 和相对于 $\tau_0(*)$ 的触发域 ND_i。ND_i 是在状态类 C_i 下所有使能变迁的新触发域，$N\alpha_i(t)$ 和 $N\beta_i(t)$ 分别称作相对于状态类 C_0 的最早和最晚触发时间。在状态类 ND_i 下，仍然使用触发域 D_i 确定变迁 t_i 是否可以触发，而使用相对于 $\tau_0(*)$ 的调度间隔 $(N_{\text{NE}_i}, N_{\text{NL}_i})$ 计算调度长度。该算法主要包括以下两个步骤：

(1) 如果变迁 $t_{i+1} \in W_{\text{working}}(C_i)$，且 $\alpha_i(t_{i+1}) \le \min\{\beta_i(t_j)$，$t_j \in W_{\text{working}}(C_i)\}$，那么变迁 t_{i+1} 在状态类 C_i 下是可以调度的。

(2) 当某个变迁 t_{i+1} 被触发后，计算其相应的调度间隔：

$$(N_{\text{NE}}(t_{i+1}), N_{\text{NL}}(t_{i+1})) = (N\alpha_i(t_{i+1}) \quad \min\{N\beta_i(t_j): t_j \in W_{\text{working}}(C_i)\})$$

$$N\alpha_i(t_{i+1}, \tau_{i+1}(*)) = (N_{\text{NE}}(t_{i+1}), N_{\text{NL}}(t_{i+1}))$$

如果所有的变迁都是可以调度的，就可得调度时间 $t_{\text{sl}}(\rho) = \tau_n(*)$。

2.2　分析算法存在的问题

引入新的触发域后，可以很好地解决分布式嵌入式系统的调度分析。然而，在RBTPN模型中，状态类的触发域计算是非常复杂的。而该调度分析算法中，不仅需要计算原有的触发域 D_i，而且需要计算新的相对于 $\tau_0(*)$ 的触发域 ND_i。

在该调度分析算法中，两个触发域是不能互相代替的。前面已经说明由于使用触发域D_i进行调度分析存在着问题，才引入了触发域ND_i计算调度长度。而在RBTPN模型中，变迁的执行时间是一个时间段,导致使用触发域ND_i判断变迁是否可以调度也会产生错误，就使新的调度分析算法的运算复杂程度加倍了。

因此，下面通过引入并行间隔，仅仅使用触发域 D_i，在减小计算复杂度的同时，也可以得到正确的调度长度。

2.3　增加并行间隔

为了解决分布式系统中存在的问题，可以引入并行间隔，改进常规算法。该算法仍然使用相对的触发域，识别并处理调度序列中的并行变迁。该算法如下。

3.3.1　调度分析

调度序列 $\rho = C_0 t_1 C_1 t_2 \cdots t_n C_n$ 中，调度分析算法步骤如下：

1) 设定初始值变量i=0。

2) 如果变迁 $t_{i+1} \in W_{\text{working}}(C_i)$，继续步骤 3)。否则，$t_{i+1}$ 是不可以调度的，退出校验。

3) 如果 $\alpha_i(t_{i+1}) \le \min\{\beta_i(t_j), t_j \in W_{\text{working}}(C_i)\}$，则 t_{i+1} 在状态类 C_i 下，在下面的间隔是可以调度的：$(S_{\text{SE}_{i+1}}, S_{\text{SL}_{i+1}}) = (\alpha_i(t_i), \min\{\beta_i(t_j): t_j \in W_{\text{working}}(C_i)\})$，否则，$t_{i+1}$ 是不可以调度的，退出校验。

4) 对所有其他运行变迁 $t \in W_{\text{working}}(C_i) \cap t \ne t_{i+1}$，计算并行间隔：$(P_{\text{PE}_{i+1}}(t), P_{\text{PL}_{i+1}}(t)) = (\max\{(P_{\text{PE}_i}(t) - S_{\text{SE}_{i+1}}), 0\}$，$P_{\text{PL}_i}(t) - S_{\text{SL}_{i+1}})$，并且将该变迁加入并行变迁集，即 $t \in P_{parallel}(\rho)$。

5) 构建新的相对触发域 D_{i+1}，主要是针对以下几种变迁：

(1) 对变迁 $t_j \in W_{\text{working}}(C_i) \cap t_j \in E_{\text{enabled}}(C_{i+1})$，触发域作如下改变，即 $(\alpha_{i+1}(t_j), \beta_{i+1}(t_j)) = (\max\{(\alpha_i(t_j) - S_{\text{SL}_{i+1}}), 0\}, \beta_i(t_j) - S_{\text{SE}_{i+1}})$。

(2) 对于新使能的变迁 $t_j \in E_{\text{enabled}}(C_i) \cap t_j \notin E_{\text{enabled}}(C_i)$，将其静态的触发间隔增加到触发域 D_{i+1} 中，即 $(\alpha_{i+1}(t_j), \beta_{i+1}(t_j)) = (\alpha^s(t_j), \beta^s(t_j))$。

(3) 对于迁 $t_j \in H_{\text{halting}}(C_i) \cap t \in E_{\text{enabled}}(C_{i+1})$，触发域保持不变，即 $(\alpha_{i+1}(t_j), \beta_{i+1}(t_j)) = (\alpha_i(t_j), \beta_i(t_j))$。

(4) 对于新的非使能变迁 $t_j \in E_{enalbed}(C_i) \cap t \notin E_{\text{enabled}}(C_{i+1})$，移走与之相关联的表达式。

6) 变量 $i = i + 1$，如果 $i \le n - 1$，返回步骤2)继续校验，否则退出。

7) 如果对所有 $0 \le i \le n - 1$，t_{i+1} 都是可以调度的，则该触发序列是可以调度的。

3.3.2 计算调度长度

调度分析完毕后，可以使用下列算法计算 $\rho = C_0 t_1 C_1 t_2 \cdots t_n C_n$ 的调度长度：(1) 设定初始值 $i = 0, sl(\rho) = 0$ 。(2) 如果变迁 $t_{i+1} \in P_{\text{parallel}}(\rho)$ ，$t_{\text{sl}}(\rho) = \ t_{\text{sl}}(\rho) + (P_{\text{PE}_{i+1}}, P_{\text{PL}_{i+1}})$ ，否则 $t_{\text{sl}}(\rho) = t_{\text{sl}}(\rho) + (S_{\text{SE}_{i+1}}, S_{\text{SL}_{i+1}})$。(3) 变量 $i = i+1$，如果 $i \le n-1$，返回步骤(1)继续校验，否则退出。$t_{\text{sl}}(\rho)$ 即为调度完成时间。最后，通过比较调度长度和时限即可确定的调度序列的可行性。

3.3.3 实例

由于三个变迁 t_1、t_2和t_3 都使用同一个处理器 p_{proc_1}，所以三个变迁需要顺序执行，调度长度就等于三个变迁的执行时间之和，即 $t_{\text{sl}}(\rho) = \tau(t_1) + \tau(t_2) + \tau(t_3) = (3,5) + (1,4) + (2,5) = (6,14)$。假定任务的死线是12，则该调度方法有可能无法满足死线。为此可以引入处理器 p_{proc_2}，如图1b和1c所示。

对 $\rho = t_1 t_2 t_4 t_3$ 进行调度分析如图1b所示。初始时 $C_0 = (M_0, D_0)$,其中 $M_0 = (1,0,0,0,0)$，$D_0 = \{t_1(3,5)\}$。

(1) $t_1 \in W_{\text{working}}(C_0) = \{t_1\}$，由于运行变迁是唯一的，所以 t_1 在间隔 $(S_{\text{SE}_1}, S_{\text{SL}_1}) = (3,5)$ 之间是可以调度的。变迁 t_1 触发后，计算状态类 $C_1 = (M_1, D_1)$，得到 $M_1 = (0,\ 1,\ 1,\ 0,\ 0,\ 0)$， $D_1 = \{t_2(1,4), t_4(1,2)\}$。

(2) $t_2 \in W_{\text{working}}(C_1) = \{t_2, t_4\}$，并且 $\alpha_1(t_2) = 1 \le \min\{\beta_1(t_2)$，$\beta_1(t_4)\} = \min\{4,2\} = 2$，所以变迁 t_2 可以在间隔 $(S_{\text{SE}_2}, S_{\text{SL}_2}) = (\alpha_1(t_2), \min\{\beta_1(t_j): t_j \in W_{\text{working}}(C_1)\}) = (1,2)$ 之间进行调度。变迁 t_2 触发后，计算状态类 $C_2 = (M_2, D_2)$，得到：

$$M_2 = (0,0,1,1,0)\ D_2 = \{t_3(\max\{(1-2),0\}, 2-1)\} = \{0,1\}$$

$$(P_{\text{PE}_2}(t_4), P_{\text{PL}_2}(t_4)) = (\max\{(P_{\text{PE}_1}(t_4) - S_{\text{SE}_2}), 0\}$$

$$\max\{(P_{\text{PL}_1}(t_4) - S_{\text{SL}_2}), 0\}) = (\max\{(\alpha^s(t_4) - S_{\text{SE}_2}), 0\},$$

$$\beta^s(t_4) - S_{\text{SL}_2}) = (0,0)$$

$$P_{\text{parallel}}(\rho) = \{t_4\}$$

(3) $t_4 \in W_{\text{working}}(C_2) == \{t_4\}$，由于运行变迁是唯一的，所以变迁 t_4 可以在区间 $(S_{\text{SE}_3}, S_{\text{SL}_3}) = (0,1)$ 之间进行调度。变迁 t_4 触发后，计算状态类 C_3，得到：

$$M_3 = (0,0,0,1,1)$$

$$D_3 = \{t_3(2,5)\}$$

$$(P_{\text{PE}}(t_4), P_{\text{PL}}(t_4)) = (P_{\text{PE}_2}(t_4), P_{\text{PL}_2}(t_4)) = (0,0)$$

$$P_{\text{parallel}}(\rho) = \{t_4\}$$

(4) $t_3 \in W_{\text{working}}(C_2) = \{t_3\}$，由于运行变迁是唯一的，所以变迁 t_3 可以在区间 $(S_{\text{SE}_3}, S_{\text{SL}_3}) = (2,5)$ 之间进行调度。变迁 t_3 触发后，计算状态类 C_4，得到：

$$M_4 = (0,0,0,2,0)$$

$$D_4 = \varnothing$$

调度时间 $t_{\text{sl}}(\rho) = (3,5) + (1,2) + (0,0) + (2,5) = (6,12)$。同理，可得如图1c所示的调度长度 $t_{\text{sl}}(\rho) = (3,5) + (1,2) + (1,4) + (0,0) = (5,11)$。可见，新算法可以正确地处理分布式系统。

3 结 论

由于使用RBTPN模型建模分布式实时嵌入式系统的静态调度后，原有的调度分析算法存在着不足，通过引入并行间隔，改进了传统的调度分析算法，在保持其优点的同时，将其有效地扩展到了分布式实时嵌入式系统的调度分析中。

本文研究工作得到了河南科技大学人才科学研究基金项目(06-8)的资助，在此表示感谢。

参 考 文 献

[1] MURATA T. Petri nets: Properties, analysis, and applications[J]. Proceedings of the IEEE, 1989, 77(4): 541-580.

[2] EDWARDS S, LAVAGNO L, LEE E A, et al. Design of embedded systems: Formal models, validation, and synthesis[J]. Proceedings of the IEEE, 1997, 85: 366-390.

[3] ESSER R. An object oriented Petri net approach to embedded system design[D]. Swiss: Federal Institute of Technology Zurich, 1996.

[4] CORTES L A. Modeling and formal verification of embedded systems based on a Petri net representation[J]. Journal of Systems Architecture, 2003, 49: 571-598.

[5] NAEDELE M. Petri net models for single processor real-time scheduling[R]. TIK-Report No.61, 1998.

[6] OLIVER O H. Roux. A *T*-time Petri net extension for real-time task scheduling modeling[J]. JESA Modeling of Reactive Systems, 2002, 36(7): 973-986.

[7] 张海涛, 艾云峰. 基于Petri网的分布式实时嵌入式系统调度的建模[J]. 计算机工程, 2006, 32(18): 6-9.

[8] XU D X, HE X D. Compositional scheduling analysis of real-time systems using time Petri nets[J]. IEEE International on Software Engineering, 2002, l28: 984-996.

[9] 张海涛, 艾云峰. 基于Petri网的分布式实时嵌入式系统的调度分析[J]. 吉林大学学报, 2007, 37(3): 616-620.

编 辑 熊思亮