软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*,2022,33(9):3512−3528 [doi: 10.13328/j.cnki.jos.006394]

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

# 基于周期虚拟缩减的实时任务调度和分析方法*

刘洪标[1,2], 乔 磊[2], 杨孟飞[3], 陈 熙[2], 马 智[2], 李少峰[1,2]

[1](西安电子科技大学 计算机科学与技术学院, 陕西 西安 710071)

[2](北京控制工程研究所, 北京 100190)

[3](中国空间技术研究院, 北京 100094)

通信作者: 乔磊, E-mail: fly2mars@163.com

**摘 要**: 针对航天器等安全关键系统中实时任务调度和可调度性分析的实际问题, 提出基于任务周期虚拟缩减的可调度性判定方法, 构建 SHT (strong-hard task) 任务模型对强硬实时任务进行精确描述, 并根据任务时间特性分配优先级. 虚拟化所有强实时任务为一个硬实时任务, 对此硬实时任务周期虚拟缩减并计算出其最差虚拟执行时间, 然后按 RMS 可调度性判定公式判定. 给出了判定方法的严格证明, 可对包含 $n$ 个 SHT 任务的任务集进行快速可调度性判定, 此算法时间复杂度仅为 $O(n^2)$. 在我国空间站计算机进行了对比验证, 实验表明判定效率优于现有可调度性判定方法, 平均运行时间开销降低了 41.8%, 可调度率提高了 5.7%.

**关键词**: 动态扩展; 实时任务; 周期虚拟缩减; 优先级分配; 可调度性

**中图法分类号**: TP316

中文引用格式: 刘洪标, 乔磊, 杨孟飞, 陈熙, 马智, 李少峰. 基于周期虚拟缩减的实时任务调度和分析方法. 软件学报, 2022, 33(9): 3512–3528. http://www.jos.org.cn/1000-9825/6394.htm

英文引用格式: Liu HB, Qiao L, Yang MF, Chen X, Ma Z, Li SF. Real-time Task Scheduling and Analysis Method Based on Virtual Zoom Out Period. Ruan Jian Xue Bao/Journal of Software, 2022, 33(9): 3512–3528 (in Chinese). http://www.jos.org.cn/1000-9825/6394.htm

## Real-time Task Scheduling and Analysis Method Based on Virtual Zoom Out Period

LIU Hong-Biao[1,2], QIAO Lei[2], YANG Meng-Fei[3], CHEN Xi[2], MA Zhi[2], LI Shao-Feng[1,2]

[1](School of Computer Science and Technology, Xidian University, Xian 710071, China)

[2](Beijing Institute of Control Engineering, Beijing 100190, China)

[3](China Academy of Space Technology, Beijing 100094, China)

**Abstract**: Regarding the practical problems of the real-time task scheduling and analysis in safety-critical systems such as spacecraft, this study proposes a schedulability determination method based on virtual zoom out period, constructing a strong-hard task (SHT) model to accurately describe real-time tasks, and allocates priority based on task's time characteristics. Virtualizing all strong real-time task as a hard real-time task, virtually reduces the period of the hard real-time task and calculates the worst virtual execution time, and then determines the schedulability according to the RMS schedulability judgment formula. This paper presents a rigorous proof of the method, which can make a fast schedulability determination on an SHT task set containing $n$ tasks, and the time complexity of this algorithm is only $O(n^2)$. Comparative verification was carried out on the China space station computer, and the experiments show that the schedulability determination efficiency is better than the existing methods. The average running time overhead is reduced by 41.8%, and the schedulable ratio is increased by 5.7%.

**Key words**: dynamic expansion; real-time task; virtual zoom out period; priority assignment; schedulability

## 1 引 言

航天器等安全关键无人系统是计算资源受限的嵌入式系统. 随着智能化发展, 在面临外部环境和内部状态变

* 基金项目: 国家自然科学基金 (61632005, 62032004)
收稿时间: 2021-03-31; 采用时间: 2021-06-04; jos 在线出版时间: 2022-05-24

化且不改变硬件资源的情况下, 将对系统功能进行动态扩展和重构、同时保证系统功能正确性和实时性[1]. 在航天器控制系统里, 数据采集、控制计算和控制输出等高安全关键任务具有独特时序要求, 不同于一般的实时任务, 此类任务具有共同的周期, 称为系统控制周期 (system control period), 它们之间后者依赖前者, 需要严格地按照不同的固定时间点启动执行. 而系统动态扩展会外部动态加入新的固定点启动和其他的实时任务, 在这一过程中, 带来任务动态加入的调度和可调度性分析问题. 目前大多数航天器不具备在线分析能力, 调度策略为基于表的固定点调度策略[2], 即地面事先分析确定好所有任务执行时序, 所有任务周期相同且长度固定、任务在固定时间点启动、任务顺序执行, 这种方式不能支持新任务的动态加入, 特别是加入的任务种类具有多样性 (强实时任务、硬实时任务、软实时任务、最大努力任务) 以及加入任务数量具有不确定性. 随着多种类型任务数量的增加, 地面已无法离线分析和设计航天器任务的具体运行情况并且难以远距离实时更新航天器系统, 必须要求航天器在线进行新加入任务的优先级分配和快速可调度性分析, 使得航天器具备自主智能性, 同时保证系统的确定性和实时性.

针对上述应用场景, 本文结合国内外最新研究进展, 对其中的关键问题进行了深入研究, 研究的主要贡献与创新点概括如下.

(1) 建立了 SHT (strong-hard task) 任务模型, 精确描述了强实时任务和硬实时任务时间特性, 为可调度性分析奠定了基础.

(2) 提出了强实时任务和硬实时任务的优先级分配方法, 保证强实时任务不被抢占情况下的计算资源利用的最大化.

(3) 提出一种基于虚拟周期缩减的可调度性判定 VZOP (virtual zoom out period) 算法, 解决了强实时任务和硬实时任务动态加入的调度和分析问题, 在线对新加入任务进行优先级分配和可调度性判定, 保证了系统的实时性. 从理论上严格证明了此判定算法的可行性, 且此算法时间复杂度为 $O(n^2)$, 实验表明, 判定效率优于现有可调度性判定方法.

本文第 1 节介绍背景, 第 2 节介绍相关工作, 第 3 节对强实时任务和硬实时任务进行数学模型描述, 第 4 节对任务进行优先级分配, 第 5 节详细表述了基于虚拟周期缩减思想的可调度性分析算法和对其进行了性能分析, 并给出了其可行性的理论证明, 第 6 节是实验及结果分析, 将该算法与已有算法进行了运行时间开销和可调度率指标对比, 最后一节是对本文的工作总结.

## 2 研究背景

目前实时操作系统调度问题主要分为调度策略和可调度性判定. 在调度策略方面, 主要分为时钟驱动调度、优先级调度和轮转调度[3]. Liu 等人针对硬实时任务的调度和可调度性判定问题, 提出了 RMS (rate-monotonic scheduling) 单调速率调度算法[4], 属于固定优先级调度, 根据任务周期的大小 (代表任务请求速率) 来分配任务优先级, 是固定优先级分配方式中最优调度算法, 给出了硬实时任务集的可调度性判定充分性条件: $\sum_{i=1}^{n} C_i/T \leqslant n\left(\sqrt[n]{2}-1\right)$, $n$ 为任务集中的任务个数, 时间复杂度仅为 $O(n)$, 为实时调度理论发展奠定了基础. Abdelzaber 等人提出了 DM (deadline monotonic) 单调截止时间算法[5], 证明为约束截止时间偶发任务最优调度算法, 任务截止期越短, 分配的优先级越高, synthetic 利用率上界是 $1/(1+\sqrt{1/2})$, 可用于开放式系统运行时的在线分析, 此外还证明了动态优先级调度算法 EDF (earliest deadline first) 的利用率上界为 1, 是最优的动态优先级调度算法. Meng 等人针对高安全关键任务的调度, 在基于多核 653 调度框架的系统中, 提出了基于权重轮转的调度方法[6], 任务按照利用率比例分配权值, Jo 等人其后对其调度开销作了定量分析[7].

在可调度性判定方面, 实时任务集的可调度性判定, 是经典的理论难题, Ekberg 等人通过多项式时间规约到 EDF 调度问题的方式, 证明了偶发任务采用固定优先级调度的可调度性判定同样是 NP-hard 问题[8]. 响应时间分析 RTA (response time analysis) 是最常用的可调度性分析方法[9,10], 能精确的给出任务集的可调度性判定, 属于可调度性判定的充要条件, 它通过计算任务最差响应时间 WCRT (worst-case response time), 与任务截止时间相比较, 判断任务的可调度性, 对固定优先级抢占式调度的响应时间分析, 使用迭代公式: $R_i = C_i + \sum_{j<i} \lceil R_i/T_j \rceil \times C_j$, 通过对

$R_i$ 和 $D_i$ 的比较, 判定出任务是否可调度, 响应时间分析由文献 [9,10] 给出, 得到了学术界广泛的研究和工业界的应用, 该方法时间复杂度较高, 当时间参数为有理数时, 为 NP-hard 问题[11], 时间参数为整数情况下, 时间复杂度为 $O(n^2 \times D_{\max})$, 其中 $n$ 为任务集的任务数, $D_{\max}$ 为任务集中截止期的最大值[12], 在后续研究中进一步扩展, 使其包含阻塞[13,14], 释放抖动[15], 任意截止时间[16,17]等. 关于可调度性测试效率问题, 其后更多研究都是通过找到一个响应时间分析里用到的更好的初值来提高测试速度[18−20]. BCL (Bertogna M, Cirinei M, Lipari G) 算法[21]是另外一种高效率的可调度性判定方法, 能近似的给出任务集的可调度性判定, 属于可调度性判定的充分条件, 它通过计算每个任务的周期内所有高优先级任务造成的抢占量, 从而判断任务是否可调度, 时间复杂度为 $O(n^2)$. Baruah 等人提出了偶发并行任务的有向无环图模型 DAG (directed acyclic graph)[22], 采用有向无环图表示一个任务内部的并行状态与时序依赖关系. Bonifaci 等人给出了该模型在多处理器上可调度性分析的充分条件, 然而其利用率上界在单核中仅 0.25[23]. Baruah 等人证明了时序依赖任务的利用率上界不超过 $\epsilon$ 的情况下, 其可调度性问题是强 co-NP-hard 问题, $\epsilon$ 是一个任意小的常数[24,25]. 黄姝娟等人提出了一种基于 ST (simple tree) 的实时周期任务调度模型[26], 该模型用数据结构树来维护任务之间的依赖关系, 有效地提高系统利用率以及降低截止期错失率, 然而该方法假设 ST 树与树之间无任何关联关系且所选算法还是存在任务截止期错失情况, 限制了其实际应用. 孙景昊等人针对独立或者时序依赖的偶发任务, 指出固定优先级调度方式实现简单, 但是处理器利用率相比动态优先级调度方法较低, 于是采用动态优先级方式调度偶发任务, 使用整数规划方法对系统进行可调度性分析, 实验结果表明在动态优先级调度中采用该方法分析结果更为精确[27]. Zhang 等人指出偶发任务采用固定优先级非抢占式 FPNS (fixed priority non-preemptive scheduling) 调度方式具有实现简单, 可预测性好的优势, 提出了最长截止期优先 IA-LDF (longest deadline first) 算法[28], 采用响应时间 RTA 分析方法对该算法进行可调度性分析, 实验表明, 该算法一定程度上提高了调度效率, 然而并未考虑存在时序依赖和固定点调度等任务的情况.

综上, 大部分研究工作主要针对硬实时任务进行可调度性分析, 未考虑系统中存在时序依赖和固定点调度等任务的可调度性分析问题, 或者处理器利用率上界过低而不能适用于计算资源受限的航天器等安全关键系统的实际要求. 另一方面, 已有分析方法在对任务进行可调度性分析的时间复杂度较高, 对实时性要求较高的航天器系统也不适用, 难以满足在线分析要求.

## 3 任务模型

实时任务具有严格的截止期约束, 截止期错失将导致灾难性后果[3]. 在航天器控制系统里, 存在数据采集、控制计算和控制输出等高安全关键任务, 不同于一般的实时任务, 此类任务具有共同的周期 $T_c$, 称为系统控制周期 (system control period), 它们之间后者依赖前者, 需要严格地按照不同的固定时间点启动执行, 由于截止期等于最差执行时间, 因此执行过程中不可被其他任务抢占, 才不会造成截止期错失, 这类任务我们称为强实时任务 (strong real-time task). 而其他实时任务具有各自的周期, 任务间无固定执行顺序, 任意时刻到达, 保证在其截止期前能完成即可, 这类任务我们称为硬实时任务 (hard real-time task).

由于经典的周期任务模型 (periodic) 无法精确刻画出强实时任务的时间特性, 所以本文在周期任务模型[4]的基础上增加了任务初始到达偏移 (initial arrival offset) $A_i$ 参数, 表示任务 $\tau_i$ 初始到达时刻与系统 0 时刻的差值, 其中 $0 \leqslant A_i < T_i$, 成为了新的任务模型: 强硬实时任务模型 (strong hard task model, SHT 模型), 具体如图 1 所示, $C_i$ 表示任务 $\tau_i$ 的最坏执行时间, $D_i$ 表示 $\tau_i$ 的相对截止时间, $T_i$ 表示 $\tau_i$ 的周期, 任务的一次到达称为任务的一个实例 (task instance), $\tau_i^{j+1}$ 表示任务 $\tau_i$ 的第 $j$+1 个实例, 实例 $\tau_i^j$ 到达时刻是 $A_i + jT_i$, 其中 $j \in N$ . 根据 $D_i$ 与 $T_i$ 的相对关系, 若 $D_i = T_i$ , 则 $\tau_i$ 称为隐式截止时间 (implicit deadline) 任务, 若 $D_i \leqslant T_i$ , 则称 $\tau_i$ 为约束截止时间 (constrained deadline) 任务, 若 $D_i > T_i$ , 则称 $\tau_i$ 为非约束截止时间 (arbitrary deadline) 任务. 如果任务集 $S$ 中所有任务初始到达偏移相等, 即 $A_i = A_j$, 其中 $\tau_i, \tau_j \in S$ , 则 SHT 任务模型退化为经典的周期任务模型. 所以 SHT 任务 $\tau_i$ 可以表示为一个四元组 $(T_i, C_i, D_i, A_i)$.

任务 $\tau_i$ 响应时间 $R_i$ 表示任务完成时刻与到达时刻的时间差. 令 $\Delta t$ 表示 $\tau_i$ 到达时刻到完成时刻区间内被其他高优先级任务抢占的时间, 则:

$$R_i = C_i + \Delta t \tag{1}$$

任务$\tau_i$称为可调度的 (schedulable) 是指它的每个实例能在截止期前完成, 即:

$$R_i \leqslant D_i \tag{2}$$

实时任务集$S_R$称为可调度的是指$S_R$内所有任务都是可调度的, 即:

$$\forall \tau_i \in S_R, R_i \leqslant D_i \tag{3}$$

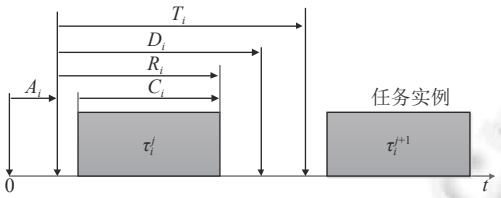


图 1 SHT 任务模型

任务$\tau_i$的处理器利用率$u_i$表示为最差执行时间$C_i$与周期$T_i$的比值, 即$u_i = C_i/T_i$.

本文所考虑的实时任务$\tau_i$用 SHT 任务模型表述, $n$ 个 SHT 任务构成的实时任务集为$S_R = \{\tau_1, \tau_2, \ldots, \tau_n\}$, 任务间无共享资源访问, 按其时间特性不同, 实时任务可以分为两类: 强实时任务和硬实时任务, 其对应的任务集分别表示为$S_{St}, S_H$, 即$S_R = S_{St} + S_H$.

(1) 强实时任务, 指具有约束截止时间, 共同的周期, 且执行过程不可被抢占, 满足:

$$D_i = C_i, D_i < T_i, T_i = T_j, A_i \neq A_j, \tau_i, \tau_j \in S_{St} \tag{4}$$

(2) 硬实时任务, 指具有隐式截止时间, 满足$D_i = T_i, A_i = A_j, \tau_i, \tau_j \in S_H$.

## 4 优先级分配策略

优先级调度策略[29,30]可以分为固定优先级调度和动态优先级调度, 固定优先级调度又分为固定优先级非抢占式调度 (FPNS), 固定优先级抢占式调度 (FPPS) 和固定优先级延迟抢占调度 (FPDS). 本文采用固定优先级抢占式调度 (FPPS), 所有强实时任务具有相同周期, 不同的初始到达偏移, 由于其截止期等于最差执行时间, 因此任务到达必须立刻执行, 而且在执行过程不能被其他任务抢占, 否则会错过截止期, 所以强实时任务相较于硬实时任务分配的优先级更高. 强实时任务按照任务初始到达偏移分配优先级, 初始到达偏移越小, 分配优先级越高, 反之则越低. 所有硬实时任务的周期不相等, 优先级采用 RMS 算法[4]分配方式, 即任务的周期 $T$ 越小, 分配的优先级越高, 反之则越低.

令$\pi_i$表示任务$\tau_i$的优先级, $\pi_i$越大, 表示任务$\tau_i$的优先级越高. 如果$\pi_j, \pi_k \in S_{St}$, 有$\pi_j > \pi_k$当且仅当$A_j < A_k$. 如果$\pi_j, \pi_k \in S_H$, 有$\pi_j > \pi_k$当且仅当$T_j < T_k$. 由于强实时任务比硬实时任务分配更高优先级, 因此保证了强实时任务固定点启动, 结合[4]中定理 2 可知, 在满足强实时任务不错过截止期情况下, 最大化的利用了处理器计算资源.

例 1: 如图 2 所示, 实时任务集$S_R$包括 4 个任务, $\tau_1 = (25,4,4,0), \tau_2 = (25,3,3,13), \tau_3 = (13,2,13,0), \tau_4 = (24,4,24,0), \tau_1, \tau_2 \in S_{St}$, 为强实时任务, 具有共同的系统控制周期$T_c = T_1 = T_2 = 25, \tau_1$截止期$D_1 = C_1 = 4$, 初始到达偏移$A_1 = 0, D_2 = C_2 = 3$, 初始到达偏移$A_2 = A_1 + 13 = 13, \tau_3, \tau_4 \in S_H$, 为硬实时任务, $\tau_3$周期$T_3 = 13$, 截止期$D_3 = T_3 = 13$, 初始到达偏移$A_3 = 0, \tau_4$周期$T_4 = 24$, 截止期$D_4 = T_4 = 24$, 初始到达偏移$A_4 = A_3 = 0$. 在 0 时刻, $\tau_1, \tau_3, \tau_4$同时到达, $\tau_2$在 13 时刻到达.

由于$\tau_1, \tau_2$为强实时任务, $\tau_3, \tau_4$为硬实时任务, 根据优先级分配规则, $\tau_1, \tau_2$优先级都高于$\tau_3, \tau_4$. 强实时任务中, 初始偏移时刻$A_1 < A_2$, 因此优先级$\pi_1 > \pi_2$; 硬实时任务中, 任务周期$T_3 < T_4$, 因此优先级$\pi_3 > \pi_4$. 因此, 总体优先级分配结果$\pi_1 > \pi_2 > \pi_3 > \pi_4$, 即任务$\tau_1, \tau_2, \tau_3, \tau_4$优先级依次降低.

在 0 时刻, $\tau_1, \tau_3, \tau_4$同时到达, 由于$\tau_1$优先级最高, 因此抢占处理器, 先执行至当前实例结束, 期间$\tau_3, \tau_4$处于就绪状态; 在时刻 4, 接着低优先级任务$\tau_3$执行至当前实例结束; 在时刻 6, 接着更低优先级任务$\tau_4$执行至当前实例

结束; 在 (10, 13) 时间, 无就绪任务, 处理器空闲; 在时刻 13, $\tau_2,\tau_3$到达, 由于$\tau_2$优先级更高, 因此抢占处理器, $\tau_2$执行至当前实例结束; 在时刻 16, 接着更低优先级任务$\tau_3$执行至当前实例结束; 在 (18, 24) 时间, 无就绪任务, 处理器空闲; 在时刻 24, $\tau_4$到达, 执行到时刻 25, 高优先级任务$\tau_1$到达, 立刻抢占处理器, 执行至时刻 26, 低优先级$\tau_3$到达, 但是此时就绪任务中$\tau_1$仍然是优先级最高的, 因此任务$\tau_1$继续执行至当前实例结束; 在时刻 29, 系统中就绪任务有任务$\tau_3,\tau_4$, 然而$\tau_3$优先级更高, 因此$\tau_3$执行至当前实例结束; 在时刻 31, 仅最低优先级的$\tau_4$处于就绪状态, 因此任务$\tau_4$继续执行至当前实例结束; 在 (34, 38) 时间, 无就绪任务, 处理器空闲; 处理器依此反复执行, 可以看到强实时任务$\tau_1,\tau_2$一旦到达, 总是能立刻得到执行, 且执行过程中不会被其他任务抢占, 因此满足了其时序要求, 硬实时任务$\tau_3,\tau_4$也能最大化利用了处理器资源.

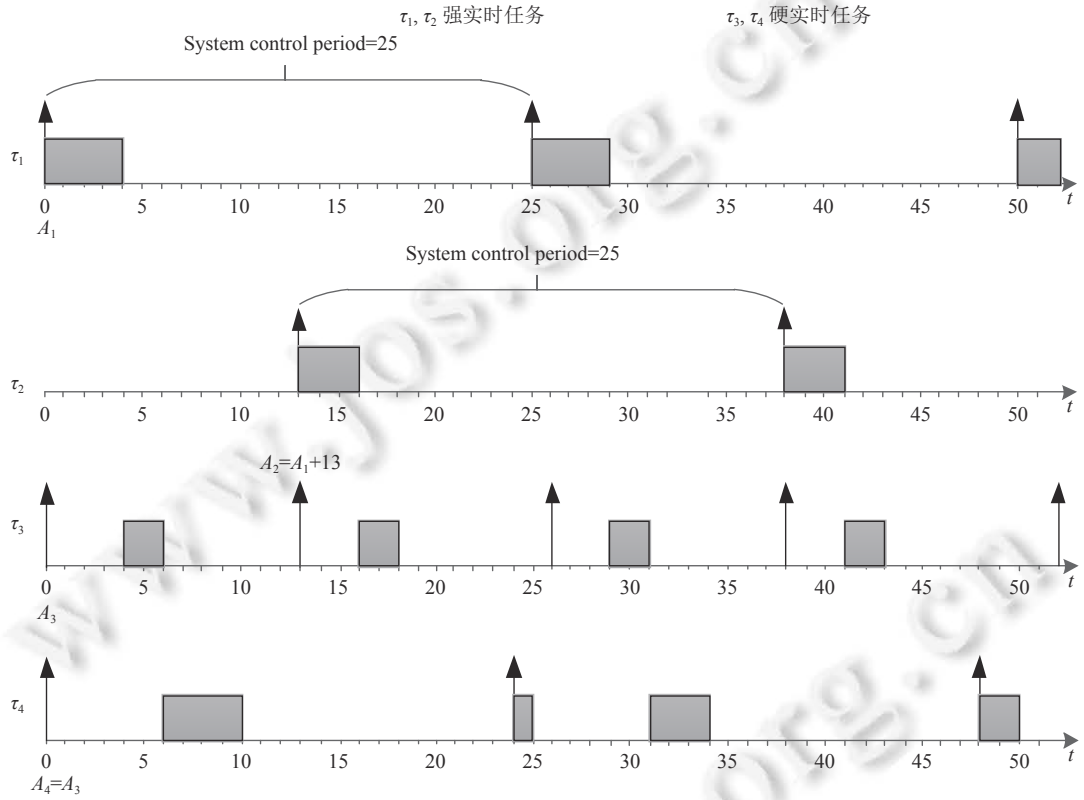


图 2 强实时任务和硬实时任务

## 5 基于周期虚拟缩减的可调度性分析方法

由于强实时任务的系统控制周期可能大于硬实时任务周期, 无法直接使用 RMS 的可调度性判定方法. 本文将强实时任务虚拟化为一个硬实时任务, 然后对周期进行虚拟缩减, 此时的实时任务集里全部是硬实时任务, 且优先级分配符合 RMS 规则, 然后使用其可调度性判定方法即可.

### 5.1 基本概念

**定义 1**. 虚拟缩减周期 (virtual zoom out period) $VZOP_{S_t}$: 表示强实时任务集$S_{S_t}$中所有任务虚拟化为一个硬实时任务, 然后其周期缩减后的长度.

**定义 2**. 最差虚拟执行时间 (worst virtual execution time) $WVET_{S_t}(S_{S_t},VZOP_{S_t})$: 表示强实时任务集$S_{S_t}$虚拟缩减为一个周期$VZOP_{S_t}$的硬实时任务对应的最差执行时间. 计算步骤描述如下:

(1) 将$S_{St}$合并为一个硬实时任务.

(2) 计算区间长度$VZOP_{St}$的任意区间内的最大负载请求, 显然, 仅有可能以$A_i$作为区间起始点的区间内取得此最大负载请求, 因此, 以$A_i$作为区间起始点, $A_i+VZOP_{St}$作为区间结束点, 令$I_i=[A_i,A_i+VZOP_{St}]$, 划分出$|S_{St}|$个区间, 其中$\tau_i\in S_{St}$,$|S_{St}|$表示$S_{St}$的任务数.

(3) 计算出任务$\tau_j$对区间$I_i$的负载请求$E_j$, 分两种情况:

① $i>j$: $E_j=\min\left\{\dfrac{A_i+VZOP_{St}-A_j-T_c+|A_i+VZOP_{St}-A_j-T_c|}{2},C_j\right\},\tau_j\in S_{St}$.

② $i\leqslant j$: $E_j=\min\left\{\dfrac{A_i+VZOP_{St}-A_j+|A_i+VZOP_{St}-A_j|}{2},C_j\right\},\tau_j\in S_{St}$.

其中, $T_c$表示系统控制周期, 令$f(i,j)=\begin{cases}0,i\leqslant j\\1,i>j\end{cases}$, 因此表示为:

$$E_j=\min\{\frac{A_i+VZOP_{St}-A_j-f(i,j)\times T_c+|A_i+VZOP_{St}-A_j-f(i,j)\times T_c|}{2},C_j\},\tau_j\in S_{St}.$$

(4) 累加所有任务$\tau_j$对$I_i$的负载请求, 得到$I_i$的总负载请求$W_i=\sum_{j=1}^{|S_{St}|}E_j$.

(5) 计算所有区间$I_i$的负载请求最大值, 即为最差虚拟执行时间$WVET_{St}=\max\limits_{1\leqslant i\leqslant|S_{St}|}W_i$.

因此其计算公式为:

$$WVET_{St}=\max_{1\leqslant i\leqslant|S_{St}|}\sum_{j=1}^{|S_{St}|}\min\{\frac{A_i+VZOP_{St}-A_j-f(i,j)\times T_c+|A_i+VZOP_{St}-A_j-f(i,j)\times T_c|}{2},C_j\},\tau_j\in S_{St}.$$

最差虚拟执行时间计算方法如算法 1.

**算法 1**. $WVET_{St}(S_{St},VZOP_{St})$.

输入: 强实时任务集$S_{St}$, 虚拟缩减周期$VZOP_{St}$;

输出: 强实时任务集$S_{St}$的控制周期虚拟缩减为$VZOP_{St}$后对应的最差虚拟执行时间.

```
1. res = 0
2. FOR i : |S_St|
3.     W_i = 0 //每个区间的负载请求累加值
4. FOR τ_j : S_St
5.     E_j = WVET_St
6.     W_i+ = E_j //每个任务对当前区间的负载请求值
7. IF res < W_i //求出所有区间的负载请求累加值的最大值
8.     res = W_i
9. RETURN res
```

例 2: 如图 3 所示. $\tau_1=(25,4,4,0),\tau_2=(25,3,3,13),\tau_1,\tau_2\in S_{St}$. 假设虚拟缩减周期$VZOP_{St}=13$, 将强实时任务$\tau_1,\tau_2$虚拟为一个硬实时任务$\tau_0=(T_0,C_0,D_0,A_0),\tau_0\in S_H$, 周期$T_0=VZOP_{St}=13$. 第一步将所有强实时任务虚拟化为一个硬实时任务$\tau_0$, 划分为两个区间$I_1:[0,13],I_2:[13,26]$, 计算$\tau_1,\tau_2$对区间$I_1$的负载请求$E_1=4,E_2=0$, 因此区间$I_1$总的负载请求$W_1=E_1+E_2=4$. 计算$\tau_1,\tau_2$对区间$I_2$的负载请求$E_1=1,E_2=3$, 因此区间$I_2$总的负载请求$W_2=E_1+E_2=4$. 求区间$I_1,I_2$总的负载请求最大值, 即为最差虚拟执行时间, $C_0=WVET_{st}(S_{St},13)=\max\{W_1,W_2\}=4$. 因此所有强实时任务虚拟缩减为硬实时任务$\tau_0$, 最差执行时间$C_0=4$, 截止期$D_0=T_0=13$, 初始到达偏移$A_0=0,\tau_0=(13,4,13,0),\tau_0\in S_H$.

### 5.2 可调度性证明

固定优先级调度中, 仅高优先级任务会对低优先级任务造成抢占延迟, 从而对低优先级任务的可调度性产生

影响. 因此对当前任务的可调度性分析中, 仅需考虑高优先级任务的抢占影响.

**定理 1**. 强实时任务集$S_{St}=\{\tau_1,\tau_2,\ldots,\tau_n\}$, 其中, $0\leqslant A_1<A_2<\ldots<A_n<T_c$, 若$\forall\tau_i\in S_{St},i\neq 1,A_i\geqslant A_{i-1}+D_{i-1}$, 则$S_{St}$一定可调度.

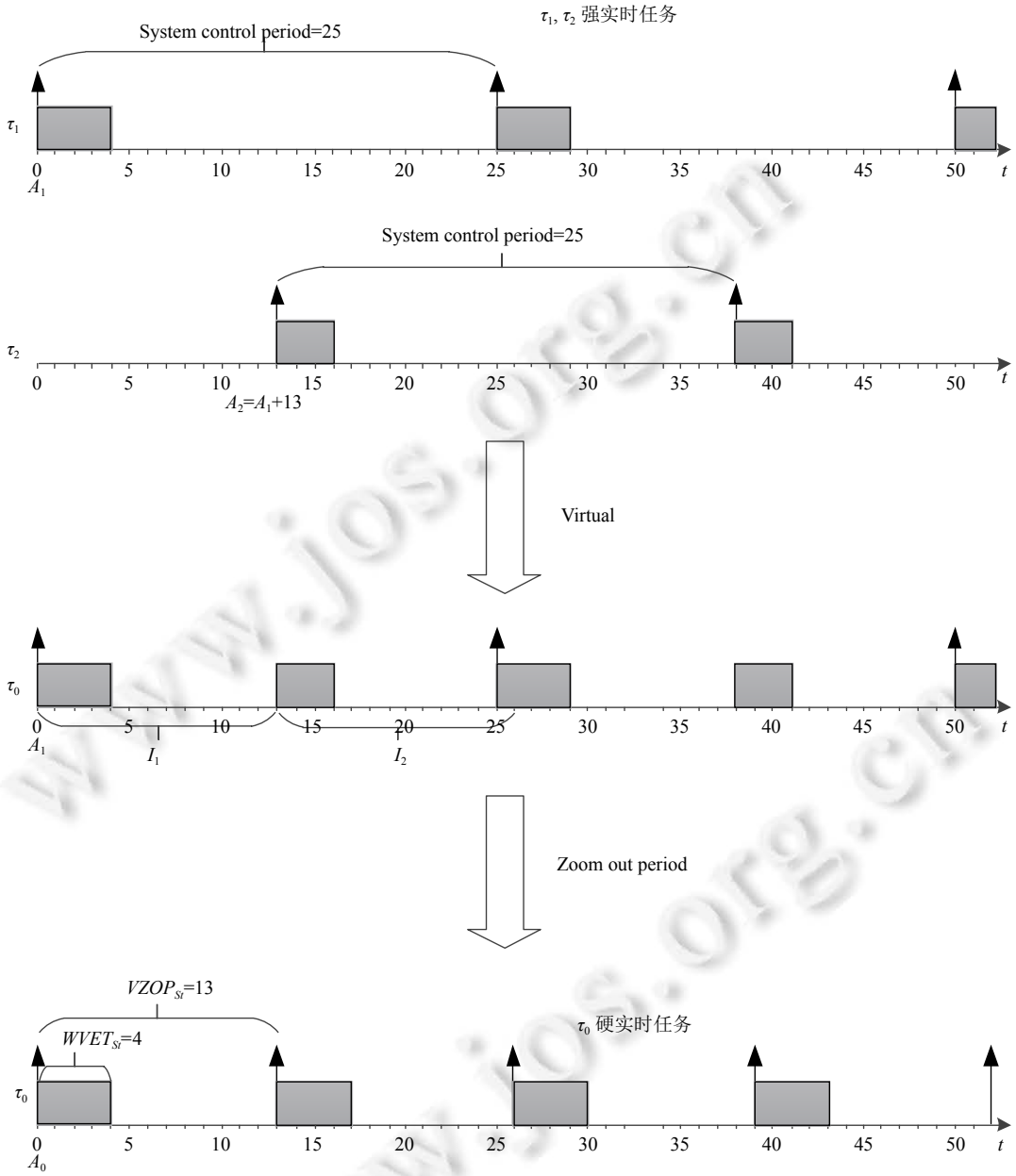


图 3　强实时任务集虚拟缩减为一个硬实时任务

证明: 根据优先级分配规则可知, 有$\pi_1>\pi_2>\ldots>\pi_n$. 对于$\forall\tau_i\in S_{St}$的可调度性分析, 采用数学归纳法证明.

(1) 当$i=1$时, 根据强实时任务特性公式 (4), 有:

$$D_1=C_1 \tag{5}$$

由于$\tau_1$为最高优先级任务, 所以到达后, 立刻执行, 且执行过程中不会被其他任务抢占, 抢占时间$\Delta t=0$, 结合

公式 (1), 因此:

$$R_1 = C_1 \tag{6}$$

结合公式 (5)、公式 (6) 有 $R_1 = D_1$, 根据任务可调度性判定公式 (2) 可知, $\tau_1$ 是可调度的.

(2) 假设 $\forall \tau_i \in S_{St}, k < n, i = k$ 时, $\tau_i$ 是可调度的, 根据公式 (2) 即有:

$$R_k \leqslant D_k \tag{7}$$

(3) 当 $i = k+1$ 时, 根据已知条件 $A_{k+1} \geqslant D_k$, 结合公式 (7) 有 $A_{k+1} \geqslant R_k$, 表明更高优先级任务 $\tau_k$ 已执行完成, $\tau_{k+1}$ 才到达, 且 $\pi_{k+1}$ 为当前最高优先级, 因此释放后会立刻执行, 且执行过程中不会被抢占, 抢占时间 $\Delta t = 0$, 结合公式 (1), 因此:

$$R_{k+1} = C_{k+1} \tag{8}$$

根据强实时任务特性公式 (4), 有:

$$D_{k+1} = C_{k+1} \tag{9}$$

结合公式 (8)、公式 (9) 有 $R_{k+1} = D_{k+1}$, 根据任务可调度性判定公式 (2) 可知, $\tau_{k+1}$ 是可调度的.

因此 $\forall \tau_i \in S_{St}, R_i \leqslant D_i$, 根据任务集可调度性判定公式 (3) 可知, 任务集 $S_{St}$ 一定可调度, 证毕.

**引理 1**. 任务集 $S_H = \{\tau_1, \tau_2, \ldots, \tau_n\}$, 其中 $T_1 < T_2 < \ldots < T_n$, 采用 RMS 调度算法, 若此任务集可调度, 当最高优先级任务 $\tau_1$ 执行期间自挂起, 即 $C_1$ 分为不连续的 $p$ 段 $C_{1,1}, C_{1,2}, \ldots, C_{1,p}, p \in N^+$, 满足 $C_1 = C_{1,1} + C_{1,2} + \ldots + C_{1,p}$, 此时任务集 $S_H$ 仍可调度.

证明: 根据优先级分配规则可知, 有 $\pi_1 > \pi_2 > \ldots > \pi_n$, 将 $\tau_1$ 执行效果等效分解为 $p$ 个周期同为 $T_1$ 的子任务 $\tau_{1,1} = (T_1, C_{1,1}, C_{1,1}, 0), \ldots, \tau_{1,p} = (T_1, C_{1,p}, C_{1,p}, A_{1,p}), \tau_{1,1}, \ldots, \tau_{1,p} \in S_{St}, A_{1,i} + C_{1,i} \leqslant A_{1,i+1} \leqslant T_1 - C_{1,p}, 1 < i < p, \tau_{1,1}, \ldots, \tau_{1,p}, \tau_2, \ldots, \tau_n$ 构成新实时任务集 $S'_R$, 优先级分配 $\pi_{1,1} > \ldots > \pi_{1,p} > \pi_2 > \ldots > \pi_n$. 根据文献 [4] 中的定理 1, 任务的临界时刻发生在所有优先级高于该任务的任务同时发生请求的时刻, 所有高优先级任务同时到达会造成低优先级任务响应时间达到最长. 由于 $S'_R$ 中 $\tau_{1,i}$ 不再与低优先级任务 $\tau_2, \ldots \tau_n$ 同时到达, 不存在临界时刻, 因此造成 $\tau_2, \ldots, \tau_n$ 的响应时间更短, $\tau_2, \ldots, \tau_n$ 一定仍然可调度. 结合定理 1 可知, $\tau_{1,1}, \ldots, \tau_{1,p}$ 一定可调度. 综上, 新任务集 $S'_R$ 可调度, 而 $\tau_1$ 执行效果与其子任务 $\tau_{1,1}, \ldots, \tau_{1,p}$ 相同, 因此原任务集 $S_H$ 仍可调度, 证毕.

**定理 2**. 实时任务集 $S_R = \{\tau_1, \tau_2, \ldots, \tau_n, \tau_{n+1}, \ldots, \tau_{n+m}\}$, $\tau_1, \ldots \tau_n \in S_{St}, A_1 < \ldots < A_n$, $S_{St}$ 已可调度, $\tau_{n+1}, \ldots, \tau_{n+m} \in S_H$, $T_1 = \ldots = T_n = T_c < T_{n+1} < \ldots < T_{n+m}$, 若

$$\forall \tau_i \in S_R, \sum_{i=1}^{i=n+m} C_i / T_i \leqslant (m+1)(\sqrt[m+1]{2} - 1),$$

则 $S_R$ 一定可调度.

证明: 根据优先级分配规则可知, 有 $\pi_1 > \pi_2 > \ldots > \pi_{n+m}$. 将所有强实时任务 $\tau_1, \tau_2, \ldots, \tau_n$ 执行效果等效虚拟为一个硬实时任务 $\tau_0 = (T_0, C_0, D_0, A_0), T_0 = T_c, C_0 = \sum_{k=1}^{k=n} C_k, D_0 = T_0, A_0 = 0$. 令 $S'_H = \{\tau_0, \tau_{n+1}, \ldots, \tau_{n+m}\}$, 全为硬实时任务, 由已知 $T_0 < T_{n+1} < \cdots < T_{n+m}$, 符合 RMS 调度算法规则, 根据 RMS 算法可调度性判定公式可知, 若 $\sum_{\tau_i \in S'_H} C_i / T_i \leqslant (m+1)(\sqrt[m+1]{2} - 1)$, 则 $S'_H$ 是可调度的. 由引理 1 可知, $\tau_1, \tau_2, \ldots, \tau_n$ 虚拟化为 $\tau_0$ 前, $S_R$ 一定也是可调度的. 综上, 因此 $S_R$ 一定可调度, 证毕.

**定理 3**. 实时任务集 $S_R = \{\tau_1, \tau_2, \ldots, \tau_n, \tau_{n+1}, \ldots, \tau_{n+m}\}$, $\tau_1, \ldots, \tau_n \in S_{St}, A_1 < \ldots < A_n$, $S_{St}$ 已可调度, $\tau_{n+1}, \ldots, \tau_{n+m} \in S_H$, $T_{n+1} < \ldots < T_{n+m}, T_{n+1} < T_1 = \ldots = T_n = T_c$, 若:

$$\forall \tau_i \in S_R, \frac{WVET_{St}(S_{St}, VZOP_{St})}{VZOP_{St}} + \sum_{i=n+1}^{i=n+m} C_i / T_i \leqslant (m+1)\left(\sqrt[m+1]{2} - 1\right),$$

则 $S_R$ 一定可调度.

证明: 根据优先级分配规则可知, 有 $\pi_1 > \pi_2 > \ldots > \pi_{n+m}$. 将所有强实时任务 $\tau_1, \tau_2, \ldots, \tau_n$ 通过周期虚拟缩减方式虚拟为一个硬实时任务 $\tau_0 = (T_0, C_0, D_0, A_0), T_0 = VZOP_{St} = T_{n+1}, C_0 = WVET_{St}(S_{St}, VZOP_{St}), D_0 = T_0, A_0 = 0$. 令 $S'_H = \{\tau_0, \tau_{n+1}, \ldots, \tau_{n+m}\}$, 全为硬实时任务, 由已知 $T_0 \leqslant T_{n+1} < \ldots < T_{n+m}$, 符合 RMS 调度算法规则, 根据 RMS 算法

可调度性判定公式可知, 若 $\sum_{\tau_i \in S'_H} C_i/T_i \leqslant (m+1)(\sqrt[m+1]{2}-1)$, 则 $S'_H$ 是可调度的. 由算法 1 可知, 周期缩减过程是搜索该周期长度区间内的最大执行时间, 因此新的硬实时任务 $\tau_0$ 总能满足原强实时任务 $\tau_1,\tau_2,\ldots,\tau_n$ 的负载请求, 结合引理 1 可知, 原实时任务集 $S_R$ 一定也是可调度的, 证毕.

### 5.3 可调度性分析算法

系统原实时任务组成的集合为 $S_{R1}=\{\tau_1,\tau_2,\ldots,\tau_n\}$, 新加入的实时任务集合 $S_{R2}=\{\tau_{n+1},\tau_{n+2},\ldots,\tau_{n+m}\}$, 共同组成新的任务集合 $S_{R3}=\{\tau_1,\tau_2,\ldots,\tau_n,\tau_{n+1},\tau_{n+2},\ldots,\tau_{n+m}\}$, 按照优先级分配策略分配所有任务的优先级, 其中 $\forall\tau_i\in S_{R3},\tau_i=(T_i,C_i,D_i,A_i),\pi_1>\pi_2>\ldots>\pi_{n+m}$. 任务集 $S_{R2}$ 加入 $S_{R1}$ 时的可调度性分析算法为 *AdmissionControl*, 其算法步骤描述如下:

(1) 将 $S_{R3}$ 分为 $S_{St}$ 和 $S_H$ 两个集合, 其中 $S_{st}=\{\tau_1,\tau_2,\ldots,\tau_p\}$, 表示强实时任务集合, 满足 $A_1\leqslant A_2\leqslant\ldots\leqslant A_p, S_H=\{\tau_{p+1},\tau_{p+2},\ldots,\tau_{n+m}\}$, $1\leqslant p\leqslant n+m$, 表示硬实时任务集合.

(2) 在 $S_{St}$ 中从前往后依次选取任务 $\tau_j, j\in[1,p]$, 若选取结束, 说明强实时任务集 $S_{St}$ 可调度 (见定理 1), 则转步骤 4, 否则转步骤 (3);

(3) 如果 $\tau_{j+1}$ 的初始到达偏移 $A_{j+1}$ 大于等于 $A_j+D_j$, 即 $A_{j+1}\geqslant A_j+D_j$, 转步骤 (2). 否则任务集准入失败, 算法结束;

(4) 如果集合 $S_H$ 为空集, 则新的任务集合 $S_{R3}$ 满足可调度性, 返回任务集 $S_{R2}$ 准入成功, 算法结束. 否则转步骤 (5);

(5) $S_H$ 中找出周期最小的任务的周期 $T_{\min}$, 如果 $T_{\min}\geqslant T_c$, 转步骤 (6), 否则转步骤 (7);

(6) 如果 $\forall\tau_k\in S_{R3}, \sum_{k=1}^{n+m}\frac{C_k}{T_k}\leqslant(n+m-p+1)(\sqrt[n+m-p+1]{2}-1)$, 说明实时任务集 $S_{R3}$ 可调度 (见定理 2), 则返回任务集 $S_{R2}$ 准入成功, 算法结束. 否则, 返回任务集 $S_{R2}$ 准入失败, 算法结束.

(7) 设定 $S_{St}$ 的虚拟缩减周期 $VZOP_{St}=T_{\min}$, 计算出最差虚拟执行时间 $WVET_{St}(S_{St},VZOP_{st})$, 如果 $\forall\tau_k\in S_H$, $\frac{WVET_{St}(S_{St},VZOP_{St})}{VZOP_{St}}+\sum_{k=p+1}^{n+m}\frac{C_k}{T_k}\leqslant(n+m-p+1)(\sqrt[n+m-p+1]{2}-1)$, 说明实时任务集 $S_{R3}$ 可调度 (见定理 3), 则返回任务集 $S_{R2}$ 准入成功, 算法结束. 否则, 返回任务集 $S_{R2}$ 准入失败, 算法结束.

可调度性分析算法伪代码如算法 2.

**算法 2**. BOOL *AdmissionControl*($S_{R2}$,$S_{R1}$).

输入: 新加入的实时任务集 $S_{R2}$, 原实时任务集 $S_{R1}$;

输出: 任务集 $S_{R2}$ 动态加入的可调度性分析结果, TRUE 或者 FALSE.

```
1. S_R3 = S_R1 + S_R2
2. Split(S_R3, S_St, S_H) //将总任务集S_R3分为强实时任务集S_St和硬实时任务集S_H
3. FOR τ_j : S_St
4.    IF A_{j+1} < A_j + D_j
5.       RETURN FALSE
6.    IF |S_H| == 0
7.       RETURN TRUE
8.    T_min = FindMin(S_H) //找出S_H中最小周期T_min
9.    IF T_min ⩾ T_c
10.      IF ∀τ_k ∈ S_R3, Σ_{k=1}^{n+m} C_k/T_k ⩽ (n+m-p+1)(2^{1/(n+m-p+1)} - 1)
11.         RETURN TRUE
12.      ELSE
13.         RETURN FALSE
```

14. ELSE

15. $VZOP_{St} = T_{\min}$

16. IF $\forall \tau_k \in S_H, \dfrac{WVET_{St}(S_{St}, VZOP_{St})}{VZOP_{St}} + \sum_{k=p+1}^{n+m} \dfrac{C_k}{T_k} \leqslant (n+m-p+1)(\sqrt[n+m-p+1]{2}-1)$

17. RETURN TRUE

18. ELSE

19. RETURN FALSE

例 3: 如图 4 所示. 实时任务集 $S_R$ 包括 4 个任务, $\tau_1 = (25,4,4,0), \tau_2 = (25,3,3,13), \tau_3 = (13,2,13,0), \tau_4 = (24,4,24,0), \tau_1, \tau_2 \in S_{St}, \tau_3, \tau_4 \in S_H$. 根据算法 2, 虚拟缩减周期 $VZOP_{St} = 13$, 将强实时任务 $\tau_1, \tau_2$ 虚拟缩减为一个硬实时任务 $\tau_0 = (13,4,13,0), \tau_0 \in S_H$, 即周期 $T_0 = VZOP_{St} = 13$, 根据算法 1 的虚拟最差执行时间计算方法, $C_0 = WVET_{St}(S_{St}, 13) = 4$, 截止期 $D_0 = T_0 = 13$, 初始到达偏移 $A_0 = 0, \tau_0, \tau_3, \tau_4$ 构成新的实时任务集 $S'_R$, 此任务集符合 RMS 调度算法, 根据算法 2, 可快速计算出 $S'_R$ 可调度, 从而判定实时任务集 $S_R$ 一定也可调度.

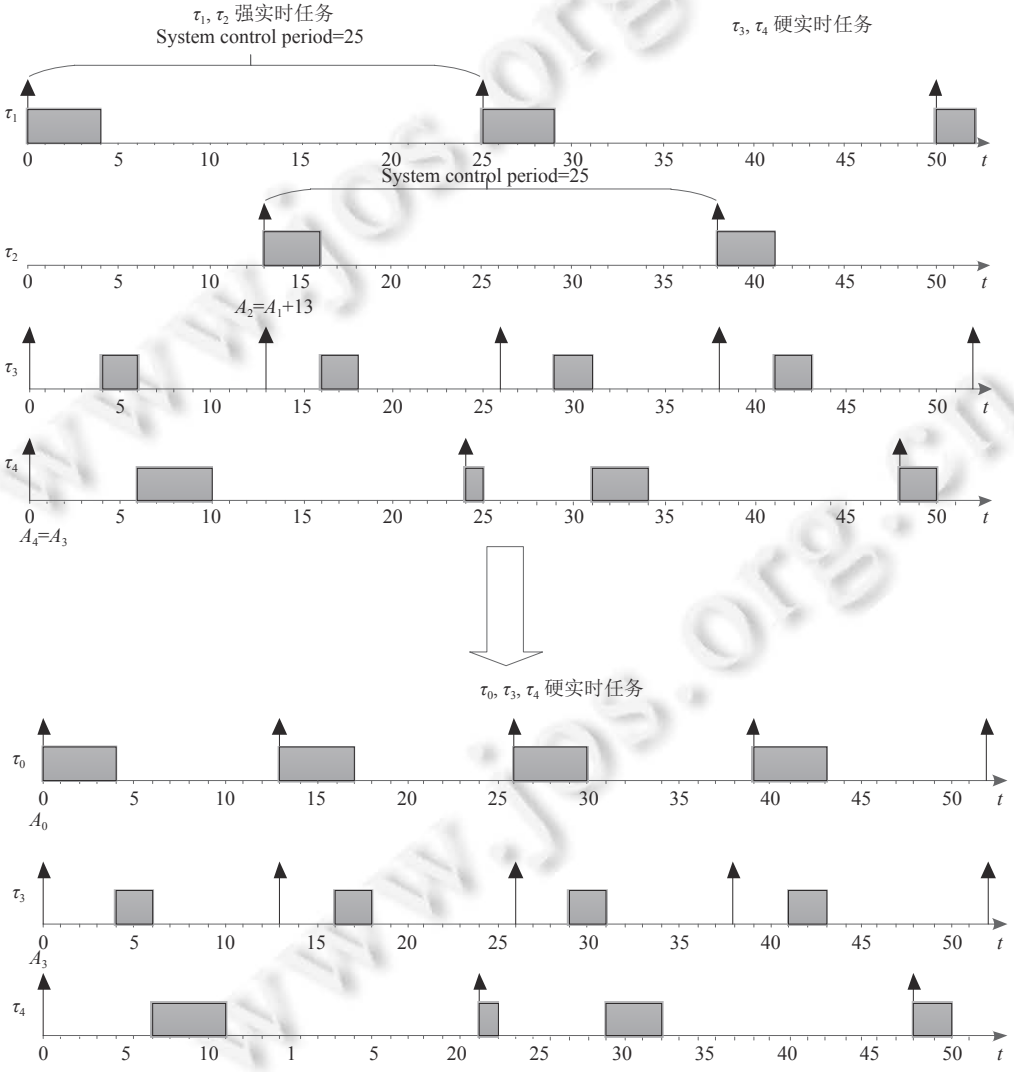


图 4 实时任务集虚拟缩减结果

### 5.4 性能分析

原实时任务组成的集合为 $S_{R1}$, 包含 $n$ 个任务, 新加入的实时任务集合 $S_{R2}$, 包含 $m$ 个任务, 共同组成新的任务集合 $S_{R3}$, 即判定 $m+n$ 个任务的可调度性. 令 $x=m+n$, 即为算法输入规模, 其中包含 $p$ 个强实时任务, $x$–$p$ 个硬实时任务, $1 \leqslant p \leqslant x$.

第 1 步. 任务集合并, 时间复杂度为 $O(x)$;

第 2 步. 任务集拆分, 时间复杂度为 $O(x)$;

第 3, 4 步. 遍历强实时任务集和进行可调度性判定, 时间复杂度为 $O(p)$;

第 5 步. 遍历硬实时任务集找出最小周期值, 时间复杂度为 $O(x-p)$;

第 6 步. 累加所有任务利用率, 时间复杂度为 $O(x)$;

第 7 步. 需计算强实时任务集 $S_{st}$ 最差虚拟执行时间, 时间复杂度为 $O(p^2)$, 累加所有任务利用率, 时间复杂度为 $O(x-p)$, 但与第 6 步并列, 执行其一.

因此算法总的计算时间复杂度为 $O(x)+O(x)+O(p)+O(x-p)+O(x)=O(x)$; 或者 $O(x)+O(x)+O(p)+O(x-p)+O(p^2)+O(x-p)=O(x)+O(p^2)$, 当强实时任务数为最大值 $x$ 时, 即 $p=x$, 此时时间复杂度为 $O(x^2)$. 时间复杂度取最坏值, 即 $O(x)+O(p^2)$. 因此, 时间复杂度不仅和总任务数有关, 还和其中的强实时任务在任务集里所占比例有关, 一般情况下, 时间复杂度介于 $O(x)$ 与 $O(x^2)$之间, 强实时任务所占比例越小, 时间复杂度越低. 由于强实时任务周期进行了虚拟缩减, 根据此虚拟周期计算出对应的最差虚拟执行时间, 再计算出强实时任务的总利用率往往比实际利用率要高, 导致分析结果变得悲观, 任务集利用率较高时, 可调度率快速下降. 但是, 该方法却提高了可调度性分析速度, 使得该方法可以在线实时的对动态加入任务进行可调度性分析, 实现开放式实时操作系统任务动态加入的可调度性分析功能, 非常适合于计算资源受限的实时嵌入式操作系统中, 如航天器计算机操作系统等.

## 6 实验及结果分析

### 6.1 实验方案

在强硬实时任务集中, 本文的可调度性分析 VZOP 算法与已有相关的 RTA[9,10]、BCL 算法[21]进行性能对比, 进行了 4 组对比实验, 实验 1、实验 2 比较运行时间开销指标 $t$, 实验 3、实验 4 比较可调度率指标 $r$. 在我国空间站计算机进行了实际对比验证, 实验平台如表 1 所示.

表 1 系统实验环境

| 参数 | 编程平台配置 | 运行平台配置 |
|---|---|---|
| 设备 | PC | 空间站计算机 |
| CPU | Intel i5-3470 | BM3803单核 |
| 主频 | 3.2 GHz | 80 MHz |
| 内存 | 8 GB | SDRAM 4 MB |
| 操作系统 | Win7 | SpaceOS 2.0 |

空间站计算机是典型的计算资源受限嵌入式系统, 使用基于 SPARC 架构国产的 32 位抗辐照单核处理器 BM3803, 其上运行自主研发的 SpaceOS 实时操作系统, 具有较高的可靠性, 已经广泛应用于我国多个重大航天工程中.

**实验 1**. 任务数 $n$ 的不同, 对算法 VZOP 和 RTA、BCL 的运行时间开销 $t$ 的影响. 其中各参数如下.

● 任务集数: 产生随机任务集, 任务集包含 $n$ 个任务, 任务集内强实时任务和硬实时任务各占一半, $n$ 取值范围 [1, 50], 步长为 1, 总共 50 个步长. 为了防止任务集随机产生时极端值的干扰, 对于每个步长产生 20 个任务集, 且可调度任务集占一半, 取这 20 个任务集可调度性判定耗时的平均值作为当前步长的结果值, 因此总的会产生 1 000

个任务集.

● 任务集利用率 $U$: 任务集利用率 $U$=0.6, 每个任务利用率用 UUnifast 算法产生[31].

● 任务周期 $T$: 任务的周期长度服从均匀分布, 取值范围 [1, 9999], 第一次随机产生的周期确定为控制周期 $T_c$, 对于强实时任务, 周期都相同且为控制周期, 对于硬实时任务, 每次都随机产生, 可各不相同.

● 任务最差执行时间 $C$: 每个任务的利用率乘以该任务随机产生的周期 $T$, 则得到每个任务的最差执行时间 $C$, 若 $C$ 小于 0, 则重新产生该任务周期值, 直到最差执行时间 $C$ 大于 0 为止.

● 任务到达偏移 $A$: 对于强实时任务, 服从均匀分布, 取值范围 [0, $T_c$), 对于硬实时任务, 都为 0.

● 任务截止期 $D$: 对于强实时任务, 截止期为到达偏移 $A$ 加最差执行时间 $C$, 对于硬实时任务, 截止期等于周期.

**实验 2**. 硬实时任务在任务集中所占比例 $p$ 的不同, 分别测试 $p$=0, $p$=0.2, $p$=0.5, $p$=0.8, $p$=1 时, 对算法 VZOP 和 RTA、BCL 的平均运行时间开销 $t\_a$ 的影响. 其他参数如下.

● 任务集数: 产生随机任务集, 任务集包含 $n$ 个任务, 任务集内硬实时任务所占比例取值为 $p$, $n$ 取值范围 [1, 50], 步长为 1, 总共 50 个步长. 为了防止任务集随机产生时极端值的干扰, 对于每个步长产生 20 个任务集, 且可调度任务集占一半, 取这 20 个任务集可调度性判定耗时的平均值作为当前步长的结果值, 因此总的会产生 1 000 个任务集. 累计这 50 个步长的耗时除以 50 作为平均运行时间开销 $t\_a.p$ 分别取 5 个值, 因此总共产生 5 000 个任务集;

● 任务集利用率 $U$: 任务集利用率 $U$=0.6, 每个任务利用率用 UUnifast 算法产生[31];

● 任务周期 $T$: 任务的周期长度服从均匀分布, 取值范围 [1, 9999], 第一次随机产生的周期确定为控制周期 $T_c$, 对于强实时任务, 周期都相同且为控制周期, 对于硬实时任务, 每次都随机产生, 可各不相同;

● 任务最差执行时间 $C$: 每个任务的利用率, 乘以该任务随机产生的周期 $T$, 则得到每个任务的最差执行时间 $C$, 若 $C$ 小于 0, 则重新产生该任务周期值, 直到最差执行时间 $C$ 大于 0 为止;

● 任务到达偏移 $A$: 对于强实时任务, 服从均匀分布, 取值范围 [0, $T_c$) 对于硬实时任务, 都为 0;

● 任务截止期 $D$: 对于强实时任务, 截止期为到达偏移 $A$ 加最差执行时间 $C$, 对于硬实时任务, 截止期等于周期.

**实验 3**. 任务集利用率 $U$ 的不同, 对算法 VZOP 和 RTA、BCL 的可调度率 $r$ 的影响. 其中各参数如下.

● 任务集数: 产生随机任务集, 任务集内强实时任务和硬实时任务各占一半, 任务集利用率 $U$ 取值范围 (0, 1], 每个任务利用率用 UUnifast 算法产生[31], 步长为 0.04, 因此总共 25 个步长. 对于每个步长产生 100 个任务集, 统计这 100 个任务集中可调度性判定为可调度所占比例作为该任务集利用率下对应的可调度率, 因此总的会产生 2 500 个任务集;

● 任务集任务数 $n$=20;

● 任务周期 $T$: 任务的周期长度服从均匀分布, 取值范围 [1, 9999], 第一次随机产生的周期确定为控制周期 $T_c$, 对于强实时任务, 周期都相同且为控制周期, 对于硬实时任务, 每次都随机产生, 可各不相同;

● 任务最差执行时间 $C$: 每个任务的利用率, 乘以该任务随机产生的周期 $T$, 则得到每个任务的最差执行时间 $C$, 若 $C$ 小于 0, 则重新产生该任务周期值, 直到最差执行时间 $C$ 大于 0 为止;

● 任务到达偏移 $A$: 对于强实时任务, 服从均匀分布, 取值范围 [0, $T_c$) 对于硬实时任务, 都为 0;

● 任务截止期 $D$: 对于强实时任务, 截止期为到达偏移 $A$ 加最差执行时间 $C$, 对于硬实时任务, 截止期等于周期.

**实验 4**. 任务数 $n$ 的不同, 分别测试 $n$=5, $n$=10, $n$=20, $n$=30 时, 对算法 VZOP 和 RTA、BCL 的平均可调度率 $r\_a$ 的影响. 其中各参数如下:

● 任务集数: 产生随机任务集, 任务集内强实时任务和硬实时任务各占一半, 任务集利用率 $U$ 取值范围 (0, 1], 每个任务利用率用 UUnifast 算法产生[31], 步长为 0.04, 因此总共 25 个步长. 对于每个步长产生 100 个任务集, 统计这 100 个任务集中可调度性判定为可调度所占比例作为该任务集利用率下对应的可调度率, 因此总的会产生

2500 个任务集, $n$ 分别取 4 个值, 因此会产生 10 000 个任务集. 累计这 25 个步长的可调度率除以 25 为平均可调度率 $r\_a$;

- 任务集任务数 $n$;
- 任务周期 $T$: 任务的周期长度服从均匀分布, 取值范围 [1, 9999], 第 1 次随机产生的周期确定为控制周期 $T_c$, 对于强实时任务, 周期都相同且为控制周期, 对于硬实时任务, 每次都随机产生, 可各不相同;
- 任务最差执行时间 $C$: 每个任务的利用率, 乘以该任务随机产生的周期 $T$, 则得到每个任务的最差执行时间 $C$, 若 $C$ 小于 0, 则重新产生该任务周期值, 直到最差执行时间 $C$ 大于 0 为止;
- 任务到达偏移 $A$: 对于强实时任务, 服从均匀分布, 取值范围 $[0, T_c)$ 对于硬实时任务, 都为 0;
- 任务截止期 $D$: 对于强实时任务, 截止期为到达偏移 $A$ 加最差执行时间 $C$, 对于硬实时任务, 截止期等于周期.

### 6.2 实验结果比较

图 5 是实验 1 的结果, 以任务数 $n$ 作为自变量, VZOP 与 RTA、BCL 关于运行时间开销指标进行了对比. 横坐标表示任务数, 纵坐标表示算法运行时间开销, 单位为 ms. 可以看出, 随着任务数的增加, VZOP 算法运行时间开销基本线性增长, 而 RTA、BCL 明显为二次增长. 而当任务集中少于 5 个任务时, VZOP 算法时间开销略微大于 RTA、BCL 算法, 而 RTA、BCL 算法时间开销无明显差异. 当任务集中多于 5 个任务时, VZOP 运行时间开销更小, 且随着任务数的增加, VZOP 算法涨幅最小, BCL 其次, RTA 涨幅最大, 当任务数 50 个时, RTA 运行耗时达到了 5.8 ms, BCL 为 1.6 ms, 而 VZOP 仅为 0.8 ms. VZOP 算法对任务集的可调度性判定始终都在 1 ms 内完成.

图 6 是实验 2 的结果, 以硬实时任务所占比例 $p$ 作为自变量, VZOP 与 RTA、BCL 关于平均运行时间开销指标进行了对比. 横坐标表示比例, 纵坐标表示算法平均运行时间开销, 单位为 ms. 可以看出, 当硬实时任务所占比例为 0 时, 即任务集中全是强实时任务, 运行时间几乎相等, 随着硬实时任务所占比例增加, RTA 和 BCL 算法运行时间增加, VZOP 算法运行时间反而降低了, 当硬实时任务所占比例为 1 时, RTA 平均运行时间达到了 15.72 ms, BCL 为 3.13 ms, VZOP 仅为 0.28 ms.

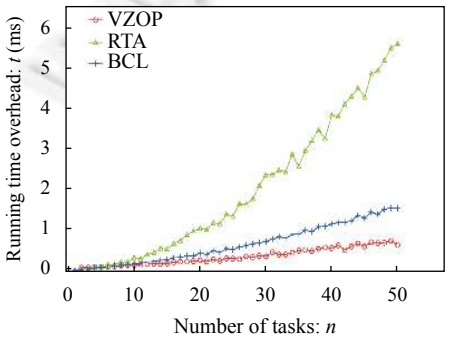


图 5 VZOP 与 RTA、BCL 运行时间开销对比

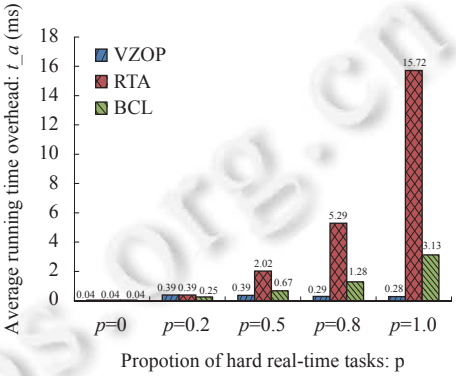


图 6 硬实时任务所占比例对平均运行时间开销影响

图 7 是实验 3 的结果, 以任务集总利用率 $U$ 作为自变量, VZOP 与 RTA、BCL 关于可调度率指标进行了对比. 横坐标表示任务集总利用率, 纵坐标表示任务集可调度率. 可以看出, 随着任务集总利用率的增加, VZOP 算法与 RTA、BCL 算法可调度率都降低. 当任务集总利用率小于 0.5 时, VZOP 算法可调度率高于 RTA、BCL; 当任务集总利用率大于 0.5 时, VZOP 算法可调度率开始低于 RTA、BCL, 且下降速度较快, 当利用率约为 0.72 时, VZOP 可调度率降低为 0, 而 RTA 和 BCL 算法可调度率一直相同, 当总利用率达到 0.8 时, BCL 算法可调度率开始低于 RTA, 随着总利用率的进一步增加, 都趋于 0.

图 8 是实验 4 的结果, 以任务数 $n$ 作为自变量, VZOP 与 RTA、BCL 关于平均可调度率指标进行了对比. 横坐标表示任务数, 纵坐标表示任务集平均可调度率. 可以看出, 随着任务数的增加, VZOP 算法与 RTA、BCL 算法

平均可调度率都降低, 但是 VZOP 算法相对 RTA、BCL 算法平均可调度率反而在升高. 当任务集中的任务数为 5 时, 平均可调度率都很高, VZOP 算法为 0.53, 分别低于 RTA 和 BCL, 0.09, 0.08. 当任务数为 30 时, VZOP 算法为 0.35, 分别高于 RTA 和 BCL, 0.06, 0.07.

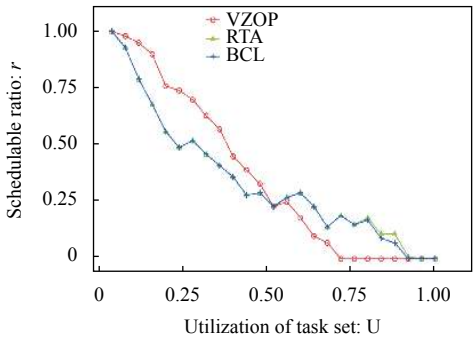


图 7 VZOP 与 RTA、BCL 可调度率对比

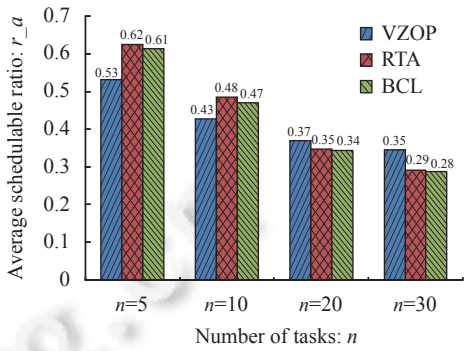


图 8 任务数不同对平均可调度率的影响

### 6.3 实验结果分析

对于实验 1, 由于 VZOP 算法对所有强实时任务进行了虚拟缩减为一个硬实时任务, 此时符合 RMS 算法优先级分配规则, 从而可采用 RMS 的累加利用率方式进行可调度性判定, 仅做一次性的加减乘除运算, 使得其判定时间近似线性增长. 而 RTA、BCL 算法仅仅将强实时任务当作一个硬实时任务 (见引理 1), 会对任务集中每个硬实时任务进行判定, 通过计算更高优先级任务的抢占量来得到任务最差响应时间 (其中 RTA 是迭代至响应时间不再变化, 而 BCL 仅采用区间抢占量计算方式), 将此响应时间与截止期进行比较, 从而判定此任务是否可调度, BCL 为二次增长, 在任务模型参数为整数且周期长度为有限值条件下, RTA 也为二次增长. 由于 VZOP 算法对强实时任务进行虚拟缩减增加了运行时间开销, 当任务集中任务较少时, 运行时间开销略微大于 RTA、BCL, 但是当任务数增多时, RTA、BCL 运行时间开销快速增加, 明显高于 VZOP 算法. 对于实验 2, 然而随着硬实时任务所占比例的增加, 使得强实时任务虚拟化过程速度加快, 其最差虚拟执行时间计算量减小, 因此其平均运行时间开销反而降低, 符合第 5.4 节的理论时间复杂度分析, 强实时任务所占比例越小, 时间复杂度越低, 当 $p$=0.5, 即强硬实时任务各占一半时, 平均运行时间开销比 BCL 降低了 41.8%.

对于实验 3, VZOP 算法将所有强实时任务虚拟化为一个硬实时任务, 且对其周期进行虚拟缩减, 并计算出该周期下的最差执行时间, 此时任务集中所有任务按照周期来分配优先级, 任务集转为最优调度[4], 因此可调度率高于 RTA、BCL 算法. 然而, VZOP 考虑的是强实时任务分布最坏情况, 从而计算出的利用率可能会比实际偏大, 此外, 当强实时任务虚拟化完成后, 采用 RMS 算法的利用率累加和判定上界判定条件仍然是考虑总利用率情况下的硬实时任务最坏分布情况, 导致判定结果较实际情况变得悲观, 可调度率下降较快. 然而 RTA、BCL 算法没有进行周期虚拟缩减, 仅是将所有强实时任务合并为一个硬实时任务, 不存在总利用率虚拟增大的情况, 由于强实时任务的执行时间增大使得硬实时任务的响应时间变大, 从而可能导致硬实时任务不可调度, 因此可调度率下降较慢. 此外, VZOP 算法利用率上界是任务数的函数, 实验 2 中强硬实时任务集中任务数为 20 时, 其上界约为 0.715, 大于此利用率都会判定为不可调度, 可调度率为 0. 对于实验 4, 任务数的增加导致高优先级任务间抢占量增加, 低优先级任务容易错过截止期, 因此平均可调度率下降, 然而由于 VZOP 算法将强实时任务虚拟化为一个周期最短的硬实时任务, 优先级采用 RMS 算法分配方式, 反而转为理论最优调度[4], 因此平均可调度率下降速度较慢, 符合理论分析. 当 $n$=20, 即任务集包含 20 个任务时, 平均可调度率比 RTA 提高了 5.7%.

总结, 在对强硬实时任务集进行可调度性判定中, 随着任务数和硬实时任务所占比例的增加, VZOP 算法运行时间开销近似线性增长, 明显优于 RTA、BCL 算法. 然而, 随着总利用率的增加, 其可调度率低于 RTA、BCL 算法, 但是当任务集中任务数较多时, 其平均可调度率高于 RTA、BCL 算法. 因此, VZOP 算法采用周期虚拟缩减的

方式, 以分析精度换取了时间复杂度. 在任务集总利用率越低, 任务数越多, 硬实时任务占比越高的计算资源受限的实时系统中, VZOP 算法更适合在线的对新加入任务进行可调度性判定, 从而使得处理器利用率更高, 同时其较低运行时间开销也使得对系统性能影响更小, 保障了系统的实时性.

## 7 总 结

本文针对航天器等安全关键系统中实时任务动态加入时的调度和可调度性分析的实际问题, 根据任务时间特性的不同, 将实时任务分为了强实时任务和硬实时任务, 构建 SHT 任务模型对强实时和硬实时任务进行了精确描述, 根据任务的实时性要求对任务进行了优先级分配. 对于优先级分配结果, 本文提出了基于任务周期虚拟缩减的可调度性判定方法, 对任务集的可调度性进行快速判定, 从而实现任务动态加入的可调度性分析功能, 保障了系统的实时性和安全性. 本方法将所有强实时任务虚拟为一个硬实时任务, 同时根据其他硬实时任务周期大小, 对此硬实时任务周期进行虚拟缩减并计算出其最差虚拟执行时间, 使其满足 RMS 调度算法规则, 然后按 RMS 可调度性判定公式进行可调度性判定. 本文给出了判定方法的严格证明, 且该算法时间复杂度近似线性, 是目前的可调度性判定方法中时间开销较低的算法, 在我国空间站计算机进行了对比验证, 实验表明判定效率优于现有可调度性判定方法, 平均运行时间开销降低了 41.8%, 平均可调度率提高了 5.7%, 非常适合于计算资源受限的开放式系统中实时任务动态加入的可调度性分析过程, 例如航天器计算机操作系统等. 目前本方法仅针对单核处理器, 下一步将扩展到面向多核处理器实时任务动态加入的可调度性分析方法研究.

**References:**


[1] Yi W. Design and dynamic update of real-time systems. In: Proc. of the 2019 IEEE Real-time Systems Symp. (RTSS). Hong Kong: IEEE, 2019. 1–3. [doi: 10.1109/RTSS46320.2019.00011]

[2] Gong J, Yang MF, Qiao L, Yang H, Gu B, Peng F, Wang J, Xu J, Liu B. Timed scheduling method for tasks in operating system through preemptive priority and round-robin: CN, 201310746023.2, 2014-04-09. (in Chinese with English abstract)

[3] Sheridan-Barbian KK. A survey of real-time operating systems and virtualization sollutions for space systems [MS. Thesis]. Monterey: Naval Postgraduate School, 2015.

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

[5] Abdelzaher TF, Sharma V, Lu C. A utilization bound for aperiodic tasks and priority driven scheduling. IEEE Trans. on Computers, 2004, 53(3): 334–350. [doi: 10.1109/TC.2004.1261839]

[6] Meng Y, Zou LK. A weighted round robin based scheduling method for ARINC653 system. In: Proc. of the Computer Science and Engineering Technology (CSET2015), Medical Science and Biological Engineering (MSBE2015). Hong Kong: IEEE, 2016. 252–259. [doi: 10.1142/9789814651011_0037]

[7] Jo HC, Park JK, Jin HW, Lee SH, Yoon HS. Quantitative analysis of ARINC-653 scheduling overheads on multi-core systems. In: Proc. of the 37th IEEE/AIAA Digital Avionics Systems Conf. London: IEEE, 2018. 1–5. [doi: 10.1109/DASC.2018.8569867]

[8] Ekberg P, Yi W. Fixed-priority schedulability of sporadic tasks on uniprocessors is NP-hard. In: Proc. of the 2017 IEEE Real-Time Systems Symp. (RTSS). Paris: IEEE, 2017. 139–146. [doi: 10.1109/RTSS.2017.00020]

[9] 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]

[10] Audsley N, Burns A, Richardson M, Tindell K, Wellings AJ. Applying new scheduling theory to static priority pre-emptive scheduling. Software Engineering Journal, 1993, 8(5): 284–292. [doi: 10.1049/sej.1993.0034]

[11] Eisenbrand F, Rothvoß T. Static-priority real-time scheduling: Response time computation is NP-hard. In: Proc. of the 2008 Real-time Systems Symp. Barcelona: IEEE, 2008. 397–406. [doi: 10.1109/RTSS.2008.25]

[12] Baruah S, Bertogna M, Buttazzo G. Multiprocessor Scheduling for Real-Time Systems. Cham: Springer, 2015.

[13] Sha L, Rajkumar R, Lehoczky JP. Priority inheritance protocols: An approach to real-time synchronization. IEEE Trans. on Computers, 1990, 39(9): 1175–1185. [doi: 10.1109/12.57058]

[14] Baker TP. Stack-based scheduling of realtime processes. Real-time Systems, 1991, 3(1): 67–99. [doi: 10.1007/BF00365393]

[15] Tindell K. Using offset information to analyse static priority pre-emptively scheduled task sets. Technical Report, YCS-92-

182, Department of Computer Science, University of York, 1992

[16] Lehoczky JP. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: Proc. of the 11th Real-time Systems Symp. Lake Buena Vista: IEEE, 1990. 201–209. [doi: 10.1109/REAL.1990.128748]

[17] Tindell KW, Burns A, Wellings AJ. An extendible approach for analyzing fixed priority hard real-time tasks. Real-time Systems, 1994, 6(2): 133–151. [doi: 10.1007/BF01088593]

[18] Sjodin M, Hansson H. Improved response-time analysis calculations. In: Proc. of the 19th IEEE Real-time Systems Symp. Madrid: IEEE, 1998. 399–408. [doi: 10.1109/REAL.1998.739773]

[19] Bril RJ, Verhaegh WFJ, Pol EJD. Initial values for online response time calculations. In: Proc. of the 15th Euromicro Conf. on Real-time Systems. Porto: IEEE, 2003. 13–22. [doi: 10.1109/EMRTS.2003.1212722]

[20] Lu WC, Lin KJ, Wei HW, Shih WK. Period-dependent initial values for exact schedulability test of rate monotonic systems. In: Proc. of the 2007 IEEE Int'l Parallel and Distributed Processing Symp. Long Beach: IEEE, 2007. 1–8. [doi: 10.1109/IPDPS.2007.370352]

[21] Bertogna M, Cirinei M, Lipari G. Improved schedulability analysis of EDF on multiprocessor platforms. In: Proc. of the 17th Euromicro Conf. on Real-time Systems. Balearic Islands: IEEE, 2005. 209–218. [doi: 10.1109/ECRTS.2005.18]

[22] Baruah S, Bonifaci V, Marchetti-Spaccamela A, Stougie L, Wiese A. A generalized parallel task model for recurrent real-time processes. In: Proc. of the 33rd IEEE Real-time Systems Symp. San Juan: IEEE, 2012. 63–72. [doi: 10.1109/RTSS.2012.59]

[23] Bonifaci V, Marchetti-Spaccamela A, Stiller S, Wiese A. Feasibility analysis in the sporadic DAG task model. In: Proc. of the 25th Euromicro Conf. on Real-time Systems. Los Alamitos: IEEE, 2013. 225–233. [doi: 10.1109/ECRTS.2013.32]

[24] Baruah SK, Rosier LE, Howell RR. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. Real-time Systems, 1990, 2(4): 301–324. [doi: 10.1007/BF01995675]

[25] Baruah SK, Howell RR, Rosier LE. Feasibility problems for recurring tasks on one processor. Theoretical Computer Science, 1993, 118(1): 3–20. [doi: 10.1016/0304-3975(93)90360-6]

[26] Huang SJ, Zhu YA, Li BZ, Lu W. Real-time scheduling method for dependency period tasks. Chinese Journal of Computers, 2015, 38(5): 999–1006 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2015.00999]

[27] Sun JH, Sun JC, Guan N, Deng QX. Integer programming approach for schedulability of sporadic real-time systems. Ruan Jian Xue Bao/Journal of Software, 2017, 28(2): 411−428 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5025.htm [doi: 10.13328/j.cnki.jos.005025]

[28] Zhang WZ, Bai EC, Li J. Speeding up the schedulability analysis and priority assignment of sporadic tasks under uniprocessor FPNS. IEEE Trans. on Industrial Informatics, 2020, 16(10): 6382–6392. [doi: 10.1109/TII.2020.2968590]

[29] Andersson B, Baruah SK, Jonsson J. Static-priority scheduling on multiprocessors. In: Proc. of the 22nd IEEE Real-time Systems Symp. (RTSS 2001). London: IEEE, 2001. 193–202. [doi: 10.1109/REAL.2001.990610]

[30] Yang ML. Research on real-time scheduling algorithms with shared resources [Ph.D. Thesis]. Chengdu: University of Electronic Science and Technology of China, 2016 (in Chinese with English abstract).

[31] Bini E, Buttazzo GC. Measuring the performance of schedulability tests. Real-time Systems, 2005, 30(1–2): 129–154. [doi: 10.1007/s11241-005-0507-9]

## 附中文参考文献:

[2] 龚健, 杨孟飞, 乔磊, 杨桦, 顾斌, 彭飞, 王婧, 徐建, 刘波. 一种优先级抢占时间片轮转操作系统中任务定时调度方法: 中国, 201310746023.2, 2014-04-09.

[26] 黄姝娟, 朱怡安, 李兵哲, 陆伟. 具有依赖关系的周期任务实时调度方法. 计算机学报, 2015, 38(5): 999–1006. [doi: 10.3724/SP.J.1016.2015.00999]

[27] 孙景昊, 孙景昶, 关楠, 邓庆绪. 偶发实时系统可调度性分析问题的整数规划方法. 软件学报, 2017, 28(2): 411–428. http://www.jos.org.cn/1000-9825/5025.htm [doi: 10.13328/j.cnki.jos.005025]

[30] 杨茂林. 共享资源约束下的多核实时调度算法研究[博士学位论文]. 成都: 电子科技大学, 2016.

**刘洪标**(1995－), 男, 博士, 主要研究领域为嵌入式操作系统, 任务调度.

**陈熙**(1997－), 女, 硕士, 主要研究领域为嵌入式操作系统, 任务调度.

**乔磊**(1982－), 男, 博士, 研究员, CC 高级会员, 主要研究领域为操作系统模型设计, 存储管理, 文件系统.

**马智**(1994－), 男, 博士, 主要研究领域为嵌入式操作系统, 系统架构.

**杨孟飞**(1962－), 男, 博士, 研究员, 博士生导师, CCF 高级会员, 主要研究领域为空间飞行器嵌入式系统, 控制系统, 总体技术.

**李少峰**(1992－), 男, 博士, 主要研究领域为嵌入式操作系统, 内存管理.