软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(2):279−296 [doi: 10.13328/j.cnki.jos.004781]

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

多处理器实时系统可调度性分析的 UPPAAL 模型*

代声馨, 洪 玫, 郭 兵, 杨秋辉, 黄 蔚, 徐保平

(四川大学 计算机学院,四川 成都 610065)

通讯作者: 洪玫, E-mail: hongmei@scu.edu.cn

摘 要: 随着多处理器实时系统在安全性攸关系统中的广泛应用,保证这类系统的正确性成为一项重要的工作.可调度性是实时系统正确性的一项关键性质.它表示系统必须满足的一些时间要求.传统的可调度性分析方法结论保守或者不完备,为了避免这些方法的缺陷,提出使用模型检测的方法来实现可调度性分析.提出了一个用于多处理器实时系统可调度性分析的模板,将与系统可调度性相关的部分包括实时任务、运行平台和调度管理模块都用时间自动机建模,并使用 UPPAAL 验证可调度的性质是否总被满足.符号化模型检测方法被用于推断可调度性,但是由于秒表触发的近似机制,符号化模型检测方法不能用于证明系统不可调度.作为补充,统计模型检测方法被用于估算系统不可调度的概率,并在系统不可调度时生成反例.此外,在系统可调度时,通过统计模型检测方法获取一些性能相关的信息.

关键词: 可调度性;模型检测;UPPAAL;多处理器实时系统;时间自动机

中图法分类号: TP316

中文引用格式: 代声馨,洪玫,郭兵,杨秋辉,黄蔚,徐保平.多处理器实时系统可调度性分析的 UPPAAL 模型.软件学报,2015,26(2):279−296. http://www.jos.org.cn/1000-9825/4781.htm

英文引用格式: Dai SX, Hong M, Guo B, Yang QH, Huang W, Xu BP. Schedulability analysis model for multiprocessor real-time systems using UPPAAL. Ruan Jian Xue Bao/Journal of Software, 2015,26(2):279−296 (in Chinese). http://www.jos.org.cn/1000-9825/4781.htm

Schedulability Analysis Model for Multiprocessor Real-Time Systems Using UPPAAL

DAI Sheng-Xin, HONG Mei, GUO Bing, YANG Qiu-Hui, HUANG Wei, XU Bao-Ping

(College of Computer Science, Sichuan University, Chengdu 610065, China)

Abstract: As multiprocessor real-time systems are increasingly applied in safety critical systems, it's important to ensure the correctness of such systems. One key attribute of the correctness of real-time system is the schedulability that guarantees to satisfy the timing requirements. The traditional methods for determining the schedulability are either pessimistic or unsafe. To tackle the drawbacks of those methods, this paper proposes a scheme to achieve the schedulability analysis using model checking. It provides a schedulability analysis framework for multiprocessor real-time system. All the components involved in the schedulability of the system, including the tasks, the execution infrastructure and the dispatching management unit, are modeled as timed automata and implemented in UPPAAL. Further, UPPAAL is employed to verify whether the schedulability property is always satisfied. Symbolic model checking is applied to determine schedulability. However, because of the over-approximation for stopwatches, symbolic model checking cannot be used to disprove schedulability. As a supplement, statistical model checking is used to estimate the probability of non-schedulability and generate concrete counterexamples going along with non-schedulability. Statistical model checking is also used to obtain some performance information when system is schedulable.

Key words: schedulability; model checking; UPPAAL; multiprocessor real-time system; timed automata

实时系统被广泛地应用于汽车、飞机等安全性攸关的系统中,实时系统的正确性对整个系统的安全运行有

* 基金项目: 国家自然科学基金(61332001, 61272104); 四川省应用基础研究项目(2014JY0112)
收稿时间: 2014-07-01; 修改时间: 2014-10-31; 定稿时间: 2014-11-26

至关重要的影响.实时系统分为硬实时和软实时两类,硬实时系统对验证其正确性提出了新的要求,不仅要求运行结果的逻辑正确性,而且要求时间正确性(必须满足时间约束).可调度性分析就是用于验证系统中的任务是否满足其规定的时间约束的重要方法,因此,可调度性分析已经成为验证实时系统正确性的关键步骤.

目前,任务可调度性分析方法主要有两种:理论证明,主要包括 CPU 利用率边界测试和响应时间分析;模拟测试,在模拟环境下反复运行系统,模拟可能的调度序列,验证可调度性.其中,

- 理论证明的结果能够保证正确性,但都过于保守,被理论证明方法确定为不可调度的任务集,可能实际上是可调度的.考虑到多处理器、不确定的任务运行时间以及抢占式策略这些因素,理论证明的计算将非常复杂甚至不能胜任.
- 模拟测试的方法分析成本较高,且不能保证分析结果的完备性,只能保证被执行到的情况下系统是可调度的;并且,模拟测试常设定所有任务都同时创建,在多处理系统中,这可能导致错误的结果[1].

因此,可调度性分析面临着分析成本和分析精确度两个方面的挑战.随着片上多处理器系统(multiprocessor systems-on-chip)[2,3]在嵌入式系统中的广泛应用,系统运行在包含多个不同的处理器的平台上,不同处理器上的任务有时需要协作以完成系统功能,也就是说,当运行在不同处理器上的任务之间存在依赖关系时[2,4,5],如图 1 中 τ_3 依赖于 τ_0,理论证明方法很难将这种依赖关系计算在内,而模拟测试中所有任务同时创建的假设更容易得到错误的结论.

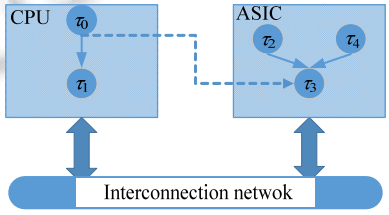


Fig.1 Multiprocessor systems-on-chip architecture

图 1 片上多处理器系统架构

根据在处理器间迁移能力的不同,多处理器调度策略可以分成 3 类:已划分的(partitioned)、限制迁移的(restricted migration)、可全迁移的(full migration)[6].本文针对任务已划分的多处理器实时系统的可调度性分析问题,提出使用模型检测方法实现自动分析.但是,将系统可调度性分析的问题描述为模型检测工具需要的模型和性质是一个并不简单的步骤.当前主要有两种可选方案:将已有的设计模型转换为模型检测工具需要的模型,或者是建立基于模型检测工具的可配置的可调度性分析模板.第 1 种方案能够针对特定的系统定制可调度性分析的模型,但当前不能实现自动转换,只能手动转换,工作繁琐;第 2 种方案提供了快速建立系统可调度性分析模型的途径.本文通过对实例系统的分析并结合可调度性分析理论,抽象出多处理器实时系统可调度性分析模型;规范化抽象出来的模型建立模板;再选择更多系统实例,根据模板建立模型,用于可调度性分析,通过与其他可调度性分析方法结果的比较验证模板的有效性.选用模型检测工具 UPPAAL[7]对系统建模,将系统抽象为实时任务、运行环境和调度管理模块的时间自动机模型,并通过自动机间通信和共享变量构成时间自动机网络.使用两种方法对系统做可调度性分析:

- 一是使用符号化模型检测方法验证系统是否是可调度的,UPPAAL 遍历系统模型的整个状态空间,验证可调度的性质是否总是被满足,如果被满足,则系统是可调度的.
- 二是通过统计模型检测方法估算系统不可调度的概率,UPPAAL 自动模拟系统模型,验证不可调度的性质是否被满足.最后,通过统计算法估算性质被满足的概率.

此外,在系统可调度时,可以通过其他统计性质获得一些性能相关的信息.

通过本文提供的可配置且可扩展的分析模板,能够快速构建类似系统的分析模型并完成可调度性分析;同时,文中提供了建立可调度性分析模型过程中一些基本问题的建模方法,如抢占式策略、不确定的任务运行时

间以及任务依赖关系的管理等.通过将可调度性分析问题转换为模型检测的性质验证问题,使分析过程自动化,降低了分析成本,保证了结果的完备性,提高了分析效率;同时,模型检测方法可以得到更准确的分析结果,提高了实时系统的可靠性.

本文第 1 节简要介绍可调度性分析的基本概念.第 2 节详细介绍使用 UPPAAL 实现可调性分析的模板.第 3 节阐述分别使用符号化模型检测方法及统计模型检测方法完成系统的可调度性分析的流程.第 4 节通过实例验证文中所提出模板的有效性.第 5 节综述相关工作.第 6 节总结本文工作并提出未来的研究方向.

1 实时系统任务可调度性分析方法

1.1 任务可调度性和可调度性分析

实时系统中的任务分为周期性任务和非周期性任务两种类型.周期性任务有一个固定的到达时间间隔 T,T 就是任务的周期.如果任务在 t 时刻被创建,那么该任务将在 $t+T$ 时刻再次被创建.非周期性任务被不规则地创建,但是它们有最小的到达时间间隔 T.如果任务在 t 时刻被创建,那么该任务的下次创建时刻将可能在 $t+T$ 时刻或其之后的任意时刻[8].

实时任务的约束类型可以分为 3 类:时间约束,即任务的时间属性;依赖约束,即任务间的依赖关系;资源约束,即任务所映射到的处理器.

任务的时间属性如图 2 所示.

- 初始的时间偏移 $ioffset\in\mathbb{N}$,系统开始运行到任务第 1 个周期开始之间的时间间隔;
- 最好/最坏的运行时间 $0<bect\leqslant wcet\in\mathbb{N}$,每个周期内任务在最好或最坏情况下的运行时间;
- 响应时间 $respt\in\mathbb{R}$,任务从创建到运行结束的时间;
- 周期 $period\in\mathbb{N}$;
- 截止时间 $deadline\in\mathbb{N}$;
- 运行时间 $exec\in\mathbb{R}$ 且 $exec\in[bect,wcet]$.

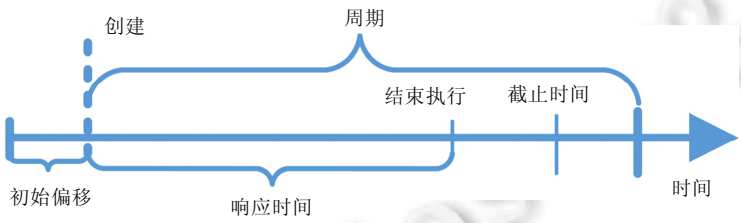


Fig.2 Timing attributes of task

图 2 任务的时间属性

任务的依赖约束使用任务图 $G=(T,\prec)$表示,如图 3 所示,其中,$\prec$表示在任务集合上的偏序关系.因此,任务图是一个有向无环图.图中的节点表示任务,边表示任务之间的依赖关系.例如,$\tau_0\prec\tau_1$ 表示 τ_1 只能在 τ_0 运行结束后才能开始执行.使用 $\Pi(\tau_4)$ 表示任务 τ_4 的直接依赖任务的集合,当且仅当 $\Pi(\tau_4)$ 中的所有任务都结束后,τ_4 才能开始执行.

实时系统的调度策略可以分为静态任务优先级、静态作业优先级和动态策略[8]:

- 在静态任务优先级策略下,任务的所有作业都具有固定的优先级;
- 在静态作业优先级策略下,任务的不同作业可能有不同的优先级,但是每个作业具有固定的优先级;
- 在动态策略下,同一作业在不同时刻可能具有不同的优先级.

同时,调度策略可以是抢占式、非抢占式或合作的[8]:

- 在抢占式策略下,调度器会临时挂起当前运行的任务,而让具有更高优先级的任务运行;

- 在非抢占式策略下,任务一旦开始运行,就会持续运行,直到结束;
- 在合作的策略下,任务只可以在运行过程中的某些时刻被抢占.

本文将以实时系统中两个最常用的调度策略为例,对于其他调度策略的模型将不在文中详细叙述:

- 单速率调度(rate monotonic scheduling,简称 RMS):周期短的任务优先;
- 最早截止时间优先(earliest deadline first,简称 EDF):最早到达其截止时间的任务优先.

在一定的调度策略下,若任务的最坏响应时间不大于任务的截止时间,则这个任务是可调度的.一个任务集是可调度的,表示在当前调度策略下,集合中的所有任务都是可调度的[8].

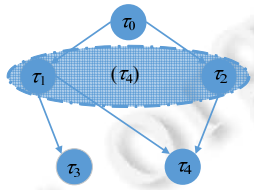


Fig.3 Task graph

图 3 任务图

1.2 传统的可调度性分析方法

Liu 和 Layland[9]给出了验证采用固定优先级(如单速率调度策略)调度策略的单处理器系统可调度性的处理器利用率上界如下:

$$\sum_{i=1}^{m}(C_i/T_i)\leqslant m\left(2^{\frac{1}{m}}-1\right),$$

其中,m 为系统中的任务数,C_i 和 T_i 分别为任务的运行时间和周期.

当任务的截止时间小于或等于周期时,任务 i 的最坏响应时间 r_i 可通过以下递归函数得到:

$$r_i=C_i+\sum_{j\in hp(i)}\left\lceil\frac{r_i}{T_j}\right\rceil,$$

其中,C_i 是任务的运行时间,$hp(i)$表示优先级高于任务 i 的任务集合,T_j 表示任务 j 的周期.

Tindell 和 Clark[10]提出了针对抢占式的静态优先级策略的可调度性分析方法,称为 holistic 方法.此方法的全面性在于它将单处理器可调度性分析理论的成果与通信分析结合起来,从而得到系统的全貌(holistic view).文中提出了针对固定优先级的分布式硬实时系统的可调度性分析方法,任务 i 的最坏响应时间 r_i 为

$$r_i=\max_{q=0,1,2,\ldots}(J_i+w_i(q)-qT_i),$$

其中,J_i 是任务 i 到达和被释放之间的最坏情况延迟时间,称为释放抖动(jitter);C_i 和 T_i 分别为任务的运行时间和周期;$w_i(q)$由以下公式给出:

$$w_i(q)=(q+1)C_i+\sum_{\forall j\in hp(i)}\left\lceil\frac{J_i+w_i(q)}{T_j}\right\rceil C_j+\tau(w_i(q)),$$

其中,τ是计时器中断的时间消耗,其最坏情况下计算时间和周期分别为 C_{clk} 和 T_{clk}.在一个宽度为 w 的时间窗口内,可得最坏情况计时器中断次数 L 和最坏情况下被移出挂起队列的任务数 K:

$$L=\left\lceil\frac{w}{T_{clk}}\right\rceil,K=\sum_{\forall j\in hp(i)}\left\lceil\frac{J_i+w}{T_j}\right\rceil.$$

定义第 1 个任务从挂起队列中取出的时间为 C_{QL},每个后续任务取出的时间为 C_{QS}.可得τ的计算公式如下:

$$\tau_i(w)=\begin{cases}LC_{clk}+KC_{QL}, & L\geqslant K\\ LC_{clk}+\min(L,K)C_{QL}+\max(K-L,0)C_{QS}, & L<K\end{cases}.$$

1.3 基于模型检测技术的可调度性分析

模型检测能够用于系统的可调度性分析[2,4,11−13].基于时间自动机[14]的模型能够同时表示实时系统的逻辑属性和时间属性,通过系统建模,将可调度性分析问题转化为时间自动机的可达性问题.时间自动机的状态空间可以划分为有限个的状态区域(zones)[15],从而实现将无限问题的求解转换为有限问题的求解.

差异约束矩阵(difference bound matrices)提供了对时间自动机的状态区域的正则表达,通过该数据结构,完成对状态区域操作结果的求解以及在状态区域上性质的验证[15].

2 基于 UPPAAL 的系统建模

本文提供了基于 UPPAAL 的多处理器实时系统可调度性分析的模板.系统的总体假设如下:

(1) 系统中的程序为周期性任务的集合.

(2) 系统中的多个处理器通过总线链接.

(3) 每个实时任务被分配到一个固定的处理器上运行.

(4) 一个实时任务可能依赖于一组其他任务,只有满足所有依赖,任务才能运行.任务间可能存在跨处理器的依赖,即,一个任务依赖于另一个处理器上的任务.

(5) 不同处理器上的任务依赖消息通过总线传递.

(6) 任务执行时间的取值在区间[最好执行时间,最坏执行时间]内随机选取.

2.1 符号化模型检测工具UPPAAL及其建模语言

UPPAAL[7]是一个用于实时系统建模、模拟和验证的工具.在 UPPAAL 中,一个系统被表示为由一组时间自动机[14]、全局数据、变量及 CCS(calculus of communicating systems)风格的同步管道组成的模型.通过定义信号 *c*,时间自动机之间可以完成同步,其中,发布消息使用 *c*!,而接收消息使用 *c*?.同时,UPPAAL 支持 C 语言风格的语法使用户能够实现自定义的函数.最新版本的 UPPAAL 加入了对秒表(stopwatches)[16]及统计模型检测[17]的支持.最初,UPPAAL 支持对时间计算树逻辑(CTL)[18]表达式的子集的验证;当前,新的 UPPAAL-SMC[19]引擎支持对 WMTL(weighted metric temporal logic)[20]表达式的验证.UPPAAL-SMC 中实现了几种统计模型检测的算法,并支持 4 种统计性质[21]的验证:

- 概率估计:‘Pr[’ (Clock | ‘#’) ‘<=’ CONST ‘](’ (‘⟨·⟩’ | ‘[·]’) Expression ‘)’,在方括号中条件为真的情况下,估算括号中性质满足的概率.
- 假设测试:‘Pr[’ (Clock | ‘#’) ‘<=’ CONST ‘](’ (‘⟨·⟩’ | ‘[·]’) Expression ‘)’ (‘<=’ | ‘>=’) PROB,测试某一性质被满足的概率是否小于或大于某一阈值.
- 概率比较:‘Pr[’ (Variable | ‘#’) ‘<=’ CONST ‘](’ (‘⟨·⟩’ | ‘[·]’) Expression ‘)’ (‘<=’ | ‘>=’) ‘Pr[’ (Variable | ‘#’) ‘<=’ CONST ‘](’ (‘⟨·⟩’ | ‘[·]’) Expression ‘)’,比较两个性质被满足的概率.当左边表达式的概率大于右边时结果为 1,当右边大于左边时结果为 0,其余情况为 0.5.
- 数值估计:‘E[’ (Clock | ‘#’) ‘<=’ CONST ‘;’ CONST ‘]’ ‘(’ (‘min:’ | ‘max:’ Expression ‘)’,在模拟设定次数后,估算表达式的最小(最大)期望值.

2.2 实时系统及其环境建模

一个多处理器实时系统由应用程序、运行平台及调度管理模块组成:应用程序包含一组有限的周期性任务,并且任务之间存在依赖关系(可以抽象为任务图);运行平台包括多个处理器、链接多个处理器的总线;调度管理模块负责调度任务在处理器上运行.在可调度性分析的模板中,对多处理器实时系统建模.另外,为了实现对任务依赖关系管理的建模,增加一个依赖管理器模型.模板自顶向下的组成如下:

- *System*=*Application*||*Infrastructure*||*Scheduler*;
- *Application*=$\|_{\tau\in T}$*Task* (τ)||*DependencyManager*;
- *Infrastructure*=||*Bus*||*Processors*;

- $Scheduler=\|Scheduler\|_{\rho\in P}Policy\ (\rho)$.

其中,‖表示时间自动机的并行组合.

在模型中,每一个任务都是任务模板参数化属性(如周期、截止时间等)后的实例.任务间的依赖关系则通过两个矩阵(模板中通过数组实现)表示:依赖矩阵 *staDap* 表示任务间的静态依赖关系,也就是任务图,其中,*staDap*[*i*][*j*]表示任务 *i* 是否依赖于任务 *j*;用动态矩阵 *dynDap* 表示某一时刻任务的依赖关系.依赖管理器模型用于动态更新 *dynDap*,并通过比较两个矩阵的对应项来确定在该时刻任务的依赖是否被满足.依赖矩阵 *staDap* 根据待验证系统中任务之间的关系配置,动态矩阵由模板自动生成和管理.总线模型用于传输跨处理器的任务依赖消息.处理器并不单独使用时间自动机建模,而使用一个队列保存该处理器上的就绪任务,通过判断队列是否为空,表示处理器是否被使用,并且规定,在处理器队列首位的任务就是当前正在运行的任务;同时,设计一个处理器属性数组以保存处理器的属性,这个数组根据待验证系统配置数据;调度器模型用于管理所有的处理器队列,每一种调度策略都表示为一个时间自动机模型并与调度器模型通信,调度器模型调用相应的调度策略模型,将新的就绪任务插入到处理器的队列中.

2.3 实时任务建模

一个任务模板由一个时间自动机(如图 4 所示)及保存任务属性的数据结构(见表 1)组成.任务的时间自动机模板需要一个参数,即任务编号 *id*,任务的属性保存在一个全局的任务数组 *Tasks*[*N*](*N* 表示任务个数)中,并根据待验证系统做相应的配置,其中,任务 *id* 被用作该数组的索引.

每个任务都使用 3 个时钟变量记录其每个周期中的时间属性:执行时间 *exec*、响应时间 *respt* 及本周期开始以来的时间 *time*[*id*].为了实现时间的暂停,秒表被用于记录执行时间和响应时间,在状态不变量中,使用 *exec*[*id*]′==*e* 控制秒表运行,只有在 *e* 大于 0 时秒表才会活动.*time*[*id*]持续增长,并在每个周期结束后重置.当 *time*[*id*]大于等于截止时间时,任务转换到 *Error* 状态.

初始时,任务停留在初始状态直到 *InitOffset* 时刻.然后,任务开始其第 1 个周期,任务转换到等待依赖状态,并发出检查依赖是否满足的请求;当收到 *dep_ack*[*id*]信号后,任务转换到依赖满足状态并立即发出 *check_pe*[*id*]信号请求处理器,同时转换到 *Ready* 状态等待运行;当任务拥有就绪队列中最高优先级时,调度器模型通过 *run*[*id*]信号使任务开始运行;任务运行结束后发出广播信号 *finish*[*id*],同时转换到 *Finish* 状态,等待下一个周期开始.当任务在 *running* 状态时,在抢占式策略下,调度器模型可以通过 *preempt*[*id*]信号挂起当前任务.

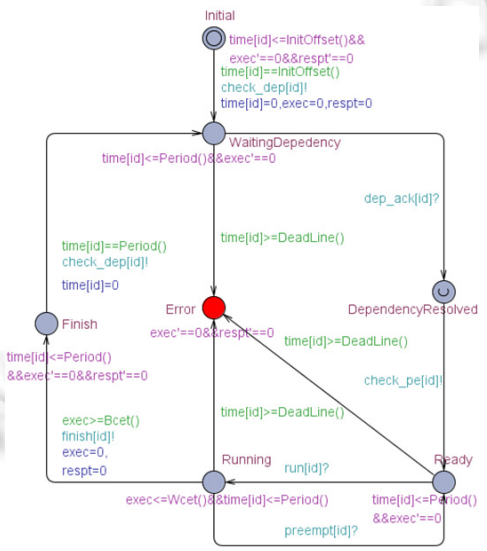


Fig.4 Task model

图 4 任务模型

Table 1 Task attributes

表 1 任务属性

任务集 T 中的任意任务可以表示为如下多元组:

$\tau=\langle id,initoffset,bect,wect,deadline,period,pe\rangle$.

其中,

$Id:T\to\mathbb{N}$,任务的编号;

$Initoffset:T\to\mathbb{N}$,初始时间偏移;

$Bect:T\to\mathbb{N}_{\geqslant 0}$,最好情况执行时间;

$Wect:T\to\mathbb{N}_{\geqslant 0}$,最坏情况执行时间;

$Deadline:T\to\mathbb{N}_{\geqslant 0}$,截止时间;

$Period:T\to\mathbb{N}_{\geqslant 0}$,周期;

$Pe:T\to\mathbb{N}_{\geqslant 0}$,任务映射到的处理器编号;

$\to$:表示集合间的映射关系.

2.4 依赖管理建模

依赖管理器模型用于管理动态矩阵 *dynDap*,并接受来自任务的检查依赖是否满足的请求.当任务的依赖被满足后向其发送确认信号,如图 5 所示.

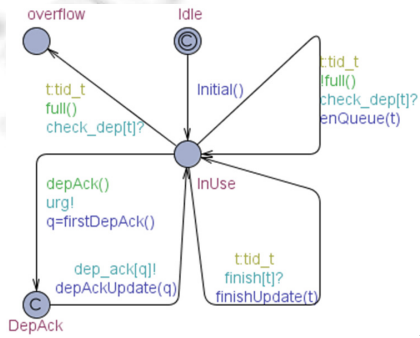


Fig.5 Dependency manager model

图 5 依赖管理器模型

使用一个队列 *DepQueue* 保存发出检查依赖请求的任务编号.当收到一个任务发出的 *check_dep*[*t*]信号后,依赖管理器将任务 *id* 加入到 *DepQueue* 队列中.如果队列已满,则转换到 *overflow* 状态.在该模型中,*depAck*(·)函数用于检查队列中是否存在一个任务 i 其依赖的所有任务 $j\in\Pi(i)$,动态矩阵中,*dynDap*[*i*][*j*]为真,*firstDepAck*(·)函数返回依赖被满足的第 1 个任务的 *id*,并同时将其弹出队列,然后向任务发出确认信号;最后,更新 *dynDap*[*q*][·]一行:如果 *staDap*[*q*][*j*]为真,则设置 *dynDap*[*q*][*j*]为假.当接收到任务发出的 *finish*[*t*]信号后,依赖管理器模型更新 *dynDap*[·][*t*]一列:当任务 i 和任务 t 在同一个处理器上时,如 *staDap*[*i*][*t*]为真,则将 *dynDap*[*i*][*t*]设置为真;如果任务 i 和任务 t 在不同的处理器上,如图 1 中 τ_3 依赖于 τ_0,依赖管理器模型则产生一个处理器间依赖消息并加入到总线队列中.可以通过修改动态矩阵的更新函数实现特定的依赖关系建模.

模型通过 *DepQueue* 队列及一个坚定(committed)状态 *DepAck* 实现对动态矩阵的互斥访问:所有的依赖满足的更新操作都串行执行,由坚定状态的定义——一个坚定状态不能延迟,并且下一个转换必须从坚定状态往外转换[7],因此,当 *depAckUpadte*(·)执行时,*finishUpdate*(·)将不能执行,从而不会出现两个写操作同时执行的情况.

2.5 总线建模

总线用于传输处理器间依赖消息,使用一个时间自动机(如图 6 所示)和保存消息的队列表示.模型需要两个变量 *bus_bect* 和 *bus_wcet* 作为参数,分别表示消息传输的最好和最坏运行时间,这两个参数根据待验证系统配置.在该模型中,*receive*(·)函数根据消息更新动态矩阵中的对应项.模型主要有以下状态:

- *Idle*,总线队列为空;
- *InUse*,正在传输依赖消息;
- *Received*,消息传输完成,同时用于实现动态矩阵的互斥访问.

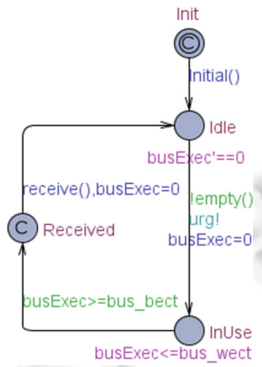


Fig.6 Bus model

图 6 总线模型

2.6 调度器建模

调度器模型完成处理器和调度策略之间的关联.调度器模型调用调度策略模型,将就绪任务插入到对应的处理器队列中,并在任务运行结束后将其弹出处理器队列,同时选择下一个任务开始运行.调度器模型根据处理器属性中配置的调度策略,将对应的调度策略模型加入到系统模型中.调度器模型如图 7 所示,模型读取处理器属性数组 *pes*[*P*](*P* 为处理器个数),每个处理器的属性见表 2.

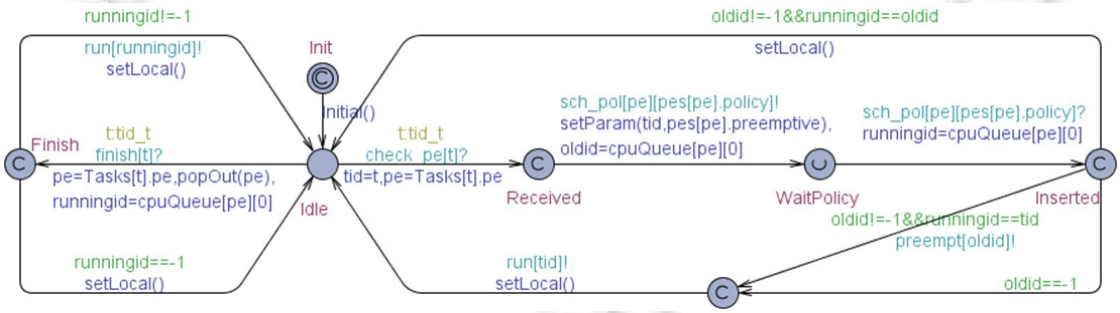


Fig.7 Scheduler model

图 7 调度器模型

Table 2 Processor attributes

表 2 处理器属性

处理器集合 *P* 中的任意处理器可以表示为以下多元组:

$$\rho=\langle pid,policy,preemptive\rangle.$$

其中,

Pid:*P*→ℕ,处理器编号;

Policy:*P*→ℕ,调度策略编号;

Preemptive:*P*→ℕ,调度策略是否可抢占.

Initial(·)函数将处理器队列中的每一项都初始化为–1.当接收到任务 *t* 发起的 *check_pe*[*t*]信号后,调度器模型与任务映射到的处理器的调度策略模型通信,并使用 *setParam*(·)函数传递参数,同时记录当前在处理器队列

首位的任务 *oldid*.调度策略模型将就绪任务插入到处理器队列的合适位置后发出响应.当收到该响应后,调度器模型转换到 *Inserted* 状态,并记录当前在处理器队列队首的任务号 *runningid*.最后,调度器模型更改相关任务的状态:如果插入前处理器队列为空(*oldid*==−1),则发出 *run*[*tid*]信号使新就绪任务开始运行;如果插入前处理器队列不为空,且新就绪任务抢占原任务(*oldid*!=−1 && *runningid*==*tid*),则调度器模型通过 *preempt*[*oldid*]挂起原任务,然后使新就绪任务开始运行;如果就绪任务没有抢占原任务(*oldid*!=−1 && *runningid*==*oldid*),则不需要改变任务状态.

任务结束时发出*finish*[*t*]信号,当调度器模型接收到*finish*[*t*]信号后,将任务 *t* 弹出对应的处理器队列,并设置队列中的下一个任务(如果存在)开始运行.

2.7 调度策略建模

调度策略可被视为一种排序算法,该算法根据任务的属性管理处理器队列中任务的顺序.在模型中,静态任务优先级策略可以通过函数调用实现,而静态作业优先级策略及动态策略则必须使用一个单独的自动机实现.分别使用单速率调度策略和最早截至时间优先调度策略分别代表这两类调度策略的建模方法,其他调度策略的模型将不再详述.调度器模型与调度策略模型相互独立,所以其他调度策略也可以使用相似方法来建模并加入到系统模板中,方便系统模板的扩展.

2.7.1 单速率调度策略建模

单速率调度策略模型如图 8 所示:当接收到从调度器发来的信号后,模型通过 *readParam*(·)函数读取参数,然后将就绪任务插入到处理器队列中;*pol_RMS*(·)函数迭代地比较新就绪任务与队列中任务的周期长短,以找到合适的插入位置 *pos*;最后,调用全局函数 *insertQueue*(*pid*,*tid*,*pos*),将任务插入到处理器队列中,其中,*pid* 是处理器编号,*tid* 是就绪任务 *id*,*pos* 是插入到处理器队列的位置.对抢占式策略的处理通过以下代码实现:

int pos=(*preempt* ? 0:!*empty*(*pid*)).

该代码决定迭代的起始位置:当调度策略为抢占式时,则从第 1 项开始迭代;当调度策略为非抢占式时,起始位置由!*empty*(*pid*)决定.进一步分析!*empty*(*pid*),*empty*(*pid*)函数返回处理器队列是否为空(1 代表为空,0 代表不为空).因此,在非抢占式的情况下,如果队列为空,那么迭代从第 1 项开始;如果队列不为空,则从第 2 项开始.由于在处理器队首的任务即为现在正在运行的任务,因此,以上迭代从队首开始就表示抢占式调度策略;反之,从队列中的第 2 项开始迭代表示非抢占式策略.

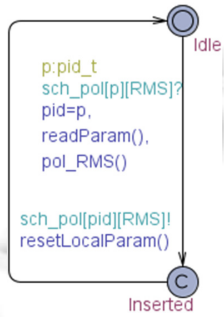


Fig.8 RMS model

图 8 RMS 模型

2.7.2 最早截止时间优先策略建模

本文更改文献[4]中最早截止时间优先策略的模型,从而与系统的其他部分协作,如图 9 所示.最早截止时间优先策略模型有 3 个状态:

- *Idle*,表示模型没有活动;
- *Iteration*,表示模型正在迭代的检查当前队列;

- *Inserted*,表示已经将就绪任务插入到了处理器队列中.

3 个局部变量 *pid*,*preempt* 和 *tid* 用于存储参数,另一个局部变量 *pos* 作为迭代器.

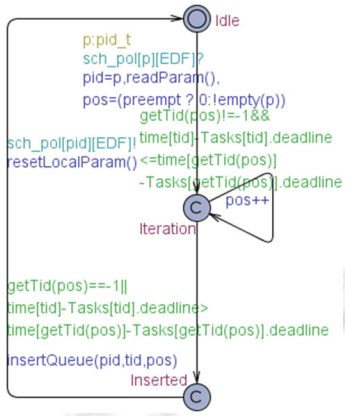


Fig.9 EDF model

图 9 EDF 模型

为了计算当前时间到任务截止时间之间的间隔,需要比较一个整形变量和一个时钟变量.UPPAAL 不允许在函数中操作时间变量,但在边的条件中允许该操作,因此只能在边的条件中完成比较,这就是静态作业优先级策略及动态策略必须使用单独的时间自动机建模的原因.为了比较就绪任务与队列中第 *pos* 项任务到达截止时间的快慢,通过检查以下条件实现:

getTid(*pos*)!=−1 && *Tasks*[*tid*].*deadline*−*time*[*tid*]>*Tasks*[*getTid*(*pos*)].*deadline*−*time*[*getTid*(*pos*)].

其中,*getTid*(*pos*)返回在处理器队列 *cpuQueue*[*pid*][*pos*]项中的任务 *id*.UPPAAL 不允许从一个常量减去一个时间变量,但可以用一个时间变量减去一个常量[4],因此重写以上条件如下:

getTid(*pos*)!=−1 && *time*[*tid*]−*Tasks*[*tid*].*deadline*<=*time*[*getTid*(*pos*)]−*Tasks*[*getTid*(*pos*)].*deadline*.

迭代终止于两种情况:达到了队列的末尾或者就绪任务比第 *pos* 项任务有更快的截止时间.调用全局函数 *insertQueue*(*pid*,*tid*,*pos*),将任务插入到处理器队列中.

2.8 全局自动机模型

全局自动机管理所有记录处理器是否被使用的秒表 *cpuUsed*[*pid*],其中,*empty*(*i*)函数返回处理器 *i* 对应的队列是否为空.由于可调度性分析的复杂性,当使用 UPPAAL 验证复杂的实时系统时,可能会导致内存用尽.Jensen[22]提出了扫描线方法(sweep line)用于缓解该问题,扫描线方法的基本思想是:在状态空间遍历的过程中,从内存中删除状态从而降低最大内存使用量.本文在全局自动机中创建与 Mikucionis[11]提出的模板相似的一个进度(progress),记录系统运行的周期数,并在达到预定的限制后重置该周期,如图 10 所示.

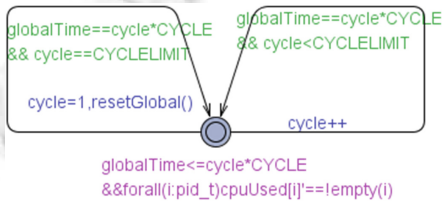


Fig.10 Global automata model

图 10 全局自动机模型

通过在系统中定义 *progress*{*cycle*;}来创建一个进度,每 *CYCLE* 时间后 *cycle* 增加,其中,*CYCLE* 是一段预定

义的时间片.当达到 *CYCLELIMIT* 限制后,*cycle*,*gloablTime* 及 *cpuUsed*[*pid*]中的所有项都被重置,此时,UPPAAL 从内存中删除某些状态,最终降低最大内存使用量.Mikucionis 等人[11]探究了 *CYCLELIMIT* 的设定规则,一个基本的假设是,至少运行所有任务的超周期(所有任务周期的最小公倍数)后重置 *cycle*.

2.9 关于建模的经验

本节分享一些在建立分析模型过程中的经验.

经验 1. meta 型变量对于减少状态空间非常有用,但不够稳定.

meta 型变量在赋值后的某一时间会被自动重置,因此可以通过 meta 型变量实现模型间同步时的数据传输,但必须在模型中使用局部变量储存这些数据.另外,使用 meta 变量后可能会导致诊断路径不能正常生成,如 David[4]的模型中使用一个 meta 变量 *ready_task* 最终导致“Cannot generate the trace”错误,将其改为全局变量后就能正常地生成诊断路径.当然,meta 型变量可能并不是造成这种错误的唯一原因,秒表机制也许是另一个源头.总的来说,对 meta 型变量的使用需要非常仔细,否则会导致不可预料的错误.

经验 2. 不变量可能会导致时间锁状态.

在 UPPAAL 中,所有时钟都以相同速率增长,因此,不变量不仅影响其关联到的状态,而且影响到整个时间自动机网.如果一个不变量要求时间自动机必须离开某一状态,但是从该状态出发的转换却不能活动,那么此时,整个模型会进入时间锁(time-locked)状态.如图 11 所示,如果模型开始运行时状态机 *S* 立即转换到 *S*1,那么超过第 1 时刻后,*S*1 的不变量要求自动机必须离开 *S*1 状态,但是与之同步通信的状态机 *T* 还在 *T*0 状态,不能完成同步通信,此时,模型进入一个时间锁状态.

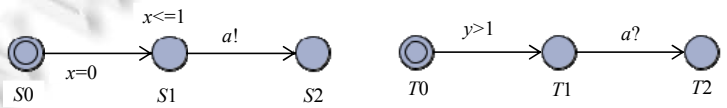


Fig.11 Invariant lead to timed locked state

图 11 不变量导致时间锁状态

经验 3. 坚定状态过于严格,可能导致时间锁状态.

最开始为了实现原子操作,我们将调度器中的 *WaitPolicy* 状态设置为坚定状态,虽然在 UPPAAL 下能够正常验证,但使用 UPPAAL-SMC 做分析时出现时间锁状态.导致该时间锁的原因分析如下:在调度器中,从 *Received* 到 *WaitPolicy* 的转换与调度策略模型使用 *sch_pol* 信号同步;前一个转换的目的状态 *WaitPolicy* 是一个坚定状态,根据坚定状态的定义,下一个转换必须从一个坚定状态出发;但是后一个转换是从一个普通状态出发,从而导致时间锁状态.使用紧急状态替换坚定状态可以避免该问题.

3 可调度性分析方案

本文分别使用两种方法来实现可调度性分析:符号化模型检测方法和统计模型检测方法.符号化模型检测方法用于得到定性的结论——这个系统是否可调度?统计模型检测方法用于估算在一定条件下系统不可调度的概率.与符号化模型检测方法相比,统计模型检测方法得到的结果并不保证绝对的精确;统计模型检测方法是在符号化模型检测方法不能完成可调度性分析(如内存使用量超过可用的内存大小)时的一个折中的方案,其并不能完全替代符号化模型检测方法.系统的一些性能数据,如最坏响应时间的期望值、处理器使用率等,也能通过统计模型检测方法得到.

3.1 用符号化模型检测实现可调度性分析

符号化模型检测方法验证“系统中的所有任务都是可调度的!”这句断言是否总是成立,转换为模型检测的性质如下:

$$A[\cdot]\ \text{forall}\ (i{:}tid\_t)\ \text{not}\ Task(i).Error \tag{1}$$

其中,*tid_t* 表示任务 *id* 的类型,*Error* 状态是任务模型中定义的超过截止时间后任务的状态.所以,该性质表示“在所有路径上所有任务都不会达到 *Error* 状态”.如果以上性质得到满足,则系统在任何情况下都是可调度的.

此外,任务的最坏响应时间(WCET)可以通过验证以下性质得到到:

$$\text{sup: } Task(0).resp, Task(1).resp, Task(2).resp, \ldots \tag{2}$$

其中,sup[21]性质返回 UPPAAL 遍历系统模型的整个状态空间后得到的表达式的最小上界,*respt* 是用于记录任务响应时间的秒表.

3.2 用统计模型检测实现可调度性分析

虽然符号化模型检测方法能够保证结果的绝对精确,但是由于状态爆炸的问题,当系统模型非常大的时候,符号化模型检测方法可能不能得出结论.统计模型检测方法用于在这种情形下得到一个牺牲精确性以换得问题可解的折中结论.统计模型检测方法回答的是“系统可调度的概率是多少?”,但估算一个硬实时系统可调度的概率意义不大,即使这个概率很大,依然不能推断系统是可调度的,因此,将待验证的问题转化为“系统不可调度的概率是多少?”.使用统计模型检测方法估算系统不可调度的概率,只要这个概率大于 0,则系统一定是不可调度的.在统计模型检测中,通过模拟系统模型并最后估算性质满足的概率,如果 UPPAAL 返回的概率区间的下限不为 0,则表示 UPPAAL 至少找到了一条调度路径,其中存在任务超过其截止时间.统计模型检测算法的内部,通过一些如 Cross-Entropy[23]等方法确保稀有情况会被模拟到.性质表达如下:

$$\Pr[<=t]\ (\langle\cdot\rangle\ \text{exists}\ (i{:}tid\_t)\ Task(i).Error) \tag{3}$$

该性质估算前 *t* 时间单元内,系统中存在一个任务达到 *Error* 状态的概率.验证该性质可以配置两个统计参数,以控制结果的置信度及精确性[21]:

- 显著性水平α(probability of false negatives,1−α是置信度)用于控制估算结果的置信度,如α为 0.01,则估算的结果是 99%可信的.
- 概率的不确定性ε(probability uncertainty)用于控制估算结果的精确性,ε越小,则估算的结果越接近实际的概率.

另一种对系统可调度的需求是要求系统不可调度的概率必须小于某一阈值,可用如下的性质来表示:

$$\Pr[<=t]\ (\langle\cdot\rangle\ \text{exists}\ (i{:}tid\_t)\ Task(i).Error)<=p,\ p\in[0,1] \tag{4}$$

验证系统在前 *t* 时间单元内系统不可调度的概率是否小于 *p*,验证该性质可以配置 3 个统计参数:显著性水平α、限定阈值的无差别区间的上界和下界的上概率偏差及下概率偏差(lower/upper probabilistic deviation).

完成可调度性分析之后,系统的一些性能信息也可通过统计模型检测性质得到.

通过以下性质得到最小/最大响应时间的期望值:

$$E[<=t;NUM]\ (\min|\max{:}Task(tid).respt) \tag{5}$$

该性质在模拟 *NUM* 次系统模型后,估算任务响应时间的最小或最大的期望值,并同时给出响应时间的概率密度分布等统计信息.

通过以下性质模拟系统模型并记录处理器使用时间:

$$simulate\ NUM[<=t]\ \{cpuUsed[0],cpuUsed[1],cpuUsed[2],cpuUsed[3]\} \tag{6}$$

该性质模拟 *NUM* 次系统模型在[0,*t*]时间区间内的情况,记录处理器被使用的时间;通过处理器被使用的时间与全局时间的比值,获得处理器的利用率.

4 实验验证

本节使用两个实验验证前文所提出的模型的有效性.实验环境为 CPU Intel i7-3632QM (2.20GHz),Java 1.7.0_55,UPPAAL 4.1.18.

从文献[5]中提到的智能手机的 GSM 系统中抽取出 21 个任务为例,分别使用符号化模型检测方法和统计模型检测方法对该实例做可调度性分析.实例系统的任务图和处理器如图 12 所示,其中,节点表示任务,边表示任务间依赖关系(虚线边表示处理器间依赖),处理器用矩形表示.为简洁起见,任务的时间属性将不在本文中详

细列举.处理器的属性,即,调度策略和可抢占性见表 3,总线的最好/最坏传输延迟分别为 2 和 4.

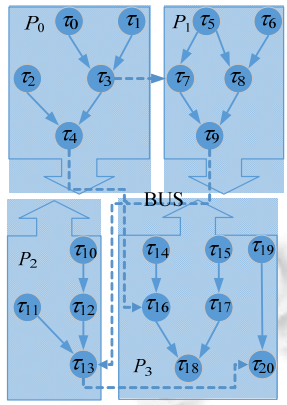


Fig.12 Experiment example

图 12 实验实例

Table 3 Processor attributes of the example

表 3 实例的处理器属性

处理器	策略	可抢占
P_0	EDF	TRUE
P_1	RMS	TRUE
P_2	EDF	TRUE
P_3	RMS	TRUE

配置任务属性数组、依赖矩阵、处理器属性数组以及总线模型的最好和最坏传输延迟建立系统的可调度性分析模型.为了验证实例的可调度性,验证性质(1).UPPAAL 在运行 2 335s 并使用 282 184KB/582 696KB 的最高驻存/虚拟内存后,确认性质(1)被满足,表示该实例是可调度的.另外,分别使用已有的 Brekling 和 David 提供的模型验证该实例,其时间和内存消耗分别为 3 126s,2 958s 和 285 274KB/60 876KB,326 598KB/634 920KB.在 Brekling 提供的模板中,各模型的功能划分不清楚,扩展很困难,如果系统中使用了模板中没有提供的调度策略,就难以验证;David 的模板中对每一个跨处理器的依赖加入一个额外的任务,相当于增加了 4 个任务,增大了系统模型的状态空间.

为了获得每个任务的最坏响应时间,验证性质(2).实验中,在一条性质得到所有任务的最坏响应时间会耗费大量时间,可以通过相同性质逐一得到每一个任务的最坏响应时间.通过比较任务的最坏响应时间与截止时间,可以看出,在本实例中所有任务都是可调度的.

在使用符号化模型检测方法的基础上,使用统计模型检测方法估算实例不可调度的概率.本文以时间 t=20000 及不同 α和 ε设置下验证性质(3),结果见表 4.

Table 4 Probability of the example is not schedulable

表 4 实例不可调度的概率

参数	概率	时间(s)	最高驻存/虚拟内存(KB)
α=0.05, ε=0.05	[0,0.0973938]	115.86	237 740/499 952
α=0.05, ε=0.01	[0,0.019956]	490.594	242 600/510 332
α=0.01, ε=0.05	[0,0.0986743]	136.938	252 564/531 440
α=0.01, ε=0.01	[0,0.0199441]	709.334	247 564/520 832

如表 4 所示,结果的下限保持为 0,表示 UPPAAL 模拟运行的过程中没有找到一条路径,其中任务的截止时间被违反.同时可以看到:修改统计参数对最终结果有重要影响;并且概率不确定参数对结果的影响大于置信度参数;且越精确的分析,得到的结果的上限越接近 0.造成该结果的原因很简单,因为该实例已经被符号化模型检

测方法确认为是可调度的,所以进一步提高统计的精确度,结果的上限将进一步趋近于 0.

再验证在前 20 000 时间单元之内,存在任务超过其截止时间的概率是否小于 1%:

Pr[<=20000] (⟨·⟩ exists (*i*:*tid_t*) *Task*(*i*).*Error*)<=0.01.

在 95%的置信度及 0.0001 的上概率偏差和下概率偏差下,UPPAAL 在模拟运行 1 458 次系统模型,花费 15 681.578s 和 61 568KB/131 940KB 的最高驻存/虚拟内存后,得出"Pr[<=20000] (⟨·⟩ exists (*i*:*tid_t*) *Task*(*i*).*Error*)>=0.991 with confidence 0.95",同时显示性质被满足.

在完成对实例的可调度性分析后,通过 UPPAAL 提供的统计性质获得一些系统的性能信息.

验证 *E*[<=20000;1000] (max:*Task*(16).*respt*)性质以获得任务 16 的最坏响应时间的期望值,得到结果 12.4942±0.0532254,在 UPPAAL 的 Plot Composer 中显示概率密度分布如图 13 所示.与 sup 性质得到的结果相比,这里得到的结果是一个统计结果.通过概率密度分布图可以得到任务响应时间的分布区间,并找出最有可能的响应时间,可用于对系统做性能分析.

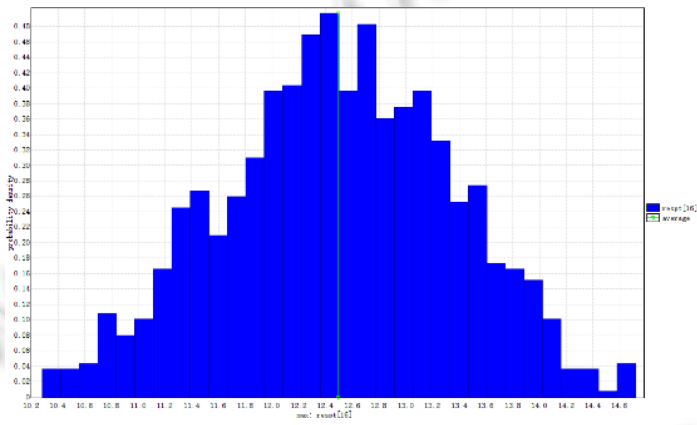


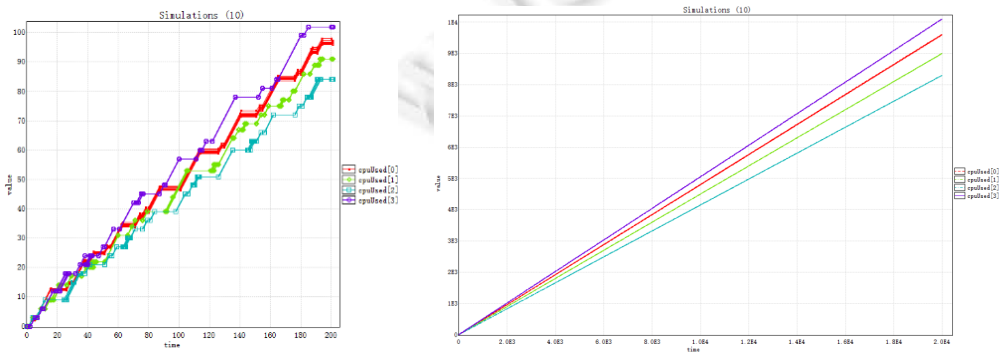
Fig.13 Probability density distribution of *Task*(16).*respt*

图 13 *Task*(16).*respt* 概率密度分布

然后,通过性质(6)模拟系统模型 10 次,记录处理器使用时间:

simulate 10 [<=200] {*cpuUsed*[0],*cpuUsed*[1],*cpuUsed*[2],*cpuUsed*[3]}.

在 Plot Composer 中显示如图 14(a)所示.为了计算处理器利用率,我们将时间边界扩张到 20 000,并将次数限制增加到 100 次,以便得到忽略掉空闲时间的近似结果,结果如图 14(b)所示.由图可知:斜线的斜率即为对应处理器的利用率,分别为 48%,45%,41%,50%.



(a) (b)

Fig.14 Processor utilization

图 14 处理器利用率

在对包含 21 个任务的系统完成验证后,使用本文提供的模板,验证文献[5]中的 GSM 系统以及与其共享两个处理器的 MP3 程序(总共 103 个任务和 4 个处理器)组成的复杂系统.首先,使用 UPPAAL 验证性质(1),在运行 19 分钟后出现“内存用尽”错误.在显著性水平为 0.0001(置信度为 99.9%)、概率不确定性为 0.01 的条件下,验证 Pr[<=20000] (⟨·⟩ exists (*i*:*tid_t*) *Task*(*i*).*Error*),UPPAAL 在运行 40 分钟后,最终得到的概率区间为[0.98004, 1],表示系统是不可调度的.

5 相关工作

响应时间分析方法[24,25]已经得到了广泛的运用.该方法只适用于包含严格类型的任务(任务间相互独立且运行时间固定)的系统,任务的最坏响应时间(WCRT)通过最坏执行时间(WCET)、最坏到达周期的间隔及最坏阻塞时间计算得到.最后,通过比较任务的最坏响应时间及截至时间得到可调度性的结论.这类方法是基于对任务的执行时间和阻塞时间的保守估计得出结论,因此总是得到安全但保守的结论,也就是说,存在将可调度的任务集合判断为不可调度的情况.

文献[10]中提出,分布式系统的响应时间可以通过 holistic 方法得到.该方法合并单处理器调度分析和通信分析的结果,从而得到整体的的结果.任务的最坏响应时间通过可调度性分析得到,而消息传递的最坏响应时间在基于使用时分多路复用(TDMA)通信协议的假设下得到.

文献[26]中提出了通过异步处理扩展的时间自动机实现可调度性分析的方法.该工具能够处理单处理器环境下的包含简单依赖关系的任务集,同时提供抢占式策略支持,但是该工具要求任务执行时间为固定值且不支持多处理器环境.工具基于 Fersman 等人[27]提出的任务自动机模型,每一个任务都与时间自动机的一个状态相关联,并使用一个任务队列表示多个任务的状态,自动机根据相应的调度策略将新就绪的任务插入到任务队列中.同时,Fersman 等人[27,28]证明了使用任务自动机做可调度性分析时可判定和不可判定的情况,其中,当任务执行时间不确定时以及在抢占式策略下是不可判定的.Krcal 等人[29]将任务自动机扩展到多核情况下,证明了当任务不在多个处理器队列之间迁移的假设下,系统的可调度性在非抢占调度器或固定的任务运行时间情况下是可判定的.

文献[2]中提出了分析运行在片上系统平台上的嵌入式系统的同步和时间行为的模型.文中首先提出了用于系统的形式化分析的系统级模型,并估算了使用最早截止时间优先策略下的计算树大小.然后,文中给出了基于时间自动机的系统模型:每个任务被映射到相应的处理器上;每个处理器运行固定的调度策略;处理器间的任务依赖通过加入额外的任务实现.文中最后给出了一个包含 4 个任务和 2 个处理器的实例,并使用 UPPAAL 验证其中的所有任务都不会超过截止时间.

文献[4]中提出了用于验证包含一组任务和资源的实时系统可调度性的框架.该框架支持任务的多种属性,包括时间偏移量和任务依赖关系等,同时使用秒表机制实现抢占式策略.文中对于任务间依赖关系的处理较为简单,每个任务使用一个布尔变量表示任务是否结束,并周期性地更新该变量,同时增加任务的周期内时间偏移,以等待父任务更新该变量.该文通过加入一个先进先出资源及运行在该资源上的附加任务实现跨处理器间任务依赖,最后使用符号化模型检测算法验证系统的可调度性.

文献[11]提出了一个使用 UPPAAL 验证 Herschel-Planck 卫星系统的实例.该卫星系统中包括一个处理器、一个基于抢占式的固定优先级策略的调度器以及一组周期性任务.任务分为应用程序和基础程序两类并分别建立模型,然后将系统中的所有任务逐一实例化,对于每一个任务,详细描述其操作和每一步操作所需要的时间和资源.文中使用了扫描线方法降低验证所需要的最大内存,最后,通过符号化模型检测算法验证系统的可调度性.David 等人[13]扩展了 Mikučionis 的模型,并进一步使用统计模型检测算法验证系统的可调度性;并分析了每一步操作的实际执行时间不为固定时间,而是在[*f*·*WCET*,*WCET*](其中,*f* 为百分比)区间内随机选择的情况下,不同的 *f* 选值对系统可调度性的影响.

文献[12]中提出了一个对多层次的单处理器实时系统建模和可调度性分析的组合调度框架,其中,分层结构、调度策略、任务的具体行为和资源都是可配置的,任务的具体行为通过操作列表给出.文中分别使用

UPPAAL 和 UPPAAL-SMC 验证系统的模型,并给出了使用该框架验证航空电子系统的实例.

文献[1,30]分别使用 UPPAAL 和 NuSMV 验证了采用全局调度策略的多核系统的可调度性.文中假设任务之间没有依赖关系,同时,每个任务拥有固定的执行时间,并且系统采用非抢占的固定优先级策略;在以上假设情况下,文中提供了限制迁移和全迁移调度策略的模型.基于同样的假设,文中又提供了使用 NuSMV 验证使用固定优先级和最短时间优先策略的多核系统的模型,并且提供了一个精确的任务集可调度性分析方法,同时能够得到任务的最好响应时间和最坏响应时间.

6 总 结

本文提出了用于多处理器实时系统可调度性分析建模和验证的模板.模板支持复杂的任务间依赖、不确定的执行时间、多种调度策略及可能的抢占式策略.与系统可调度性相关的部分包括任务、运行平台和调度管理模块都使用时间自动机建模.并同时使用符号化模型检测和统计模型检测两种方法验证系统模型,以分析系统的可调度性.通过本文提供的系统模板,将可调度性分析问题和模型检测方法连接起来,得益于模型检测自动化和高可靠的特点,实现自动并且准确的可调度性分析.

未来我们将扩展本模板,加入对软件资源的支持,并处理优先级反转问题;完善系统模型,形成一套实时系统可调度性分析的模板;研究 UPPAAL 的内部实现,建立更有效的模板,能够使用更少的时钟变量和自动机状态完成同样的系统验证;开发前端转换器,给用户提供一个更友好的界面:用户在界面中输入待验证系统的属性,然后转换器根据内部模板自动生成可调度性分析的模型,并调用模型检测工具完成可调度性分析;开发结果解释器,将模型检测工具得到的结果解释为更易被理解的形式.

致谢 诚挚感谢各位评审专家所给出的宝贵意见.感谢美国波特兰州立大学的 Xiaoyu Song 教授对本团队的帮助及对本文工作提出的中肯建议.

References:

[1] Guan N, Gu ZH, Lü MS, Deng QX, Yu G. Schedulability analysis of global fixed-priority or EDF multiprocessor scheduling with symbolic model-checking. In: Pettit R, ed. Proc. of the 11th IEEE Symp. on Object Oriented Real-Time Distributed Computing (ISORC 2008). IEEE Computer Society, 2008. 556−560. [doi: 10.1109/ISORC.2008.74]

[2] Brekling A, Hansen MR, Madsen J. Models and formal verification of multiprocessor system-on-chips. The Journal of Logic and Algebraic Programming, 2008,77(1):1−19. [doi: 10.1016/j.jlap.2008.05.002]

[3] Li RF, Liu Y, Xu C. A survey of task scheduling research progress on multiprocessor system-on-chip. Journal of Computer Research and Development, 2008,45(9):1620−1629 (in Chinese with English abstract).

[4] David A, Illum J, Kim GL, Skou A. Model-Based framework for schedulability analysis using UPPAAL 4.1. In: Mosterman PJ, ed. Proc. of the Model-Based Design for Embedded Systems. Boca Raton: CRC Press, 2010. 93−119.

[5] Schmitz MT, Al-Hashimi BM, Eles P. System-Level Design Techniques for Energy-Efficient Embedded Systems. Springer-Verlag, 2004. [doi: 10.1007/b106642]

[6] Carpenter J, Funk S, Holman P, Srinivasan A, Anderson J, Baruah S. A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms. In: Joseph YT, ed. Handbook of Scheduling: Algorithms, Models and Performance Analysis. Boca Raton: Chapman and Hall/CRC Press, 2004. 641−659.

[7] Behrmann G, David A, Larsen KG. A tutorial on UPPAAL. In: Bernardo M, ed. Proc. of the Formal Methods for the Design of Real-Time Systems. Springer-Verlag, 2004. 200−236. [doi: 10.1007/978-3-540-30080-9_7]

[8] Davis RI, Burns A. A survey of hard real-time scheduling for multiprocessor systems. ACM Computing Surveys, 2011,43(4):1−44. [doi: 10.1145/1978802.1978814]

[9] Liu CL, Layland JW. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM), 1973,20(1):46−61. [doi: 10.1145/321738.321743]

[10] Tindell K, Clark J. Holistic schedulability analysis for distributed hard real-time systems. Microprocessing and Microprogramming, 1994,40(2-3):117−134. [doi: 10.1016/0165-6074(94)90080-9]

[11] Mikučionis M, Larsen KG, Rasmussen JI, Nielsen B, Skou A, Palm SU, Pedersen JS, Hougaard P. Schedulability analysis using UPPAAL: Herschel-Planck case, study. In: Margaria T, ed. Proc. of the Leveraging Applications of Formal Methods, Verification, and Validation. Springer-Verlag, 2010. 175−190. [doi: 10.1007/978-3-642-16561-0_21]

[12] Boudjadar A, David A, Kim JH, Kim LG, Mikučionis M, Nyman U, Skou A. Hierarchical scheduling framework based on compositional analysis using UPPAAL. In: Fiadeiro JL, ed. Proc. of the Formal Aspects of Component Software. Springer-Verlag, 2014. 61−78. [doi: 10.1007/978-3-319-07602-7_6]

[13] David A, Larsen KG, Legay A, Mikučionis M. Schedulability of Herschel-Planck revisited using statistical model checking. In: Margaria T, ed. Proc. of the Leveraging Applications of Formal Methods, Verification and Validation. Applications and Case Studies. Springer-Verlag, 2012. 293−307. [doi: 10.1007/978-3-642-34032-1_28]

[14] Alur R, Dill DL. A theory of timed automata. Theoretical Computer Science, 1994,126(2):183−235. [doi: 10.1016/0304-3975(94)90010-8]

[15] Bengtsson J, Wang Y. Timed automata: Semantics, algorithms and tools. In: Desel J, ed. Proc. of the Lectures on Concurrency and Petri Nets: Advances in Petri Nets. Berlin: Springer-Verlag, 2004. 87−124. [doi: 10.1007/978-3-540-27755-2_3]

[16] Cassez F, Kim LG. The impressive power of stopwatches automata. In: Palamidessi C, ed. Proc. of the CONCUR 2000—Concurrency Theory. Berlin: Springer-Verlag, 2000. 138−52. [doi: 10.1007/3-540-44618-4_12]

[17] Legay A, Delahaye B, Bensalem S. Statistical model checking: An overview. In: Barringer H, ed. Proc. of the Runtime Verification. Springer-Verlag, 2010. 122−135. [doi: 10.1007/978-3-642-16612-9_11]

[18] Clarke EM, Jr. Grumberg O, Peled DA. Model Checking. London: MIT Press, 1999.

[19] Bulychev P, David A, Larsen KG, Mikučionis M, Poulsen DB, Legay A, Wang Z. UPPAAL-SMC: Statistical model checking for priced timed automata. In: Herbert W, ed. Proc. of the 10th Workshop on Quantitative Aspects of Programming Languages and Systems. Tallinn: Electronic Proceedings in Theoretical Computer Science, 2012. 1−16. [doi: 10.4204/EPTCS.85.1]

[20] Bulychev P, David A, Larsen KG, Legay A, Li GY, Poulsen DB, Stainer A. Monitor-Based statistical model checking for weighted metric temporal logic. In: Bjorner N, ed. Proc. of the Logic for Programming, Artificial Intelligence, and Reasoning. Springer-Verlag, 2012. 168−182. [doi: 10.1007/978-3-642-28717-6_15]

[21] UPPAAL HELP. 2014. http://www.uppaal.org/

[22] Jensen K, Kristensen LM, Mailund T. The sweep-line state space exploration method. Theoretical Computer Science, 2012,429:169−179. [doi: 10.1016/j.tcs.2011.12.036]

[23] Clarke EM, Zuliani P. Statistical model checking for cyber-physical systems. In: Bultan T, ed. Proc. of the Automated Technology for Verification and Analysis. Springer-Verlag, 2011. 1−12. [doi: 10.1007/978-3-642-24372-1_1]

[24] Buttazzo GC. Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications. 3rd ed., New York: Springer-Verlag, 2011. [doi: 10.1007/978-1-4614-0676-1]

[25] Joseph M, Pandya P. Finding response times in a real-time system. The Computer Journal, 1986,29(5):390−395. [doi: 10.1093/comjnl/29.5.390]

[26] Amnell T, Fersman E, Mokrushin L, Pettersson P, Wang Y. TIMES: A tool for schedulability analysis and code generation of real-time systems. In: Larsen KG, ed. Proc. of the Formal Modeling and Analysis of Timed Systems. Springer-Verlag, 2003. 60−72. [doi: 10.1007/978-3-540-40903-8_6]

[27] Fersman E, Krcal P, Pettersson P, Yi W. Task automata: Schedulability, decidability and undecidability. Information and Computation, 2007,205(8):1149−1172. [doi: 10.1016/j.ic.2007.01.009]

[28] Fersman E, Mokrushin L, Pettersson P, Yi W. Schedulability analysis of fixed-priority systems using timed automata. Theoretical Computer Science, 2006,354(2):301−317. [doi: 10.1016/j.tcs.2005.11.019]

[29] Krcal P, Stigge M, Yi W. Multi-Processor schedulability analysis of preemptive real-time tasks with variable execution times. In: Raskin JF, ed. Proc. of the Formal Modeling and Analysis of Timed Systems. Springer-Verlag, 2007. 274−289. [doi: 10.1007/978-3-540-75454-1_20]

[30] Guan N, Gu ZH, Deng QX, Gao SH, Yu G. Exact schedulability analysis for static-priority global multiprocessor scheduling using model-checking. In: Obermaisser R, eds. Proc. of the Software Technologies for Embedded and Ubiquitous Systems. Springer-Verlag, 2007. 263−272. [doi: 10.1007/978-3-540-75664-4_26]

附中文参考文献:

[3] 李仁发,刘彦,徐成.多处理器片上系统任务调度研究进展评述.计算机研究与发展,2008,45(9):1620−1629.

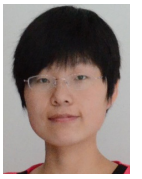
代声馨(1991－),男,四川自贡人,博士生,CCF 学生会员,主要研究领域为嵌入式实时系统,软件形式化验证.

洪玫(1963－),女,教授,CCF 高级会员,主要研究领域为软件工程,软件质量保障与测试.

郭兵(1970－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为嵌入式系统,绿色计算,软件工程.

杨秋辉(1970－),女,博士,副教授,CCF 会员,主要研究领域为软件测试,软件工程.

黄蔚(1990－),女,硕士生,CCF 学生会员,主要研究领域为软件工程,软件测试,模型检测.



徐保平(1989－),女,硕士生,主要研究领域为软件工程,软件测试,模型检测.