博 士 学 位 论 文

面向混合关键性系统与 DRT 系统的实时调度问题研究

研究生：

导　师：

东北大學

二〇一五年五月

Typeset by LaTeX 2_ε at May 28, 2015

With package NEUthesis ? of NEU-RTES.ORG

分类号_____ 密 级_____

UDC _____

学 位 论 文

面向混合关键性系统与 DRT 系统的实时调度问题研究

作 者 姓 名：

指 导 教 师：

申请学位级别：博 士 学 科 类 别：工 学

学科专业名称：计算机软件与理论

论文提交日期：2015 年 5 月 论文答辩日期：2015 年 7 月

学位授予日期： 答辩委员会主席：

评 阅 人：

东 北 大 学

2015 年 5 月

A Dissertation in Computer Software and Theory

Research on Real-Time Scheduling of Mixed-Criticality And DRT Systems

Northeastern University

May, 2015

独创性声明

本人声明，所呈交的学位论文是在导师的指导下完成的。论文中取得的研究成果除加以标注和致谢的地方外，不包含其他人已经发表或撰写过的研究成果，也不包括本人为获得其他学位而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者：

日　　期：

学位论文版权使用授权书

本学位论文作者和指导教师完全了解东北大学有关保留、使用学位论文的规定：即学校有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人同意东北大学可以将学位论文的全部或部分内容编入有关数据库进行检索、交流。

作者和导师同意网上交流的时间为作者获得学位后：

半年 □　　一年 □　　一年半 □　　两年 □

学位论文作者签名：　　导师签名：

签字日期：　　签字日期：

面向混合关键性系统与 DRT 系统的实时调度问题研究

摘 要

现代实时嵌入式系统不断发展的一个重要趋势是在同一硬件平台中集成多种不同关键性级别的应用。与此同时，嵌入式系统硬件也在经历从单核平台向多核平台的变革之中，未来还将发展至众核平台。但是这种混合关键性系统的调度问题即便是对于单核平台也是极具挑战性的。目前，大多数复杂的嵌入式系统很难被传统基于周期的模型所精确描述。实时任务有向图（DRT）模型具备很强的描述能力，可以用于没有循环时间行为的复杂实时系统的建模。但是针对该模型的精确时间分析在时间复杂度上通常是不接受的（指数级复杂度）。

本文研究了面向混合关键性系统和 DRT 系统的实时调度算法的设计与分析等问题。在混合关键性系统研究方面，提出了一种高效的单核处理器运行时调度算法，和两种多核、多处理器划分调度算法。在 DRT 系统研究方面，提出了两种有效的近似响应时间分析方法，并通过计算加速比进行了量化评价，还提出了一种有效的有向图整形算法来提升系统的可调度性。

本文的主要贡献点可以被总结如下：

（1）提出了基于 OCBP 策略的固定作业优先级单处理器混合关键性实时调度算法 LPA。与其它 OCBP 族的算法相比，LPA 算法显著提升了系统的运行时时间效率（线性时间复杂度）、空间效率和可调度性。LPA 算法的核心思想是尽可能晚的调整作业的运行时优先级，从而避免了与实际调度决策不相关的冗余优先级调整工作。本文还提出了更精确的混合关键性系统忙碌周期上界的计算方法。使用随机生成任务集合的实验结果表明本文提出算法显著提升了运行时时间效率、空间效率和可调度性。

（2）提出了新型混合关键性划分调度算法 MPVD，将单处理器上可调度性能最佳的算法 EY-VD 扩展至多处理器平台。MPVD 算法的核心思想是将不同关键性级别的任务尽可能均匀地分配到各个处理器中，以更好地利用不同关键性级别任务间的差异性，从而提升系统的可调度性。本文还分析了 MPVD 的不足，并提出了两个优化技术来进一步提升算法的性能。使用随机生成任务集合的实验结果表明本文提出算法的可调度性显著高于已有算法。

（3）提出了新型混合关键性多处理器划分调度策略 OCOP。本文首先结合 EY-VD 算法和传统划分调度策略提出了混合关键性系统划分调度算法 MC-PEDF。该算法的

性能显著高于其它已有划分调度算法，但本文研究发现传统的划分策略不能很好的利用任务在不同关键性级别中工作量的差异性。为了解决该问题，本文放松了划分调度禁止所有运行时作业迁移的限制，提出了 OCOP 划分调度策略。OCOP 允许系统关键性模式切换时，调整任务在处理器间的分配，从而显著提升了不同关键性模式中的系统资源利用率。最后本文本文还提出了使用 OCOP 策略的新划分调度算法 MC-MP-EDF。实验结果表明 MC-PEDF 和 MC-MP-EDF 算法在可调度性上优于先前的多处理器混合关键性实时调度算法，而采用 OCOP 划分调度策略的 MC-MP-EDF 算法则具有更好的可调度性能。

（4）提出了两种伪多项式时间复杂度的近似分析 DRT 任务系统响应时间的方法 RBF 和 IBF，并通过分别计算加速比，量化评价了这两种近似方法的性能。本文证明了 RBF 近似响应时间分析方法的精确加速比为 2（即便是双任务的简易系统）。IBF 方法的加速比是随干涉任务数量 k 单调递增的函数。该函数在 k 趋于无穷大时收敛于 2，并在 $k=2$ 时取值为 1。因此 IBF 方法在分析双任务系统时，结果是精确的。随机任务实验结果表明本文提出的近似分析方法具有很好的时间效率，同时仅有很小的精度损失。

（5）提出了一种高效率的 DRT 任务有向图整形算法以提升系统的可调度性。算法的主要思想是在任务有向图的特定顶点上添加一个人为设定的释放时间延迟，从而得到一个释放工作量更平滑的新图，同时新图还能保持原有的时间约束条件。延迟特定顶点的释放时间能够使得有向图释放的某些路径上的工作量更平滑，同时会造成其它路径的工作量更集中。本文提出算法能高效率地为每个顶点搜索适当的释放时间延迟取值。实验结果表明本文提出算法能够显著提升 DRT 系统的可调度性，并且能够在很短时间内处理大规模的任务集合。

增加一整体结论性段落

关键词： 实时系统；混合关键性；多处理器；划分调度；EDF；DRT；响应时间分析

Research on Real-Time Scheduling of Mixed-Criticality And DRT Systems

Abstract

An increasingly important trend in the design of real-time and embedded systems is the integration of components with different levels of criticality onto a common hardware platform. At the same time, these platforms are migrating from single cores to multi-cores and, in the future, many-core architectures. Unfortunately, the scheduling problem of mixed-criticality systems appears to be challenging, even on single-processor platforms. Most of the complex embedded systems cannot be exactly described by traditional period based models. Real-time task graphs are used to model complex real-time systems with non-cyclic timing behaviors. However the exactly analysis of such graph-based systems are often intractable.

This dissertation studied the design and analysis of real-time sheduling algorithm for mixed-criticality systems and DRT systems. As to the mixed-criticality scheduling, proposed an efficient online scheduling algorithm for single-processor systems, two partioned scheduling algorithms for multi-cores/processors. As to the DRT systems, proposed two efficient approximate response time analysis methods with speedup factor evaluation, and an effective graph transformation method to improve system schedulability.

The main contribution of this dissertation can be summarized as follows:

(1) This dissertation proposed a novel OCBP-based algorithm LPA, to schedule mixed-criticality sporadic tasks on preemptive single processor systems. Comparing with the previous OCBP-based algorithms, it can improve the online time efficiency, online space efficiency, as well as schedulability. The central idea of LPA is to make online priority adjustment as lazy as possible, in order to avoid redundant priority adjustments that are not relevant to the actual scheduling decisions. Experiments with synthetic workloads show the performance improvement of our new algorithm in online time efficiency, online space efficiency and schedulability.

(2) This dissertation proposed a novel partitioned scheduling algorithm MPVD to extend the state-of-the-art single-processor mixed-criticality scheduling algorithm EY-VD to multiprocessor platforms. The key idea of MPVD is to evenly allocate tasks with different criticality levels to different processors, in order to better explore the asymmetry between different crit-

icality levels and improve the system schedulability. Then we propose two enhancements to further improve the schedulability of MPVD. Experiments with randomly generated task sets show significant performance improvement of our proposed approach over existing algorithms.

(3) This dissertation proposed a novel partitioning policy for mixed-criticality scheduling on multiprocessor platforms. First, we integrate EY-VD into traditional workload partitioning schemes to get a multiprocessor mixed-criticality scheduling algorithm MC-PEDF. Although MC-PEDF performs better than previous solutions, we find that the traditional workload partitioning schemes are not suitable for mixed-criticality systems as it does not explore the asymmetricity of workload on different criticality levels. To overcome this problem, we propose a novel workload partitioning policy OCOP (one criticality one partition). OCOP allows tasks to be reassigned to a different processor when criticality mode switch occurs, thus can better balance the resource utilization among processors on different criticality levels. Based on OCOP, we propose our second partitioned scheduling algorithm MC-MP-EDF. Experiments with randomly generated workload show that MC-MP-EDF can drastically improve the system schedulability comparing with MC-PEDF and other previous algorithms, especially for systems with more processors.

(4) This dissertation proposed two approximate response time analysis methods RBF and IBF to evaluate the DRT models, both of which have pseudo-polynomial complexity. We quantitatively evaluate their analysis precision using the metric speedup factor. We prove that RBF has a speedup factor of 2, and this is tight even for dual-task systems. The speedup factor of IBF is an increasing function with respect to k, the number of interfering tasks. This function converges to 2 as k approaches infinity and equals 1 when k = 1, implying that the IBF analysis is exact for dual-task systems. We also conduct experiments to empirically evaluate the precision and efficiency of RBF and IBF with randomly generated task sets. Results show that the proposed approximate analysis methods have very high efficiency with low precision loss.

(5) This dissertation proposed a novel DRT task graph transformation method to improve system schedulability. The idea is to insert artificial delays to the release times of certain vertices of a task graph to get a new graph with a smoother workload, while still meeting the timing constraints of the original task graph. Delaying the release time of a vertex may smoothen the workload of some paths of the task graph, but at the same time make the workload of other paths even more bursty. We developed efficient techniques to search for an appropriate release time delay for each vertex. Experiments with randomly generated task systems show that the

proposed transformation method can make a significant number of task systems that was originally unschedulable to become schedulable, and the transformation procedure is very efficient and can easily handle large-scale task graph systems in very short time.

In summary, this dissertation studied various real-time scheduling problems for mixed-criticality systems and DRT systems. The results of this dissertation contribute to theoretical foundations, also provide reference value for the design and analysis for mixed-criticality and DRT systems.

Keywords: real-teime system; mixed-criticality systems; multi-processor; partitioned scheduling; EDF; DRT; response time analysis

目 录

附表清单

插图清单

第 1 章 绪 论

1.1 研究背景及意义

实时系统[1–3] 泛指那些对外部环境触发的事件必须在精确的时间约束内作出反应的计算系统。因此实时系统的运行时行为正确性不仅由计算结果决定，还取决于结果产生的时间是否满足设计要求[4]。过晚的时间相应可能是毫无价值，甚至是十分危险的。如今越来越多的复杂系统已经部分或完全地依赖于计算机控制，由此实时系统将扮演者愈发重要的作用。

现代实时嵌入式系统的一个重要的发展趋势为，在同一硬件平台中集成具有不同关键性级别的多个应用。而硬件平台的发展正处于从单核向多核的迁移过程中，将来会进一步发展至众核架构的平台。关键性则表示一个系统的应用在不同程度系统错误情景中保证正确性的级别。混合关键性系统包含两个或更多的不同关键性级别。一般标准中（例如 IEC 61508、DO-178B、DO-254 和 ISO 26262）通常不会超过 5 个关键性级别。为了满足诸如成本、空间、重量、发热和功耗等非功能需求的约束，工业界正将大多数复杂的嵌入式系统（例如汽车和飞行器）升级为到混合关键性系统。事实上，欧盟汽车工业的 AUTOSAR[5] 和飞行器领域的 ARINC[6] 等软件标准都涉及到了混合关键性的问题。

混合关键性系统要解决的根本问题是如何以规范的方式来平衡应用在不同模式下割裂的正确性需求，与资源有效地充分利用之间的矛盾。该问题涉及到建模和验证等理论问题，以及与软硬件设计与实现相关的系统问题。在传统的单处理器系统中性能优异的调度算法（如 RM[7]、EDF[8]）已被证实不能直接适用于混合关键性系统中。因为在混合关键性系统中，这些算法表现出很差的可调度性能，因此对混合关键性领域调度算法研究是全新的挑战。而多核、多处理器平台中的混合关键性系统调度也面临着同样的问题，并且调度算法的设计与分析更为复杂。

另一方面，传统的实时任务系统模型通常被定义为由周期性重复执行的任务构成的集合[8, 9]。但是实际系统中存在很多并非完全是周期性重复计算的行为，而这些行为很难被这些简单的周期任务模型所精确描述。例如依赖于可变速率行为的内燃机燃料注射控制器系统[10]，和视频编解码器中基于帧的执行[11]。针对这些复杂结构的一种自然表示方法就是任务图。近些年来，很多基于图的任务模型[7, 11–16] 被不断的提出，以为精确地描述复杂的嵌入式系统。然而对于大多数已有的实时任务模型（不包括简单的周期性模型），对静态优先级调度的精确响应时间分析在时间复杂度上

都是难于接受的。在这些模型中只有非常简单的 L&L [8] 和 sporadic [9] 模型能够在伪多项式时间内计算出响应时间。而在模型中引入分支或相位调整后[17]，原问题便转变为强反 NP 难问题（Strongly coNP-hard）。在文献 [18, 19] 中提出了非常有效的优化技术，能够减掉指数级状态空间的很大一部分。但是，这些方法在一般情况下仍是指数级的复杂度，并且在处理大规模系统时依然可能面临状态空间爆炸的问题。

由于现代嵌入式系统设计的日益复杂，传统基于周期的实时系统模型很难满足精确描述复杂系统的需求。基于有向图的 DRT 模型具备强大的系统描述能力，但是对于该模型的分析却面临着时间复杂度过大，相关技术不成熟等问题。

本文将基于现有的研究成果，围绕混合关键性系统在单核和多核处理器平台中的实时调度问题，和高效的 DRT 系统分析与可调度性改进方法展开研究工作。

1.2 国内外研究现状

2007 年，Vestal 发表了第一篇混合关键性系统领域的论文[20]，并将传统的固定优先级实时调度算法扩展到混合关键性系统中。文献 [20] 使用了有所限制性的单处理器工作流模型，并使用了响应时间分析[21] 的方法。该文表明传统实时调度中最优的固定优先级分配算法，如速率单调算法（RM[7]）和截止期单调算法（DM[22]）在混合关键性系统中都不是最优的。但是 Audsley 提出的最优优先级分配算法（OPA[23]）在混合关键性系统中是适用的。Baruah 等在文献 [24] 中已证明判断一个给定的作业集合使用最优的算法是否可以被调度是强 NP 难问题。因此对于混合关键性任务实时调度的研究重点集中在找到在实践中有很好性能的次优调度算法。

运行时系统关键性级别的变化与在不同操作模式间切换的系统行为比较类似（尽管存在一些显著的不同[25, 26]）。在高关键性模式下，只有较少的任务需要调度，但是这些任务却拥有更长的执行时间或着更短的周期。在模式切换协议相关的文献[27–33]中存在的一个重要问题是：系统能够在任意的模式下成功调度，却不能够在模式切换期间满足截止期约束[33]。该问题同样存在于混合关键性系统在经历关键性级别切换时的系统行为中。

近年来在混合关键性系统研究领域已经发表的大量论文中，大部分均是基于单处理平台和相互独立的任务集合。其中早期的很多文章仅考虑了有限数量混合关键性作业的限制性模型 [24, 34–43]，而其中大被研究已经被适用于更广泛系统任务模型的工作所取代。文献 [44] 中，Dorin 等证明了 Vestal 方法中使用 Audsley 优先级分配算法[23]是最优的。文献 [44] 还将混合关键性模式扩展至包含释放抖动行为，并介绍了如何进行敏感性分析。

Baruah 等人在文献 [45] 中提出适用于混合关键性领域的 OCBP（Own Criticality Based Priority）调度算法，该算法主要适用于调度由有限数量的作业组成的作业集合，然后运用一种基于 Audsley 算法 [46] 的方法对每个作业赋予相应的优先级，这是一种固定作业优先级调度算法，很显然这种算法的局限性很大。在文献 [39] 中 Li 和 Baruah 在 OCBP 算法基础上提出 LB 调度算法，LB 调度算法采用了忙碌周期（busy period）的技术，并对在忙碌期内释放的所有作业赋予适当的优先级。虽然 LB 算法可以将 OCBP 计算优先级原则应用到偶发性任务模型中，但该算法的运行时时间开销较大（伪多项式级别），难以被实际系统所接受。存针对以上不足 Guan 等在文献 [47] 中提出了 PLRS （Priority List Reuse Scheduling）调度算法。PLRS 算法分析了 LB 算法中运行时重复计算作业优先级的情况，充分利用离线优先级的计算结果，有效降低了运行时调度开销，提高了系统性能。虽然最后 PLRS 调度算法已经很好解决 LB 存在的问题，其时间复杂度也降到了 $O(n^2)$，但是其在对任务优先级重新计算过程中仍有很多无意义的计算，浪费系统资源，降低了系统利用率。

在文献 [48, 49] 中，Zhao 等人扩展了 AMC-rtb 方法，将抢占阈值[50] 的概念添加到模型中。Burns 和 Davis 在文献 [51] 中进行了另一个综合 AMC-rtb 和已有调度理论的研究。他们考虑使用了递延抢占[52, 53]，并较之于完全抢占 AMC-rtb 展示出显著的性能提升。在文献 [54–59] 中进行了进一步的研究工作。

De Niz 等在文献 [60] 中提出了用于调度偶发性混合关键性任务的零松弛时间算法。这种算法的关键点在于通过动态的调整作业优先级来避免低关键性作业对高关键性作业造成的运行时调度干扰。这种方法已经被扩展到用于不可抢占共享资源平台[61] 和分布式平台，在这些平台上，混合关键性任务需要被分配到不同的执行单元[62]。但是 Huang 等 在文献 [63] 中发现如果一个低关键性（高优先级）任务执行超过了它的截止期，那么一个高关键性（低优先级）任务可能会错失其截止期。他们提出低关键性任务必须在其截止期之前被终止执行或被赋予背景优先级。他们在文献 [64, 65] 中进一步提升了算法的性能。Neukirchner 等在文献 [66, 67] 中采用并扩展了多种策略进行激活模式的监测。他们提出的多模式方法被证明是安全有效的。

在 2008 年，Baruah 和 Vestal 首先在文献 [68] 中研究了混合关键性系统使用 EDF（Earliest Deadline First）进行调度的问题。Park 和 Kim 在文献 [41] 中介绍了基于松弛时间的 EDF 作业调度算法 CBEDF （Criticality Based EDF）。Baruah 在文献 [45] 中提出一种基于适用于混合关键性系统的改进型 EDF 算法 EDF-VD，该算法提出了一个虚拟截止期（Virtual Deadlines）的概念，每个高关键性任务有两个虚拟相对截止期（均不大于真实的相对截止期）。其中一个被视为系统处于低关键性模式时高关键性任务

的相对截止期，另一个是系统处于高关键性模式时高关键性任务的相对截止期。当实时任务系统的资源利用率满足一定条件时，可以利用此 EDF 算法调度。Ekberg 和 Yi 在文献 [69, 70] 中提出了用于计算采用虚拟相对截止期的实时任务在不同关键性模式下 DBF 的方法，并基于该方法提出了一个有效的为混合关键性实时任务系统的高关键性任务计算虚拟相对截止期的贪心算法 EY-VD。文献 [69] 中的方法具有比之前的策略[71] 更好的性能。在文献 [72] 中 Easwaran 提出了更为精确的分析方法，但该方法是否能适用于多于两个关键性级别的系统仍然是未知的。Yao 等在文献 [73] 中提出了进一步的优化技术。他们使用了改进的 EDF 可调度性判断方法 QPA[74]，和基于遗传算法的虚拟截止期搜索算法。其它的混合关键性系统 EDF 调度算法的研究还出现在文献 [75–78] 中。

Anderson 等人于 2009 年首次发表了关于多核、多处理器平中混合关键性系统的论文[79]，并在 2010 年进行了进一步讨论[80]。他们将关键性分为五个级别，并对于不同关键性级别的任务采用不同的调度算法进行运行时调度。该课题组还对多处理器平台中的操作系统开销问题进行了评估[81]。Mollison 等人在文献 [82] 中将不同关键性级别的任务封装在一起，然后在多核系统中用关键性单调优先级分配策略调度这些分组。Bommert[83] 和 Liu[84] 等人同样采用该框架对混合关键性并行任务的调度问题开展了相关工作。文献 [85] 中还进行了在多处理器平台上的混合关键性同步系统的实现工作。

Lakshmanan 等在文献 [61] 中采用 Compress-on-Overload 划分策略，将单处理器中的松弛调度方法[60] 扩展到多处理器系统，从而解决了混合关键性任务的划分调度问题。Tamas-Selicean 和 Pop 在文献 [86–88] 中解决了静态调度（循环执行）和时间分配方面的分布式任务分配问题。他们发现可以通过提高一些任务的关键性级别来改进可调度性，并使得单一关键性的划分更为平衡。但该法必须使用搜索优化程序（模拟退火算法[87] 和禁忌搜索算法[86, 88]）以确保通过最低的资源使用率满足可调度性要求。Zhang 等人在文献 [89] 中通过使用另一种搜索程序（遗传算法[90]）进行任务分配以保证能量消耗最小（满足绝对安全和时间约束）。Alonso 等人在文献 [91] 中提出了辅助任务划分的工具箱。

Kelly 等人在文献 [92] 中则采用一种更为直接的任务划分方法：他们在考虑多处理器平均划分的前提下，基于任务利用率或任务关键性级别降序准则对 First-Fit、Best-Fit 方法等预排序方法进行了比较。通过 Vestal 原始分析法对每个处理器进行调度测试，最终认为以任务关键性级别降序的 First-Fit 方法最优。Rodriguez 等人在文献 [93] 中对采用 EDF-VD 框架的 EDF 调度算法的多种可能方案进行了综合评价。他

们的一个结论是对于高关键性任务采用 Worst-Fit 划分策略，并对低关键性任务采用 First-Fit 划分策略组合方法时更为有效的。Gratia 等人在文献 [94] 中提出了混合关键性系统全局分配方案。他们通过使用 RUN 调度程序[95]（采用分层调度框架）来调解高关键性和低关键性任务。Bletsas 和 Petters 等人在文献 [96, 97] 中研究了介于完全划分调度和完全全局调度之间的准划分调度问题。Axer 等人在文献 [98] 中针对双关键性容错系统，提出了高关键性任务被复制而低关键性任务不被复制的运行时调度方案，适用于 MPSoC（多处理器片上系统）上相互独立的周期性任务调度。并提出了任务划分可靠性分析的方法。Lee 等人 在文献 [99] 中提出一种更为理论化的基于流式任务模型的方案。流式任务模型[100] 执行每个任务的速率与其利用率成正比。如果不考虑运行时上下文切换的成本，那么改方法能够提供一种多处理器平台混合关机性调度的最优方案。他们还提出了该模型的的一个可实现版本，并在模拟实验中具有良好的性能表现。

Li 和 Baruah 在文献 [101] 中研究了 EDF-VD 算法[102] 在多处理器系统中的扩展, 并结合传统的 fpEDF 算法[103] 提出了全局调度算法 Global-fpEDF，实验评价表明这两种算法的联合效果较好。作者在文献 [104] 中进一步将 EDF-VD 算扩展为划分调度算法 MC-Partition，并比较了混合关键性系统中的划分调度和全局调度，结果表明划分调度是迄今为止采取的最有效的办法。尽管如此，Pathan 在文献 [105] 中对全局调度固定的优先级系统进行了分析，他们采用单处理器调度和多处理器调度相整合的方法，将多处理器系统中传统的全局调度算法扩展到混合关键性系统中，并证明该方法为最佳的优先级排序。

Kritikakou 等在文献 [106, 107] 中提出了一种混合关键性系统多核调度的新方法。他们发现在不同处理器核心上运行任务时，由于硬件平台使用共享总线以及内存控制器等因素的影响，高关键性任务将受到低关键性任务的干扰。通过监控高关键性任务的执行时间，在未来的干涉不能被容忍时触发预警，并终止所有低关键性任务的执行。文献[107] 在多处核平台上进行了实现，并通过实验验证了该方案的有效性。Socci 等在文献 [108] 中对上述工作进行了支持有限约束的扩展，但是该工作仅适用于作业而非任务。

对于执行在具有确定性行为硬件平台上的系统，没有特定的任务都拥有唯一的实际最差执行时间（Worst-Case Execution Time，WCET）。但是通常并不能获得 WCET 的绝对精确值。对于执行在具有时间随机性行为硬件平台[109] 中的系统，每个任务则拥有概率的最差执行时间（pWCET）[110–113]。采用 pWCET 的概率分布能够有效地定义同一任务在不同关键性级别下因最大容错率的差异导致的不同 WCET 评价。

在文献 [114–117] 中研究了可变周期的混合关键性任务模型。该模型中任务被定义为事件处理器。任务的关键性级别越高，需要处理的事件的频率也就越大，对处理器资源的需求也就越大。

在实时系统设计阶段，通常选择抽象的实时任务模型来评价系统的非功能属性，比如截止期的时间约束。设计者选择合适的模型时，既要考虑模型具有足够的可表达性来尽可能精确地描述系统行为，还要兼顾对于所选模型的分析足够有效率，以满足系统规模增大时依然能够在可接受时间内返回分析结果。著名的 Liu&Layland 模型[7] 将实时任务抽象为几个简单的相关参数（执行时间和周期）组成的周期执行模型。该模型的分析效率非常高，但是对于任务类型的限制十分严格。相反的，时间自动机模型[118] 强大的形式化描述能够精确的为复杂系统建模，但其可调度性测试的时间开销却极其繁重而难以接受的。

将分析效率作为重要的考量因素的前提下，研究者们提出了很多表达性很强，并且分析效率很高的实时任务模型。早期的 Liu & Layland 模型的泛化版本包括偶发任务模型[119]，和多帧模型（Multiframe）[120]。而多帧模型后来又被统一到广义多帧模型（Generalized Multiframe，GMF）[121] 和循环分支模型 [9] 中。概括而言，GMF 解除了任务的周期与截止期间的耦合关系，提供了一个不同作业类型（帧）的集合，并允许偶性发作业在不同帧内循环释放。尽管进行了泛化，但可调度性分析的复杂度仍控制在伪多项式级别，因此分析开销是可接受的。通过放松同一任务释放不同类型作业次序的限制，GMF 模型又被扩展为 RRT（Recurring Real-Time）模型[10]。RRT 允许分支代码被建模成有向无环图（Directed Acyclic Graph，DAG），因此大大增强了模型的描述能力。在文献 [12] 中,GMF 被扩展为 ncGMF（non- cyclic GMF）模型。在该模型中不同类型的作业可以以任意的次序释放，因此运行时同一任务释放的不同类型作业不一定是循环的。综合 RRT 模型与 ncGMF 模型，在文献 [11] 中提出了 ncRRT（non-cyclic RRT）模型。ncRRT 通过允许重新开始 DAG 遍历的方式将 RRT 模型增加了非循环释放的行为属性。通过采用在文献 [122] 中的动态规划方法，文献 [11] 证明了 ncRRT 模型的可调度分析复杂度依然时伪多项式级别的。

一个比之前介绍的模型更为泛化的模型是 Stigge 等人在文献 [19] 中提出的实时任务有向图模型（Digraph Real-Time，DRT）。DRT 能够对任意可以通过有向图描述的实时任务进行建模，具备强大的描述能力。然而利用文献 [17] 中类似的规约技术，在文献 [123] 中证明对该模型进行可调度分析的问题是强反 NP 难（Strongly *co*NP-hard）的。在文献 [124] 中提出了一种与文献 [125] 中类似的工作量抽象技术来对有向图任务集合进行近似的分析。但是在文献 [124] 中仅通过模拟实验评价了算法性能，并没有

进行量化地性能分析。

在网络研究领域，整形（shaping）是一种被广泛研究的技术。该技术通过延迟数据包使其满足所期望的传输原则[126, 127]。通过整形可以平滑突发传输流，优化缓冲区需求，改善延迟，增加特定种类数据包的可用带宽等。整形的思想同样被应用于实时嵌入式系统的设计。Wandeler 等在文献 [128, 129] 中扩展了网络演算（Network Calculus ）[130] 中贪心整形（greedy shaper）的技术来模块化分析实时系统的性能。Richter 等 在文献 [131] 中介绍了一种限制性整形方法 EAFs（Event Adaption Functions）。Phan 和 Lee 在文献 [132] 中设计了一种新型的带抖动的周期任务整形方法。

另一个可调度性分析的工具时实时演算（Real-Time Calculus）[133]。实时演算能够使用到达曲线模型对事实系统进行组成可调度分析。该分析方法的原理与需求上界函数方法类似。

1.3 本文研究内容与贡献

本文的研究内容及贡献主要有以下几个方面：

1.3.1 基于混合关键系统

（1）提出了基于 OCBP 策略具有线性运行时时间复杂度的固定作业优先级单处理器混合关键性实时调度算法 LPA。之前基于 OCBP 算法的 LB、PLRS 等算法虽然成功将 OCBP 算法扩展到偶发任务模型，但其较高运行时复杂度限制了在实际系统中的应用。本文提出的 LPA 算法，在运行时尽可能晚的对各个任务的优先级进行调整，从而避免繁重且不必需的运行时优先级调整计算，只是当作业被释放时进行轻微优先级调整，进而有效的改善了系统复杂度（线性时间复杂度）。本文还提出了更精确的混合关键性系统忙碌周期上界的计算方法，使用该方法不但可以改善运行时的空间效率，还可以一定程度上提升系统的可调度性。

（2）提出了分别对高关键性任务和低关键性任务采用不同策略的混合划分调度算法 MPVD（Mixed-criticality Partitioning with Virtual Deadlines）。该算法首先使用最差适应（Worst-Fit）划分策略来分配高关键性任务，然后使用首次适应（First-Fit）划分策略来分配低关键性任务，并在运行时采用 EY-VD 算法进行调度。通过混合划分策略，能够使高关键性的任务被均匀地分配到不同处理器（核心）中，以使得 EY-VD 算法能够有更多的空间来平衡不同关键性级别下的工作量，并提升系统的可调度性。MPVD 算法的性能会随着处理器（核心）数量的增加而出现明显的下降。为了解决该问题，本章提出了两个优化算法来进一步提升算法的性能。首先考虑到由于高关键性

任务在所有处理器中的均匀分配，可能导致无法为利用率较高的低关键性任务适配到拥有足够剩余资源的处理器，造成充足的处理器（核心）剩余资源总量无法被充分利用。针对该问题，本章提出了为利用率较高的低关键性任务预留资源的策略。另外，本章还提出了一种优化的虚拟截止期调整算法来进一步提升 MPVD 算法的性能。

（3）提出了为不同的系统关键性模式采用不同的任务集合划分方案的 OCOP 混合关键性多处理器划分调度策略。本文首先基于传统划分调度策略，结合单处理器 EDF-VD 算法，提出了多处理器混合关键性系统中的基于非固定优先级的划分调度算法 MC-PEDF。由于混合关键性系统在不同系统关键性级别下的任务工作量分布有很大差异，OCOP 放松了传统多处理器划分策略禁止所有运行时作业迁移的限制，允许在系统关键性模式切换时为任务重新分配处理器，从而显著提升了系统在不同关键性模式中的处理器资源利用率。基于 OCOP 本文还提出了新的划分调度算法 MC-MP-EDF。实验结果表明 MC-PEDF 和 MC-MP-EDF 算法在可调度性上优于先前的多处理器混合关键性实时调度算法，而采用 OCOP 划分调度策略的 MC-MP-EDF 算法则具有更好的可调度性能。

1.3.2 基于实时任务有向图模型

（1）提出了两种分析 DRT 任务系统响应时间的近似分析方法 RBF 和 IBF，并通过加速比分析，量化评价了该两种近似方法的性能。其中基于加速比的的性能评价被广泛应用于众多调度问题的近似算法分析中。本章的主要成果可总结如下：

- RBF 近似响应时间分析方法的精确加速比为 2（即便是仅包含两个任务的系统）。
- IBF 近似响应时间分析方法的加速比为 $1+\frac{\sqrt{k^2-k}}{k}$，其中 k 为干涉任务（优先级高于当前分析任务者）的数量。

因为当 $k=1$ 时有 $1+\frac{\sqrt{k^2-k}}{k}=1$，所以对于只有两个任务的系统 IBF 方法能够得到精确解。另外由于 $1+\frac{\sqrt{k^2-k}}{k}$ 为以 k 为自变量的单调递增函数，因此 IBF 方法的分析精度随着干涉任务数量的增加而降低。而当 k 趋于无穷时，IBF 的加速比也趋近于 2（与 RBF 一致）。这两种方法均为伪多项式时间复杂度，并可以很高效地处理大规模的任务系统。

（2）提出了一种高效率的 DRT 任务有向图整形算法。本文提出的方法通过对 DRT 任务对应有向图中的每个顶点依次进行整形操作，并调整与被整形顶点相关联的边上参数取值。通过发掘任务图中的一些性质进行合理的抽象，算法能够快速并且有效的完成整形操作。通过对一些关键顶点快速的设置合适的作业释放时间延迟参

数，能够使得整形后的 DRT 任务释放的工作量更加均匀。从而使得一些不可调度的 DRT 任务系统变得可调度。实验结果表明本文提出的整形算能够显著提升 DRT 系统的可调度性能。

1.4 本文组织结构

本文共分 7 章，各章具体内容组织如下：

第 1 章是绪论部分，主要阐述了本文的研究背景及意义，混合关键性系统和 DRT 系统实时调度主要内容和相关技术，分析了国内外研究现状与尚存在的问题，介绍了本文的主要内容和贡献，最后说明了本文的篇章结构。

第 2 章主要研究了但处理器平台中基于 OCBP 的混合关键性偶发任务系统的实时调度问题。分析了造成之前提出算法运行时效率底下的原因，提出了线性时间复杂度的运行时调度算法 LPA，和更精确的忙碌周期上界计算方法，显著降低了系统运行时的时间、空间调度开销病提升了系统的可调度性能。

第 3 章主要研究了多处理器平台中基于 EY-VD 的混合关键性偶发任务系统的划分调度问题。分析了传统的划分策略应用到混合关键性系统时的不足，提出了针对不同关键性任务的采用不同划分策略的混合划分调度算法 MPVD。并提出了两种优化方法，显著提升了算法的性能。

第 4 章主要研究了多处理器平台中混合关键性偶发任务系统划分调度策略问题。针对混合关键性系统的特点，提出了允许系统关键性模式切换时为任务重新分配处理器的 OCOP 划分调度策略。并通过随机生成任务集合的实验评价了基于该策略设计的划分调度算法的性能。

第 5 章主要研究了实时任务有向图系统的近似响应时间分析方法。提出了两种近似分析方法 RBF 与 IBF，并通过计算加速比和模拟实验分析评价了该两种方法的精确度，以及时间效率。

第 6 章主要研究了通过调整实时任务有向图的参数来提升系统可调度性能的方法。提出了高效（伪多项式时间复杂度）地进行 DRT 任务整形的方法，显著提升了系统的可调度性能。

第 7 章总结全文，并提出了未来研究方向以及可以继续深入研究的内容。

第 2 章 线性时间复杂度混合关键性调度算法

伴随着信息产业的飞速发展，为了满足日益复杂的需求，嵌入式系统集成了越来越多的功能。为了满足系统功耗、体积、成本等非功能因素的约束，现代嵌入式系统通常将多种不同各类的具有不同关键性的功能集成到同一硬件平台当中。

在传统的单处理器系统中，存在很多性能优异的调度算法（如 EDF [8]），但这些算法已被证实在混合关键性领域不能直接适用。因为在混合关键性系统中，这些算法表现出很差的调度性能，因此对混合关键性领域调度算法研究是全新的挑战。Vestal [20] 于 2007 年首先形式化的提出混合关键性系统的模型及其实时调度的问题。该问题引起了学术界的广泛关注和大量学者们的深入研究，并获得了大量的研究成果。其中 Sanjoy Baruah 等人率先提出适用于混合关键性系统的 OCBP [45]（Own Criticality Based Priority）单核调度算法，其为作业级的固定优先级混合关键性实时调度算法，可以成功地对混合关键性任务进行调度，同时表现出比较良好性能，但它工作在比较理想的环境下：确定的作业释放时间和有限的作业数量。然而典型的实时系统都是基于偶发性任务模型（释放时间不确定，但是有一个最小的释放时间间隔）。

为寻找适合混合关键性偶发任务模型的调度算法，Li 和 Baruah 提出了基于 OCBP 的扩展调度算法，本文称之为 LB [39]，它可以应用于混合关键性偶发性任务的调度，经过理论证明该算法的时间复杂度为伪多项式级别，这对于实时嵌入式系统的调度而言难以接受；后来，Guan 等人提出 PLRS [47] 调度算法，通过对 LB 运行时优先级调整方式的改进，将算法的时间复杂度降低到平方级别，运行时性能得到显著提升。

虽然 PLRS 调度算法将时间复杂度降到平方级别，但对实时嵌入式系统来说还是过于繁重，尤其它在系统运行中发生作业抢占时，进行了过多无效的作业优先级重复计算，增加了系统时间开销；同时由于 LB、PLRS 在将无限作业数转化为有限作业数时采用了忙碌周期概念，但采用的计算忙碌周期上界的算法过于悲观，导致目标作业集中待分配优先级作业数量过多，从而增加了系统的运行时空间开销并降低了系统的可调度性，这对于资源有限的嵌入式系统影响较大。因此设计新的计算方法，求出更为精确的改进忙碌周期上界，对系统在时间、空间及可调度性上性能的提升都具有重要的现实意义和应用价值。

针对上述提到的已有算法不足，本文主要以降低系统时间开销、空间开销及提高可调度性等为目标，提出新型的混合关键性偶发任务系统实时调度算法。具体设计目标包括如下几个方面：

- 降低系统时间开销，因为现有调度算法在作业发生抢占时优先级重复计算的次数过多，通过对该情况改善来有效降低系统时间开销，目标是将系统的时间复杂度降到线性的。
- 降低系统空间开销，LB、PLRS 调度算法利用的忙碌周期上界过于悲观，通过研究更为精确的上界计算方法来降低系统空间开销。目标计算更小更精确的上界，以此来减少存储的作业优先级的数量。
- 提高可调度性，同样地，在调度算法中利用悲观的忙碌周期上界使得系统的可调度性降低，通过重新求该上界来提高可调度性。

通过以上几方面的改进对于处理器紧张、资源有限的嵌入式实时系统来说意义很大；同时因为目前处理器主流都是多核的，所以在多核处理器上对混合关键性的研究也将非常有意义，相信也会是一个挑战。

2.1 系统模型与定义

本小结将给出单处理器可抢占平台中，混合关键性偶发任务模型（Mixed-Criticality Sporadic Task Model）的形式化定义。与传统的偶发性任务模型类似，我们将混合关键性偶发任务模型定义为一个由相互独立的潜在无限数量的混合关键性作业序列组成的集合。

定义 2.1 (混合关键性任务)：每个混合关键性任务被定义为一个四元组：$\tau_i = (T_i, D_i, C_i, \zeta_i)$，该元组各元素语义的说明如下：

- $T_i \in R^+$，表示实时任务 τ_i 任意连续释放的两个作业间的最小释放时间间隔。
- $D_i \in R^+$，表示实时任务 τ_i 的相对截止期。
- $C_i \in N^+ \rightarrow N^+$，表示实时任务 τ_i 的最差执行时间函数，返回 τ_i 在不同关键性下系统评估的最差执行时间取值。
- $\zeta_i \in \{1,2,\ldots L\}$，表示关键性级别。本文约定该值越大，则对应的关键性级别越高，并用 L 表示该混合关键性系统中关键性级别的最大取值。

需要注意的是，任务的相对截止期与其最小释放间隔之间没有任何的约束条件，即 D_i 可以大于、小于或等于 T_i。

令 J_i^j 表示由任务 τ_i 在运行时释放的第 j 个作业，其释放时间由 $r_i^j \in R^+$ 表示。据此，J_i^j 的绝对截止期为 $d_i^j = r_i^j + D_i$，完成时间为 $f_i^j : r_i^j < f_i^j \leq d_i^j$。而由任务 τ_i 释放的所有运行时作业均共享相同的最差执行时间函数 C_i 和关键性级别 ζ_i。

本章提出的 LPA 算法为作业级固定优先级实时调度算法，即每个运行时作业在

释放时刻被分配一个优先级，并一直保持其优先级直至执行完毕或被系统终止执行。遵从传统实时调度理论的惯例，本章令更小的优先级取值表示更高的优先级。

2.1.1 混合关键性偶发任务系统的运行时行为

混合关键性运行时作业的语义定义如下：一个有任务 τ_i 在时刻 r_i^j 释放的混合关键性作业 J_i^j 需要运行时执行 γ_i^j 个时间单位。而 γ_i^j 的精确取值在作业 J_i^j 执行完毕前是不能够被准确预知的。根据运行时测量的 γ_i^j 取值，定义混合关键性系统经历 $\lambda-criticality$ 运行时行为的充分必要条件为

$$\lambda = \max_{\forall \gamma_i^j} \left\{ \ell | \min \left\{ \ell | \gamma_i^j \le C_i(\ell) \right\} \right\}. \tag{2.1}$$

特别的，如果任意作业 J_i^j 已经执行了 $C_i(L)$（最高关键性下的最差执行时间），却仍没有执行完毕，则定义该行为为运行时错误。在本章后面的讨论中假设系统不会经历运行时错误。

由式 (2.1) 可推出对于任意运行时作业 J_i^j，更长的运行时执行时间 γ_i^j 可能导致更高的系统运行时关键性级别。而系统中经历最高关键性运行时行为的作业决定了系统的实际运行时行为。

2.1.2 混合关键性系统的可调度性

混合关键性系统的可调度性取决于所有系统运行时行为下可调度性。由此定义调度算法 $\mathscr{A}$ 是混合关键性可调度的条件为：

定义 2.2 (混合关键性可调度性)：给定一个调度算法 $\mathscr{A}$，一个混合关键性任务系统 τ 在 $\mathscr{A}$ 调度下是混合关键性可调度的充分必要条件为：系统在经历任意的 λ-关键性运行时行为时均满足

$$\forall \tau_i : \zeta_i \ge \lambda \Rightarrow \forall J_i^j : J_i^j \text{ 均在其截止期} d_i^j \text{ 时刻或之前执行完成。}$$

特别注意的是，当系统经历 λ-行为的运行时行为时，所有关键性级别低于 λ 作业不再需要满足其截止期约束。系统设计者仅保证混合关键性任务 τ_i 在系统经历不超过关键性级别 ζ_i 的运行时行为时的时间正确性，而 τ_i 释放的作业在更高关键性的系统运行时行为下不需要在截止期之前执行完毕。

2.2 LPA 算法

每当系统在运行时发生任务抢占时刻，无论 PLRS 或 LBA 算法都会进行大量的优先级重新分配计算。这也是造成两者运行时效率低下的主要原因。但在实际系统运行时，很多作业会因其释放之前发生过多次抢占而造成其优先级被重复计算多次。而

优先级计划表中的很多优先级甚至会因当前忙碌周期的结束而不会被使用，但这些优先级同样会因为当前忙碌周期内的多次任务抢占而被重复计算多次。因此，优先级重新计算中的很大一部分工作量都是与运行时调度决策无关的冗余计算。

为了降低运行时调度的时间复杂度，本节提出一个新的调度算法 LPA（Lazy Priorities Adjustment）。LPA 尽可能晚的调整作业优先级，以尽可能的避免在作业抢占时刻的冗余优先级调整计算，达到提升运行时调度效率的目的。但运行时作业抢走发生时，LPA 不会立即为所有任务的作业执行优先级重新调整的繁重计算，代之的是为每个任务记录下该次抢占事件。当调度器需要为当前释放作业分配优先级时，LPA 能够根据优先级抢占记录来高效地为释放当前作业的任务更新优先级列表。通过该方法，LPA 能够避免大量繁重且非必须的在线优先级重新计算工作，并能够在作业释放的时刻进行高效低优先级分配。

另外本章还提出了用于更精确地计算混合关键性任务系统最大忙碌周期长度上界的新方法。相较于 LB 与 PLRS 算法，LPA 算法使用精确（小）的忙碌周期长度，不仅可以降低在线调度算法的空间开销，甚至能够在一定程度上提高系统的可调度性（可调度接受率）。关于忙碌周期的分析，将在下一节中详细介绍。

2.2.1 离线优先级分配算法

与 LB 和 PLRS 算法相似，LPA 算采用基于忙碌周期（Busy Period）方法来解决为偶发任务系统中潜在无限多数量作业需要分配优先级的问题。在任意时刻，调度器只需要为当前忙碌周期内可能释放的有限个作业分配优先级。

这是由于任何工作保持（Work-Conserving）调度算法，处理器当其仅当系统中已释放的所有作业全部执行完毕或者被终止时才会空闲，而在两个连续的处理器忙碌周期之间必然存在一段处理器空闲的时间。因此，任何在一个新的忙碌周期开始之前释放的任务或者已经执行完毕，或者被系统终止（比如，系统的关键性级别升高，调度器会终止所有较低关键性级别的作业的执行），都不会对新忙碌周期内释放作业的调度造成任何的影响。也即是说，不同忙碌周期内的实时任务调度是相互独立的。

离线优先级分配算法的第一步是计算一个 混合关键性偶发任务系统的忙碌周期最大长度。Li 和 Baruah 在文献 [39] 中介绍了一个计算忙碌周期最大长度上界的算法，本文将在 小节2.3介绍一个计算更精确上界的算法。

通过忙碌周期长度上界（记作 Γ），可以计算出系统中每个混合关键性实时任务 τ_i 在任意忙碌周期能够释放最大作业数量 $n_i = \left\lceil \frac{\Gamma}{T_i} \right\rceil$。特别的，用 I 表示一个忙碌周期内可能释放的所有作业的集合，I 内包含作业的数量为 $N = \sum_{\tau_i \in \pi} n_i$。

LPA 的离线调度算法使用 OCBP 方法为最大忙碌周期内释放作业集合 I 中所有作业分配优先级。算法使用 δ_i 表示由任务 τ_i 释放的在 I 中尚未分配优先级的作业数量。算法初始化时，设置 $\delta_i = n_i$。算法的每次迭代都会检查每个任务中标号最大的未分配优先级作业，当找到第一个满足式 (2.2)中条件的作业 $J_k^{\delta_k}$ 时，将当前的最低优先级分配给 $J_k^{\delta_k}$，本次迭代结束。

$$\sum_{\tau_{j\in\pi}} (C_j(\zeta_k)\cdot\delta_j) \le T_k\cdot(\delta_k - 1) + D_k \tag{2.2}$$

式 (2.2)左边部分表示所有将分配更高优先级或等于作业 $J_k^{\delta_k}$ 优先级的所有作业在 ζ_k-关键性系统行为下的工作量之和。式 (2.2)右边部分表示作业 $J_k^{\delta_k}$ 的绝对截止期与当前忙碌周期开始时间之间的最短长度。一般来说，算法可能在某次迭代总找到多于 1 个符合要求的作业。此时，算法可以选择其中任意一个来分配当前的最低优先级。在分配了当前最低优先级给作业 $J_k^{\delta_k}$ 之后，算法会将参数 δ_k 的取值更新为 $\delta_k - 1$，并继续重复上述过程直至在某次迭代时没有作业能够满足被赋予最低优先级的条件。如果所有集合 I 中的作业都分配了各自的优先级，则 LPA 算法的离线操作成功结束，并返回优先级分配结果，否则返回错误。本章定义表 Λ 来记录离线优先级分配算法的结果。

根据上述离线优先级分配算法的过程，可以推倒出如下的引理：

引理 2.1：对于任务 τ_i 在同一忙碌周期内释放的任意两个作业 J_i^m 和 J_i^n，如果 $n_i > m > n \ge 0$，则有 $\Lambda_i(m) > \Lambda_i(n)$。

例 2.1：考虑如表2.1中所示的混合关键性任务集。假设 $n_1 = 5$，$n_2 = 3$，$n_3 = 2$，$n_4 = 1$。第一阶段，依次检查所有的候选作业 $\{J_1^5, J_2^3, J_3^2, J_4^1\}$ 是否满足被分配最低优先级的条件。首先，检查作业 J_1^5。由于 $\zeta_1 = 1$，式 (2.2)的左边部分等于

$$C_1(1)\times\delta_1 + C_2(1)\times\delta_2 + C_3(1)\times\delta_3 + C_4(1)\times\delta_4 = 53。$$

另一方面，式 (2.2)的右边部分等于

表 2.1 任务集示例

Table 2.1 An Example Task Set

Task	T_i	D_i	ζ_i	$C_i(1)$	$C_i(2)$
τ_1	10	10	1(LOW)	1	1
τ_2	20	20	2(HIGH)	1	2
τ_3	30	30	1(LOW)	15	15
τ_4	50	50	2(HIGH)	15	25

$$T_1 \times (\delta_1 - 1) + D_1 = 50 < 53。$$

因此 J_1^5 不满足式 (2.2)中的条件，从而需要继续检查下一个候选作业 J_2^3。由于 $\zeta_2 = 2$，式 (2.2)的左边部分等于 $\sum_{\forall \tau_i} C_i(2) \cdot \delta_i = 66$，但是右边部分等于 60。因此 J_2^3 仍然不满足被分配最低优先级的条件，需要继续检查下一个候选作业 J_3^2。对于 J_3^2，左边等于 3，而右边等于 60，因此式 (2.2)的条件满足，该作业可以被分配最低优先级 11。然后更新 $\delta_3 = \delta_3 - 1 = 1$，并进行下一次的迭代。通过上述的步骤，可以为一个忙碌周期内释放的所有作业分配优先级，分配方案如下表所示。

Λ_1	1 2 4 8 9
Λ_2	3 6 10
Λ_3	5 11
Λ_4	7

需要注意的是，通过离线算法得到的优先级分配方案并不能直接用于在线的优先级分配。离线计算的优先级分配计划只是作为运行时在线优先级分配算法的辅助信息，直接简单地按该计划来分配运行时实际释放的作业分配优先级，并不能保证系统的可调度性。LPA 的运行时调度器会更具实时的系统状态信息，来动态调整优先级分配计划以保证系统的可调度性。

2.2.2 运行时调度算法

虽然在上一小节介绍了离线分配优先级的算法，但是即便该算法成功返回了优先级分配结果，也不能直接通过该静态作业优先级方案来保证系统的运行时可调度性。这是因为离线优先级分配算法的正确性是建立在所有任务在忙碌周期的起始时刻同时释放各自的第一个作业，并且后继作业 J_i^{c+1} 严格的释放时间严重满足 $r_i^{c+1} = r_i^c + T_i$ 等假设之上。但是偶发任务系统模型的运行时行为并不满足这些假设。因为任意偶发性任务可以在满足最小释放间隔的约束下在任意时刻释放新的作业。接下来，本小节将接受 LPA 的运行时调度算法来解决上述问题。

LPA 算法是一个作业级别的固定优先级可抢占调度算法。当系统初始化时，系统的运行时关键性级别 ℓ 将被设置为最小值 1。而在处理器执行完所有已释放的作业进行空闲状态时，ℓ 也将被重新设置为 1。系统的运行时关键性级别 ℓ 会在任意作业 J_i^c 已经执行了 $C_i(\ell)$，但仍然没有结束执行时被提升至 $\ell + 1$。同时，LPA 算法保证在任意时刻都将处理器分配给由关键性级别不小于当前系统关键性级别 ℓ 的任务释放释放的未执行完作业中优先级最高的作业。也即是说，当系统的运行时关键性级别提升至

$\ell+1$ 时，所有满足 $\zeta_i \leq \ell$ 的任务释放的作业都将被运行时调度算法忽略掉。

对于每个混合关键性任务 τ_i，定义变量 idx_i 来表示当前忙碌周期中该任务将要释放作业 J_i^c 的顺序标号。LPA 运行时调度算法不会直接使用优先级计划表中对应的 $\Lambda_i(c-1)$ 值来作为改作业的运行时优先级。其中，$\Lambda_i()$ 的标号从 0 开始，因此 Λ_i 中的第 c 个元素为 $\Lambda_i(c-1)$)。LPA 算法会使用一个调整取值 $\Lambda_i(c-\alpha_i):0\leq\alpha_i\leq c$ 来作为作业 J_i^c 的运行时优先级。而 LPA 算法的主要目标是更有效率地为每个释放作业 J_i^c 计算一个适合的便宜参数 α_i，并根据此参数来计算当前忙碌周期中作业的运行时优先级 $prt(J_i^c)$ 。

每当系统进入一个新的忙碌周期时，LPA 算法会将每个任务 τ_i 的参数 idx_i 和 α_i 的取值分别重置为 1。也即是说，在当前忙碌周期内，每个任务的第一个释放作业 J_s^1 将被分配优先级计划表 Λ_s 中的第一个元素值 $\Lambda_s(0)$ 作为其运行时优先级。而后续释放的作业将被使用参数 $\alpha_i=1$ 来计算优先级，直至系统监测到运行时抢占的发生。而不同于 LB 算法和 PLRS 算法的是，LPA 算法的最显著特点是并不会在每次抢占发生的时刻均为所有可能释放的作业重新计算优先级。取而代之的是，LPA 算法为每个任务 $\alpha_i=1$ 定义了一个偏移变量 δ_i，来记录在时间区间 $(r_i^{c-1},r_i^c]$ 内所有被抢占作业优先级的最大值。通过变量 δ_i，可以在将来作业真实释放的时刻对其优先级进行快速地调整操作。这也是 LPA 算法的运行时时间复杂度能够控制在线性界别的关键所在。

为了更快捷地为每个运行时作业计算合适的偏移变量 α_i，LPA 算法还维护一个辅助集合 $\Omega_i=\{(x_1,y_1),(x_2,y_2),\cdots\}$。对于集合 Ω_i 中的每个二元组 (x_j,y_j)，元素 x_j 记录了当前忙碌周期中曾经被使用过的一个偏移变量 α_i' 值，而元素 y_j 记录了当该二元组被拆入时的被抢占作业的优先级 δ_i'。在图2.1中描述了 LPA 算法的在线优先级调整过程 **AdjustPrt** 。

当一个运行时作业 J_k^c 释放时，LPA 算法会执行优先级调整过程 **AdjustPrt** 来检查之前释放的所有作业是否都已经全部执行完毕。如果是，则系统进入一个新的忙碌周期，而 LPA 算法将所有调度相关变量都更新为初始值。如果不是并且该释放作业的关键性级别不小于当前系统的关键性级别，那么 LPA 算法将继续执行 **AdjustPrt** 过程来为该作业分配一个合适的运行时优先级，并将其插入到就绪队列等待调度。否则 LPA 算法将会舍弃掉该作业不予调度。

AdjustPrt过程首先使用变量 α' 来记录当前偏移变量 α_k 的取值，然后尝试赋予实时作业 J_k^c 予优先级 $P_k=\Lambda_k(idx_k-\alpha_k)$，并比较 P_k 与变量 δ_k 和当前系统执行作业的优先级高低。如果 P_k 为最高优先级（取值最小），**AdjustPrt**过程将更新变量 α_k 的值为 idx_k，并将 P_k 的取值更新为 $\Lambda_k(0)$。然后 **AdjustPrt**将继续比较 P_k 与当前作业优先级的

1: **if** there are no unterminated jobs except J_k^c **then**
2: **for** each $\tau_i \in \pi$ **do**
3: $(idx_i, \alpha_i, \delta_i, \Omega_i) \leftarrow (1,1,0,\emptyset)$
4: **end for**
5: $prt(J_k^c) \leftarrow \Lambda_k(0)$; $idx_k \leftarrow idx_k + 1$
6: **else**
7: $P_{cur} \leftarrow priority\ of\ the\ current\ job$
8: $\alpha' \leftarrow \alpha_k$; $P_k \leftarrow \Lambda_k(idx_k - \alpha')$
9: **if** $P_k < \max\{\delta_k, P_{cur}\}$ **then**
10: $\alpha_k \leftarrow idx_k$; $P_k \leftarrow \Lambda_k(0)$
11: **end if**
12: **if** $P_k < P_{cur}$ **then**
13: update $\delta_i \leftarrow \max\{\delta_i, P_{cur}\}$ for $\forall \tau_i$
14: **end if**
15: Modify($\Omega_k, \alpha', \alpha_k, \delta_k$) {check and update Ω_k}
16: $prt(J_k^c) \leftarrow P_k$; $(idx_k, \delta_k) \leftarrow (idx_k + 1, 0)$
17: **end if**

图 2.1 运行时优先级分配算法 AdjustPrt
Fig. 2.1 The online priority assignment routine AdjustPrt.

高低，并以此来判定是否发生作业抢占。如果 P_k 为更高优先级，则 **AdjustPrt**过程将按照算法描述的第 13 行来为每个任务更新抢占记录变量 δ_i 的取值。

辅助集合 Ω_k 和偏移变量 α_k 的更新操作将通过函数 **Modify** 来实现。该函数主要由两部分组合，具体伪代码描述如 图2.2 所示。函数的前半部分（从第 1 行开始，至第 7 行结束）首先检查之前存储的偏移变量值 α' 与当前偏移变量值 α_k 的大小。当 $\alpha_k < \alpha_k$ 时，表明 α_k 刚刚被调度程序更新，此时需要将之前的偏移变量 α' 存储到 α' 中，以供将来新作业释放时使用。函数 **Modify**的第 2 行至第 5 行会将记录的被抢占优先级 δ_i 不大于当前被抢占优先级 δ_k 的所有元组从辅助集合 Ω_k 中删除掉，同时将变量 α' 的取值更新为删除元组中记录的被抢占优先级的最小值。然后函数 **Modify** 在第 8 行至第 10 行检查当前的偏移变量 α_k 是否需要被辅助集合 Ω_k 中的某个值所代替。之后将辅助集合 Ω_k 中记录的偏移变量 δ_i 不小于当前被偏移变量 δ_k 的所有元组全部删除掉。

最后，算法 **AdjustPrt** 将释放的运行时作业 J_k^c 的运行时优先级赋为 P_k，并将变量 idx_k 和 δ_k 分别更新为 $idx_k + 1$ 和 0 。

对于任意混合关键性偶发实时任务集合，如果能够被 LPA 的离线优先级算法成功为其最大忙碌周期内的所有作业分配优先级，那么 LPA 的运行时调度算法则可以保证该集合在运行时是混合关键性可调度的。对于该性质的证明请参考小节2.4。

Modify($\Omega_k, \alpha', \alpha_k, \delta_k$)

1: **if** $\alpha' < \alpha_k$ **then**
2: **for** each (x_i, y_i) in Ω_k satisfies $y_i \leq \delta_k$ **do**
3: remove (x_i, y_i) from Ω_k
4: $\alpha' = \min\{\alpha', x_i\}$
5: **end for**
6: add (α', δ_k) to Ω_k
7: **end if**
8: **for** (x_i, y_i) in Ω_k satisfies $y_i \leq \Lambda_k(idx_k + 1 - \alpha_i)$ **do**
9: $\alpha_k = \min\{\alpha_k, x_i\}$
10: **end for**
11: remove any tuple (x_i, y_i) satisfying $x_i \geq \alpha_k$ from Ω_k

图 2.2 算法 *AdjustPrt* 中调用的 *Modify* 函数伪代码描述

Fig. 2.2 The Modify function invoked in AdjustPrt.

2.2.3 LPA 算法实例

本小节将通过实例来具体说明 LPA 算法的运行时优先级调整算方法 $AdjustPrt(J_i^c)$ 在忙碌周期 *BI* 中的工作原理。实例采用如表2.1 所示的混合关键性任务系统，该系统在忙碌周期 *BI* 内的释放时序如 图2.3所示。本实例通过例2.1计算得到忙碌周期 *BI* 中的优先级分配计划表，并基于该表介绍算法 $AdjustPrt(J_i^c)$ 如何在新的运行时作业释放时刻，来进行作业优先级调整并得到保障系统运行时可调度的优先级。

本小节不是一般性地假设新的忙碌周期在时间 0 时刻开始，并且实时任务 τ_3 和 τ_4 在 0 时刻同时释放该忙碌周期内的第一个运行时作业，并且所有的运行时调度参数全部被重置为初始值（$\forall \tau_i \in \pi : idx_i = 1$，$\alpha' = \alpha_i = 1$，$\delta_i = 0$，$\Omega_i = \emptyset$）。遵从算法 **AdjustPrt**在 图2.1 中的描述，实时作业 J_3^1 将获得运行时优先级 $\Lambda_3(0) = 5$，而该优先级高于实时作业 J_4^1 的优先级 $prt(J_4^1) = \Lambda_4(0) = 7$。因此作业 J_3^1 将首先占用处理器执行。而后续释放的作业 J_1^1 和 J_2^1 将分别被分配运行时优先级 $\Lambda_1(0) = 1$ 和 $\Lambda_2(0) = 3$。

直至此刻，由于没有发生运行时作业的抢占执行，因此算法 **AdjustPrt** 计算得到

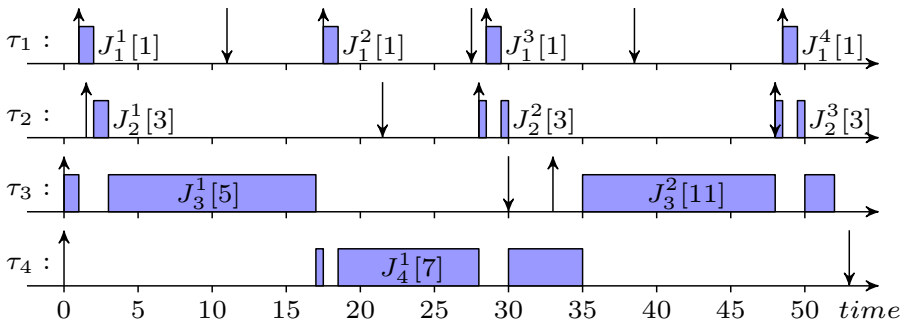


图 2.3 一个忙碌周期中的运行时调度次序实例

Fig. 2.3 An example of scheduling sequence during a busy period

的 $\alpha' = \alpha_i$，故不会对辅助集合 Ω_i 和偏移变量 α_i 进行任何调整。

当实时任务 τ_1 在时刻 17.5 释放其第 2 个运行时作业 J_1^2 时，当前的偏移变量 α' 等于 $\alpha_1 = 1$。此时函数 $AdjustPrt(J_1^2)$ 尝试设置作业 $J_1^2\Lambda_1(idx_1 - \alpha') = 2$。由于该优先级高于正在执行的作业 J_4^1 的优先级（7），因此该函数调整当前的偏移变量 α_1 为 $idx_1 = 2$，并重新分配该作业 J_1^2 的优先级 P_1 为 $\Lambda_1(0) : 1 < prt(J_4^1) : 7$，然后为每个任务 τ_i 分别更新其抢占记录参数 δ_i 的取值为 $\max\{\delta_i, 7\}$。因为 $\alpha_1 = 2 > 1 = \alpha'$，所以函数 **Modify**将新元组 $(\alpha' : 1, \delta_1 : 7)$ 添加到辅助集合 Ω_1 中。

当运行时作业 J_2^2 在时刻 28 被释放时，函数 $AdjustPrt(J_2^2)$ 将执行类似的操作。在该函数被调用时，$\alpha' = \alpha_2 : 1$, $prt(J_4^1) = 7$, $\Omega_2 = \{\}$, $\Lambda_2(idx_2 - \alpha_2) = 6 < \delta_2 = 7$，因此函数的第 10 行设置 $\alpha_2 = idx_2 : 2$ and $P_2 = \Lambda_2(0) : 3$，然后为每个任务更新器抢占记录参数。因为 $\alpha_2 > \alpha'$，所以函数 **Modify** 执行如图2.2所示伪代码的第 2 行至第 6 行，将新元组 $(\alpha' : 1, \delta_2 : 7)$ 添加到辅助集合 Ω_2。但是由于 $\Lambda_2(idx_2 : 2 + 1 - x_1 : 1) = 10 > y_1 : 7$，因此该函数执行第 8 行至第 11 行，重置变量 α_2 的取值为 $x_1 = 1$，并从辅助集合 Ω_2 中删除元组 $(1, 7)$（即 $\Omega_2 \leftarrow \emptyset$）。

2.2.4 运行时时间复杂度

本小节将 LPA 算法的运行时时间复杂度进行分析。由于 PLRS 调度算法在每次抢占点发生的时刻，都要遍历所有任务的所有作业进行优先级调节，导致其运行时时间复杂度为 $O(N^2)$（其中 N 为任务数量）。而由图2.1中 LPA 算法的运行时优先级调整算法 **AdjustPrt** 的描述可知，该算法仅包含两个非嵌套循环（第 2 行至第 4 行，和第 13 行）。并且每个循环体的迭代次数均为 N （任务数量）。接下来，本小节将分析函数 **Modify** 的复杂度亦为 $O(N)$，从而得到全算的时间复杂度为 $O(N)$ 的结论。

引理 2.2：*在运行时任意时刻，任意辅助集合 $\Omega_i : \tau_i \in \pi$ 中包含元组的数量不会大于 N。*

证明：根据 LPA 算法的运行时调度过程可知，在运行时作业 J_i^c 的释放时刻 r_i^c，每当一个新元组 (x, y) 被插入到辅助集合 Ω_i 中时，必然存在一个作业 J_k^p 在时间区间 $(r_i^{c-1}, r_i^c]$ 内被抢占，并且没有优先级低于 $prt(J_k^p)$ 的其他作业在该时间区间内被抢占。

接下来通过反证法来证明该引理。假设在当前忙碌周期中的某一时刻 t^1，辅助集合 Ω_i 被插入了第 $(N+1)$ 个元组 (x_{N+1}, y_{N+1})，而导致该操作发生的对应作业为 J^1。因为系统中所有任务的数量为 N，所以必然存在一个之前被插入辅助集合的元组 (x_m, y_m)，满足导致该元组插入的相关作业 J^2 与作业 J^1 由相同任务在当前忙碌周期

内释放，且释放时间为 t^2。因为元组 (x_m, y_m) 在 t^1 时刻仍然包含在辅助集合 Ω_i 之中，根据图2.2中函数 **Modify**的定义可知，没有比优先级 y_m 在区间 $[t^2, t^1]$ 内被抢占。因此对于每个在时间区间 $[t^2, t^1]$ 内执行的作业 J，均满足 $prt(J) < prt(J^2)$。从而可推出 $prt(J) < prt(J^2)$。根据引理2.5可推出作业 J^2 在作业 J^1 的释放时刻 r^1 不可能为活跃作业，也即是说 J^2 必然在时间区间 (t^2, t^1) 内完成执行。但是当 J^2 执行完毕时，如果当前忙碌周期没有结束则必须有更低优先级的活跃任务接替 J^2 占用处理器执行。这与 $prt(J) < prt(J^2)$ 的假设是相矛盾的。由此，引理得证。 □

定理 2.3：LPA 算法的运行时优先级分配过程的时间复杂度为 $O(N)$。

证明：根据引理2.2可知，在任意时刻任意辅助集合 Ω_i 包含元组的最大值为 N（系统中的任务数量）。因此图2.2中所示的三个 Ω_i 次数循环（第 2 至 5 行，第 8 至 10 行，第 11 行）的运行时复杂度为 $O(N)$。结合对算法 **AdjustPrt**其它部分的运行时复杂度分析结果可推出，LPA 全算法的运行时优先级分配过程的时间复杂度为 $O(N)$。由此定理得证。 □

2.3 忙碌周期上界计算

在文献[39]中，Li 和 Baruah 介绍了一种通过任务集合负载来计算双关键性系统最大忙碌周期上界的方法。该方法的计算结果是安全的，但是存在严重地过度近似。由于基于 OCBP 的方法（包括 LB，PLRS 和 LPA）都需要存储离线计算得到的优先级分配表，且该表的大小与忙碌周期的大小线性相关，因此通过使用更精确的忙碌周期上界算法，能够降低运行时的空闲开销。事实上，更精确的忙碌周期上界算法不仅能提升系统的运行时空间效率，还能够显著地提升系统的可调度性。本小节的后面部分将对此进行分析。

本小节提出一个高效的算法来为任意关键性界别数量的混合关键性系统计算更为精确的最大忙碌周期长度上界。

定义 2.3：给定一个混合关键性偶发任务系统 τ，定义其在系统关键性级别不超过 ℓ 时的工作量上界（记作 Γ_ℓ）为关键性级别不大于 ℓ 的所有任务在任意忙碌周期内能够执行的最大累积工作量。需要注意的是 Γ_L 等于系统的最大忙碌周期长度上界。特别的，补充定义 $\Gamma_0 = 0$。

定理 2.4：给定一个混合关键性偶发任务系统 τ，和其在 $(\ell-1)$ 关键性级别下的工作量上界 $\Gamma_{\ell-1}$，则满足

$$令\phi_\ell = \frac{\Gamma_{\ell-1} + \sum_{\zeta_i \geq \ell} C_i(\ell)}{1 - \sum_{\zeta_i \geq \ell} \frac{C_i(\ell)}{T_i}} \tag{2.3}$$

$$\Gamma_\ell = \Gamma_{\ell-1} + \sum_{\zeta_i = \ell} C_i(\ell) \cdot \left(1 + \left\lfloor \frac{\phi_\ell}{T_i} \right\rfloor\right) \tag{2.4}$$

证明：考虑任意一个在时刻 t_s 开始的忙碌周期 BI。系统关键性级别在该忙碌周期内的某一时刻 t^* 升级到 ℓ，并一直保持的该忙碌周期的结束时刻 t_e。令 W_i 表示 BI 内任务 τ_i 执行的工作量。现在将系统在忙碌周期 BI 内累积的所有执行工作量划分为三部分：$W^{l-} = \sum_{\forall \zeta_i < \ell} W_i$，$W^{\ell} = \sum_{\forall \zeta_i = \ell} W_i$ 和 $W^{\ell+} = \sum_{\forall \zeta_i > \ell} W_i$。因为 $W^{\ell-}$ 只能在时间区间 $[t_s, t^*)$ 内执行，而该时间区间必然包含在某个系统运行时关键性级别不超过 $\ell - 1$ 的忙碌周期内，所以可推出如下不等式

$$W^{\ell-} \leq \Gamma_{\ell-1}. \tag{2.5}$$

接下来通过反证法来继续证明该定理。令 bl 表示忙碌周期 BI 的长度，易知 $bl = W_m^{\ell-} + W_m^{\ell} + W_m^{\ell+}$。根据式 (2.5)可推出

$$\begin{aligned}
\sum_{\forall \zeta_i = \ell} C_i(\ell) \cdot (1 + \left\lfloor \frac{bl}{T_i} \right\rfloor) \quad &\geq \quad W_m^{\ell} \\
&> \quad \sum_{\forall \zeta_i = \ell} C_i(\ell) \cdot \left(1 + \left\lfloor \frac{\phi}{T_i} \right\rfloor\right) \\
\Rightarrow \quad bl \quad &> \quad \phi \\
\Leftrightarrow \quad bl \quad &> \quad \frac{\Gamma_{\ell-1} + \sum_{\forall \zeta_i \geq \ell} C_i(\ell)}{1 - \sum_{\forall \zeta_i \geq \ell} \frac{C_i(\ell)}{T_i}} \\
\Leftrightarrow \quad bl \quad &> \quad \Gamma_{\ell-1} + \sum_{\forall \zeta_{i \geq \ell}} C_i(\ell) \cdot \left(1 + \frac{bl}{T_i}\right) \\
&\geq \quad \Gamma_{\ell-1} + \sum_{\forall \zeta_{i \geq \ell}} C_i(\ell) \cdot \left(1 + \left\lfloor \frac{bl}{T_i} \right\rfloor\right) \\
&\geq \quad W^{\ell-} + W^{\ell} + W^{\ell+} \\
\Leftrightarrow \quad bl > bl &
\end{aligned}$$

根据该矛盾，定理得证。 □

根据 定理2.4，本小节提出了一个递归算法来计算最大忙碌周期长度上界 Γ_L。算法描述如 图2.4所示。

利用式 (2.3)中的参数 ϕ_ℓ，可以得到如下计算忙碌周期内每个任务 τ_i 释放作业数量的等式

$$N_i = \left\lceil \frac{\phi_{\zeta_i}}{T_i} \right\rceil. \tag{2.6}$$

ComputeGamma(ℓ)

1: **if** $\ell = 0$ **then**
2: **return** 0
3: **end if**
4: $\Gamma \leftarrow ComputeGamma(\ell - 1)$
5: $\gamma \leftarrow \dfrac{\Gamma + \sum_{\zeta_i \geq \ell} C_i(\ell)}{1 - \sum_{\zeta_i \geq \ell} \dfrac{C_i(\ell)}{T_i}}$
6: **return** $\Gamma_{\ell-1} + \sum_{\zeta_i = \ell} C_i(\ell) \cdot \left(1 + \left\lfloor \dfrac{\gamma_\ell}{T_i} \right\rfloor\right)$

图 2.4 计算关键性级别 ℓ 下的工作量

Fig. 2.4 Compute *Criticality*-ℓwork load.

与文献 [39] 中提出的方法相比，通过更精确的忙碌周期上界，式 (2.6) 能够得到更小的作业数量。因此本小节的方法能够提升系统的运行时空间效率。除此之外，通过更精确的忙碌周期上界还可以提升系统的可调度性。接下来，本小节将通过实例分析来展示其中的原因。首先考虑如下表所示的混合关键性系统。

$Task$	T_i	D_i	ζ_i	$C_i(1)$	$C_i(2)$
τ_1	15	15	2	8	14
τ_2	80	80	1	9	9

低关键性 [39] 中介绍的方法，可以计算得到该系统的忙碌周期上界为 $bl = 3309$。而在该长度的时间区间内能够包含 221 个高关键性任务 τ_1 释放的作业，和 42 个由低关键性任务 τ_2 释放的作业。因为该系统中高关键性任务 τ_1 的利用率非常高（93.33%），该任务释放的作业不能够承受太多高优先级的低关键性作业。因此，在离线优先级分配算法的开始阶段将会把当前的最低优先级分配给低关键性任务的作业。但是，大量高优先级高关键性任务作业同样会导致低关键性任务的作业错失截止期。在本小节实例的实验中，但系统剩余 215 个高关键性任务 τ_1 释放作业和 24 个低关键性任务 τ_2 释放作业未分配优先级时，存在如下关系

$$C_1(2) \cdot 215 + C_2(2) \cdot 24 = 3226 > T_1 \cdot 214 + D_1 = 3225 \tag{2.7}$$

$$C_1(1) \cdot 215 + C_2(1) \cdot 24 = 1936 > T_2 \cdot 23 + D_2 = 1920. \tag{2.8}$$

因此，无论是高关键性作业 J_1^{215} 或是低关键性作业 J_2^{24}，均不能被赋予当前的最低优先级。也即是说，离线优先级分配释放，系统不可调度。当通过对该实例的进一步观察可以发现 $d_2^{24} = 1920$，而高关键性任务 τ_1 在低关键性作业 J_2^{24} 的截止期之前最多释放 $\lceil \frac{1920}{15} \rceil = 128$ 个作业。然而根据式 (2.7) 的测试会计算 215 个 τ_1 释放作业的累积工作量，从而导致了过度的工作量估计。

但是通过本节提出改进算法，能够计算得到更为精确的上界 $bl_{HI}=345$。而在该长度的时间区间内仅包含 23 个高关键性任务 τ_1 释放的作业，和 5 个低关键性任务 τ_2 释放的作业。这样便显著减小了对于累积工作量的过度估计，从而能够使得离线优先级分配算法成功返回结果。因此该实例的任务集合采用 LPA 算法是可调度的。

2.4 LPA 算法可调度性的证明

在本小节将形式化地证明当 LPA 的离线优先级算法能够成功返回优先级分配计划表示，LPA 算法的在线优先级调整算法能能够保证系统的混合关键性可调度性。

定义 2.4 (忙碌集合)：在某个特定忙碌周期内，假设存在一个运行时作业 $J_i^{\alpha}:\alpha\in N$ 获得 $\Lambda_i(0)$ 作为其运行时优先级。定义任务 τ_i 的忙碌集合为 $S_{\alpha}=\{J_i^c \mid c\geq\alpha\}$。

定义 2.5 (问题窗口)：给定一个运行时作业 J_k^c，其被赋予运行时优先级 $\Lambda_k(c-\alpha)$。定义作业 J_k^c 的问题窗口为时间区间 $(r_k^{\alpha},f_k^c]$，其中 r_k^{α} 表示被分配 $\Lambda_k(0)$ 为运行时优先级的作业 J_k^{α} 的释放时间。

需要注意的是，在当前忙碌周期中每个曾经使用过的 α_i 取值 θ，其只有在某个特定作业 J_i^{θ} 被释放并且被分配了优先级 $\Lambda_i(0)$ 时，才可能被重新使用。

定义 2.6 (干涉数量上界)：给定一个运行时作业 J_k^c 和一个任务 τ_i，定义干涉数量上界 $BIC(J_k^c,\tau_i)$ 为任务 τ_i 在当前忙碌周期释放的所有作业中，优先级大于或等于作业 J_k^c 的优先级且在 J_k^c 的问题窗口中执行的作业的最大数量。

定义 2.7 (活跃作业)：一个在当前忙碌周期中已经被释放的作业，如果尚未执行完毕或因系统关键性级别升高而被终止执行难，则定义这样的作业为活跃作业。

引理 2.5：给定任意一个运行时作业 $J_i^c\in S_{\alpha}$，其在时刻 r_i^c 释放并在 f_i^c 时刻执行完毕，即 J_i^c 在时间区间 (r_i^c,f_i^c) 内为活跃作业。对于每个在作业 J_i^c 活跃期间释放的作业 $J_i^h:h>c$，均满足 $J_i^h\in S_{\alpha}$。

证明：该引理使用反证法来证明。假设作业 J_i^h 为时间区间 (r_i^c,f_i^c) 内释放的第一个满足如下条件的作业：

$$prt(J_i^h)<\Lambda_i(h-\alpha).$$

假设作业 J_i^h 的释放时刻为 r_i^h。根据算法 **AdjustPrt**的定义可推出，在时刻 r_i^h，满足下面等式：

$$\delta_i>P_k\geq\Lambda_i(h-\alpha)>\Lambda_i(c-\alpha)=prt(J_i^c).$$

因此，在时间区间 $(r_i^c,r_i^h]\subset(r_i^c,f_i^c)$ 内，必定存在一个优先级为 δ_i 的被抢占作业

J_l。也即是说，作业 J_l 的优先级为 $\delta_i > prt(J_i^c)$，故其在 $(r_i^c, r_i^h]$ 必然会执行一段时间。J_i^c 在该时间区间内还是活跃的。

但是任何低优先级的作业都不可能在运行时抢占更高优先级作业的执行，因此作业 J_i 不能够在作业 J_i^c 的活跃时间区间 (r_i^c, f_i^c) 内执行。

上述矛盾证明了该引理的正确性。 □

引理 2.6：对于任意一个运行时作业 J_k^c，如果其由算法 **AdjustPrt** 赋予了优先级 $\Lambda_k(c-\alpha)$，那么必然满足所有优先级低于 $prt(J_k^c)$ 的运行时作业都不能够在 J_k^c 的问题窗口内执行。

证明：当满足 $c=\alpha$ 时，作业 J_k^c 的问题窗口区间为 $(r_k^c, f_k^c]$，并且易知没有优先级低于 $prt(J_k^c)$ 的运行时作业能够在该问题窗口内执行。

接下来使用反证法来证明当 $c>\alpha$ 时的情况。假设作业 J^* 的优先级低于 $prt(J_k^c)$（$prt(J^*) > prt(J_k^c)$），并为在问题窗口 $(r_k^\alpha, f_k^c]$ 能够执行一段时间的最低优先级作业。因为 $prt(J^*) > prt(J_k^c)$，J^* 不能够在时间区域 $(r_k^c, f_k^c]$ 内执行，所以只需要分析作业 J^* 在区间 $(r_k^\alpha, r_k^c]$ 内执行的情况。一旦作业 J^* 开始执行，则表明系统中已经不存在优先级高于 $prt(J^*)$ 的活跃作业。而如果在其执行阶段没有更高优先级的作业释放并抢占其执行，那么 J^* 一定会在某一时刻 $t^* \in (r_k^\alpha, r_k^c]$ 结束执行。此时要么有更低优先级的活跃作业继续执行，要么当前忙碌周期结束。而上述两种情况都与本证明中的假设相矛盾。因此作业 J^* 必定要在某一时刻 $t^p \in (r_k^\alpha, r_k^c]$ 被新释放的高优先级作业抢占，并在时刻 f_k^c 之前一直保持活跃。令 f_k^c 表示任务 τ_i 在时间区间 $[t^p, r_k^c]$ 内释放的第一个作业。根据上面的讨论可推出 $P_k < prt(J^*) = \delta'$ at r_k^n。从而根据函数 **Modify** 的定义可知，变量 α_k 被设置为 $n>\alpha$，一个新元组 (α', δ') 将被添加到辅助集合 Ω_k 中。另外，由于 J^* 是在区间 $(r_k^\alpha, r_k^c]$ 内执行的最低优先级作业，因此可知在区间 $(r_k^\alpha, r_k^c]$ 内满足 $\delta_k < \delta'$。

当满足 $\alpha' \neq \alpha$ 时，因为所有满足 $y \le \delta'$ 的元组 (x,y) 都将在作业 J_k^n 的释放时刻被函数 **Modify**从辅助集合 Ω_k 中删除，并且 Ω_k 中将不再包含存储偏移变量记录 α 的元组。因此在当前忙碌周期内，变量 α_k 不可能在时刻 r_k^n 之后再被重新设置为 α。这与某个将来释放作业 J_k^c 被赋予优先级 $\Lambda_k(c-\alpha')$ 的假设相矛盾。

当满足 $\alpha' = \alpha$ 时，因为 $\forall idx < c : \Lambda_k(idx+1-\alpha) \le \Lambda_k(c-\alpha) < \delta'$，偏移变量的取值 α 不能够在作业 J_k^c 释放之前被重新使用。而这同样与假设相矛盾。

综上，引理得证。 □

引理 2.7：给定任意一个运行时作业 $J_i^c \in S_\alpha$，对于属于与 J_i^c 相同忙碌周期的由任务 τ_i 释放的每个作业 $J_i^h : h > c$，如果在时间区域 $(r_i^\alpha, r_i^h]$ 内没有优先级低于 $\Lambda_i(h-\alpha)$

的其它作业占用处理器执行，那么一定满足 $J_i^h \in S_\alpha$，即 $prt(J_i^h) \geq \Lambda_i(h-\alpha)$。

证明：该引理通过反证法来证明。假设作业 $J_i^h : h > c$ 是由任务 τ_i 释放的第一个满足约束 $prt(J_i^h) < \Lambda_i(h-\alpha)$（即 $\alpha_i^h > \alpha$）的运行时作业。根据 图2.2 中函数 **Modify**的定义可知，如果没有之后释放的作业被分配优先级 $\Lambda_i(0)$，那么变量 α_i 的取值将会单调递减，因此 $prt(J_i^c) = \Lambda_i(c-\alpha_i) \geq \Lambda_i(c-\alpha)$。因为 $prt(J_i^h) < \Lambda_i(h-\alpha)$，故可推出 $prt(J_i^h) = \Lambda_i(0)$，进而推出上式的必要条件 $P_k < \max\{\delta_i, P_{cur}\}$ 必然成立。

如果 $\delta_i > P_{cur}$，那么在时间区间 (r_i^c, r_i^h) 内必然存在一次作业抢占，并且被抢占作业的优先级为 $\delta_i > P_k$。否则，当前执行作业的优先级低于 P_k。综上两种情况，都可推出在时间区间 (r_i^c, r_i^h) 内存在某个优先级低于 P_k 的作业执行了一段时间。

该结论与时间区间 $(r_i^\alpha, r_i^h]$ 内不存在优先级低于 $\Lambda_i(h-\alpha)$ 的作业能够占用处理器执行的假设是相矛盾的。由此，引理得证。 □

引理 2.8：假设一个运行时作业 J_k^c 在时刻 r_k^c 释放，并且按照 LPA 算法被分配了运行时优先级 $\Lambda_k(c-\alpha_k) : 1 \leq \alpha_k \leq c$。那么对于 $\forall \tau_i \in \pi$，均满足

$$BIC(J_k^c, \tau_i) \leq \| \{x | \Lambda_i(x) \leq prt(J_k^c)\} \| . \tag{2.9}$$

其中 $\|s\|$ 表示集合 s 中包含元素的数量。

证明：根据算法 **AdjustPrt**的优先级分配方法可知，必然存在一个更早释放的作业 $J_k^{\alpha_k}$ 被分配了优先级 $\Lambda_k(0)$。由引理2.7可知，每个在时间区间 $(r_k^{\alpha_k}, d_k^c]$ 内执行的作业 J_i^j 均满足如下不等式：

$$prt(J_i^j) \leq prt(J_k^c)。 \tag{2.10}$$

为了计算作业 J_k^c 的时间窗口内高优先级作业的数量，令变量 x_i 存储 Λ_i 中第一条记录的索引，并且满足条件 $\Lambda_i(x_i) > prt(J_k^c) = \Lambda_k(c-\alpha_k)$，即 $x_i = \min\{x | \Lambda_i(x) > prt(J_k^c)\}$。特别的，对于 $x \geq N_i$，有 $\Lambda(x) = +\infty$。根据引理2.1，可推出

$$\| \{x | \Lambda_i(x) \leq prt(J_k^c)\} \| = x_i。 \tag{2.11}$$

根据任务 τ_i 在时刻 $r_k^{\alpha_k}$ 是否存在已释放的活跃任务，可以将混合关键性任务集合 τ 分为两个划分 π_{active} 和 π_{silent}：对于任意任务 $\tau_i \in \tau$，如果在时刻 $r_k^{\alpha_k}$ 存在其释放的活跃任务 J_i，则 $\tau_i \in \pi_{active}$；否则 $\tau_i \in \pi_{silent}$。下面将对这两种情况分别进行讨论。

1) 考虑 π_{active}：

对于任意任务 $\tau_i \in \pi_{active}$，不失一般性地令 J_i^a 表示由该任务释放并在时刻 r_k^o 依然活跃的最小序号作业（最早释放者）。假设作业 J_i^a 在时刻 r_i^a 释放，并且满足 $J_i^a \in S_{ai}$。显然 $a \geq ai \geq 1$，因此 $\Lambda_i(a + x_i - ai) \geq \Lambda_i(x_i)$。

当 $prt(J_i^a) > prt(J_k^c)$ 时，因为作业 J_k^c 比作业 J_i^a 的优先级更高，所以根据引理2.6可知 J_i^a 不会在作业 J_k^c 的问题窗口 $(r_k^{\alpha_k}, f_k^c]$ 内执行。因此作业 J_i^a 必然在时间区间 $(r_k^{\alpha_k}, f_k^c]$ 内保持活越性，并且任务 τ_i 在 $(r_k^{\alpha_k}, f_k^c]$ 内释放的所有作业均属于集合 S_{ai}。所以对于每个作业 J_i^j，如果其满足 $j \geq a$ 和 $r_i^j \in (r_k^{\alpha_k}, f_k^c]$，那么可推出 $prt(J_i^j) \geq \Lambda_i(j-ai) \geq \Lambda_i(a-ai) = prt(J_i^a) > prt(J_k^c)$，即任务 τ_i 释放的所有作业都不会在作业 J_k^c 的问题窗口中执行。从而可推出

$$BIC(J_k^c, \tau_i) = 0 \leq x_i. \tag{2.12}$$

当 $prt(J_i^a) < prt(J_k^c)$ 时，根据 引理2.5 可推出在时刻 $r_k^{\alpha_k}$ 由任务 τ_i 释放的所有作业都属于集合 S_{ai}，并且满足 $\delta_i \leq prt(J_i^a)$。根据 引理2.6可知，所有优先级低于 $prt(J_k^c)$ 的运行时作业都不能够在时间区间 $(r_k^o, f_k^c]$ 内执行，因此也不会被高优先级作业抢占。[1] 因此在时间区间 $(r_k^{\alpha_k}, f_k^c]$ 满足条件 $\delta_i \leq \max\{prt(J_i^a), prt(J_k^c)\} = prt(J_k^c) < \Lambda_i(x_i)$。再根据引理2.7可推出，任意由任务 • τ_i 释放的作业 $J_i^h : h \geq x_i + ai > c$ 均属于忙碌集合 S_{ai}，即满足 $prt(J_i^h) \geq \Lambda_i(x_i) > prt(J_i^c)$。综合上面结论有如下不等式成立：

$$BIC(J_k^c, \tau_i) \leq (x_i + ai) - a \leq x_i \tag{2.13}$$

2) 考虑 π_{silent}：

对于任意任务 $\tau_i \in \pi_{silent}$，不失一般性地令 $J_i^s \in S_{si}$ 表示任务 τ_i 在时间区间 $(r_k^{\alpha_k}, f_k^c]$ 内释放的第一个运行时作业。与之前分析类似，同样引入变量 x_i。容易验证条件 $s \geq si \geq 1$ 和 $\Lambda_i(s + x_i - si) \geq \Lambda_i(x_i)$ 是成立的。根据引理2.6可推出，在时间区间 $(r_i^s, f_k^c] \subset (r_k^{\alpha_k}, f_k^c]$ 内满足 $\delta_i \leq prt(J_k^c) < \Lambda_i(x_i)$。再根据引理2.7可知，任意由任务 τ_i 释放的作业 $J_i^h : h \geq x_i + si > c$，均属于忙碌集合 S_{si}。又因为 $J_i^s \in S_{si}$，所以对于任意任务 τ_i 在时间区间 $J_i^h : h \geq x_i + si > c$ 内释放的作业均满足：如果 $h \geq x_i + si$，则 $prt(J_i^h) \geq \Lambda_i(x_i) > prt(J_i^c)$。

综上可推出如下不等式成立：

$$BIC(J_k^c, \tau_i) \leq x_i + si - s \leq x_i - 1. \tag{2.14}$$

根据 式 (2.12)，式 (2.13)，式 (2.14) 和式 (2.11)可推出，对于 $\forall \tau_i \in \pi$ 均满足

$$BIC(J_k^c, \tau_i) \leq x_i - 1 = \| \{x | \Lambda_i(x) \leq prt(J_k^c)\} \|.$$

由此引理得证。 □

定理 2.9：对于任意混合关键性系统 τ，如果 LPA 的离线算法能成功对其进行离

[1] Note that there may be more than one jobs from different tasks release at same time instant t, and the scheduler will handle these events respectively with arbitrary order. To simplify the analysis, we treat the earlier handled job as earlier released one and vice versa.

线优先级分配，则 LPA 的运行优先级分配算法能够保证 τ 是混合关键性可调度的。

证明：该定理使用反证法来证明。假设作业 $J_k^{m+\alpha-1}$ 被赋予运行时优先级 $\Lambda_k(m-1)$，并是第一个错失截止期其 $d_k^{m+\alpha-1}$ 的作业。根据 引理2.6 和 引理2.8可推出，在作业 $J_k^{m+\alpha-1}$ 的问题窗口 $(r_k^{\alpha}, d_k^{m+\alpha-1}]$ 内，满足

$$\begin{aligned}\sum_{\tau_i\in\pi} C_i(\zeta_k)\times BIC(J_k^c,\tau_i) &\le \sum_{\tau_i\in\pi} C_i(\zeta_k)\times(x_i-1)\\ &\le \sum_{\forall i,j:\Lambda_i(j)\le\Lambda_k(m)} C_i(\zeta_k)\\ &\le T_k\times(m-1)+D_k\\ &\le d_k^m - r_k^{\alpha}\end{aligned}$$

这与 式 (2.2) 相矛盾。该定理得证。　□

2.5 实验结果与分析

本小节将对 LPA 算法和另一个基于 OCBP 的算法 PLRS 在可调度性，运行时时间效率和运行时空间效率上的性能进行综合评价分析。本节实验采用了单处理器平台上双关键性的隐式截止期偶发任务模型。前面已经对 LPA 算法流程进行介绍，也分析了其在时间复杂度、空间复杂度、可调度性等方面较现有算法的显著改进，最后也在理论上证明了利用 LPA 算法调度的正确性；在本章将用 LPA 与 PLRS 算法进行对比实验，通过实验来具体分析 LPA 较 PLRS 算法在时间复杂度、空间复杂度、可调度性的改进情况；该实验将在偶发性任务系统的单处理器平台来完成，实验中任务采用双关键性（HI、LO）、隐式截止期。

2.5.1 随机任务集合生成

我们采用与文献 [69] 中类似的生成混合关键性随机任务集的方法。一个随机实时任务集初始时被设置为空集 $\pi\leftarrow\emptyset$，然后逐次添加新的随机混合关键性实时任务。随机任务的生成主要受 5 个参数的控制：随机任务为高关键性任务的最大概率 P_{HI}；高关键性任务的高关键性下的最差执行时间与低关键性下最差执行时间的最大比例 R_{HI} 务在低关键性下最差执行时间的最大值 $C(\mathrm{LO})$；最大的实时任务周期 T^{max}；和单位速率处理器的个数 m。每个新的随机实时任务按照如下步骤生成：

（1）τ_i 服从 P_{HI} 的概率取值 HI，否则取值 LO；

（2）$C_i(\mathrm{LO})$ 的取值在 $\{1,2,\cdots,C_{\mathrm{LO}}^{max}\}$ 范围内服从均匀分布；

（3）如果该随机任务为低关键性任务则 $C_i(\mathrm{HI})=C_i(\mathrm{LO})$；

否则 $C_i(\mathrm{HI})$ 在 $\{C_i(\mathrm{LO}),C_i(\mathrm{LO})+1,\cdots,R_{\mathrm{HI}}\cdot C_i(\mathrm{LO})\}$ 范围内均匀分布；

（4）T_i 的取值在 $\{C_i(\zeta_i), C_i(\zeta_i)+1, \cdots, T^{max}\}$ 范围内服从均匀分布；

（5）由于我们采用隐式截止期的模型, 所以有 $D_i = T_i$。

我们定义一个双关键性实时任务集 π 的平均利用率为:

$$U_{avg} = \frac{U_{\mathrm{LO}}(\pi) + U_{\mathrm{HI}}(\pi)}{2} \tag{2.15}$$

每个任务集在生成时都有一个平均资源利用率基准 $U^{\star}$。由于产生拥有准确资源利用率的随机任务集比较困难，所以我们允许任务集的平均利用率有一个可接受的误差范围：$U^{\star}_{min} = U^{\star} - 0.005, U^{\star}_{max} = U^{\star} + 0.005$。若满足 $U_{avg}(\pi) < U^{\star}_{min}$, 就继续产生更多的任务并将它们添加到任务集 π。如果将一个任务加入 π 后导致 $U_{avg}(\pi) > U^{\star}_{max}$ 则随机任务集生成算法将整个任务集抛弃，并从一个空的任务集重新开始生成。如果将一个随机任务加入 π 后，满足 $U^{\star}_{min} \le U_{avg}(\pi) \le U^{\star}_{max}$，则一个随机任务集生成完毕，除非任务集中的所有任务都有相同的关键性或者 $U_{\mathrm{LO}}(\pi) > 0.99$ 或 $U_{\mathrm{HI}}(\pi) > 0.99$。在这种情况下，此任务集也将被抛弃。

在每次实验中生成随机任务集的各个参数设置分别为 $P_{\mathrm{HI}} = 0.5$，$R_{\mathrm{HI}} = 2$，$C^{max}_{\mathrm{LO}} = 10$，$T^{max} = 100$。

2.5.2 时间开销

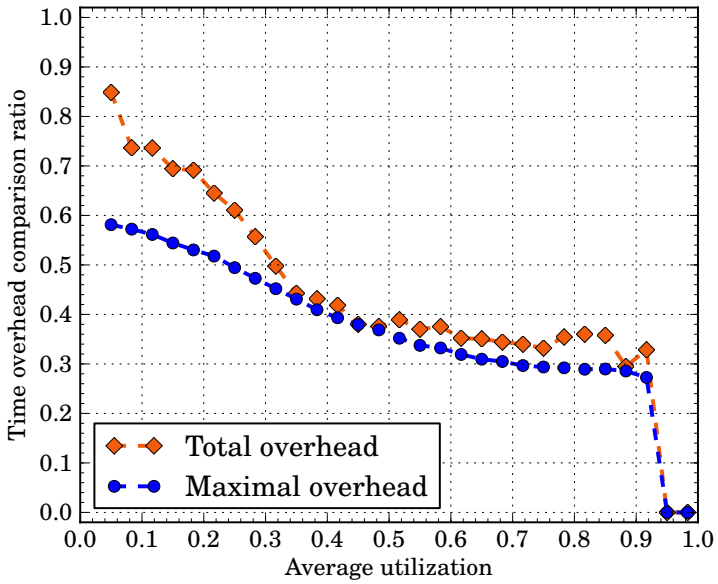


图 2.5 $P_{HI} = 0.5$，$R_{HI} = 2$，$C^{max}_{LO} = 10$，$T^{max} = 100$

Fig. 2.5 $P_{HI} = 0.5$, $R_{HI} = 2$, $C^{max}_{LO} = 10$ and $T^{max} = 100$

本节首先将通过模拟调度过程来对运行时调度的时间开销予进行评价。对于每个随机任务集，随机生成 5000 个运行时释放的作业，并统计模拟调度器调度这些作业的时间开销。

本节针对如下两个方面比较 LPA 算法与 PLRS 算法的进行调度时间开销：

- 所有开销：表示模拟器 1000 个作业中总的运行时间开销；
- 最大开销：表示模拟器 1000 个作业中运行时间开销最大的作业。

图2.5 展示了 LPA 算法与 PLRS 算法的运行时调度时间开销对比实验的结果。图中每个点的数据都基于超过 5000 个随机任务集合样本。其中 *x*-轴 表示随机任务集合的平均利用率，*y*-轴 表示 LPA 算法第总体（或最大）时间开销与 PLRS 算法对应开销的比率。例如图中曲线 *Maximal overhead* 的一点 (0.81,0.29)，*x*-轴 坐标值为 0.81 表示生成随机任务集合的目标利用率为 81% *y*-轴 坐标值为 0.29 表示在平均情况下，LPA 算法的最大运行时调度开销为 PLRS 算法开销的 29%。

观察曲线的性质可知，LPA 在不同利用率下的时间开销都比 PLRS 算法的时间开销要小，从而可知 LPA 算法在运行时的时间效率远远高于 PLRS 算法，尤其当系统的利用率高的时候改善效果越显著。

2.5.3 空间开销

本小节继续评价算法在运行时空间开销上的性能。实验通过统计在忙碌周期内可能释放的最大作业数量来计算优先级分配表 Λ 的大小，进而评价运行时的空间开销。图2.6展示了实验结果。图中每个点均包含了至少 5000 个随机任务集合样本的信息。其中 *x*-轴 坐标值表示随机任务集合的平均利用率；*y*-轴 坐标值表示 LPA 算法存储的优先级分配表大小与 PRLS 算法存储大小的比率。图2.6中的实验结果表明，由于使用了更为精确的忙碌周期长度计算方法，LPA 算法相较于 PLRS 算法能够显著降低运行时调度的空间开销。当系统的利用率越高时，空间开销降低的效果越显著。

2.5.4 可调度接受率

最后一个实验将比较 LPA 算法相较于 PLRS 算法在随机任务集合可调度接受比率上的提升效果。实验结果如 图2.7 所示。其中 *x*-轴 坐标值表示随机任务集合的平均利用率；*y*-轴 坐标值表示随机任务集合的可调度接受率，即随机任务集合中可被算法调度的数量占总体样本数量的比率。图2.7中的每个点均包含至少 10000 个随机随机任务集合样本的信息。实验结果表明本章提出的 LPA 算法能够显著提高随机任务集合的可接受率。这是因为通过本章提出的更为精确的忙碌周期计算方法，能够显著减小需要分配优先级的作业数量，从而降低了在进行工作量分析时的悲观程度。对此的详细分析可参考小节2.3 中的说明。

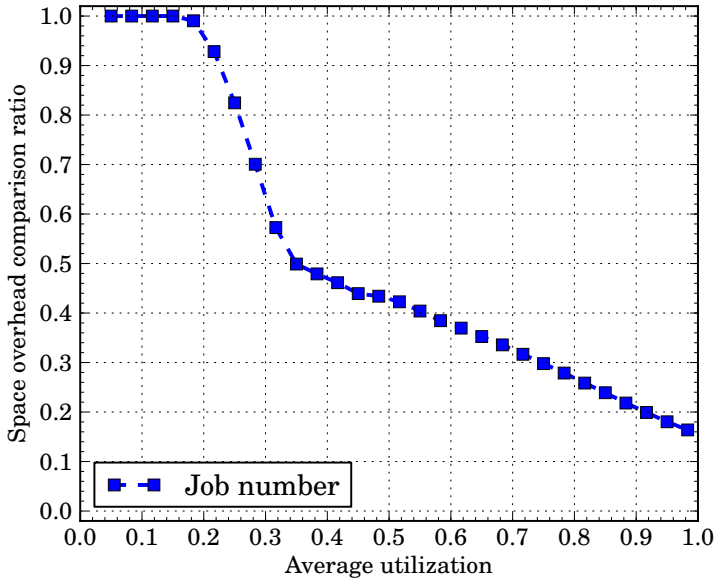


图 2.6 $P_{HI}=0.5$，$R_{HI}=2$，$C_{LO}^{max}=10$，$T^{max}=100$

Fig. 2.6 $P_{HI}=0.5$, $R_{HI}=2$, $C_{LO}^{max}=10$ and $T^{max}=100$

2.6 小结

本章提出了一个基于 OCBP 算法的混合关键性偶发任务系统调度算法 LPA。与之前提出的基于 OCBP 的调度算法相比，LPA 能够显著地提升运行时调度的时间、空间效率，以及系统的可调度性。LPA 算法的主要思想是尽可能晚的对运行时作业优先级计划表进行调整，以避免与调度决策没有直接关联的冗余优先级调整开销。通过随机生成任务集合的实验也验证了本章提出的 LPA 算法在运行时调度时间、空间效率和可调度性方面的优良性能。

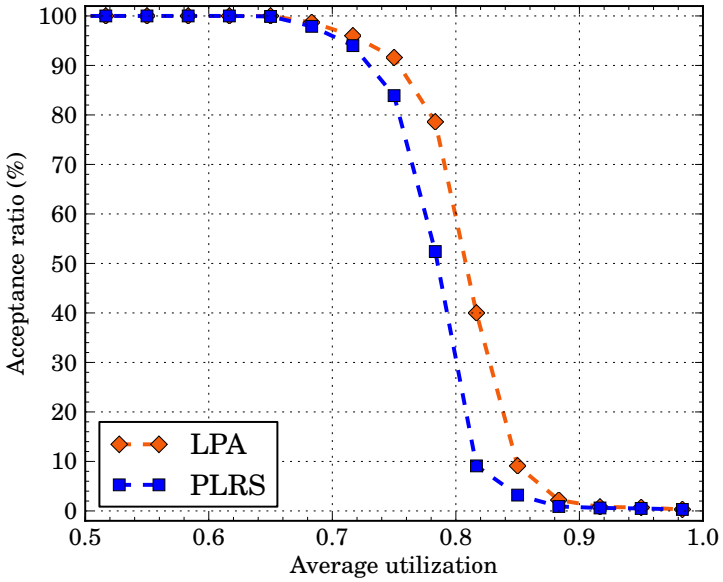


图 2.7 $P_{HI}=0.5$，$R_{HI}=2$，$C_{LO}^{max}=10$，$T^{max}=100$

Fig. 2.7 $P_{HI}=0.5$, $R_{HI}=2$, $C_{LO}^{max}=10$ and $T^{max}=100$

第 3 章 基于虚拟截止期的划分调度算法

随着信息技术日新月异的发展，现代嵌入式实时系统中集成功能的规模和复杂性也呈现爆炸式增长的趋势。例如当今汽车电子系统中已经集成了数十个甚至上百个的微处理器，而航空电子系统中微处理器的数量则更为庞大。为了适应未来日益复杂、庞大的系统功能需求，以及满足嵌入式实时系统硬件本身体积、成本、能耗等诸多方面的约束，将多个在传统设计中部署在独立子系统中的不同关键性功能应用集成到共享处理器资源的单一硬件平台成为当今嵌入式实时系统设计的趋势和发展方向。而多处理器平台的出现极其迅猛发展，极大提升了处理器性能的同时也为嵌入式系统功能集中化的设计方案提供了硬件性能的保障。然而较之于传统的实时调度问题，混合关键性系统中的实时调度和可靠性认证等问题更为复杂和困难。而传统的经典调度算法(如 RM，EDF 等) 在混合关键性系统中的资源利用率十分低下，不能直接应用于系统设计。

2007 年，Vestal 在文献 [20] 中首先形式化定义了单核处理器平台上的混合关键性系统（Mixed Criticality Systems）中的实时调度问题。之后，混合关键性系统的研究迅速成为近年来实时系统领域的热点问题并引起了大量知名学者的关注。然而大量的工作都集中于单处理器平台上的混合关键性系统调度问题。近年来，多处理器平台中的混合关键性系统调度问题受到了广泛地关注。根据是否允许相同任务释放的作业在运行时执行在不同的处理器上，传统的单关键性多处理器调度算法通常被分为全局调度和划分调度。而文献 [134] 证明了在强实时系统中划分调度具有更好的可调度性。Kelly 等在文献 [92] 中首次提出了可抢占多处理器平台中基于固定优先级算法的划分调度问题，并分别对比了采用资源利用率降序排列（Decreasing Utilization，DU）和关键性降序排列（Decreasing Criticality，DC）两种实时任务集排序策略以及 FFD，WSD 等启发式划分策略对划分调度性能的影响。Baruah 等在文献 [104] 中提出了基于非固定优先级算法 EDF-VD[45]，并采用基于 DC 策略的混合关键性划分调度算法 MC-Partition。

Vestal 在文献 [20] 中形式化定义了单核处理器平台上的混合关键性系统。将传统的实时调度方法（比如 EDF）直接应用到混合关键性任务系统中可能会导致非常低下的可调度性能。文献 [36, 39, 47, 54, 60] 分别研究了不同关键性级别下调度问题的差异性，并提出了各自的方法来提升混合关键性系统的可调度性。而 Ekberg 和 Yi 在文献 [69] 中提出了一个可调度性能十分出众的算法（本章称之为 EY-VD）。EY-VD 算法基

于 EDF-VD (EDF with virtual deadlines)[45] 中虚拟截止期的思想，通过更自由地调节高关键性任务的虚拟截止期来平衡不同关键性级别下系统的可调度性。小节3.1.3将对 EY-VD 算法进行简要的介绍。文献 [104] 已经将 EDF － VD 算法扩展到了多处理器系统中。本章将研究把可调度性更优异的 EY-VD 算法扩展到多处理器平台中的方法。

传统的（单关键性）多处理器调度算法通常被分为两类[135]：全局调度算法和划分调度算法。其中全局调度算法允许任意任务在运行时释放的任意作业都能够在任意可用的处理器上执行，并允许作业在不同处理器间的迁移；而划分调度算法会将每个任务分配到各自特定到处理器当中，在运行时每个任务释放的所有作业仅能够在其被分配到处理器中执行，不允许任何的作业迁移。而最近的多处理器调度研究成果表明，在硬实时系统中划分调度算法通常拥有相较于全局调度算法更好的可调度性[134]。因此本章主要研究使用划分调度策略的混合关键性多处理系统调度问题。

任务划分策略的选择对于划分调度算法的性能影响是巨大的。在传统实时系统的划分调度算法中，采用首次适应（First Fit，FF）划分策略的算法通常表现出更好的性能。而文献 [92] 通过对不同划分策略和任务分配次序组合对混合关键性划分调度算法性能影响的评价，得出了 FF 划分策略和关键性降序任务分配次序对组合（FF and criticality-decreasing，FFDC）具有最佳可调度性能的结论。但是本章的研究表明，FF 划分策略不能很好的利用不同关键性模式下系统调度的差异性，而采用该策略扩展的 EY-VD 划分调度算法不能得到满意的可调度性能。

因此本章提出了对于高关键性任务和低关键性任务分别采用不同策略的混合划分调度算法 MPVD（Mixed-criticality Partitioning with Virtual Deadlines）。该算法首先使用最差适应（worst-fit (WF)）划分策略来分配高关键性任务，然后使用首次适应（first-fit）划分策略来分配低关键性任务。通过混合划分策略，能够使高关键性的任务能够被均匀地分配到不同处理器（核心）中，以使得 EY-VD 算法能够有更多的空间来平衡不同关键性级别下的工作量，并提升系统的可调度性。

MPVD 算法的性能会随着处理器（核心）数量的增加而出现明显的下降。为了解决该问题，本章提出了两个优化算法来进一步提升算法的性能。首先考虑到由于高关键性任务在所有处理器中的均匀分配，可能导致无法为利用率较高的低关键性任务分配到拥有足够剩余资源的处理，造成充足的所有处理器（核心）剩余资源总量无法被充分利用。针对该问题，本章提出了为利用率较高的低关键性任务预留资源的策略。另外，本章还提出了优化的虚拟截止期调整算法来进一步提升 MPVD 算法的性能。

随机生成任务集合的模拟实验结果表明，与已有的多处理器混合关键性调度算法相比，本章提出的 MPVD 算法（特别是继承了两个优化策略的版本）能够显著地提升

系统的可调度性。

3.1 基本概念

3.1.1 混合关键性任务和混合关键性作业

本章为了简化系统行为的描述和算法表达与分析，只研究拥有两个关键性级别的双关键性系统模型。并使用符号 LO 表示低关键性级别，使用 HI 表示高关键性级别。而本文的结论均可以扩展到任意关键性个数的情况。

与小节2.1中对混合关键性系统模型的定义类似，本章将每个混合关键性任务定义为四元组：$\tau_i = (T_i, D_i, C_i, \zeta_i)$，该元组中各元素的语义说明如下：

- $T_i \in R^+$，表示实时任务 τ_i 任意连续释放的两个作业间的最小释放时间间隔。
- $D_i \in R^+$，表示实时任务 τ_i 的相对截止期。
- $C_i \in N^+ \rightarrow N^+$，表示实时任务 τ_i 的最差执行时间函数，返回 τ_i 在不同关键性下系统评估的最差执行时间取值。
- $\zeta_i \in \{\text{LO}, \text{HI}\}$，表示任务的关键性界别，其中 LO 表示低关键性级别，HI 表示高关键性级别。

需要注意的是，每个任务的最小释放时间间隔与其相对截止期之间不存在约束关系。

为了成功调度上述的双关键性系统，所有高关键性任务释放的作业无论在低关键性模式还是高关键性模式下都必须满足截止期约束，而低关键性任务释放的作业仅需要在低关键性模式下满足截止期约束。

为了描述系统在不同关键性模式下的工作量差异，使用符号 U_i^{LO} 和 U_i^{HI} 分别表示任务 τ_i 在低关键性模式和高关键性模式下的资源利用率：

$$U_i^{\text{LO}} \triangleq C_i(\text{LO})/T_i \ , \quad U_i^{\text{HI}} \triangleq C_i(\text{HI})/T_i 。$$

使用 $\text{LO}(\pi) \triangleq \{\tau_i \in \pi | \zeta_i = \text{LO}\}$ 表示任务集合 τ 中所有低关键性任务的资源利用率之和，并使用 $\text{HI}(\pi) \triangleq \{\tau_i \in \pi | \zeta_i = \text{HI}\}$ 表示所有高关键性任务的资源利用率之和。并分别定义低关键性模式与高关键性模式下的总体资源利用率 $U_{\text{LO}}(\pi)$ 和 $U_{\text{HI}}(\pi)$ 如下：

$$U_{\text{LO}}(\pi) \triangleq \sum_{\tau_i \in \pi} U_i^{\text{LO}} \ , \ U_{\text{HI}}(\pi) \triangleq \sum_{\tau_i \in \text{HI}(\pi)} U_i^{\text{HI}} 。$$

3.1.2 需求上界函数 DBF

需求上界函数 (Demand-Bound Function, DBF) 描述任意指定长度时间间隔内, 系统产生的最大执行时间需求上界, 即系统所需提供的能够保证实时性约束的最小处理器

资源数量.

定义 3.1 (**需求上界函数** -DBF)：一个需求上界函数 dbf(τ_i , Δ) 表示一个实时任务 τ_i 在给定长度为 Δ 的时间间隔内可能产生的最大执行时间需求的上界. 其中执行时间的需求为所有由实时任务 τ_i 释放且调度窗口完全包含在长度为 Δ 的时间间隔内的作业所需要的最大执行时间 (即 WCET) 之和。

DBF 是传统实时任务模型下分析实时系统工作量可调度性（Schedulability）的有效方法，并且针对诸多主流的实时任务模型，都有精确的计算方法。例如在偶发实时任务模型中，DBF 可以通过文献 [8] 中的算法求得。Ekberg 等在文献 [69] 中将其应用扩展到单处理器混合关键性偶发实时任务系统模型中，并给出了基于 EY-VD 算法的混合关键性实时任务分别在低关键性和高关键性下的 DBF 计算方法。下面我们将对该法进行简要介绍。

当混合关键性系统运行在低关键性级别 ($\ell = \mathrm{LO}$) 时，每个混合关键性任务 τ_i 可视为普通的单关键性实时任务 $(C_i(\mathrm{LO}), D_i(\mathrm{LO}), T)$。其中 $D_i(\mathrm{LO})$ 是实时任务 τ_i 在低关键性级别下的相对截止期，当 τ_i 为低关键性实时任务时 $D_i(\mathrm{LO}) = D_i$；当 τ_i 为高关键性实时任务时，$D_i(\mathrm{LO}) \le D_i(\mathrm{HI}) = D_i$。可以得到：

$$\mathrm{dbf}_{\mathrm{LO}}(\tau_i, t) = \max\left\{0, \left(\left\lfloor \frac{t - D_i(\mathrm{LO})}{T_i} + 1 \right\rfloor\right) \cdot C_i(\mathrm{LO})\right\} \tag{3.1}$$

当系统运行时切换到高关键性级别时，这里面可能包含一个遗留作业。而从高关键性任务 τ_i 释放的遗留作业的调度窗口最小可能为 $D_i(\mathrm{HI}) - dilo$。据此可以得到高关键性任务 DBF 的最大值：

$$\mathrm{full}(\tau_i, t) = \max\left\{0, \left(\left\lfloor \frac{t - D_i(\mathrm{HI}) - D_i(\mathrm{LO})}{T_i} + 1 \right\rfloor\right) \cdot C_i(\mathrm{HI})\right\} \tag{3.2}$$

函数 $\mathrm{full}(\tau_i, t)$ 的计算过程中没有考虑遗留作业在系统的关键性变化前已经执行的时间。式 (4.3)用于计算该时间，其中 $n = t \mod T_i$。

$$\mathrm{done}(\tau_i, t) = \begin{cases} \max\{0, C_i(\mathrm{LO}) - n + D_i(\mathrm{HI}) - D_i(\mathrm{LO})\} & D_i(\mathrm{HI}) > n \ge D_i(\mathrm{HI}) - D_i(\mathrm{LO}) \\ 0 & otherwise \end{cases} \tag{3.3}$$

根据式 (4.2)和式 (4.3)，可以得到高关键性实时任务 τ_i 在高关键性模式下的需求上界函数为：

$$\mathrm{dbf}_{\mathrm{HI}}(\tau_i, t) = \mathrm{full}(\tau_i, t) - \mathrm{done}(\tau_i, t) \tag{3.4}$$

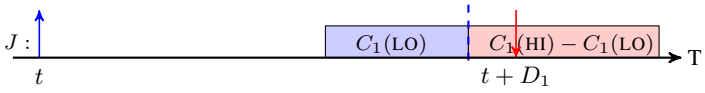
$\mathrm{dbf}_{\mathrm{LO}}$ 和 $\mathrm{dbf}_{\mathrm{HI}}$ 的计算复杂性均是伪多项式的。

3.1.3 EY-VD 方法

本章提出的多核（处理器）划分调度算法在每个处理器上通过采用虚拟截止期的 EDF 算法进行运行时调度，并使用基于 EY-VD[69] 的方法来对高关键性任务进行虚拟截止期调整和可调度性判定。接下来首先总结一下 EY-VD 算法的基本原理。当系统运行于低关键性模式时，对于每个高关键性任务 τ_i，调度算法使用其虚拟截止期 $D_i(\mathrm{LO})$（比原始截止期 D_i 更短）来确定其释放作业的运行时 EDF 优先级。这样能够使得高关键性任务释放的作业更早的执行完其低关键性工作量，从而为系统进入高关键性模式后剩余的高关键性工作量能够在其实际截止期前执行完毕创造了条件。考虑如下所示的任务集合实例：

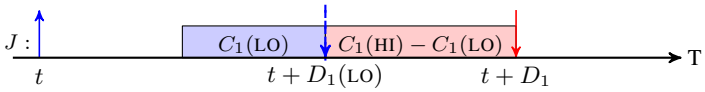
Task	T_i	D_i	ζ_i	$C_i(\mathrm{LO})$	$C_i(\mathrm{HI})$	U_i^{LO}	U_i^{HI}
τ_1	10	10	HI	3	7	0.3	0.7
τ_2	5	5	LO	3	3	0.6	0.6

假设在低关键性模式下，高关键性任务 τ_1 在 t 时刻释放了一个运行时作业 J。如图3.1(a)所示，根据传统的 EDF 调度策略，作业 J_1 在最差情况下会在其绝对截止期 $t+D_1$ 之前 1 个时间单位完成其低关键性下的工作量。如果作业 J 的在第关键性模式下执行过载，则系统切换到高关键性模式执行。如 图3.1(a)所示，在上述情况下作业 J 在其绝对截止期 $t+D_1$ 之前没有足够的时间来执行完其高关键性模式下的剩余工作量 $C_1(\mathrm{HI})-C_1(\mathrm{LO})$。如果设置 τ_1 的虚拟截止期 $D_1(\mathrm{LO})=6$，则可以保证作业 J 完成其低关键性工作量的时间不会晚于 $t+D_1(\mathrm{LO})$。如图3.1(b)所示，在此情况下作业 J 有充足的时间在其绝对截止期 $t+D_1$ 之前执行完其高关键性模式下的剩余工作量。



(a) $D_1(\mathrm{LO})=10$

(a) $D_1(\mathrm{LO})=10$

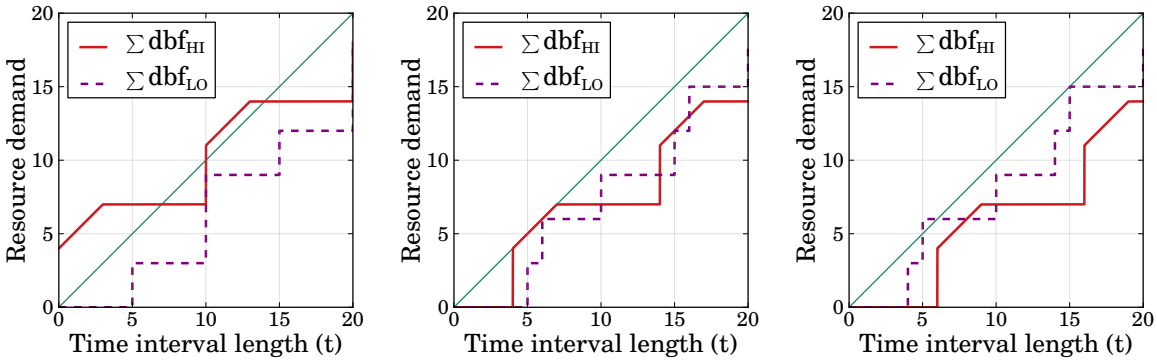


(b) $D_1(\mathrm{LO})=6$

(b) $D_1(\mathrm{LO})=6$

图 3.1　不同虚拟截止期设置对可调度性的影响

Fig. 3.1　Impact of different virtual deadlines



(a) $D_1(\mathrm{LO})$=10　　(b) $D_1(\mathrm{LO})$=6　　(c) $D_1(\mathrm{LO})$=4

(a) $D_1(\mathrm{LO})$=10　　(b) $D_1(\mathrm{LO})$=6　　(c) $D_1(\mathrm{LO})$=4

图 3.2 不同虚拟截止期设置对资源需求的影响

Fig. 3.2 Impact on demand of different virtual deadlines

虽然设置更小的虚拟截止期取值有利于高关键性任务在高关键性模式下满足可调度性，但由于高关键性任务在低关键性模式下必须满足更严格的截止期约束，因此系统在低关键性模式下的可调度性会受到不利的影响。在上面的例子中，如果设置虚拟截止期 $D_1(\mathrm{LO})=4$，那么系统在低关键性模式下就变得不可调度。在 EY-VD 算法中，这种现象被形式化地通过高关键性需求函数 $\mathrm{dbf}_{\mathrm{HI}}(\tau_i,\Delta)$ 和低关键性需求函数 $\mathrm{dbf}_{\mathrm{LO}}(\tau_i,\Delta)$ 来评价。基于上述两个函数，可以给出系统在低关键性模式下可调度的一个充分必要条件如下：

$$\forall t \geq 0: \sum_{\tau_i \in \pi} \mathrm{dbf}_{\mathrm{LO}}(\tau_i,t) \leq t. \tag{3.5}$$

系统在高关键性模式下可调度的一个充分必要条件如下：

$$\forall t \geq 0: \sum_{\tau_i \in \pi \wedge \zeta_i = \mathrm{HI}} \mathrm{dbf}_{\mathrm{HI}}(\tau_i,t) \leq t. \tag{3.6}$$

对于任意高关键性任务 τ_i，如果减小其虚拟截止期 $D_i(\mathrm{LO})$，那么会造成高关键性需求上界函数 $\mathrm{dbf}_{\mathrm{HI}}$ 减小，但同时也会导致低关键性续期上界函数 $\mathrm{dbf}_{\mathrm{LO}}$ 增加。图3.2展示了高关键性任务 τ_1 的虚拟截止期被设置不同取值对需求上界函数的影响。如图3.2(a)所示，当虚拟截止期的取值太小时，系统在低关键性模式下是不可调度的。如图3.2(c)所示，当虚拟截止期的取值太大时，系统在高关键性模式下是不可调度的。如图3.2(b)所示，当虚拟截止期 $D_i(\mathrm{LO})$ 被设置为合适的取值时，系统在高关键性和低关键性模式下都是可调度的。

因此为高关键性任务设置合适的虚拟截止期取值成为调节系统在不同关键性模式下可调度性的关键。但寻找最优化的虚拟截止期设置需要枚举指数级的状态空间，这在时间复杂度上是不可接受的。EY-VD 算法使用了一种有效的启发式算法来调节虚拟

截止期。该算法为每个高关键性任务 τ_i 调节虚拟截止期 $D_i(\mathrm{LO})$ 时，均从 D_i 到 $C_i(\mathrm{LO})$ 单调递减的进行迭代，从而控制总体的时间复杂度。在每次迭代中，EY-VD 算法选都会贪婪地选择如式 (3.7) 所示的启发式条件取值最大的任务 τ_i，并将其虚拟截止期 $D_i(\mathrm{LO})$ 更新为 $D_i(\mathrm{LO})-1$。

$$\mathrm{dbf}_{\mathrm{HI}}(\tau_k, D_k(\mathrm{LO})) - \mathrm{dbf}_{\mathrm{HI}}(\tau_k, D_k(\mathrm{LO})-1). \tag{3.7}$$

3.2 MPVD 划分调度算法

文献 [45] 中研究了基于 EDF-VD 算法的划分调度问题。本节将讨论适用于在单处理器平台上性能更高的 EY 算法的多处理器划分调度策略。

3.2.1 混合划分策略

在介绍本章提出的使用混合划分策略的调度算法之前，本小节首先讨论应用 FF 划分策略来扩展 EY 算法到多处理器平台的一些不足，并由此促使本章提出了基于混合划分策略的调度算法。

在文献 [92] 中介绍了使用 FFDC（First-fit packing and decreasing-criticality task ordering）的划分调度策略。FFDC 划分策略会尽可能多的将高关键性任务分配到同一处理中，直至该处理没有足够的剩余资源容纳新的任务。然后，FFDC 策略会选取下一个空闲处理器继续分配剩余的高关键性任务。当所有的高关键性任务全部成功分配之后，FFDC 会继续按照 FF 策略分配低关键性任务。在文献 [92] 中评价的划分调度算法中，FFDC 具有最好的性能表现。但是，直接将 EY 算法与 FFDC 策略相结合并不能得到满意的性能表现。其原因主要 EY 算法的优点是能够很好的平衡高关键性任务与低关键性任务的可调度性。而 FFDC 策略会造成不同关键性任务在处理器间的分布极不平衡，因此导致 EY 算法失去了在单个处理上通过调节虚拟截止期来平衡不同关键性任务可调度性的空间。考虑如表3.1 所示的混合关键性任务集合：

该混合关键性任务集合将被分配到两个处理器中进行划分调度。根据 FFDC 划分

表 3.1 混合关键性任务集合实例
Table 3.1 An Example Mixed-criticality Task Set

Task	T_i	D_i	ζ_i	$C_i(\mathrm{LO})$	$C_i(\mathrm{HI})$	U_i^{LO}	U_i^{HI}
τ_1	10	10	HI	2	3	0.2	0.3
τ_2	10	10	HI	2	3	0.2	0.3
τ_3	10	10	LO	7	7	0.7	0.7
τ_4	10	10	LO	7	7	0.7	0.7

策略，两个高关键性任务 τ_1 and τ_2 都将被分配到处理器 P_1 中，之后继续分配低关键性任务 τ_3 和 τ_4 。由于处理器 P_1 中已分配的两个高关键性任务在低关键性的利用率之和以及达到了 0.4，而任务 τ_3 和 τ_4 的低关键性利用率均为 0.7，因此无论如何调节 τ_1 和 τ_2 的虚拟截止期，两者均不能被分配到处理器 P_1 中。又因为 τ_3 和 τ_4 的低关键性利用率之和为 $1.4 > 1$，所有这两个低关键性也不能被同时分配到处理器 P_2 中。综上可知，该混合关键性任务集合不能够通过 FFDC 策略成功划分。

如果使用最差适应（WF，worst-fit）策略来划分该任务集合，那么容易验证任务 τ_1 和 τ_3 将被分配到处理器 P_1，而任务 τ_2 和 τ_4 将被分配到处理器 P_2。通过使用 EY 算法的虚拟截止期调整，可以验证当高优先级任务 τ_1 和 τ_2 的虚拟截止期分别设置为 6 时，两个划分处理器上的子任务集合都是可调度的。综上，WF 划分策略能够将高关键性的任务分配到不同的处理器中，从而给予 EY 算法更多的机会来通过调整高关键性任务的虚拟截止期来提升系统的可调度性。

在分配高关键性任务时，划分算法会通过调整被划分任务的虚拟截止期来保证每个处理器上被分配任务在高关键下的可调度性。只要所有的高关键性任务都能够被成功分配处理器，那么所有被分配的高关键性任务的虚拟截止期也就同时被确定了。因此当开始划分低关键性任务时，每个处理器剩余的低关键性利用率资源都已经确定了。所以划分低关键性任务的问题与传统的单一关键性任务的划分问题是类似的。由于 FF 划分策略被证明是解决此类问题的最优方法，本章便选择 FF 划分策略作业低关键性任务划分的策略。

3.2.2 MPVD 划分调度算法

本小节将开始详细介绍本章提出的新的混合关键性多处理器划分调度算法 MPVD（Mixed-criticality Partitioning with Virtual Deadlines）。首先定义处理器的（高关键性或低关键性）剩余利用率资源表示 1 与当前分配到该处理器的所有任务的（高关键性或低关键性）利用率之和的差值。例如，如果只有一个利用率为 0.3 的任务被分配到处理器 P_1，那么 P_1 的剩余利用率资源为 $1-0.3=0.7$。令 $U_{\mathrm{HI}}(P_1)$ 和 $U_{\mathrm{LO}}(P_1)$ 分别表示处理器 P_1 的高关键性和低关键性下的剩余利用率资源。

MPVD 划分调度算法遵从如下的三个步骤：

（1）根据 WF 划分策略首先分配高关键性任务到各个处理器中，即每次总将待分配的高关键性任务分配到剩余高关键性利用率资源 $U_{\mathrm{HI}}(P_x)$ 最多的处理器中。而高关键性任务的分配次序按照各自高关键性利用 U_{HI} 从大到小的顺序排列。

（2）按照文献 [69] 中的方法分别为各个处理器中已分配的高关键性任务调节虚拟

截止期，以满足式 (3.6)所示的 dbf_{HI} 约束。即保证分配到每个处理中的高关键性任务子集都能够在高关键性模式下可调度。如果虚拟截止期调整算法在任意一个处理器返回失败，则 MPVD 划分调度算法随即返回失败。

（3）最后安装 FF 划分策略来为低关键性任务分配处理器。低关键性任务的分配次序按照每个任务的低关键性利用率 U_{LO} 降序排列。在此步骤中，使用 dbf_{LO} 的约束条件式 (3.5) 来检验当前待分配的低关键性任务是否能够分配到候选处理器中。即保证已分配到当前候选处理的任务和当前待分配任务在低关键性模式下都能够满足各自的截止期约束。

需要注意的是，MPVD 划分调度算法的第一步仅将所有的高关键性任务分配到合适的处理器中，并不对划分结果的可调度性提供任何保证。在所有高关键性任务全部划分完成后，才执行虚拟截止期调整算法来判定每个处理中的划分任务子集是否在高关键性模式下可调度。EY 算法中的虚拟截止期调整算法的时间复杂度为伪多项式级别，在实际执行时会消耗大量的时间。因此如果该算法在每次迭代中频繁调用将导致算法的时间效率极度低下。而 MPVD 划分调度算法在每个处理器上仅需要执行一次，因此是十分高效的。

在运行时调度阶段，在每个处理器上使用与 EY 算法类似的方式来调度被分配到该处理器中的任务。但系统处于低关键性模式时，使用 EDF 算法对所有任务进行调度，其中高关键性任务的优先级通过其虚拟截止期来决定。当系统执行在高关键性模式时，所有低关键性的任务都立即被抛弃掉，剩余的高关键性任务继续通过 EDF 算法进行调度，但其优先级通过其原始的截止期来决定。

3.3 MPVD 算法优化技术

在上一小节提出的 MPVD 算法能够使得高优先级的任务更为均衡的分配到各个处理器中，从而为 EY 算法的虚拟截止期调整过程创造了更大的空间。但是该算法在某些特定情况下仍然会导致较差的性能表现。在本小节将对造成 MPVD 算法性能下降的因素进行分析，并提针对性的改进方法来优化划分策略和虚拟截止期调整算法，从而进一步提升算法的可调度性。

3.3.1 重型低关键性任务敏感的划分策略

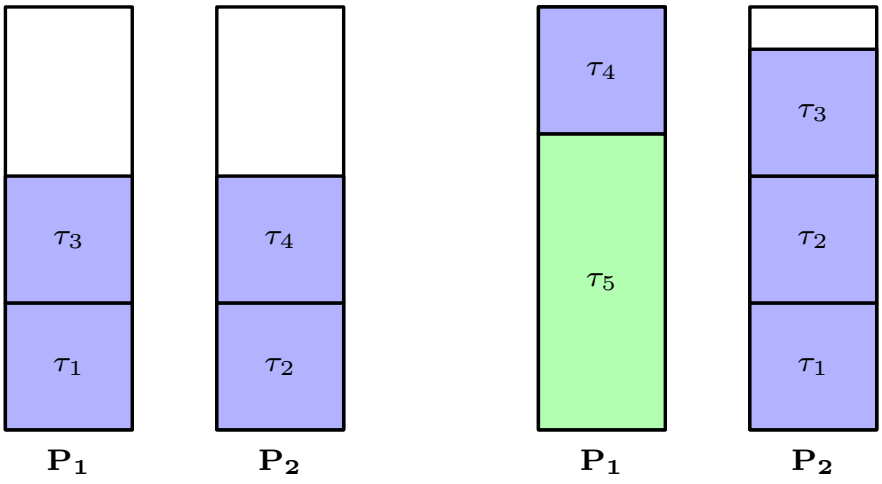
在划分过程中需要避免的最重要问题是当为一个重型（高资源利用率）任务分配处理器时，虽然所有处理的剩余资源之和还很大，却因为单个处理的剩余利用率资源都不足以满足该重型任务的需求而导致划分操作失败。在传统的单关键性任务划分过程中，采用利用率降序排列的任务次序来进行划分操作能够很有效地避免这一问题。

表 3.2 混合关键性任务集合实例 II

Table 3.2 An Example Task Set

Task	T_i	D_i	ζ_i	$C_i(\mathrm{LO})$	$C_i(\mathrm{HI})$	U_i^{LO}	U_i^{HI}
τ_1	10	10	HI	2	3	0.2	0.3
τ_2	10	10	HI	2	3	0.2	0.3
τ_3	10	10	HI	2	3	0.2	0.3
τ_4	10	10	HI	2	3	0.2	0.3
τ_5	10	10	LO	7	7	0.7	0.7

所以 MPVD 算法的第 1 阶段和第 3 阶段也都采用这一次序分别对高关键性和低关键性的任务进行划分操作。但是，由于 MPVD 算法将不同关键性任务的划分过程放到了不同的阶段，这便可能导致某些低关键性任务在所有处理器的剩余利用率资源还很充裕时，却不能够被分配到任意一个处理器中。例如考虑将表2.1中所示的混合关键性任务集合划分到两个处理中的情况。根据 MPVD 的原始算法，每个处理器将被分配 2 个高关键性的任务（如 图3.3(a)所示）。又因为在每个处理器上的低关键性剩余利用率资源（$1-0.2-0.2=0.6$）均小于低关键性任务 τ_5 的利用率 0.7，所以 τ_5 不能够被分配到任意一个处理器中，无论算法如何调整高关键性任务的虚拟截止期。



(a) MPVD　　(b) MPVD-HA

(a) MPVD　　(b) MPVD-HA

图 3.3 使用不同方法的任务划分结果

Fig. 3.3 Task allocation results in different approaches

为了解决这一问题，本小节提出一种低关键性重型任务敏感的划分策略来强化 MPVD 划分算法。该改进策略的主要思想是在高关键性任务工作量的均衡分配与保证低关键性任务可调度性之间进行折中。具体方法是在划分高关键性任务之前，首先为

低关键性重型任务预留适度的空闲资源。本小节对低关键性重型任务进行如下的形式化定义：

定义 3.2 (低关键性重型任务)：给定一个低关键性混合关键性系统 τ，一个低关键性任务 τ_k 是重型任务的充分条件是

$$U_k^{\mathrm{LO}} > 1 - \frac{U_{\mathrm{LO}}(\pi_{\mathrm{HI}})}{m}. \tag{3.8}$$

为了与 MPVD 的原始算法相区分，将采用低关键性重型任务敏感策略的新算法标示为 MPVD-HA（heavy low-criticality task aware MPVD partitioning algorithm）。其与 MPVD 算法区别为在原始算法的第 1 步之前添加如下步骤：

- 选择所有满足式 (3.8)中条件的重型低关键性任务，并将每个重型任务关联到一个单独的处理器上。如果一个重型低关键性任务 τ_b 被关联到了处理器 P_x，那么将设置 P_x 的初始高关键性剩余利用率资源为

$$U_{\mathrm{HI}}(P_x) \leftarrow 1 - U_b^{\mathrm{LO}}. \tag{3.9}$$

另外需要注意的是，MPVD-HA 在为重型低关键性任务分配预留利用率资源时，仅是修改各个对应处理器的高关键性利用率剩余资源的取值，并不是直接将重型任务分配到该处理器上。所有的低关键性任务都会在之后重新被分配处理器。

在此步骤之后 MPVD-HA 算法将按照与 MPVD 相同的过程继续进行任务划分。根据重型任务的判定条件式 (3.8)易知，当重型低关键性任务的数量大于处理器的数量 m 时，因为总的低关键性任务利用率之和已经大于 m，所以该任务集合是不能够被任何算法成功调度的。

在 表2.1所示的实例中，因为 $1 - U_{\mathrm{LO}}(\pi_{\mathrm{HI}})/2 = 0.6 < U_5^{\mathrm{LO}} = 0.7$，所以任务 τ_5 为重型低关键性任务并且被关联到处理器 P_1 中。因此 $U_{\mathrm{HI}}(P_1)$ 将被设置为 $1 - 0.7 = 0.3$。之后，遵从 MPVD 的划分算法继续将任务 τ_1, τ_2, τ_3 分配到处理器 P_2 中，将任务 τ_4 分配到处理器 P_1 中。划分结果如图3.3(b)所示。最终处理器 P_1 保留了足够的空闲资源能够使得重型低关键性任务能够在 MPVD 算法的第 3 步骤被成功分配到处理器 P_1，从而该任务集合能够被 MPVD-HA 算法成功划分。

3.3.2 虚拟截止期调整优化算法

EY 算法能够获得优异的可调性的一个主要原因，是使用了一个非常有效的启发式算法来调整高关键性任务的虚拟截止期。但是该启发式算法并不完全适用于 MPVD 划分调度。在 MPVD 算法的第 2 步调用虚拟截止期调整算法时，各个处理器中只包含已分配的高关键性任务，而缺少未来将被分配的低关键性任务信息。如果算法根据式

(3.7)来选择任务并缩小其虚拟截止期，将导致 $\mathrm{dbf}_{\mathrm{LO}}$ 的快速增长，从而不利于低关键性模式下的可调度性。为了解决这一问题，本小节定义了平衡因子参数，并以此来选择调节虚拟截止期的任务。

定义 3.3 (平衡因子)：对于一个高关键性任务 τ_k 和任意时间间隔 t，定义其平衡因子为：

$$\phi(\tau_k,t)=\frac{\mathrm{dbf}_{\mathrm{HI}}(\tau_k,D_k(\mathrm{LO}))-\mathrm{dbf}_{\mathrm{HI}}(\tau_k,D_k(\mathrm{LO})-1)}{C_k(\mathrm{LO})/(D_k(\mathrm{LO})-1)-C_k(\mathrm{LO})/D_k(\mathrm{LO})}. \tag{3.10}$$

平衡因子中的分母 $C_k(\mathrm{LO})/(D_k(\mathrm{LO})-1)-C_k(\mathrm{LO})/D_k(\mathrm{LO})$ 评价了当一个高关键性任务的虚拟截止期被减小 1 后，对于低关键性模式下的调度造成的不利影响。而本节提出的优化算法将总是选择拥有最大平衡因子取值 $\phi(\tau_k,t)$ 的高关键性任务来调整虚拟截止期。通过参考平衡因子，能够在提升高关键性模式下可调度性的同时尽可能的减小对低关键性模式下可调度性的影响。

3.4 实验结果与分析

本小节通过基于随机生成混合关键性任务集合的模拟实验来比较本章提出的算法与之前的混合关键性系统多处理器划分调度算法的性能。实验比较的主要指标是随机任务集合的可调度比率。实验中评测比较了已有的全局调度算法和划分调度算法。实验结果表明与单一关键性的传统实时任务多处理器调度算法类似，划分调度算法的性能显著的优于全局调度算法（文献 [101, 105] 中的算法）。因此，在本小节只是给出本章提出的算法与其他的划分调度算法的可调度性比较实验结果。实验评价的混合关键性调度算法包括：

- MPVD: 在 小节3.2 中介绍混合策略划分调度算法的原始版本。
- MPVD-HA: 在 小节3.3.1 中介绍的使用重型低关键性任务敏感策略的 MPVD 算法优化版本。
- MPVD-HA-BF: 小节3.3.2 中介绍的使用平衡因子优化虚拟截止期调整的 MPVD-HA 的进一步优化版本。
- DC-Audsley: 在 [92] 中介绍的基于 Audsley 方法的划分调度算法。
- EY-FF: 在小节3.2的开始部分介绍的 EY 算法采用 FFDC 策略的直接多处理器扩展算法。
- MC-Partition: 在 [104] 中介绍的基于 EDF-VD 方法的划分调度算法。

3.4.1 随机任务集合生成

我们采用与上一章实验中类似的生成混合关键性随机任务集的方法，并将其扩展至多处理器平台。一个随机实时任务集初始时被设置为空集 $\pi \leftarrow \emptyset$，然后逐次添加新的随机混合关键性实时任务。随机任务的生成主要受 5 个参数的控制：随机任务为高关键性任务的最大概率 P_{HI}；高关键性任务的高关键性下的最差执行时间与低关键性下最差执行时间的最大比例 R_{HI}；在低关键性下最差执行时间的最大值 $C(\mathrm{LO})$；最大的实时任务周期 T^{max}；和单位速率处理器的个数 m。每个新的随机实时任务按照如下步骤生成：

（1）τ_i 服从 P_{HI} 的概率取值 HI, 否则取值 LO；

（2）$C_i(\mathrm{LO})$ 的取值在 $\{1,2,\cdots,C_{\mathrm{LO}}^{max}\}$ 范围内服从均匀分布；

（3）如果该随机任务为低关键性任务则 $C_i(\mathrm{HI}) = C_i(\mathrm{LO})$，否则 $C_i(\mathrm{HI})$ 在 $\{C_i(\mathrm{LO}), C_i(\mathrm{LO})+1,\cdots,R_{\mathrm{HI}}\cdot C_i(\mathrm{LO})\}$ 范围内均匀分布；

（4）T_i 的取值在 $\{C_i(\zeta_i), C_i(\zeta_i)+1,\cdots,T^{max}\}$ 范围内服从均匀分布；

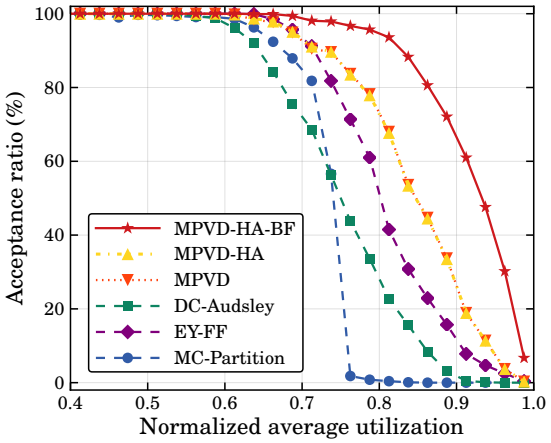
（5）由于我们采用隐式截止期的模型, 所以有 $D_i = T_i$。

每个任务集在生成时都有一个平均资源利用率基准 $U^{\star}$。由于产生拥有准确资源利用率的随机任务集比较困难，所以我们允许任务集的平均利用率有一个可接受的误差范围：$U_{min}^{\star} = U^{\star} - 0.005\cdot m$，$U_{max}^{\star} = U^{\star} + 0.005\cdot m$。若满足 $\min(U_{\mathrm{LO}}(\tau), U_{\mathrm{HI}}(\tau)) < U_{min}^{\star}$，就继续产生更多的任务并将它们添加到任务集 τ。如果将一个任务加入 π 后导致 $\max(U_{\mathrm{LO}}(\tau), U_{\mathrm{HI}}(\tau)) > U_{max}^{\star}$ 则随机任务集生成算法将整个任务集抛弃，并从一个空的任务集重新开始生成、如果将一个随机任务加入 π 后，满足 $U_{min}^{\star} \le \min(U_{\mathrm{LO}}(\tau), U_{\mathrm{HI}}(\tau))$ 和。$\max(U_{\mathrm{LO}}(\tau), U_{\mathrm{HI}}(\tau)) \le U_{max}^{\star}$ 则一个随机任务集生成完毕, 除非任务集中的所有任务都有相同的关键性或者 $U_{\mathrm{LO}}(\pi) > 0.99\cdot m$ 或 $U_{\mathrm{HI}}(\pi) > 0.99\cdot m$。在这种情况下，此任务集也将被抛弃。

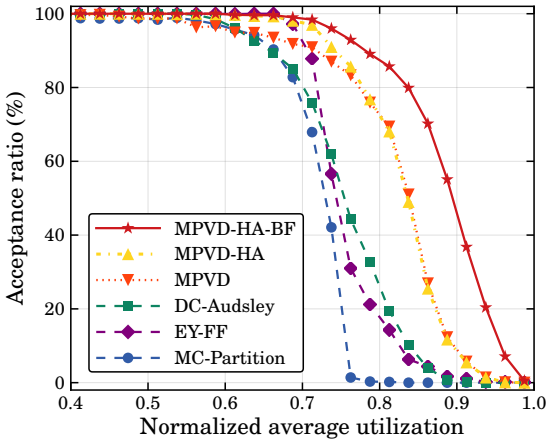
3.4.2 实验结果分析

实验中的随机任务集合生成参数设置如下：$P_{\mathrm{HI}} = 0.5$, $R_{\mathrm{HI}} = 4$, $C_{\mathrm{LO}}^{\max} = 10$ and $T^{\max} = 200$。图3.4(a)至 图3.4(d)展示了不同处理器数量设置下随机任务集合被不同算法调度时，可调度接受率随规范化评价利用率变化的趋势关系。各个图中的每个点均融合了超过 2000 个随机生成随机任务集合样本的实验结果信息。

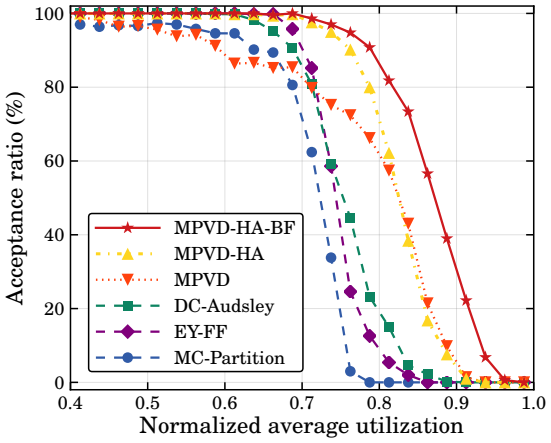
图3.4 中的结果表明本章提出的 MPVD 划分调度算法在处理器处理较小的情况下呈现出比 DC-Partition, DC-Audsley and EY-FF 更好的可调度性能。但是，当处理器的数量增加时，MPVD 算法的性能会出现明显的降级。当处理器数量为 16 时，MPVD



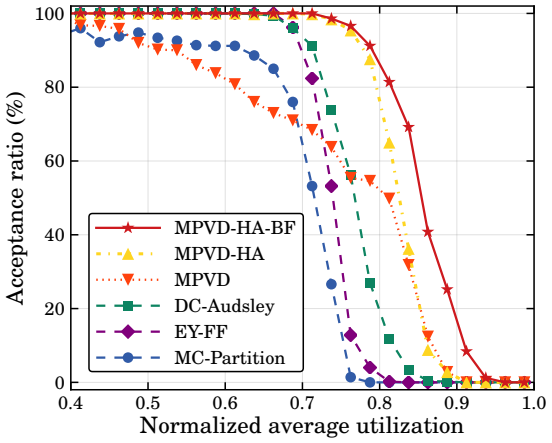
(a) 2 个处理器的试验结果
(a) Result on 2-processor system



(b) 4 个处理器的试验结果
(b) Result on 4-processor system



(c) 8 个处理器的试验结果
(c) Result on 8-processor system

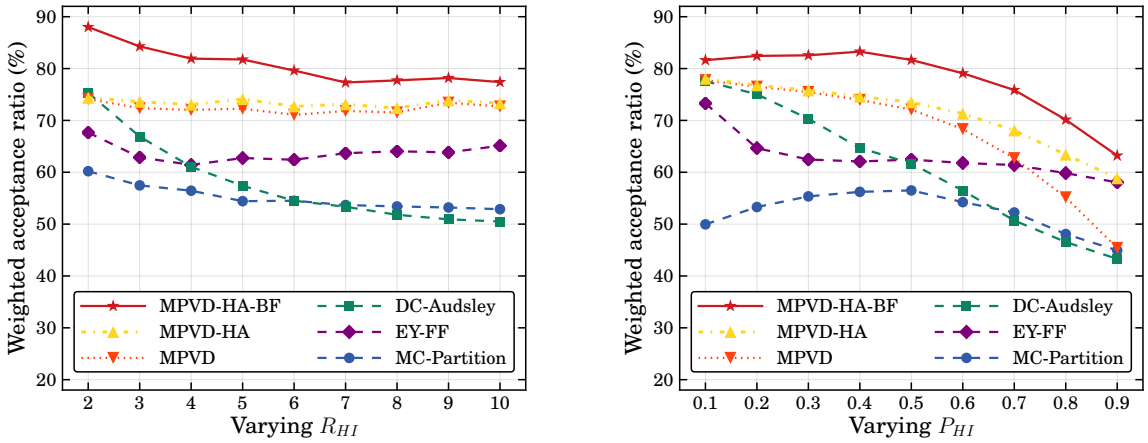


(d) 16 个处理器的试验结果
(d) Result on 16-processor system

图 3.4 实验结果（$P_{\mathrm{HI}}=0.5, R_{\mathrm{HI}}=4, C_{\mathrm{LO}}^{\max}=10 and T^{\max}=200$）
Fig. 3.4 Experiment results ($P_{\mathrm{HI}}=0.5, R_{\mathrm{HI}}=4, C_{\mathrm{LO}}^{\max}=10 and T^{\max}=200$)

算法甚至不能够成功划分规范化评价利用率很小的任务集合。造成该问题的主要原因已经在小节3.2的开始部分进行了讨论。而改进的算法 MPVD-HA 很好的解决了该问题，并相较于其他算法具有稳定地性能提升。而综合了本章提出的两种改进策略的算法 MPVD-HA-BF 也呈现出了更进一步地性能提升。图3.5中的结果页表明了在不同随机任务集合生成参数设置下，MPVD-HA-BF 算法都能表现出稳定并且显著的可调度性能提升。

图3.5(a) 和 图3.5(b) 展示了分别调整随机参数 R_{HI} 和 P_{HI} 对实验结果的影响。图3.5(a) 和 图3.5(b)的 $x-$ 轴 坐标分别表示两个调整参数的取值，$y-$ 轴 坐标表示加权可调度接受比率（*weighted acceptance ratio*）。对于每个变化的参数取值，实验都对于



(a) 调节参数 R_{HI}　　(b) 调节参数 P_{HI}

(a) Varying R_{HI} on 4-processor system　　(b) Varying P_{HI} on 4-processor system

图 3.5 实验结果（$P_{\mathrm{HI}}=0.5, R_{\mathrm{HI}}=4, C_{\mathrm{LO}}^{\max}=10 and T^{\max}=200$）

Fig. 3.5 Experiment results ($P_{\mathrm{HI}}=0.5, R_{\mathrm{HI}}=4, C_{\mathrm{LO}}^{\max}=10 and T^{\max}=200$)

20 个不同的任务集合平均利用率 U_i 分别统计了各自的平均可调度接受率结果 $A(U_i)$。并通过如下公式来计算加权可调度接受比率：$\sum_{\forall U_i} W(U_i)\cdot A(U_i)/\sum_{\forall U_i} W(U_i)$。图中的每个点均融合了超过 20000 个随机生成任务集合的信息。

3.5 小结

本章研究了混合关键性系统的多处理划分调度问题。本章基于目前单处理器上性能最好的 EY 算法[69]，提出了基于混合任务划分策略的多处理器划分调度算法，并介绍了两种优化方法。基于随机生成任务集合的模拟实验结果表明，本章提出算法与之前已提出的算法相比具有显著的可调度性能提升。

第 4 章 多处理器混合关键性系统划分调度策略

多核处理器正越来越多地应用到现代实时嵌入式系统中，以提供更强的计算能力来满足在同一硬件平台上集成不同关键性级别的多种功能的需要。混合关键性系统的调度问题即便在单处理器平台中都极具挑战性，而多核处理器平台的应用为实时系统的设计带来了极大的挑战。

然而据我们所知，目前在单处理器平台下 EY-VD[69] 算法的资源利用率远高于其他已有的混合关键性算法（包括 RM、OPA 和 EDF-VD），但仍未有基于该算法的混合关键性划分调度问题的研究。文献 [134] 证明了在传统的强实时系统中划分调度相较于全局调度具有更好的性能。Kelly 等在文献 [92] 中首次提出了可抢占多处理器平台中基于固定优先级算法的划分调度问题，认为以任务关键性级别降序的 first-fit 方法是最优的混合关键性任务划分算法。Baruah 等在文献 [104] 中提出了基于非固定优先级算法 EDF-VD[45]，并采用基于 DC 策略的混合关键性划分调度算法 MC-Partition。但已有的划分调度算法均基于传统非混合关键性多处理器系统中的划分策略，尚未有特别针对混合关键性系统优化的划分调度策略的研究。因此本文首先将单处理器平台中的非固定优先级混合关键性调度算法 EY-VD 扩展至多处理器平台，并基于传统划分策略提出了第一个划分调度算法 MC-PEDF（Mixed-Criticality Partitioned Earliest Deadline First）。然后通过对传统划分策略可能导致混合关键性系统在不同关键性模式中处理器间资源利用率不平衡问题的研究，我们提出了针对多处理器平台混合关键性系统优化的划分调度策略 OCOP（One Criticality One Partition）。OCOP 划分策略允许混合关键性实时任务在不同的系统关键性模式下被分配到不同的处理上执行，该方法较好地平衡了系统在不同关键性模式中各个处理器间的资源利用率，进而提升了划分调度算法的性能。基于 OCOP 划分策略，我们提出了第二个划分调度算法 MC-MP-EDF（Mixed-Criticality Multi-Partitioned EDF）。

本文将目前资源利用率最高的单处理器混合关键性调度算法 EY-VD 扩展到多处理器平台中。首先我们结合传统的划分调度策略提出了适用于多处理器混合关键性系统的 MC-PEDF 划分调度算法。尽管比之前的算法有更好的可调度性能，但我们发现传统的划分策略不能有效地平衡不同关键性级别下的负载，故其不完全适用于混合关键性系统。为了克服传统策略的不足，我们提出了新型的划分调度策略 OCOP（One Criticality One Partition）。OCOP 允许系统在关键性模式切换时对实时任务集进行重新划分，进而更好的平衡各个处理器在不同关键性模式中的资源利用率。基于 OCOP，

我们提出了第二个划分调度算法 MC-MP-EDF。基于随机生成任务集的仿真实验结果表明，相较于 MC-PEDF 和已有的算法，MC-MP-EDF 能够显著的提高系统的可调度性，尤其是在处理器数量较多的系统中。

4.1 基本概念

将混合关键性偶发实时任务系统定义为由一组相互独立的有限数偶发性量混合关键性实时任务构成的集合。其中每个任务都可能会产生无限数量且任意次序的混合关键性作业序列。

与小节3.1.1中对混合关键性系统模型的定义类似，本章将每个混合关键性任务定义为一个四元组：$\tau_i = (T_i, D_i, C_i, \zeta_i)$。为了简化系统行为的描述和算法的表达与分析，只研究拥有两个关键性级别的双关键性系统模型。并使用符号 LO 表示低关键性级别，使用 HI 表示高关键性级别。而本文的结论均可以扩展到任意关键性个数的情况。

为了成功调度上述的双关键性系统，所有高关键性任务释放的作业无论在低关键性模式还是高关键性模式下都必须满足截止期约束，而低关键性任务释放的作业仅需要在低关键性模式下满足截止期约束。

4.1.1 需求上界函数 DBF

需求上界函数（Demand-Bound Function，DBF）描述任意指定长度时间间隔内，系统产生的最大执行时间需求上界，即系统所需提供的能够保证实时性约束的最小处理器资源数量。

定义 4.1 (需求上界函数 -DBF)：一个需求上界函数 dbf(τ_i , Δ) 表示一个实时任务 τ_i 在给定长度为 Δ 的时间间隔内可能产生的最大执行时间需求的上界。其中执行时间的需求为所有由实时任务 τ_i 释放且调度窗口完全包含在长度为 Δ 的时间间隔内的作业所需要的最大执行时间（即 WCET）之和。

DBF 是传统实时任务模型下分析实时系统工作量可调度性（Schedulability）的有效方法，并且针对诸多主流的实时任务模型，都有精确的计算方法。例如在偶发实时任务模型中，DBF 可以通过文献 [8] 中的算法求得。Ekberg 等在文献 [69] 中将其应用扩展到单处理器混合关键性偶发实时任务系统模型中，并给出了基于 EY-VD[69] 算法的混合关键性实时任务分别在低关键性和高关键性下的 DBF 计算方法。下面我们将对该法进行简要介绍。

当混合关键性系统运行在低关键性级别（$\ell = \text{LO}$）时，每个混合关键性任务 τ_i 可视为普通的单关键性实时任务 $(C_i(\text{LO}), D_i(\text{LO}), T)$。其中 $D_i(\text{LO})$ 是实时任务 τ_i 在低关键

性级别下的相对截止期，当 τ_i 为低关键性实时任务时 $D_i(\mathrm{LO}) = D_i$；当 τ_i 为高关键性实时任务时，$D_i(\mathrm{LO}) \leq D_i(\mathrm{HI}) = D_i$。可以得到：

$$\mathrm{dbf}_{\mathrm{LO}}(\tau_i, t) = \max\left\{0, \left(\left\lfloor \frac{t - D_i(\mathrm{LO})}{T_i} + 1 \right\rfloor\right) \cdot C_i(\mathrm{LO})\right\} \tag{4.1}$$

当系统运行时切换到高关键性级别时, 这里面可能包含一个遗留作业. 而从高关键性任务 τ_i 释放的遗留作业的调度窗口最小可能为 $D_i(\mathrm{HI}) - dilo$. 据此可以得到高关键性任务 DBF 的最大值:

$$\mathrm{full}(\tau_i, t) = \max\left\{0, \left(\left\lfloor \frac{t - D_i(\mathrm{HI}) - D_i(\mathrm{LO})}{T_i} + 1 \right\rfloor\right) \cdot C_i(\mathrm{HI})\right\} \tag{4.2}$$

函数 $\mathrm{full}(\tau_i, t)$ 的计算过程中没有考虑遗留作业在系统的关键性变化前已经执行的时间。式 (4.3)用于计算该时间，其中 $n = t \mod T_i$。

$$\mathrm{done}(\tau_i, t) = \begin{cases} \max\{0, C_i(\mathrm{LO}) - n + D_i(\mathrm{HI}) - D_i(\mathrm{LO})\} & D_i(\mathrm{HI}) > n \geq D_i(\mathrm{HI}) - D_i(\mathrm{LO}) \\ 0 & otherwise \end{cases} \tag{4.3}$$

根据式 (4.2)和式 (4.3)，可以得到高关键性实时任务 τ_i 在高关键性模式下的需求上界函数为：

$$\mathrm{dbf}_{\mathrm{HI}}(\tau_i, t) = \mathrm{full}(\tau_i, t) - \mathrm{done}(\tau_i, t) \tag{4.4}$$

$\mathrm{dbf}_{\mathrm{LO}}$ 和 $\mathrm{dbf}_{\mathrm{HI}}$ 的计算复杂性均是伪多项式的.

4.2 基于传统划分策略的混合关键性划分实时调度算法

在众多的单处理器混合关键性实时调度算法中，EY-VD 算法 [69] 拥有比 RM[7]、OPA[46]、EDF-VD[24] 等其他已知的混合关键性调度算法更高的资源利用率。现有多处理器平台下的混合关键性实时调度算法研究还十分有限。本文是研究 EY-VD 算法在混合关键性多处理器平台上的扩展，并提出了一种基于划分调度的算法。

4.2.1 多处理器划分调度的基本方法

传统的隐式截止期（实时任务的相对截止期严格等于周期）周期性实时任务集在可抢占多处理器平台中的划分调度问题是强 NP 难的（经典的强 NP 难问题 Bin-Paking[136] 可以转换成该问题）。而该问题的计算复杂性很容易扩展到更加一般化的混合关键性偶发实时任务模型，因此多处理器平台中混合关键性偶发实时任务系统的划分调度问题也是强 NP 难的。在传统（非混合关键性）实时任务模型下，关于可抢占多处理器平台中的划分调度问题，已经存在广泛的研究成果。而这些成果所采用的启发式划分策略大多可以分解为如下三个典型的步骤：

（1）实时任务排序：通常在将实时任务分配到特定处理器之前都会遵从某种策略

将待分配的实时任务集排序，再依次进行分配处理器的操作；

（2）处理器选择：按照在第 1 步排序的结果，依次为实时任务集中的任务使用某种启发式策略分配处理器；

（3）划分检验：检验分配到某处理器中的实时任务子集能否被某种单处理上的实时调度算法所调度。

对于划分检验，通常是采用单处理器下实时调度算法的可调度性测试方法对分配到该处理器上的实时任务子集进行可调度性分析。在此步骤中不需要考虑多处理器的资源分配问题，而且存在大量单处理器上十分成熟的调度算法及其可调度性测试方法可以选择，因此传统的多处理器实时任务划分调度问题的研究很多都集中在实时任务的排序及划分处理器的选择策略上。处理器的选择策略和 Bin-Packing 问题的启发式策略十分类似，主要包括：

- 首次适应（First Fit，FF）：每次为实时任务选处理器时，按照固定的处理器顺序选择第一个满足划分检测条件的处理器，并将该实时任务分配到此处理器中。
- 最好适应（Best Fit，BF）：每次将实时任务分配到满足划分检测条件且剩余资源最少的处理器中。
- 最坏适应（Worst Fit，WF）：与 BF 相反，WF 每次选择满足划分检测条件但剩余资源最多的处理器。

为了使启发式分配策略达到更好的可划分性能，在实时任务选择处理器划分之前采用合适的策略对实时任务集进行预排序是必要的。特别的，文献 [92, 104] 中提出的各种混合关键性实时任务系统的划分调度算法也基本遵从上述的原则。

4.2.2 MC-PEDF 算法描述

本小节介绍本文提出的可抢占多处理器平台中，基于可调虚拟截止期 EY-VD 算法且采用传统划分策略的混合关键性偶发实时任务系统划分调度算法 MC-PEDF（Mixed-Criticality Partitioned Earliest Deadline First）。假设一个混合关键性偶发实时任务实时系统包含一个混合关键性实时任务集合（由 n 个相互独立的偶发性实时任务 $\tau_1\tau_2\ldots\tau_n$ 组成），和一个多处理器系统 Π（包含 m 个单位速率的同质处理器 $p_1p_2\ldots p_m$）。MC-PEDF 是 Bin-Packing 问题中 FF-Decreasing 启发式策略的一个变种。（经过仿真实验的分析，我们发现在采用 EY-VD 作为划分处理器调度算法时，FF 启发式策略要优于 BF 以及 WF 等启发式策略。但对比这些启发式策略不是本文的重点，故在此不做详细的分析。）该算法的伪代码描述如图4.1所示。

算法：MC-PEDF，基于传统划分策略的划分调度算法
输入：被划分任务集$\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$
输出：划分失败返回 FAILURE；
划分成功返回 SUCCESS，以及划分结果
$\Pi = \{p_1 = \{\}, p_2 = \{\}, \cdots, p_m = \{\}\}$

```
1:  Sort π with Criticality Deceasing order
2:  t ←None
3:  while t = None and π ≠ ∅ do
4:      t ← PopFirtElement(π)
5:      for each p_i ∈ Π by indicator increasing order do
6:          if CheckSchedulablityWithDBF(p_i ∪ {t}) then
7:              p_i ← p_i ∪ {t}
8:              t ← None; exit for
9:          end if
10:     end for
11: end while
12: if t ≠ None retun FAILURE
13: return  SUCCESS
```

图 4.1 MC-PEDF 算法的伪代码描述
Fig. 4.1 Pseudo code of MC-PEDF algorithm

在算法第 1 行，将混合关键性实时任务集合 τ 按照关键性降序排序，而对于相同关键性的任务按平均利用率降序排列。算法的 3 至 10 行将按照第 1 行的排序结果依次为每个实时任务分配处理器；分配处理器时，MC-PEDF 会按 5 至 9 行的方法，找到满足划分检测条件并且指标值最小的处理器并将当前任务分配之（第 7 行）。如果在某一次的迭代中所有的处理器都不能满足分配当前任务的划分检测条件，则算法会在第 3 行因 t 等于 None 而退出循环，并在第 12 行返回“FAILURE”。如果对于所有的实时任务都能被正确的分配处理器，则算法会在第 3 行因待分配实时任务集合 τ 为空集而退出循环，最后在第 13 行返回“SUCCESS”。MC-PEDF 第 6 行的 CheckSchedulablityWithDBF 函数基于 Ekberg 等提出的基于虚拟截止期的 DBF-LO 和 DBF-HI 方法（如式 3、式 6 所示）来检测划分实时任务子集的可调度性。该方法也是目前在混合关键性实时任务集可调度比率评价标准下，可抢占单处理器平台中混合关键性偶发实时任务调度性能最好的算法。

MC-PEDF 的运行时调度算法

如果 MC-PEDF 算法成功返回 SUCCESS，则系统将遵守如下原则的约束进行运行时调度；

（1）系统启动时将所有实时任务按照 MC-PEDF 的划分结果分配到各处理器中，且系统进入低关键性模式。

（2）对于每个处理器，均保证在任意时刻占用处理器执行的作业之优先级不低于其就绪队列中任何其它作业的优先级（优先级的取值越小所表示的优先级越高）。即任何时刻处理器只允许分配到其中的任务所释放的优先级最高的未完成作业占用处理器执行，当更高优先级作业到来时会抢占当前执行的作业。

（3）当系统运行在低关键性模式时，所有低关键性任务和高关键性任务释放的作业均采用其释放时间与低关键性下的虚拟相对截止期（注意，本文约定低关键性任务的虚拟相对截止期等于其低关键性下相对截止期）之和作为其运行时优先级，并被插入其所分配处理器的就绪队列等待调度执行。

（4）当系统因正在执行作业的执行时间超过低关键性下的最差执行时间而未发出执行完成信号时，系统将进行模式切换并进入高关键性模式。在模式切换时，所有低关键性任务释放的作业将被终止执行并被清除出就绪队列，而高关键性任务释放作业的优先级将被调整为该作业的释放时间与高关键性下相对截止期之和，然后系统按照第 2 条约束的原则继续运行时调度。

（5）当系统运行在高关键性模式时，所有低关键性任务释放的作业将不被添加到就绪队列而是直接终止执行；而高关键性任务释放的作业将采用其释放时间与高关键性下的相对截止期之和作为其运行时优先级，并被插入其所分配处理器的就绪队列等待调度执行。

4.2.3 MC-PEDF 划分算法的时间复杂性和正确性分析

对于算法的时间复杂度，我们易知第 1 行的排序算法复杂度不会超过 $O(n^2)$，其中 n 为实时任务集中的任务数量。另外在第六行，分配到处理器 p_i 上实时任务子集（其包含任务的数量小于等于 n）的可调度性判定和第 12 行对各个划分处理器内已分配的实时任务子集中的高关键性任务设置低关键性下的虚拟相对截止期的操作均为伪多项式时间复杂度（文献 [69] 中的结论）。由于第 12 行 for each 循环的次数为划分处理器的个数 m，故第 12 行的时间复杂度亦为伪多项式级别。现在我们分析 3-10 行的嵌套循环。其中外层循环次数不超过实时任务集的任务数量 n，内层 for each 循环次数不超过处理器的数量 m。结合对第 6 行操作的分析，该嵌套循环的时间复杂度为伪多项式级别。综上分析，MC-PEDF 算法的时间复杂度为伪多项式级别。下面我们讨论 MC-PEDF 算法的正确性。

定理 4.1：对于任意混合关键性偶发实时任务集合$\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$，和 m 个同质多处理器系统$\Pi=\{p_1,p_2,\cdots,p_n\}$，如果 MC-PEDF 划分算法返回 SUCCESS，则该

混合关键性系统在低关键性和高关键性模式下都是 MC-PEDF 可调度的。

证明：该定理可通过反证法证明。我们采用反证法证明该定理。如果 MC-PEDF 算法返回 SUCCESS，则所有实时任务一定都分配到了某个特定处理器中。我们假设存在 τ 和 Π 在算法返回 SUCCESS 时，该混合关键性系统不可调度。不失一般性地，一定存在处理器 p_k，满足分配在 p_k 中的非空实时任务子集在运行时不可调度。令在 MC-PEDF 算法划分过程中最后一个分配到 p_k 中的任务为 τ_e，则一定有 CheckSchedulablityWithDBF 返回真，即分配到 p_k 中的实时任务子集在低关键性和高关键性下都是运行时可调度的。这与假设相矛盾，也证明了该定理的正确性。　□

4.3 针对混合关键性系统的多次划分实时调度策略

在上一节我们基于传统的划分调度原则，提出了 MC-PEDF 算法。本节将分析多处理器平台中混合关键性实时任务运行时的一些特性，并以此来改进传统的划分策略，提出一种针对混合关键性系统更有效的多处理器划分策略 OCOP（One Criticality One Partition）。最后提出一种基于 OCOP 的改进型 MC-PEDF 算法 MC-MP-EDF。

4.3.1 传统划分策略的局限性

在传统的多处理器偶发实时任务模型中，因为只有一种运行时模式，所有任务的主要调度参数（最差执行时间，截止期等）都是恒定的。因此传统的划分算法可通过优化排序和分配策略来平衡各个处理器上分配的负载，以达到较高的资源利用率。

与传统的多处理器实时调度模型不同，混合关键性系统在运行时会因为运行时系统关键性级别的不同而处于不同的关键性模式。而在不同的模式下系统中需要调度的实时任务规模和特征参数都有差异。例如，当系统运行在低关键性模式时，所有低关键性任务和高关键性任务释放的作业都存在于就绪队列中，并按照各自的优先级等待调度。而当系统切换到高关键性级别时，系统需要保障高关键性任务释放的作业在更悲观的调度参数（高关键性下更长的评估最差执行时间等）下依然满足截止期约束，故强行终止所有低关键性任务释放的作业并将其从就绪队列中移除。这种差异性导致某些处理器在不同运行时模式时的资源利用率相差较大。

正因多处理器混合关键性实时任务系统中实时任务调度参数的可变性，使得传统的划分策略很难平衡各个关键性模式中的资源利用率。这使得在划分实时任务时，待划分处理器尽管已分配的实时任务子集在大多数模式下的资源利用率很低却因为在个别模式下的利用率过高而不能接受新的任务，从而导致资源利用率降低，甚至是实时任务集的不可调度，最终降低了总体的系统资源利用效率。

例 4.1：考虑将表4.1所示的实时任务集分配到 2 个单位速率处理器 p_1 和 p_2 的情况。容易验证，无论采用 DU 还是 DC 的预排序策略，排序结果都和表中的指标顺序一致。而且容易验证按照 FFD 的启发式划分策略，实时任务 τ_1 和 τ_2 将被分配到处理器 p_1，而实时任务 τ_3、τ_4 和 τ_5 将被分配到处理器 p_2，且 p_1、p_2 都满足可调度性判定条件。当分配实时任务 τ_6 时，处理器 p_1 上的低关键性模式的资源利用率为 80%，高关键性模式下的资源利用率为 100%。如果将 τ_6 分配到 p_1，虽然低关键性下的资源利用率增至 92.5% 仍小于 100%，但是在高关键性模式下不能通过 CheckSchedulablityWithDBF 的可调度性检验。故 τ_6 不能被分配到 p_1。而 p_2 上低关键性模式下的资源利用率已经为 100%，故 τ_6 也不能被分配到处理器 p_2 上。因此我们得出表4.1所示的混合关键性实时任务集在 2 个单位速率处理器上不可被 MC-PEDF 划分的结论。但是由于 p_2 上没有被分配任何高关键性任务，因此在系统进入高关键性模式时，该处理器上的资源利用率为 0。这便造成了 p_2 在不同关键性模式下处理器资源利用率的极度不平衡，也造成高关键性模式下处理器 p_1 和 p_2 的资源利用率相差极大。

表 4.1 双关键性实时任务实例
Table 4.1 An example of dual-criticality task set

实时任务	C(LO)	C(HI)	关键性	zhouqi 周期/截止期
τ_1	4	5	HI	10
τ_2	4	5	HI	10
τ_3	1	1	LO	3
τ_4	1	1	LO	3
τ_5	1	1	LO	3
τ_6	0.5	0.5	LO	4

综上分析，传统的划分策略难以平衡混合关键性系统在各个关键性模式下的资源利用率，从而限制了划分调度算法在混合关键性系统中性能的提升空间。因此如何充分利用高关键性模式下某些处理器中的空闲资源来满足高关键性实时任务的可调度性，并保障更多的低关键性实时任务在低关键性模式下可调度，从而平衡划分处理器在各个关键性模式下的资源利用率，成为本文改进混合关键性实时任务划分策略的主要目标。

4.3.2 混合关键性模型中的新型划分策略 OCOP

观察 4.1：例4.1 通过对例4.1 的进一步分析，我们发现如果将 τ_6 分配到 p_1，容易验证 p_1 在低关键性下是可调度的。而当系统切换到高关键性模式时，由于 p_2 上没有

被分配高关键性的实时任务，因此 p_2 会一直处于空闲状态，造成资源的极大浪费。而如果在系统关键性模式切换时将 p_1 中的一个高关键性任务迁移到 p_2 中继续执行，则很容易验证高关键实时任务 τ_1 和 τ_2 都在各自独立的处理器中都可以在高关键性模式下满足截止期约束。由此可见如果允许系统在关键性模式切换时将高关键性任务重新划分，将有助于提高多处理器平台的资源利用率。

基于前文对传统划分策略的分析和观察 1 的启发，我们针对混合关键性实时任务模型提出了新的多处理器划分策略 OCOP（One Criticality One Partition）。其核心思想是允许混合关键性实时任务在不同的系统关键性模式下被划分到不同的处理上执行，而系统运行在特定关键性级别时仍要求每个实时任务释放的作业必须在同一处理器中执行。由于放松了传统划分策略下实时任务释放的作业不能在处理器间迁移的约束，混合关键性系统在模式切换时可以通过更加灵活的模式切换策略来保证更高关键性模式下的可调度性。

对于任意混合关键性实时任务集合 τ，系统在不同运行时关键性模式下需要保证截止期约束的实时任务是不同的。在低关键性模式下，系统必须保证所有低关键性和高关键性实时任务（即 τ 中的所有任务）的可调度性，我们将所有这些实时任务构成的集合称为低关键性模式调度任务集。类似的，系统在高关键性模式下只需要保证所有高关键性实时任务的可调度性，我们将这些高关键性实时任务构成的集合称为高关键性模式调度任务集。本文将混合关键性系统的运行时调度分为两种时期：1）普通时期，即系统持续运行于某个特定关键性模式的一段时间间隔；2）模式切换时期，即从系统监测到当前关键性模式下的过度执行开始，到完成模式切换进入到下一个关键性模式的调度为止的一段连续时间间隔。

OCOP 划分策略对混合关键性多处理器系统的运行时调度存在如下的约束条件：

（1）系统刚启动时处于模式切换时期，完成在低关键性模式下运行的准备工作。

（2）系统在模式切换时期，允许对实时任务作业进行处理器间迁移操作，将下一个关键性模式调度任务集中的任务迁移至下一个关键性模式下的目标处理器中，并终止全部非下一个关键性模式调度任务集中的实时任务作业的执行。

（3）系统在普通时期，不允许任何实时任务作业的处理器间迁移操作，当前关键性模式调度任务集中的实时任务只能将其作业释放到该模式下分配的目标处理器中调度执行。

我们基于双关键性多处理器系统模型对 OCOP 划分策略的主要思想给出具体的非形式化描述。而 OCOP 也很容易扩展至多关键性多处理器系统模型中。

（1）低（高）关键性模式调度任务集排序：排序的目的和算法与传统划分策略中的排序类似，区别在于 OCOP 需要针对不同关键性模式调度任务集使用不同的算法进行排序，以优化不同关键性模式下划分的性能。

（2）低（高）关键性模式调度任务集的划分：OCOP 使用与传统划分类似的启发式策略对不同关键性模式的调度任务集分别进行划分，不同模式中调度任务集的划分算法可以不同。特别的由于在高关键模式中存在低关键性模式中没有执行完的高关键性遗留作业，因此对于高关键性模式调度任务集的划分策略相较于低关键性模式调度任务集的划分策略难度更高。

（3）低（高）关键性模式的划分检验：OCOP 同样需要对不同模式中的划分任务进行不同关键性级别的可调度性检验。特别的，由于高关键性遗留作业的影响，高关键性模式划分检验也颇具挑战性。

（4）划分失败的调整策略：对于传统的划分策略，划分失败直接导致算法结束，OCOP 在某个关键性模式下划分失败时会尝试对实时任务集的一些调度参数进行调整，并重新对各个关键性模式调度任务集进行划分，直至所有关键性模式调度任务集都被正确划分（划分成功），或调度参数的调整空间耗尽（划分失败）。

在 OCOP 划分策略下，低关键性模式调度任务集与高关键性模式调度任务集的划分是在两个独立的阶段进行的，这就降低了混合关键性任务在不同运行时关键性模式下调度参数的改变对划分检验造成的干扰，并能够平衡各个划分处理在不同关键性模式下的资源率，从而提高整个系统的资源利用率和划分性能。在每个关键性模式下划分失败时的实时任务调度参数调整策略又降低了初始调度参数选择不当对划分性能的影响。

4.3.3 MC-MP-EDF 算法描述

在这一小节，我们基于前文提出的 OCOP 策略提出新的多处理器平台混合关键性系统划分调度算法 MC-MP-EDF（Mixed-Criticality Multi-Partitioned EDF）。该算法在双关键性模型中的算法描述如图4.2所示。算法中 CheckSchedulable 子函数的伪代码描述如图4.3 所示。

运行时调度算法

如果 MC-MP-EDF 算法返回 SUCCESS, 则系统遵从如下的约束进行运行时调度：

（1）系统启动时将所有实时任务按照 MC-MP-EDF 的低关键性划分结果 Π LO 分配到各处理器中，且系统进入低关键性模式。

算法：MC-MP-EDF，基于 OCOP 划分策略的多次划分调度算法输入：被划分任务集 $\tau = \{\tau_1, \tau_2, \cdots, \tau_n\}$
输出：划分失败返回 FAILURE；
划分成功返回 SUCCESS，以及低关键性模式和高关键性模式下划分结果 $\Pi_{\mathrm{LO}} = \{p_1 = \{\}, p_2 = \{\}, \cdots, p_n = \{\}\}$ 和 $\Pi_{\mathrm{HI}} = \{p_1 = \{\}, p_2 = \{\}, \cdots, p_n = \{\}\}$

1: $\pi_{\mathrm{HI}} \leftarrow HS \leftarrow \{i \mid \tau_i \in \pi$ **and** $\zeta_i = \mathrm{HI}\}$
2: **for each** $i \in \pi$ **do** $D_i(\mathrm{LO}) \leftarrow D_i - (C_i(\mathrm{HI}) - C_i(\mathrm{LO}))$
3: $ct \leftarrow None$
4: **while** $TRUE$ **do**
5: **if not** $CheckSchedulable(\pi, \Pi_{\mathrm{LO}}, \mathrm{LO})$ **then**
6: **if** $ct = None$ **then return FAILURE**
7: $D_{ct}(\mathrm{LO}) \leftarrow D_{ct}(\mathrm{LO}) + 1$
8: $HS \leftarrow HS \setminus \{ct\}$
9: $ct \leftarrow None$
10: **else**
11: **if** $CheckSchedulable(\pi, \Pi_{\mathrm{HI}}, \mathrm{HI})$ **then return SUCCESS**
12: **if** $HS = \emptyset$ **then return FAILURE**
13: $D_{ct}(\mathrm{LO}) \leftarrow$ select a candidate task from HS
14: $D_{ct}(\mathrm{LO}) \leftarrow D_{ct}(\mathrm{LO}) - 1$
15: **if** $D_{ct}(\mathrm{LO}) = C_{ct}(\mathrm{LO})$ **then** $HS \leftarrow HS \setminus \{ct\}$
16: **end if**
17: **end while**

图 4.2 MC-MP-EDF 算法的伪代码描述
Fig. 4.2 Pseudo code of MC-MP-EDF algorithm

（2）每个处理器均保证任意时刻占用处理器执行的作业之优先级不低于其就绪队列中任何作业的优先级。

（3）当系统运行在低关键性模式时，所有低关键性任务和高关键性任务释放的作业均采用其释放时间与低关键性下的虚拟相对截止期之和作为其运行时优先级，并被插入其低关键性下所分配处理器的就绪队列中等待调度执行。

（4）当系统发生低关键性模式向高关键性模式切换时，所有低关键性任务释放的作业将被终止执行并被清除出就绪队列，而高关键性任务释放作业的优先级将被调整为该作业的释放时间与高关键性下相对截止期之和。并将该作业插入其高关键性下所分配的处理器的就绪队列，同时从原有就绪队列中移除该作业（对于在两种关键性下分配相同处理器的作业不进行此操作）。当所有高关键性任务释放的未完成作业都完成就绪队列迁移之后，系统继续按照第 2 条约束的原则进行调度。

当系统运行在高关键性模式时，所有低关键性任务释放的作业将不被添加到就绪

队列而是直接终止执行。而高关键性任务释放的作业将采用其释放时间与高关键性下的相对截止期之和作为其运行时优先级，并被插入其在高关键性下所分配处理器的就绪队列等待调度执行。

4.3.4 算法正确性分析

引理 4.2：假设任意的混合关键性偶发实时任务集 $\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$，和单位速率同质多处理器系统 $\Pi=\{p_1,p_2,\cdots,p_n\}$。如果 MC-MP-EDF 算法返回 SUCCESS，且该任务集采用 MC-MP-EDF 的运行时调度算法，则任意被分配到特定处理器的实时任务释放的作业在低关键性模式下都是可调度的。

证明：由 MC-MP-EDF 的划分算法流程可知，该算法只在第 11 行会返回 SUCCESS。而在每次外层 for each 循环的迭代过程中只要在第 5 行的判定条件为假时才可能执行到第 11 行，即算法返回 SUCCESS 的前提条件是在所有实时任务使用当前的低关键性虚拟截止期取值时可以通过第 5 行的低关键性 DBF 判定条件。而第 5 行的判定条件保证所有高关键性和低关键性的实时任务释放的作业都可以在虚拟截止期之前执行完低关键性下评估的最差执行时间。

又因为对于所有的低关键性实时任务其虚拟截止期等于低关键性下的截止期，而所有高关键性任务的虚拟截止期又小于等于其低关键性截止期，因此对于实时任务集合里的所有实时任务，他们所释放的作业在低关键性模式下都可以在各自的绝对截止期之前执行完毕。引理证毕。　□

引理 4.3：假设任意的混合关键性偶发实时任务集 $\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$，和单位速率同质多处理器系统$\Pi=\{p_1,p_2,\cdots,p_n\}$。如果 MC-MP-EDF 算法返回 SUCCESS，且该任务集合采用 MC-MP-EDF 的运行时调度算法，则任意高关键性实时任务释放的作业在高关键性模式下是可调度的。

证明：与引理 1 的证明类似，当算法返回 SUCCESS 时，第 11 行的判定条件可以保证在所有实时任务使用当前的低关键性虚拟截止期取值时，系统中所有的高关键性实时任务都可以被划分到某个处理器中，并且划分到同一处理器中的高关键实时任务满足 DBF-HI 的判定条件。虽然在系统切换到高关键性模式时高关键性的实时任务会进行处理器间迁移，但第 5 行的判定条件保障了所有的高关键性实时任务释放的作业在其虚拟截止期之前可以执行完低关键性下的最差执行时间，因此迁移操作不会影响 DBF-HI 判定的准确性。而 DBF-HI 判定条件也可以保证每个处理器上分配的高关键性实时任务所释放的作业都能够满足各自高关键性下的截止期约束。引理证毕。　□

算法：CheckSchedulable，判断输入的实时任务集是否可调度
输入：被划分任务集$\tau=\{\tau_1,\tau_2,\cdots,\tau_n\}$，可用处理器集合 **PS**（保存划分结果），任务的关键性级别 CL
输出：划分失败返回 FAILURE；划分成功返回 SUCCESS，以及划分结果 PS

1: Sort π by decreasing order of $C_i(CL)/D_i(CL)$
2: **for each** $p_i \in PS$ **do** $p_i \leftarrow \{\}$
3: **for each** $itr \in TS$ **do**
4: **for each** $p_i \in PS$ by indicator increasing order **do**
5: **if** $CheckSchedulablityWithDBF(p_i \cup \{\tau_{itr}\}, CL)$ **then**
6: $p_i \leftarrow p_i \cup \{\tau_{itr}\}$
7: $itr \leftarrow None$; **exit for**
8: **end if**
9: **end for**
10: **if** $itr \neq None$ return FALSE
11: **end for**
12: **return** TRUE

图 4.3 CheckSchedulable 算法的伪代码描述
Fig. 4.3 Pseudo code of CheckSchedulable algorithm

4.3.5 算法复杂性分析

通过 MC-MP-EDF 的算法描述易知，外层 for each 循环进行下一次迭代的必要条件是在第 12 行集合 HS 不为空。又因为每次迭代都会使得 HS 中某个高关键性的实时任务 τ_i 的 $D_i(\text{LO})$ 减 1 或者直接退出循环算法终止，而高关键性任务 τ_i 的 $D_i(\text{LO})$ 减至 $C_i(\text{LO})$ 时该任务将从 HS 中移除，因此外层 for each 循环的最大迭代次数为：

$$\sum_{\tau_i \in HI(\pi)} (D_i - C_i(\text{LO}) + 1) \tag{4.5}$$

通过与算法 MC-PEDF 类似的复杂性分析易知，第 5 行和第 11 行操作具有伪多项式的时间复杂性。综上分析，MC-MP-EDF 的总体时间复杂性是伪多项式的。

4.4 实验仿真与结果分析

本节将通过仿真实验来比较本文提出的两种多处理器平台中混合关键性实时任务系统的划分调度算法，以及现有划分调度算法的可调度性能。实验针对双关键性隐式截止期偶发实时任务模型和单位速率的同质多处理器平台。本文的实验实现了 Kelly 等在文献 [92] 中提出的两个资源利用率比较高的算法 DC-RM 和 DC-Audsley，并以此为参照比较了本章提出的划分调度算法 MC-PEDF 在随机任务集可调度比率评价标准中的性能。并通过观察采用 OCOP 划分策略的算法 MC-MP-EDF 的性能改进效果来评价 OCOP 的有效性。

4.4.1 随机任务集生成算法

我们采用与文献 [69] 中类似的生成混合关键性随机任务集的方法，并将其扩展到多处理器平台中。一个随机实时任务集初始时被设置为空集 $\pi \leftarrow \emptyset$，然后逐次添加新的随机混合关键性实时任务。随机任务的生成主要受 5 个参数的控制：随机任务为高关键性任务的最大概率 P_{HI}；高关键性任务的高关键性下的最差执行时间与低关键性下最差执行时间的最大比例 R_{HI} 务在低关键性下最差执行时间的最大值 $C(\mathrm{LO})$；最大的实时任务周期 T^{max}；和单位速率处理器的个数 m。每个新的随机实时任务按照如下步骤生成：

（1）τ_i 服从 P_{HI} 的概率取值 HI，否则取值 LO；

（2）$C_i(\mathrm{LO})$ 的取值在 $\{1,2,\cdots,C_{\mathrm{LO}}^{max}\}$ 范围内服从均匀分布；

（3）如果该随机任务为低关键性任务则 $C_i(\mathrm{HI}) = C_i(\mathrm{LO})$，否则 $C_i(\mathrm{HI})$ 在 $\{C_i(\mathrm{LO}), C_i(\mathrm{LO})+1,\cdots,R_{\mathrm{HI}}\cdot C_i(\mathrm{LO})\}$ 范围内均匀分布；

（4）T_i 的取值在 $\{C_i(\zeta_i), C_i(\zeta_i)+1,\cdots,T^{max}\}$ 范围内服从均匀分布；

（5）由于我们采用隐式截止期的模型，所以有 $D_i = T_i$。

我们定义一个双关键性实时任务集 π 的平均利用率为：

$$U_{avg} = \frac{U_{\mathrm{LO}}(\pi) + U_{\mathrm{HI}}(\pi)}{2} \tag{4.6}$$

每个任务集在生成时都有一个平均资源利用率基准 $U^{\star}$。由于产生拥有准确资源利用率的随机任务集比较困难，所以我们允许任务集的平均利用率有一个可接受的误差范围：$U_{min}^{\star} = U^{\star} - 0.005 U_{max}^{\star} = U^{\star} + 0.005$。若满足 $U_{avg}(\pi) < U_{min}^{\star}$, 就继续产生更多的任务并将它们添加到任务集 π。如果将一个任务加入 π 后导致 $U_{avg}(\pi) > U_{max}^{\star}$ 则随机任务集生成算法将整个任务集抛弃，并从一个空的任务集重新开始生成。如果将一个随机任务加入 π 后，满足 $U_{min}^{\star} \le U_{avg}(\pi) \le U_{max}^{\star}$，则一个随机任务集生成完毕，除非任务集中的所有任务都有相同的关键性或者 $U_{\mathrm{LO}}(\pi) > 0.99 \cdot m$ 或 $U_{\mathrm{HI}}(\pi) > 0.99 \cdot m$。在这种情况下，此任务集也将被抛弃。

在每次实验中生成随机任务集的各个参数设置分别为 $P_{\mathrm{HI}} = 0.5$，$R_{\mathrm{HI}} = 3$，$C_{\mathrm{LO}}^{max} = 10$，$T^{max} = 100$，$m = 4$或8。

4.4.2 实验结果分析

图4.4和图4.5分别比较了 4 个处理器 (m=4) 和 8 个处理 (m=8) 平台下，已有划分调度算法 DC-RM[92]、DC-Audsley[92]、MC-Partition[104] 与本文提出的 MC-PEDF、MC-MP-EDF 的随机任务集可调度比率。图中各点数据均基于至少 1000 个随机任务集样本（图4.4 各点随机任务集平均任务数量为 16 至 31，图4.5 为 31 至 63）。其中 x

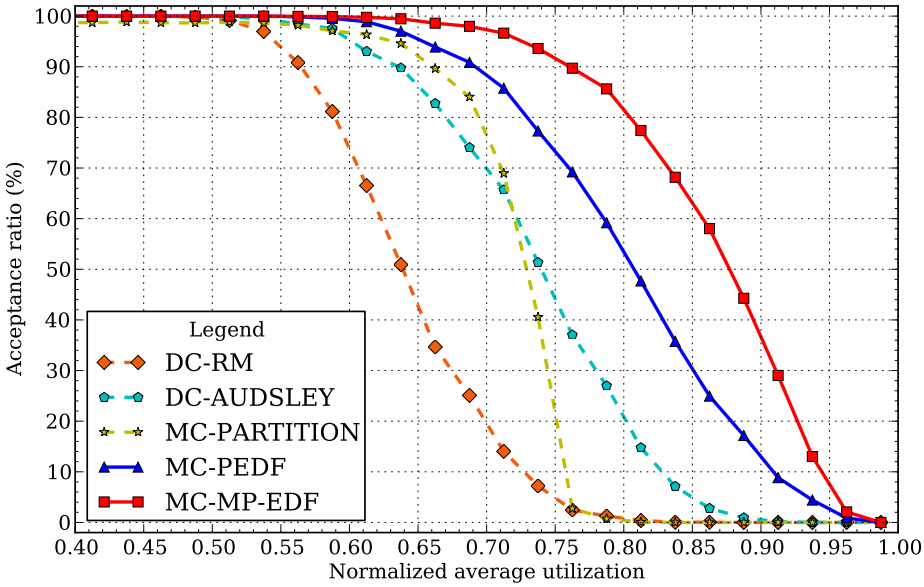


图 4.4 4 个处理器系统中的算法接受率比较

Fig. 4.4 Acceptance comparison on 4-processor platform

轴是随机任务集的规范化平均资源利用率，y 轴表示可被调度任务集数量与总样本数量的百分比。例如图4.4 中的曲线 MC-MP-EDF 上的点（0.80625,85）表示在多于 1000 组规范化平均资源利用率在（0.80125,0.81125）范围内的随机任务集样本中，可以被 MC-MP-EDF 算法调度的样本数量占总体样本数量的百分比为 85%。

实验结果表明本文提出的基于传统划分策略和 EY-VD 算法的混合关键性多处理划分调度算法 MC-PEDF 的接受率性能要明显优于之前的算法。而使用针对混合关键性系统的 OCOP 划分策略的改进性划分调度算法 MC-MP-EDF 的性能也明显的优于 MC-PEDF 算法。从图 4 与图 5 的对比中我们可以发现当混合关键性系统中处理器的数量从 4 增至 8 时，MC-MP-EDF 相对其它调度算法的优势更为突出。这也证明了 OCOP 划分策略针对多处理器平台中混合关键性系统的有效性和实用性。

4.5 小结

本文在基于传统多处理器划分调度策略实时调度算法的基础上，结合单处理器混合关键性系统中的 EY-VD 算法，提出了多处理器混合关键性系统中基于该算法的划分调度算法 MC-PEDF。并进一步研究了混合关键性系统中划分调度的特殊性质，并在分析传统划分策略局限性的基础上首次提出了针对多处理器平台的混合关键性系统划分策略 OCOP。相较于传统的多处理器划分策略，OCOP 能够有效利用混合关键性实时任务的运行时特征，更好地平衡不同处理器之间在各个运行时关键性模式中的资源利用率，以更为充分有效的利用系统资源。仿真实验的结果表明 MC-PEDF 相较于

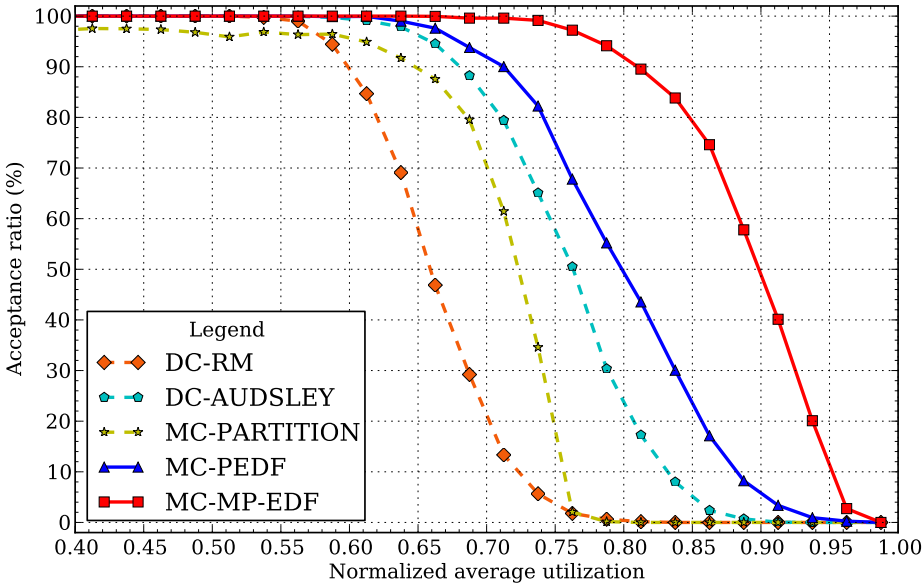


图 4.5　8 个处理器系统中的算法接受率比较

Fig. 4.5　Acceptance comparison on 8-processor platform

先前的多处理器混合关键性实时调度算法具有更高的可调度接受率；而基于 OCOP 划分策略的新算法 MC-MP-EDF 又显著提升了 MC-PEDF 的性能，并且系统中处理器的数量越多优化效果越明显。我们有理由相信 OCOP 将会成为多处理器混合关键性划分实时调度的重要分支。

第 5 章 实时任务有向图的近似响应时间分析

传统的实时任务系统模型通常被定义为由周期性重复执行的任务构成的集合[8, 9]。但是实际系统中存在很多并非完全是周期性重复计算的行为，而这些行为很难被这种简单的周期任务模型所精确描述。例如依赖于可变速率行为的内燃机燃料注射控制器系统[10]，和视频编解码器中基于帧的执行[11]。针对这些复杂结构的一种自然表示方法就是任务图。近些年来，很多基于图的任务模型被不断的提出，用来更为精确地描述复杂的嵌入式系统[7, 11–16] 。

然而对于大多数已有的实时任务模型（不包括简单的周期性模型），对于静态优先级调度的精确响应时间分析在时间复杂度上都是难于接受的。图5.1对一些常见的任务型的响应时间分复杂度进行了比较和总结，并展示了相互间的泛化关系[120]。在这些模型中只有非常简单的 L&L [8] 和 sporadic [9] 模型能够在伪多项式时间内计算出响应时间。而在模型中引入分支后相位调整后[17] ，问题便迅速变为强反 NP 难问题（strongly coNP-hard）。在文献 [18, 19] 中提出了很强的优化技术，能够减掉指数级状态空间中的很大一部分。但是，这些方法在一般情况下仍是指数级的复杂度，并且在处理大规模系统时依然可能面临状态空间爆炸的问题。

本章将研究通过近似的方法来分析基于图的实时任务模型的响应时间。由于实时任务有向图（DRT，Digraph Real-Time）模型[7] 相对于大部分已有的基于图的实时任务模型而言，更为一般化且描述能力更强，因此本章主要基于该模型进行讨论。

本章提出了两种近似响应时间分析方法 RBF 和 IBF，这两种方法均为伪多项式时间复杂度，并可以很高效地处理大规模的任务系统。

本章主要的理论贡献还在于通过加速比[137]（Speedup Factor）分析，对两种方法

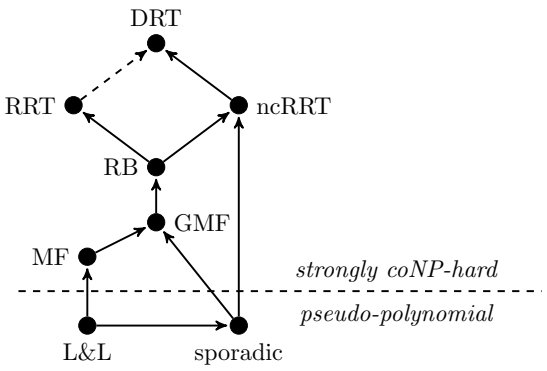


图 5.1 常见实时任务模型间的泛化关系

Fig. 5.1 Generalization relationship between different task models

进行了量化的性能评价。其中基于加速比的的性能评价被广泛应用于众多调度问题的近似算法分析中。本章的主要成果可总结如下：

- RBF 近似响应时间分析方法的精确加速比为 2（即便是仅包含两个任务的系统）。
- IBF 近似响应时间分析方法的加速比为 $1+\frac{\sqrt{k^2-k}}{k}$，其中 k 为干涉任务（优先级高于当前分析任务者）的数量。

因为当 $k=1$ 时有 $1+\frac{\sqrt{k^2-k}}{k}=1$，所以对于只有两个任务的系统 IBF 方法能够得到精确解。另外由于 $1+\frac{\sqrt{k^2-k}}{k}$ 为以 k 为自变量的单调递增函数，因此 IBF 方法的分析精度随着干涉任务数量的增加而降低。而当 k 趋于无穷时，IBF 的加速比也趋近于 2（与 RBF 一致）。

本章还通过模拟实验来分析了 RBF 与 IBF 方法的精确度与时间效率。实验结果表明，本章提出的近似分析方法（尤其是 IBF）的分析精度与指数复杂度的精确算法十分接近。并且当任务集合的大小越小时，IBF 方法的精度越高，这也与通过加速比的分析方法得出的结论一致。另一方面，本章提出两种近似分析方法的运行时效率也非常高。对于大规模的任务系统也仅需要几秒钟的执行时间，而采用精确分析方法时可能需要数小时。

5.1 系统模型与定义

5.1.1 模型定义

本小节将介绍本章研究的任务模型和一些基础概念。实时任务有向图（digraph real-time，DRT）模型[19] 描述了由 N 个相互独立的实时任务的集合 $\tau=\{T_1,T_2,\cdots,T_N\}$ 构成系统的工作量信息。其中每个任务 T 被描述为一个有向图 $G(T)$，该有向图由所有顶点构成的集合 $V(T)$ 和所有有向边的集合 $E(T)$ 组成。

顶点集合 $V(T)=\{v_1,v_2,\cdots,v_n\}$ 表示所有被任务 T 释放作业的可能类型。每个顶点 v 由一个二元组 $\langle e(v),d(V)\rangle$ 标注。其中 $e(v)\in\mathbb{N}$ 表示该顶点对应作业的最差执行时间（Worst Case Execution Time），$d(v)\in\mathbb{N}$ 表示作业的相对截止期。特别的，本章隐含地假定对于所有顶点 v 对应的作业类型，均满足 $e(v)\leq d(v)$。

有向图 $G(T)$ 的边表示由任务 T 释放的不同类型作业的释放次序。每条边 $(u,v)\in E(T)$ 均由 $p(u,v)\in\mathbb{N}$ 来标注。其中 $p(u,v)$ 表示 v 类型的后继作业与其前驱 u 类型作业之间的最小释放时间间隔。本章假定所有类型作业满足限定截止期（constrained deadline）的约束，即对于任意一个边 $(u,v)\in E(T)$，满足 $d(u)\leq p(u,v)$。

5.1.2 模型语义

每个实时任务 T 的实际运行时行为对应一个遍历有向图 $G(T)$ 的可能无限长的路径。在该路径上，每次经过一个顶点就会触发一次该类型作业的运行时释放（遵从该顶点上标注的参数）。经由该路径上每两个连续释放作业间的释放间隔服从对应边上标注的约束。本章用一个三元组 (r,e,d) 来形式化的描述一个运行时作业。其中，r 为作业的释放时间，e 为最差执行时间，d 为绝对截止期。

DRT 任务系统的语言被定义为其可能产生的作业序列 $\sigma=[(r_1,e_1,d_1),(r_2,e_2,d_2),\cdots]$。其中所有作业按其释放时间单调递增排列，即对于任意正整数 $i\le j$，满足 $r_i\le r_j$。一个作业序列 $\sigma=[(r_1,e_1,d_1),(r_2,e_2,d_2),\cdots]$ 是由实时任务 T 产生的充分必要条件为：

存在一条通过 $G(T)$ 的路径 $\pi=(v_1,v_2,\cdots)$，对于任意 $i>0$ 同时满足下列条件：

（1）$r_{i+1}-r_i\ge p(v_i,v_{i+1})$，

（2）$e_i\le e(v_i)$，

（3）$d_i=r_i+d(v_i)$。

一个作业序列 σ 是由实时任务集合 τ 中所有任务分别产生的作业序列归并组成，则该作业序列 σ 即为 τ 产生的。

本文结合图5.2中描述的任务实例来说明 DRT 任务系统的语义。当系统启动时，实时任务 T 从其顶点集合中选择任意一个作为第一个释放作业的类型。而接下来的释放作业序列必须与通过有向图 $G(T)$ 的一条具体直通路径相对应。考虑如下的作业序列 $\sigma=[(5,2,13,),(15,3,22),(25,3,31)]$，该作业序列对应的通过有向图 $G(T)$ 的路径为 $\pi=(v_5,v_2,v_4)$。

需要注意的是, 本实例演示的是符合 DRT 模型语言的偶发任务行为。作业序列 σ 中的第一个作业（对应路径 π 中第一个顶点 v_5）在时间为 5 时释放。σ 中第二个作业

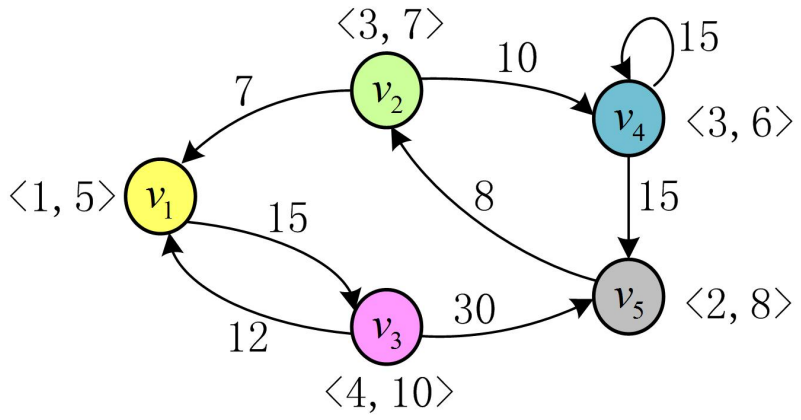


图 5.2 包含 5 种不同类型作业的 DRT 任务实例

Fig. 5.2 An example task containing five different jobs

（对应顶点 v_2）的释放时间为 15，比其理论上可能的最早释放时间晚了 2 个时间单位。而顶点 v_4 对应作业在 v_2 之后的释放则在其最早可能的释放时间 25。

5.1.3 静态优先级调度和最差响应时间

本章假设所有的运行时作业序列都执行在单处理器平台，并通过静态优先级（Static Priority）可抢占调度器进行调度。每个任务都在系统初始化时被分配一个预先设置的静态优先级。同一任务释放的所有作业都将继承该任务的优先级，并在运行时保持优先级固定不变。在运行时的任意时刻，系统都会在已经释放但尚未执行完成的作业中选择优先级最高的一个来占用处理器执行。

定义 5.1 **(最差响应时间)**：给定一个 DRT 实时任务集合 τ，其中的任一任务 T 对应的顶点集合为 $V(T)$。对于 τ 一个具体释放作业序列，$V(T)$ 中任一顶点 v 释放作业的释放时刻与完成时刻之间的时间长度定义为该作业的响应时间。而 v 的最差响应时间（Worst Case Response Time）被定义为在 τ 的所有可能的作业释放情况中，由 v 释放作业响应时间的最大值。

对于某一顶点 v，如果其最差响应时间不大于其相对截止期 $d(v)$，则该顶点是运行时可调度的。需要注意的是，任一顶点的最差响应时间都不会受到比其优先级更低作业释放工作量的影响。另外，假设某一任务的所有顶点都是可调度的，那么其中任一顶点的最差响应时间也不会受到属于同一任务其他顶点的影响（因为对于任意边 $(u,v) \in E(T)$，均满足 $d(u) \leq p(u,v)$）。因此，对于任一顶点 v 都可以独立地分析高优先级的任务集合 τ_{HI} 对它的干涉情况来计算最差响应时间。令 $R(v,\tau_{\mathrm{HI}})$ 表示顶点 v 在高优先级任务集合 τ_{HI} 干涉下的最差响应时间。

5.2 近似响应时间分析

本节将介绍两种近似响应时间分析方法：RBF（Request Bound Function）和 IBF（Interference Bound Function）。两者均为伪多项式的时间复杂度。在详细介绍这两种方法前，首先简要介绍一些基本概念以及指数复杂度的精确分析方法[125, 138]。

$G(T)$ 中的一条路径 π 产生的工作量随时间的变化关系可以被抽象描述为需求函数（Request Function）[18]。对于任意时刻 t，需求函数返回路径 π 在该时刻之前可能释放的所有作业所累积的最大执行需求（假设路径 π 中的第一个作业在 0 时刻释放）。

定义 5.2 **(需求函数)**：对于实时任务 T 对应的图 $G(T)$ 中的任意一条路径 $\pi = (v_0, v_1, \cdots, v_l)$，其需求函数被定义为

$$rf_{\pi}(t) \triangleq \max\{e(\pi') \mid \pi' \text{ is prefix of } \pi \text{ and } p(\pi') < t\} \tag{5.1}$$
$$\text{where } e(\pi) \triangleq \sum_{i=0}^{l} e(v_i) \text{ and } p(\pi) \triangleq \sum_{i=0}^{l-1} p(v_i, v_{i+1})$$

$rf_{\pi}(t)$ 是一个以 t 为自变量的非递减阶梯函数。每个水平分段的定义区间都是左开右闭的。特别的，$rf_{\pi}(0) = 0$。

令 $\Pi(T)$ 表示 $G(T)$ 中所有可能路径的集合，令 $\Pi(\tau) \triangleq \Pi(T_1) \times \Pi(T_2) \times \cdots \times \Pi(T_n)$ 表示任务集 $\tau = \{T_1, T_2, \cdots, T_n\}$ 中全体任务所有可能的路径组合。令 $\bar{\pi} = (\pi_1, \pi_2, \cdots, \pi_n)$ 表示 $\Pi(\tau)$ 中来自不同任务的路径构成的具体的一种路径组合。在分析顶点 v 在高优先级任务集合 τ_{HI} 之干涉作用下的最差响应时间时，令顶点 v 在路径组合 $\bar{\pi}$ 作用下的总需求函数（total request function）为：

$$rf_{(v,\bar{\pi})}(t) = e(v) + \sum_{\pi_i \in \bar{\pi}} rf_{\pi_i}(t) \tag{5.2}$$

需要注意的是，在式 (5.2)中，假设顶点 v 和各条高优先级路径的第一个顶点均同时在 0 时刻释放第一个作业。

根据定义5.1和式 (5.2)，顶点 v 在该特定高优先级任务路径组合 $\bar{\pi}$ 干涉下的响应时间可以被形式化描述为：

$$R(v, \bar{\pi}) = \min_{t>0} \left\{ t \mid rf_{(v,\bar{\pi})}(t) \le t \right\} \tag{5.3}$$

进一步地，顶点 v 在高优先级任务集合 π_{HI} 干涉下的总体最差响应时间即为顶点 v 在所有可能路径组合干涉下响应时间的最大值，可以被形式化描述为：

$$R(v, \tau_{\mathrm{HI}}) = \max_{\bar{\pi} \in \Pi(T)} \{R(v, \bar{\pi})\} \tag{5.4}$$

但是，$\Pi(T)$ 中路径组合的个数与任务集合 τ 的规模呈指数级增长，因此使用式 (5.4)来计算顶点 v 精确的最差响应时间的时间复杂度也是指数级的。造成路径组合规模指数爆炸的原因主要来自于两个方面：

（1）每个实时任务对应有向图生成的路径数量是随路径长度呈指数增长的；

（2）不同任务间的路径组合数量也是随任务数量呈指数增长的。

文献 [18] 提出了一种基于精炼策略的优化方法，在使用式 (5.4) 计算最差响应时间时，不需要检查所有的路径组合而是直接排除掉其中相当大的一部分。通过该方法，能够极大的提高计算最差响应时间的时间效率。但是一般而言，该精炼方法的时间复杂度仍然是指数级的，并会在处理大规模任务系统时产生可扩展性问题。事实上，计算该 DRT 模型中的精确最差响应时间问题是一个被证明的强 coNP 难问题（Strongly coNP-Hard）[17]。因此，除非 $P = NP$，否则（伪）多项式时间复杂度的方法

理论上是不存在的。

接下来，本章将介绍两种时间复杂度均为伪多项式级别的近似最差响应时间分析方法。这两种近似方法的主要思想都是为每个任务评估工作量时，采用伪多项式复杂度的抽象化方法来计算近似的安全值，已达到同时提高单个任务的工作量计算，以及不同任务间任务量叠加计算执行效率的目的。

5.2.1 RBF：需求上界函数分析方法

本小节首先介绍基于需求上界函数（Request Bound Function）的近似分析方法RBF。

定义 5.3 (需求上界函数)：对于一个实时任务 T，定义它的需求上界函数 $rbf_T(t)$ 为

$$rbf_T(t) \triangleq \max_{\pi \in \Pi(T)} \{rf_\pi(t)\} \tag{5.5}$$

对于一个顶点 v，在高优先级任务集合 τ_{HI} 干涉作用下的总体需求上界函数为

$$rbf_{(v,\tau_{\text{HI}})}(t) \triangleq e(v) + \sum_{T \in \tau_{\text{HI}}} \{rbf_T(t)\} \tag{5.6}$$

定理 5.1 (*RBF* 分析)：给定一个顶点 v 与高优先级干涉任务集合 τ_{HI}，则有 $R_{\text{RBF}}(v, \tau_{\text{HI}})$ 是顶点 v 的最差响应时间的安全上界：

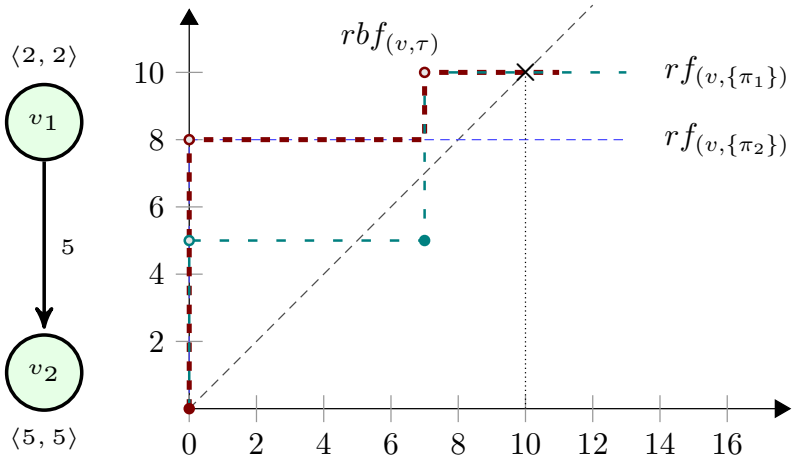
$$R(v, \tau_{\text{HI}}) \leq R_{\text{RBF}}(v, \tau_{\text{HI}}) \triangleq \min_{t>0}\{t \mid rbf_{(v,\tau_{\text{HI}})}(t) \leq t\} \tag{5.7}$$

证明：假设顶点 v 对应作业在高优先级任务集合 τ_{HI} 干涉作用下的最差响应时间 $R(v, \tau_{\text{HI}}) = R$。根据需求上界函数 $rbf_T(t)$ 的定义，可知任意的路径 $\pi \in \Pi(T)$ 都满足 $rbf_T(t) \geq rf_\pi(t)$。根据最差响应时间的定义（定义5.1），必然存在路径组合 $\bar{\pi} \in \Pi(\tau_{\text{HI}})$，满足

$$\begin{aligned}
R = R(v, \bar{\pi}) &= \min_{t>0} \left\{ t \;\middle|\; rf_{(v,\bar{\pi})}(t) \leq t \right\} \\
&= \min_{t>0} \left\{ t \mid e(v) + \sum_{\pi_i \in \bar{\pi}} rf_{\pi_i}(t) \leq t \right\} \\
&\leq \min_{t>0} \left\{ t \mid e(v) + \sum_{T \in \tau_{\text{HI}}} rbf_T(t) \leq t \right\} \\
&= \min_{t>0} \left\{ t \mid rbf_{(v,\tau_{\text{HI}})}(t) \leq t \right\} \\
&= R_{\text{RBF}}(v, \tau_{\text{HI}})
\end{aligned}$$

□

例 5.1：考虑如图5.3(a)所示仅包含一个任务的高优先级干涉任务集合，与被干涉



(a) T　　(b) $rbf_{(v,\tau_{\text{HI}})}(t)$

(a) T　　(b) $rbf_{(v,\tau_{\text{HI}})}(t)$

图 5.3 通过 $rbf(v,\tau_{\text{HI}})$ 计算 $R_{\text{RBF}}(v,\tau_{\text{HI}})$ 实例

Fig. 5.3 Illstration of coputing $R_{\text{RBF}}(v,\tau_{\text{HI}})$ by $rbf(v,\tau_{\text{HI}})$

顶点 v（$e(v)=3$）构成的例子。有向图 $G(T)$ 包含由边 (v_1,v_2) 连接的两个顶点 v_1 和 v_2 构成。其中，$e(v_1)=2$，$e(v_2)=5$，$p(v_1,v_2)=5$。通过 $G(T)$ 的路径共有两条，分别为 $\pi_1=(v_1,v_2)$，$\pi_2=(v_2)$。这两条路径各自的需求函数 $rf_{(v,\pi_1)}(t)$ 与 $rf_{(v,\pi_2)}(t)$ 与对角线分别相交于 $t=5$ 和 $t=8$。因此，顶点 v 的最差响应时间为 $R(v,\tau_{\text{HI}})=8$。但是，顶点 v 的需求上界函数 $rbf_{(v,\tau_{\text{HI}})}(t)$ 与对角线相交于 $t=10$，因此 $R_{\text{RBF}}(v,\tau_{\text{HI}})=10>8=R(v,\tau_{\text{HI}})$。

为了计算高优先级干涉任务集 τ_{HI} 的近似总体计算时间需求，在式 (5.7) 中，只是将不同任务的需求上界函数简单加和。这样便解决了因不同任务间路径组合导致的指数爆炸问题。而对于因单个任务的不同路径数量的指数爆炸问题，可以通过文献 [19] 中提出的路径抽象（Path Abstraction）技术来有效解决。

本章通过路径抽象技术来为每个任务 T_i 计算需求上界函数 $rbf_{T_i}(t)$ 的时间复杂度为 $O(t^2\cdot n)$，其中 n 为有向图 $G(T_i)$ 中顶点的数量。计算 $rbf_{T_i}(t)$ 算法的伪代码如图5.4所示。实际上对于顶点 v，为了判断其对应作业是否可调度，仅需要对不大于 $d(v)$ 的 t 计算 $rbf_{T_i}(t)$ 的值。因此，*RBF* 算法的总体时间复杂度仍然为伪多项式级别。一些文献 [19] 中的优化技术同样可以应用到图5.4所示算法中，来提高其执行的时间效率。需要注意的是，$rbf_{T_i}(t)$ 是左开右闭的阶梯函数，而通过图5.4所示算法得到的是左闭右开的阶梯函数。但所得函数可以很容易地在多项式时间内转换为对于自变量 t 左开右闭的集体函数。

CalculateRBF(T,t)

1: $RF_0 \leftarrow \{\langle 0,0,v_i\rangle | v_i \text{ vertex of } G(T)\}$
2: **for** $k=1$ **to** t **do**
3: $\quad RF_k \leftarrow \emptyset$
4: $\quad$ **for each** $\langle e,r,u\rangle \in RF_{k-1}$ **do**
5: $\quad\quad$ **for each** edges(u,v) in $G(T)$ **do**
6: $\quad\quad\quad e' \leftarrow e+e(u)$
7: $\quad\quad\quad r' \leftarrow r+pp(u,v)$
8: $\quad\quad\quad$ **if** $r' \le t$ **then**
9: $\quad\quad\quad\quad RF_k \leftarrow RF_k \cup \{\langle e',r',v\rangle\}$
10: $\quad\quad\quad$ **end if**
11: $\quad\quad$ **end for**
12: $\quad$ **end for**
13: **end for**
14: **return** $\max\{e+e(v) | \langle e,r,v\rangle \in \bigcup_{k\le t} RF_k\}$

图 5.4 计算 $rbf_T(t)$ 的算法描述

Fig. 5.4 Algorithm for computing $rbf_T(t)$

5.2.2 IBF：干涉上界函数分析方法

本章介绍的第二种近似分析方法 *IBF* 使用更为精确的干涉工作量上界抽象函数来描述每个任务的干涉工作量。本节首先介绍对于单独任务路径的干涉函数，和不同任务路径组合的干涉函数定义。

定义 5.4 (干涉函数)：给定 DRT 实时任务 T，对于通过其对应有向图 $G(T)$ 的任意一条路径 π，定义该路径的干涉函数（Interference Function）为

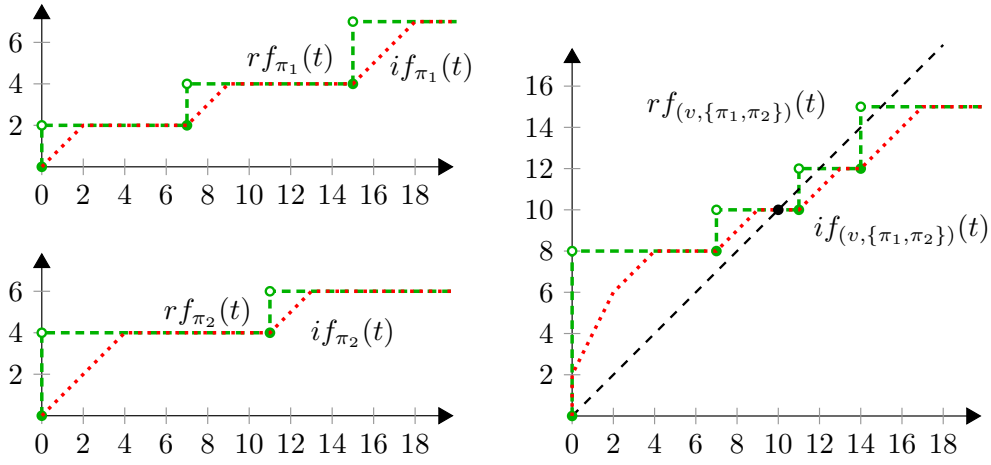
$$itf_\pi(t) \triangleq \max\{ee(\pi') \mid \pi' \text{ is prefix of } \pi \text{ and } p(\pi') < t\}$$

$$\text{where } p(\pi) \triangleq \sum_{i=0}^{l-1} p(v_i,v_{i+1}) \text{ and } ee(\pi) \triangleq \sum_{i=0}^{l-1} + \min(e(v_i), \max(0, t-p(\pi)))$$

在分析顶点 v 在高优先级任务集合 τ_{HI} 之干涉作用下的最差响应时间时，令顶点 v 在路径组合 $\bar{\pi}$ 作用下的总干涉函数（total interference function）为：

$$itf_{(v,\bar{\pi})}(t) = e(v) + \sum_{\pi_i \in \bar{\pi}} itf_{\pi_i}(t) \tag{5.8}$$

单条路径的干涉函数 $itf_\pi(t)$ 为倾斜的阶梯函数。干涉函数 $itf_\pi(t)$ 中每个分段的斜率取值为 0 或 1。不同任务路径组合的总干涉函数 $itf_{(v,\bar{\pi})}(t)$ 同样为倾斜的阶梯函数。但是对于 $itf_{(v,\bar{\pi})}(t)$ 中由多个倾斜的 $itf_\pi(t)$ 分段加和而成的分段，其斜率可能大于 1。图5.5(a)展示了两条路径 π_1 与 π_2 各自的干涉函数与需求函数，图5.5(b)展示了高优先级路径组合 (π_1,π_2) 和顶点 v （$e(v)=2$）的总干涉函数与总需求函数。



(a) 路径 π_1 和 π_2 分别对应的 *rf* 和 *itf*　　(b) 元组 (v, {π_1 , π_2}) 对应的 *rf* 和 *itf*

(a) *rf* and *itf* of π_1 and π_2　　(b) *rf* and *itf* of (v, {π_1 , π_2})

图 5.5 函数 *rf* 与 *itf* 的不同之处

Fig. 5.5 Illustration of the difference between *rf* and *itf*

事实上，总干涉函数 $itf_{(v,\bar{\pi})}$ 与总需求函数 $rf_{(v,\bar{\pi})}$ 在用于计算精确响应时间时，如果采用枚举所有路径组合的计算方法，则两者是等效的。对于干涉函数存在如下定理：

定理 5.2：顶点 v 在该特定高优先级任务路径组合 $\bar{\pi}$ 干涉下的精确响应时间可以通过下面等式计算：

$$R(v,\bar{\pi}) = \min_{t>0}\left\{t \mid itf_{(v,\bar{\pi})}(t) \le t\right\} \tag{5.9}$$

进一步地，顶点 v 在高优先级任务集合 π_{HI} 干涉下的总体最差响应时间可以通过下面等式计算：

$$R(v,\tau_{\mathrm{HI}}) = \max_{\bar{\pi}\in\Pi(\tau_{\mathrm{HI}})}\{R(v,\bar{\pi})\} \tag{5.10}$$

对比式 (5.3)的右边部分与式 (5.9)的右边部分可知，定理5.2正确性的一个充分条件为：对于任意的高优先级任务路径组合 $\bar{\pi}$，满足式 (5.3)与式 (5.9)各自右边部分的取值是相等的。通过检查函数 $itf_{(v,\bar{\pi})}(t)$ 和函数 $rf_{(v,\bar{\pi})}(t)$ 的定义易知，函数 $itf_{(v,\bar{\pi})}(t)$ 的所有水平分段都是与函数 $rf_{(v,\bar{\pi})}(t)$ 重叠的。另外，如图5.5所示，因为函数 $itf_{(v,\bar{\pi})}(t)$ 所有分段的斜率均为 0 或 1，所以函数 $itf_{(v,\bar{\pi})}(t)$ 和函数 $rf_{(v,\bar{\pi})}(t)$ 与对角线的交点一定位于函数 $itf_{(v,\bar{\pi})}(t)$ 的水平分段上。综上可知，函数 $itf_{(v,\bar{\pi})}(t)$ 和函数 $rf_{(v,\bar{\pi})}(t)$ 分别与对角线的交点都是相同的。因此由函数 $itf_{(v,\bar{\pi})}(t)$ 和函数 $rf_{(v,\bar{\pi})}(t)$ 分别计算得到的最差情况响应时间是相同的。该定理的形式化证明在此省略。

与需求函数的情况类似，枚举所有路径组合的干涉函数的计算复杂度同样是不可

接受的。接下来，本章将介绍干涉上界函数来基于该函数的近似分析方法。干涉上界函数的定义与需求上界函数类似，但是具有更好的分析精度。

定义 5.5 (干涉上界函数)：给定实时任务 T，定义其干涉上界函数（Interference Bound Function）$ibf_T(t)$ 为

$$ibf_T(t) \triangleq \max_{\pi \in \Pi(T)} \{itf_\pi(t)\} \tag{5.11}$$

对于顶点 v，在高优先级干涉任务集合 τ_{HI} 作用下的总干涉函数被定义为

$$ibf_{(v,\tau_{\text{HI}})}(T) \triangleq e(v) + \sum_{T \in \tau_{\text{HI}}} \{ibf_T(t)\} \tag{5.12}$$

定理 5.3 (IBF 分析方法)：给定顶点 v，和高优先级干涉任务集合 τ_{HI}，采用 IBF 方法计算的响应时间 $R_{\text{IBF}}(v,\tau_{\text{HI}})$ 是顶点 v 最差响应时间的安全上界：

$$R(v,\tau_{\text{HI}}) \le R_{\text{IBF}}(v,\tau_{\text{HI}}) \triangleq \min_{t>0}\{t \mid ibf_{(v,\tau_{\text{HI}})}(t) \le t\} \tag{5.13}$$

证明：根据定义5.5，对于任意实时任务 T_i 和于任意通过其对应有向图 $G(T)$ 的路径 $\pi \in \Pi(T)$，满足 $ibf_{T_i}(t) \ge itf_\pi(t)$。再根据 定理5.2 可知

$$\begin{aligned} R(v,\tau_{\text{HI}}) &= \max_{\bar{\pi} \in \Pi(T)} \{\min_{t>0}\{t \mid itf_{v,\bar{\pi}}(t) \le t\}\} \\ &\le \min_{t>0}\{t \mid \max_{\bar{\pi} \in \Pi(T)} \{itf_{v,\bar{\pi}}(t) \le t\}\} \\ &= \min_{t>0}\{t \mid ibf_{(v,\tau_{\text{HI}})}(t) \le t\} \\ &= R_{\text{IBF}}(v,\tau_{\text{HI}}) \end{aligned}$$

定理得证。 □

得到为每个任务T 计算$ibf_{(v,T)}(t)$ 的伪多项式时间复杂度算法，只需将图5.4中所列算法的最后一行修改为：

$$\max\{e + e(v) \mid \langle e,r,v\rangle \in \bigcup_{k \le t} RF_k\}。$$

事实上，*IBF* 分析方法严格优于*RBF* 方法，形式化表述为$IBF \succ RBF$。接下来将给出该结论的证明。

定理 5.4 ($IBF \succ RBF$)：对于任意任务顶点v，满足$R_{\text{IBF}}(v,\tau_{\text{HI}}) \le R_{\text{RBF}}(v,\tau_{\text{HI}})$，并且存在特定的顶点$v$ 使得$R_{\text{IBF}}(v,\tau_{\text{HI}}) < R_{\text{RBF}}(v,\tau_{\text{HI}})$。

证明：*itf* 是相较于*rf* 更为精确的路径工作量抽象，对于任意路径π 和时间$t > 0$，满足$itf_\pi(t) \le rf_\pi(t)$。因此对于任意任务T 和时间t，满足$ibf_T(t) \le rbf_T(t)$。再根据 定义5.3和 定义5.5可知$R_{\text{IBF}}(v,\tau_{\text{HI}}) \le R_{\text{RBF}}(v,\tau_{\text{HI}})$。

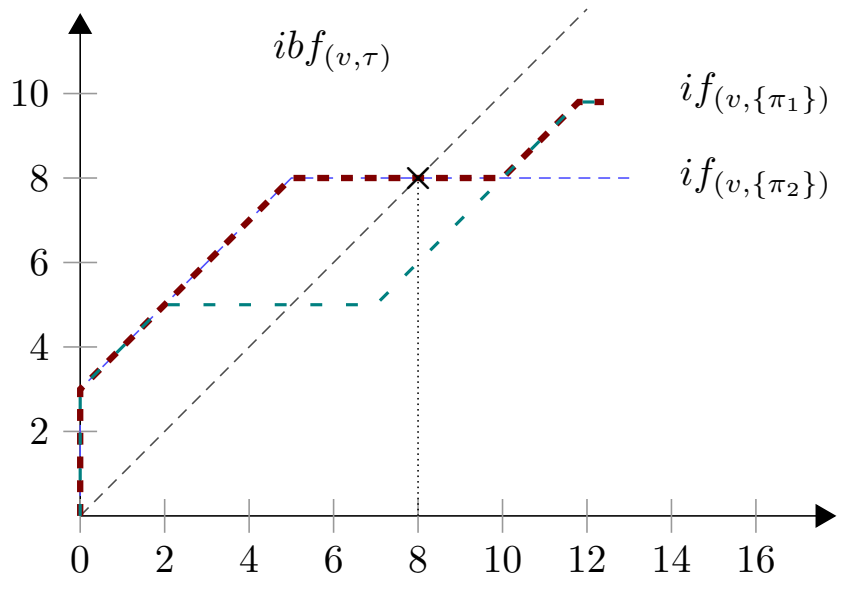


图 5.6 函数

Fig. 5.6 Illustration of $ibf_{(v,\tau_{\mathrm{HI}})}(t)$

针对图5.6中给出实例，$R_{\mathrm{IBF}}(v,\tau_{\mathrm{HI}})=8$ 小于$R_{\mathrm{RBF}}(v,\tau_{\mathrm{HI}})=10$。由此，定理得证。 □

5.2.3 一些性质

令*RF* 与*ITF* 分别表示使用需求函数*rf* 和干涉函数*itf* 进行精确响应时间分析的方法。那么，存在得到如下的关系：

$$\begin{array}{ccc} ITF & \succ & IBF \\ \| & & \curlyvee \\ RF & \succ & RBF \end{array}$$

其中，符合$\succ$ 表示不同方法之间的严格优于关系。

在本小节将介绍一些重要的辅助引理，为后面小节分析*RBF* 与*IBF* 的加速比提供支持。

首先，需要阐述的是需求上界函数 $rbf_{(v,\tau_{\mathrm{HI}})}(t)$ 与干涉上界函数 $ibf_{(v,\tau_{\mathrm{HI}})}(t)$ 对于任意特定的时间t，针对不同任务路径组合在长度为t 的时间间隔内最大累计工作量（干涉）的计算是精确的，并不会导入任何的悲观估计。该条性质的证明如下面引理所示。

引理 5.5：给定任务顶点v 和高优先级干涉任务集合τ_{HI}, 对于任意$t\geq 0$ 满足：

$$rbf_{(v,\tau_{\mathrm{HI}})}(t)=\max_{\bar{\pi}\in\Pi(\tau_{\mathrm{HI}})}\{rf_{(v,\bar{\pi})}(t)\} \tag{5.14}$$

$$ibf_{(v,\tau_{\mathrm{HI}})}(t)=\max_{\bar{\pi}\in\Pi(\tau_{\mathrm{HI}})}\{itf_{(v,\bar{\pi})}(t)\} \tag{5.15}$$

证明：首先证明式 (5.14)的正确性。根据定义5.3和式 (5.2)易知，式 (5.14)的左边

部分大于等于右边部分。接下来只需证明式 (5.14)左边部分小于等于右边部分同样成立。由式 (5.6)可推出：

$$\begin{aligned} rbf_{(v,\tau_{\mathrm{HI}})}(t) &= rbf_{T_1} + rbf_{T_2} + \cdots + rbf_{T_n} + e(v) \\ &= \max_{\pi\in\Pi(T_1)}\{rf_{\pi}(t)\} + \max_{\pi\in\Pi(T_2)} + \cdots + \max_{\pi\in\Pi(T_n)} + e(v) \end{aligned}$$

对于特定时间t，令 π_i 表示路径集合$\Pi(T_i)$ 中在长度为t 的时间内取得最大累积工作量的路径，即$\pi_i \in \Pi(T_i)$ 满足$\forall \pi \in \Pi(T_i) : rf_{\pi_i}(t) \geq rf_{\pi}(t)$。进一步可以推出：

$$\begin{aligned} rbf_{(v,\tau_{\mathrm{HI}})}(t) &= rf_{\pi_1} + rf_{\pi_2} + \cdots + rf_{\pi_n} + e(v) \\ &= rf_{(v,\{\pi_1,\pi_2,\cdots,\pi_n\})}(t) \\ &\leq \max_{\pi\in\Pi(\tau_{\mathrm{HI}})}\{rf_{(v,\bar{\pi})}(t)\} \end{aligned}$$

综上，式 (5.14)的正确性得证。同理，可以证明式 (5.15)的正确性。 □

定义 5.6 (**上升点**)：对于（倾斜）阶梯函数$f(t)$ 上的点$z \geq 0$，如果z 为上升点则必须满足如下的充分必要条件：

$$f(z) = f(z-\varepsilon) \wedge f(z) < f(z+\varepsilon)$$

其中，ε 为一个任意接近于0 的正实数。

直观上的，一个上升点可以看做是（倾斜）阶梯函数图形中一个水平分段右侧结束端的x 坐标轴的取值。如图5.7(a)所示，z_1 同为阶梯函数$rf_{(v,\bar{\pi})}(t)$ 和倾斜阶梯函数 $itf_{(v,\bar{\pi})}(t)$ 的上升点。

引理 5.6：给定一个顶点v 和高优先级干涉任务集合τ_{HI}。令 π 为路径组合$\Pi(\tau_{\mathrm{HI}})$ 中的任意一条路径。令$f(t)$ 表示总需求函数 $rf_{(v,\bar{\pi})}(t)$ 或总干涉函数 $itf_{(v,\bar{\pi})}(t)$。假设$f(x) > x$，并且定义

$$z_2 \triangleq \min_{t>x}\{t \mid f(t) \leq t\}$$

$$z_1 \triangleq \max_{t<x}\{t \mid f(t) \leq t \wedge t \text{ 是} f(x) \text{ 的一个上升点}\}$$

则满足

$$f(z_2) - f(z_1) \leq R(v,\tau_{\mathrm{HI}})$$

为了证明引理5.6，首先引入如下辅助引理。

引理 5.7：给定一个顶点 v 和高优先级干涉任务集合 τ_{HI}。假设在时间区间 $[t_1,t_2)$ 内处理器处于持续工作的忙碌状态（执行由顶点 v 或 τ_{HI} 内任务释放的作业），那么在 $[t_1,t_2)$ 内释放作业的所有工作量之和不大于 $R(v,\tau_{\mathrm{HI}})$ 。

证明：本引理通过反证法来证明。令 W 表示在区间 $[t_1,t_2)$ 内释放所有作业的累积工作量之和，并假设 $W > R$。不失一般性地，考虑 v 在 t_1 时刻释放了一个作业。因为处理器在区间 $[t_1,t_2)$ 一直处于忙碌状态，因此该区间内释放作业的工作量不能在 t_1+W 之前执行完毕。又因为 v 释放的作业为优先级最低的作业，因此该作业不能在 t_1+W 之前执行完毕。这与 R 为顶点 v 的最差响应时间的定义相矛盾。引理得证。 □

接下来将对引理5.6进行证明。

证明：引理5.6的证明可对 $f(t)=itf_{(v,\bar{\pi})}(t)$ 和 $f(t)=rf_{(v,\bar{\pi})}(t)$ 两种情况分别讨论。

首先讨论 $f(t)=itf_{(v,\bar{\pi})}(t)$ 时的情况。假设每条路径 $\pi_i \in \bar{\pi}$ 释放其第一个作业均在 0 时刻。因为 z_1 是函数 $itf_{(v,\bar{\pi})}(t)$ 上的一个上升点，所以 $itf_{(v,\bar{\pi})}(z_1)$ 表示在时间区间 $[0,z_1)$ 内释放的所有工作量之和。又因为 $itf_{(v,\bar{\pi})}(z_1) \le z_1$，所以可推出在区间 $[0,z_1)$ 内释放的所有工作量在 z_1 时刻之前全部执行完毕，即处理器在 z_1 时刻为空闲状态。另一方面由于 z_1 为上升点，因此在 z_1 时刻会有新作业的释放。由此可推出，处理器在 z_1 时刻油空闲变为忙碌状态。

接下来讨论如下命题正确性：处理器在时间区间 $[z_1,z_2)$ 内为持续忙碌状态。该命题通过反证法来证明。假设存在时刻 $z' \in (z_1,z_2)$ 满足处理器在该时刻空闲。由此可知所有在 z' 时刻前释放的作业均在 z' 时刻之前执行完毕，所以 z' 一定属于函数 $itf_{(v,\bar{\pi})}(t)$ 的某一水平分段并且 $itf_{(v,\bar{\pi})}(z') \le z'$。令 z'' 表示 z' 所述水平分段末尾的上升点，可知

$$itf_{(v,\bar{\pi})}(z'') \le itf_{(v,\bar{\pi})}(z') = z' \le z''$$

因此 z'' 是一个小于 z_2 并大于 z_1 且满足 $itf_{(v,\bar{\pi})}(t) \le t$ 的上升点，这与 z_1 的定义是相矛盾的。该矛盾证明了处理器在时间区间 $[z_1,z_2)$ 内是忙碌的。

根据引理5.7可推出，在时间区间 $[z_1,z_2)$ 内释放作业的累积工作量之和不大于 $R(v,\tau_{\text{HI}})$。又因为 z_1 为上升点，可推出 $f(z_2)-f(z_1)$ 仅包含在区间 $[z_1,z_2]$ 内的释放的工作量。综上可推出 $f(z_2)-f(z_1) \le R(v,\tau_{\text{HI}})$。

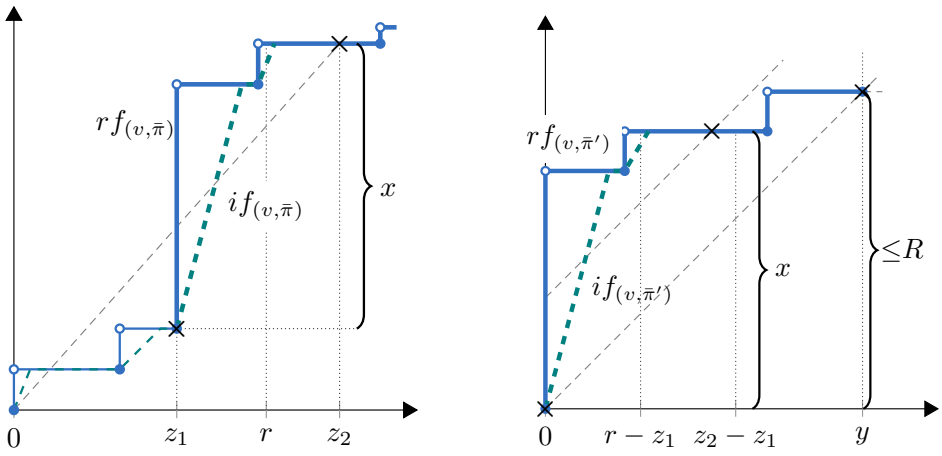
同理，可证明当 $f(t)=rf_{(v,\bar{\pi})}(t)$ 时的情况。综上，引理得证。 □

5.3 加速比分析

本小节采用加速比指标[15] 来评价近似响应时间分析方法*RBF* 和*IBF* 的性能。

定义 5.7 (加速比)：对于单位速率单处理器平台中任意 DRT 系统 τ 内的任意顶点v, 假定其精确的最差响应时间为R。如果给定的方法M 在s 倍速率单处理器平台中计算得到的最差响应最差响应时间总是满足不大于R，则定义方法M 的加速比为s。

特别的，如果对于任意小于s 的加速比s' 总能找到特定的 DRT 系统 τ 中的摸个顶



(a) $rf_{(v,\bar{\pi})}(t)$ 和 $itf_{(v,\bar{\pi})}(t)$　　(b) $rf_{(v,\bar{\pi}')(t)}$ 和 $itf_{(v,\bar{\pi}')(t)}$

(a) $rf_{(v,\bar{\pi})}(t)$ and $itf_{(v,\bar{\pi})}(t)$　　(b) $rf_{(v,\bar{\pi}')(t)}$ and $itf_{(v,\bar{\pi}')(t)}$

图 5.7 引理5.6原理示意图

Fig. 5.7 Intuition of Lemma 5.6

点v，而使用方法M在s'倍速率处理器平台求得v的最差响应时间大于其在单位速率处理器上的精确最差响应时间，则加速比s是精确的。

在单位速率处理器平台上，如有一个作业需要执行$e(v)$时间，那么它在s倍速率处理器平台上完成同样的工作量只需要$e(v)/s$的时间。因此在s倍速率处理器平台上的需求函数和干涉函数可以被重新定义为：

$$rf_{\pi}^{s}(t) \triangleq \max\{e^{s}(\pi') \mid \pi'\text{为}\pi\text{的前缀，并且}p(\pi') < t\}$$

$$itf_{\pi}^{s}(t) \triangleq \max\{ee^{s}(\pi') \mid \pi'\text{为}\pi\text{的前缀，并且}p(\pi') < t\}$$

$$\text{其中}p(\pi) \triangleq \sum_{i=0}^{l-1} p(v_i, v_{i+1})$$

$$e^{s}(\pi) \triangleq \sum_{i=0}^{l} e(v_i)/s$$

$$ee^{s}(\pi) \triangleq \sum_{i=0}^{l} e(v_i)/s + \min(e(v_l)/s, \max(0, t - p(\pi)))$$

通过类似的方法，能够得到新的函数$rf_{(v,\bar{\pi})}^{s}(t)$，$rbf_{T}^{s}(T)$，$rbf_{(v,\tau_{\mathrm{HI}})}^{s}(t)$，$itf_{(v,\bar{\pi})}^{s}(t)$，$ibf_{T}^{s}(t)$，$ibf_{(v,\tau_{\mathrm{HI}})}^{s}(t)$的定义。为了简化描述，当$s=1$时将上述各函数中的上角标$s$省略掉。

通过简单分析可以得到如下的关系：

$$rf^{s}_{\pi}(t) = rf_{\pi}(t)/s$$

$$rf^{s}_{(v,\bar{\pi})}(t) = rf_{(v,\bar{\pi})}(t)/s$$

$$rbf^{s}_{T}(t) = rbf_{T}(t)/s$$

$$rbf^{s}_{(v,\tau_{\mathrm{HI}})}(t) = rbf_{(v,\tau_{\mathrm{HI}})}(t)/s$$

但是相对应的关系对于干涉（上界）族的函数却并不总是成立。在单位速率处理平台中，该族函数仅在其图形中的水平分段区间上满足上述的关系。

引理 5.8：给定顶点v与特定高优先级干涉路径组合$\bar{\pi}$总干涉函数$itf_{(v,\bar{\pi})}(t)$，对于任意不可微分或者满足$\mathbf{d}itf_{(v,\bar{\pi})}(x)/\mathbf{d}(x) = 0$的点$x$（在函数$itf_{(v,\bar{\pi})}(t)$所有水平分段区间内的点），均满足如下关系：

$$itf^{s}_{(v,\bar{\pi})}(x) = itf_{(v,\bar{\pi})}(x)/s \tag{5.16}$$

通过分析函数$itf^{s}_{(v,\bar{\pi})}(t)$的定义，引理5.8可以很容易地证明。这里省略详细的证明。需要注意的是，类似的结论对于函数 $itf^{s}_{\pi}(t)$，$ibf^{s}_{T}(t)$，$ibf^{s}_{(v,\tau_{\mathrm{HI}})}(t)$ 同样成立。如图5.9中所示，在区间$[4,14]\cup[16,18]$ 内的点均满足式 (5.16)的关系。

5.3.1 *RBF* 方法的加速比

本小节分析 *RBF* 具有精确的加速比2。下面将通过两个定理分别证明其正确性和精确性。

定理 5.9：在分析顶点v在高优先级干涉任务结合τ_{HI}作用下最差响应时间时，*RBF* 方法的加速比为2。

证明：令$R = R(v,\tau_{\mathrm{HI}})$。该定理需要证明如果顶点$v$在单位速率处理器平台的最差响应时间为$R$，则该顶点在2 被速率处理器平台下使用 *RBF* 方法计算的最差响应时间一定小于等于R。首先讨论如下命题，对于$\forall\bar{\pi}\in\Pi(\tau_{\mathrm{HI}})$ 均满足

$$rf^{2}_{(v,\bar{\pi})}(R) \le R \tag{5.17}$$

如果式 (5.17)成立，则有

$$\max_{\bar{\pi}\in\Pi(\tau_{\mathrm{HI}})}\{rf^{2}_{(v,\bar{\pi})}(R) \le R\},$$

再根据引理5.5可推出$rbf^{2}_{(v,\tau_{\mathrm{HI}})}(R) \le R$, 则引理得证。当$rf_{(v,\bar{\pi})}(R) \le R$ 时，式 (5.17)显然是成立的。接下来，主要讨论当$rf_{(v,\bar{\pi})}(R) > R$ 的情况。令：

$$z_2 \triangleq \min_{t>R}\{t \mid rf_{(v,\bar{\pi})}(t) \le t\}$$

$$z_1 \triangleq \max_{t<R}\{t \mid rf_{(v,\bar{\pi})}(t) \le t \bigwedge t\text{为}rf_{(v,\bar{\pi})}(t)\text{的上升点}\}$$

（参考图5.7(a)中，对于当$x=R$时z_1和z_2定义的描述。）

从而，根据引理5.6可以推出

$$rf_{(v,\bar{\pi})}(z_2)-rf_{(v,\bar{\pi})}(z_1)\le R$$

又因为$rf_{(v,\bar{\pi})}(z_1)\le z_1<R$，可推出

$$rf_{(v,\bar{\pi})}(z_2)\le 2\cdot R \tag{5.18}$$

因为$R<z_2$，可以推出$rf_{(v,\bar{\pi})}(R)\le rf_{(v,\bar{\pi})}(z_2)$，结合式 (5.18)可知

$$rf_{(v,\bar{\pi})}(R)\le 2\cdot R$$

将已知关系$rf^2_{(v,\bar{\pi})}(R)=rf_{(v,\bar{\pi})}(R)/2$带入上边不等式，可得$rf^2_{(v,\bar{\pi})}(R)\le R$。

综上，定理得证。 □

定理 5.10：*RBF 分析方法的加速比不可能小于2，即便是只包含两个任务的系统。*

证明：证明该定理可通过证明如下充分条件：对于任意 $\varepsilon\in(0,1)$ 和对应的$s=2-\varepsilon$，存在特定的包含两个任务的 DRT 系统$\tau=\{T_1,T_2\}$，其中一个顶点v在单位速率处理器平台上的精确最差响应时间为r，但是v在s倍速率处理器平台上使用 *RBF* 方法计算的最差响应时间大于r。可以通过下面的方法来构造这样的系统：

- 令$r=\lceil 2/\varepsilon\rceil$，和$k=\lceil r/2\rceil$；
- 构造高优先级任务T_1，令其包含$k+1$个顶点$=\{v_1,v_2,\cdots,v_n\}$。对于每个顶点v_i，设置$e(v_i)=r-(i+1)$，$d(v_i)=r-i$。对于每个$i\in[1,k]$，添加边(v_i,v_0)来连接顶点v_i与v_0，并设置$p(v_i,v_0)=r-i$；
- 构造只包含一个顶点v的低优先级任务T_2，设置$e(v)=1$和$d(v)=r$。

通过该方法构造的 DRT 任务集合如图5.8所示。其中对于任务T_1的所有顶点均只标注了其最差执行时间，而省略了相对截止期。根据τ的设置很容易验证T_1的每个顶点都是可调度的，而且其顶点相对截止期的设置对于T_2中顶点v的响应时间分析没有任何影响。

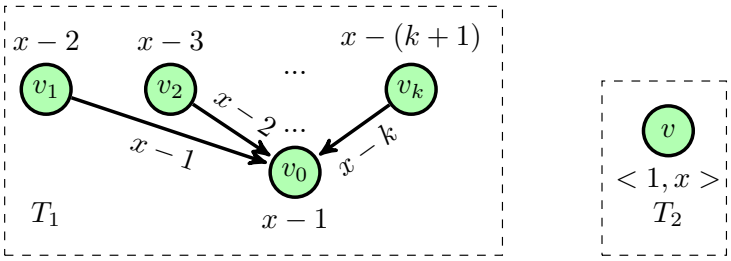


图 5.8 证明 RBF 精确加速比为 2 的任务集合实例

Fig. 5.8 A task set of two tasks T_1 and T_2 showing the speedup factor 2 of RBF is tight.

令π_i 表示通过有向图$G(T_1)$，且起始顶点为v_1 的路径。容易验证在单位速率处理器平台中，顶点$v \in T_1$ 在高优先级路径$\pi_i \in G(T_1)$ 干涉下的精确最差响应时间是$r-1$。接下来讨论对于任意的时间点$t \in [0,r]$ 满足$rbf^s_{(v,T_1)}(t) > t$。如果上述命题成立，则可推出在s 倍速率处理器平台中使用 *RBF* 方法分析顶点v 的最差响应时间时，结果将严格大于r，即定理得证。现在分别讨论$t \in [0,r/2]$ 和$t \in (r/2,r]$ 两种情况。

（1）$t \in [0,r/2]$。在这种情况下，可知$rf_{(v,\pi_0)}(t) = (r-1)+1 = r$，又因为$t \le r/2$，因此可以推出$rf_{v,\pi_0}(t) > s \cdot t$。

（2）$t \in (r/2,r]$。在这种情况下，容易验证必然存在$i \in [1,k]$ 满足$t \in (r-i, r-i+1)$。考虑路径π_i，由$t > r-i$ 可推出：

$$\begin{aligned} rf_{v,\pi_i}(t) &= e(v_i) + e(v_0) + 1 \\ &= 2 \cdot r - i - 1 \\ &= 2(r-i+1) + (i-1) - 2 \\ &> 2 \cdot (r-i+1) + (i-1) - 2 \\ &= (2-\varepsilon) \cdot (r-i+1) \\ &= s \cdot (r-i+1) \\ &\ge s \cdot t \end{aligned}$$

综上分析，在两种情况下总是存在$\pi \in \Pi(T_i)$ 满足$rf_{(v,\pi)}(t) > s \cdot t$。也即是说，可以推出：

$$\begin{aligned} rbf^s_{(v,T_i)}(t) &= rbf_{(v,T_1)}(t)/s \\ &\ge rf_{(v,\pi)}(t)/s \end{aligned}$$

因此，对于任意$t \in [0,r]$, 满足$rbf^s_{(v,T_1)}(t) > t$。进而可以推出当$s < 2$ 时，对于图5.8中的任务集合，在s 倍速率处理器平台上使用 *RBF* 方法计算的顶点v 的最差响应时间严格大于在单位速率处理器平台上的精确最差响应时间r。定理得证。　□

5.3.2 *IBF* 方法的加速比

首先需要说明的是，函数$itf^s_{(v,\bar{\pi})}(t)$ 是 Lipschitz 连续函数[17]，并且 Lipschitz 常数等于高优先级干涉任务的数量。

引理 5.11：给定顶点v 和高优先级干涉任务集合τ_{HI}，其中τ_{HI} 包含k 个任务并且执行在s 倍速率处理器平台上。令 $\bar{\pi}$ 为路径组合集合$\Pi(\tau_{\mathrm{HI}})$ 中的任意一组路径组合，则对于任意$x \ge y \ge 0$ 均满足

$$itf^{s}_{(v,\bar{\pi})}(x) - itf^{s}_{(v,\bar{\pi})}(y) \le k \cdot (x-y) \tag{5.19}$$

证明：根据函数 $itf^{s}_{(v,\bar{\pi})}$ 的定义可知

$$itf^{s}_{(v,\bar{\pi})}(x) - itf^{s}_{(v,\bar{\pi})}(y) = \sum_{\pi_i \bar{\pi}} (itf^{s}_{\pi_i}(x) - itf^{s}_{\pi_i}(y)) \tag{5.20}$$

根据函数 $itf^{s}_{\pi}(t)$ 的定义可知

$$itf^{s}_{\pi_i}(x) - itf^{s}_{\pi_i}(y) \le x - y$$

也即是说，函数$itf^{s}_{\pi_i}(t)$ 中所有倾斜分段的斜率最大为1，并且因为高优先级干涉任务集τ_{HI} 中包含k 个任务，由此可推出：

$$\sum_{\pi \in \bar{\pi}} (itf^{s}_{\pi_i}(x) - itf^{s}_{\pi_i}(y)) \le k \cdot (x-y)$$

再根据式 (5.20)该引理得证。 □

需要注意的是，当提升系统处理器的速率s 时，函数$itf^{s}_{(v,\bar{\pi})}(t)$ 的 Lipschitz 常数并不会增加，而只是会减小倾斜分段的长度。如图5.9所示的实例描述了该性质。

定理 5.12：对于任意顶点v，使用 *IBF* 方法分析其在包含k 个高优先级任务的干涉任务集合τ_{HI} 作用下的最差响应时间时，加速比为

$$1 + \frac{\sqrt{k^2 - k}}{k}$$

证明：令$R = R(v, \tau_{\mathrm{HI}})$。本定理需要证明如果在单位速率处理器平台中顶点$v$ 的精确最差响应时间为R，那么对于$s \ge 1 + \frac{\sqrt{k^2-k}}{k}$，在$s$ 倍速率处理器平台中使用 *IBF* 方法计算的最差响应时间不大于R。接下来，首先分析命题：对于任意高优先级路径组合$\bar{\pi} \in \pi(\tau_{\mathrm{HI}})$，满足

$$itf^{s}_{(v,\bar{\pi})}(R) \le R \tag{5.21}$$

如果式 (5.21)成立，则可推出

$$\max_{\bar{\pi} \in \pi(\tau_{\mathrm{HI}})} \{itf^{s}_{(v,\bar{\pi})}(R)\} \le R$$

在根据引理5.5可推出$ibf^{s}_{(v,\tau_{\mathrm{HI}})}(R) \le R$，即该定理得证。

当$itf_{(v,\bar{\pi})}(R) \le R$ 时，式 (5.21)显然是成立的。接下来，将集中讨论当$itf_{(v,\bar{\pi})}(R) > R$ 时的情况。令：

$$z_2 \triangleq \min_{t>R} \{t \mid itf_{(v,\bar{\pi})}(t) \le t\}$$

$$z_1 \triangleq \max_{t<R} \{t \mid itf_{(v,\bar{\pi})}(t) \le t \bigwedge t\text{为}itf_{(v,\bar{\pi})}(t)\text{的上升点}\}$$

（参考图5.7(a)中，对于当$x = R$ 时z_1 和z_2 定义的描述。）

根据引理5.6可推出

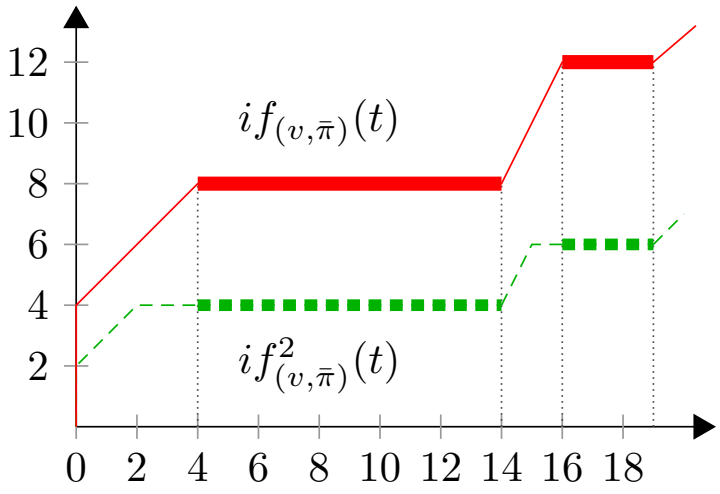


图 5.9 $itf^2_{(v,\bar{\pi})}(t)$ 和 $itf_{(v,\bar{\pi})}(t)$

Fig. 5.9 $itf^2_{(v,\bar{\pi})}(t)$ and $itf_{(v,\bar{\pi})}(t)$

$$itf_{(v,\bar{\pi})}(z_2) - itf_{(v,\bar{\pi})}(z_1) \leq R \tag{5.22}$$

令$x = itf_{(v,\bar{\pi})}(z_1)/R$，并分别讨论$x \geq \frac{k\cdot s-s}{k\cdot s-1}$ 和$x \leq \frac{k\cdot s-s}{k\cdot s-1}$ 两种情况：（1）$x \geq \frac{k\cdot s-s}{k\cdot s-1}$。根据引理5.11可推出

$$itf^s_{(v,\bar{\pi})}(R) - itf^s_{(v,\bar{\pi})}(z_1) \leq k\cdot(R-z_1) \tag{5.23}$$

根据z_1 的定义可知，z_1 是函数$itf_{(v,\bar{\pi})}(t)$ 水平分段上的点，因此根据引理5.8和$itf_{(v,\bar{\pi})}(z_1) \leq z_1$ 可推出

$$itf^s_{(v,\bar{\pi})}(z_1) = itf_{(v,\bar{\pi})}(z_1)/s \leq z_1/s \tag{5.24}$$

结合式 (5.23)和式 (5.24)可推出

$$itf^s_{(v,\bar{\pi})}(R) \leq z_1/s + k\cdot(R-z_1) \tag{5.25}$$

另外，

$$\begin{aligned} & x \geq (k\cdot s-s)/(k\cdot s-1) \\ \Rightarrow & x/s + k\cdot(1-x) - 1 \leq 0 \\ \Rightarrow & itf_{(v,\bar{\pi})}(z_1)/s + k\cdot(R - itf_{(v,\bar{\pi})}(z_1)) \leq R \\ \Rightarrow & k\cdot R - itf_{(v,\bar{\pi})}(z_1)\cdot(k-1/s) \end{aligned}$$

又因为$itf_{(v,\bar{\pi})}(z_1) \leq z_1$，$s>1$ 和$k \geq 1$ 可推出

$$k\cdot R - z_1\cdot(k-1/s) \leq R$$

将上式带入式 (5.24)，整理可得

$$itf^s_{(v,\bar{\pi})}(R) \leq R$$

（2）$x < \frac{k\cdot s-s}{k\cdot s-1}$。由命题假设可知：

$$s \geq 1+\frac{\sqrt{k^2-k}}{k}$$
$$\Rightarrow k\cdot s^2-2k\cdot s+1 \geq 0$$
$$\Rightarrow s \geq 1+\frac{k\cdot s-s}{k\cdot s-1}$$

与不等式$x < \frac{k\cdot s-s}{k\cdot s-1}$ 联立可推出

$$s \geq 1+x \Rightarrow R \geq (R+itf_{(v,\bar{\pi})}(z_1))/s$$

再根据式 (5.22)可推出$R \geq itf_{(v,\bar{\pi})}(z_2)/s$。另外，根据$z_2$ 的定义可知z_2 是函数$itf_{(v,\bar{\pi})}(t)$某一水平分段上的点，因此，根据引理5.8可推出$itf^s_{(v,\bar{\pi})}(z_2) = itf_{(v,\bar{\pi})}(z_2)/s$。综上，可得$itf^s_{(v,\bar{\pi})}(z_2) \leq R$, 又因为$R < z_2$ 可推出

$$itf^s_{(v,\bar{\pi})}(R) \leq R$$

综上分析，在两种情况下均满足$itf^s_{(v,\bar{\pi})}(R) \leq R$。因为$\bar{\pi}$ 是从所有路径组合集合中任意选择的一组组合，因此

$$\max_{\bar{\pi}\in\Pi(\tau_{\mathrm{HI}})}\{itf^s_{(v,\bar{\pi})}(R)\} \leq R。$$

根据引理5.5可推出$ibf^s_{(v,\tau_{\mathrm{HI}})}(R) \leq R$。从而在$s$ 倍速率处理器平台上 *IBF* 方法计算的最差响应时间不大于R。引理得证。 □

特别的，当$k=1$ 时，有$1+\frac{\sqrt{k^2-k}}{k}=1$。也即是说，对于高优先级干涉任务集合中任务数量为1 的情况，*IBF* 方法的加速比为1。这也得出如下推论：

推论 5.13：对于只有两个任务的 DRT 系统，使用 *IBF* 方法分析顶点的最差响应时间是精确的。

如图5.10所示，*IBF* 方法的加速比是随任务高优先级干涉任务个数k 的单调递增函数。由该性质可知 *IBF* 方法的悲观性会随着系统中任务数量的递增而增加。这是因为总干涉（上界）函数在较大k 值的情况下会在倾斜分段具有较大的斜率。而当k 趋近于正无穷时，$1+\frac{\sqrt{k^2-k}}{k}$ 的取值趋近于2，等于 *RBF* 方法的加速比。这也对应了当系统中无穷多个任务同时释放作业时，函数$ibf_{(v,\tau_{\mathrm{HI}})}(t)$ 对应的累积工作量倾斜分段将垂直于x 轴。然而，在不同任务的干涉上界函数$ibf_T(t)$ 不完全在同一时刻递增的情况下，总需求上界函数$ibf_{(v,\tau_{\mathrm{HI}})}(t)$ 的最大斜率将小于干涉任务的数量。通过对定理5.12的进一步观察可以发现，*IBF* 方法的悲观程度事实上取决于总干涉函数$ibf_{(v,\tau_{\mathrm{HI}})}(t)$ 的最大倾斜分段的斜率。只要满足总干涉函数$ibf_{(v,\tau_{\mathrm{HI}})}(t)$ 的 Lipschitz 常数为1 则 *IBF* 方法的结果就是精确的，而与系统中的任务数量无关。

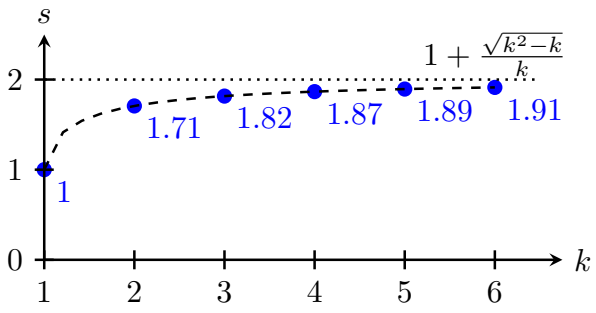


图 5.10 不同 k 值对应的 IBF 加速比取值

Fig. 5.10 The values of $1+\frac{\sqrt{k^2-k}}{k}$ for different k

5.4 实验结果与分析

本节使用随机生成的实时任务集合来进行模拟实验，并评价本章讨论的两种近似响应时间分析方法 *RBF* 和 *IBF* , 另外还将这两种方法与文献 [18] 中提出的精确分析方法（EXACT）进行任务集合可调度性接受率和时间效率等方面的比较。

本节定义有向图 $G(T)$ 中一个环的利用率为该环经过所有顶点的最差执行时间之和与该环经过所有边上释放时间间隔之和的比值。定义一个实时任务 T 的利用率为其对应有向图 $G(T)$ 所有环的利用率的最大值。定义实时任务集合的利用率为其中所有任务利用率之和。

5.4.1 随机任务集合生成方法

在生成随机任务时，首先生成一个在区间$[7,5]$ 内的随机数作为任务中顶点的个数。对于每个顶点，在区间$[1,6]$ 内选择一个随机数作为其出度。任务对应有向图中的边都是在满足各个顶点出度设置约束下随机添加的，但需要注意的是本节中的随机任务生成算法会保证有向图是强联通的。

另外每个顶点还会被分配一组从预先定义的不同类型中随机选择的配置参数。对于每一个预先定义的顶点类型，配置参数均包含该顶点标注的最差执行时间范围，和以该顶点为起始点的边上标注的最小释放间隔的范围。

对于每个顶点 v, 所有从该顶点起始的边对于的$e(v)$ 和 $p(v,u)$ 的取值都从它们分别对于的范围内随机选择，而$d(v)$ 被设置为所有外出边(v,u) 中$p(v,u)$ 的最小取值。

生成随机任务集合的过程如下：首先，构建由两个随机生成的任务组成的任务集合，并进行实验评价。接下来向上一步生成的任务集合中添加一个新生成的随机任务并进行实验评价，知道任务集合的总体利用率超过1 时结束迭代。然后重新构建一个有两个随机任务组成的集合，并按照上述方法继续迭代。不断重复上述过程，直至生成并评价了足够数量的任务集合时算法结束。

5.4.2 实验结果分析

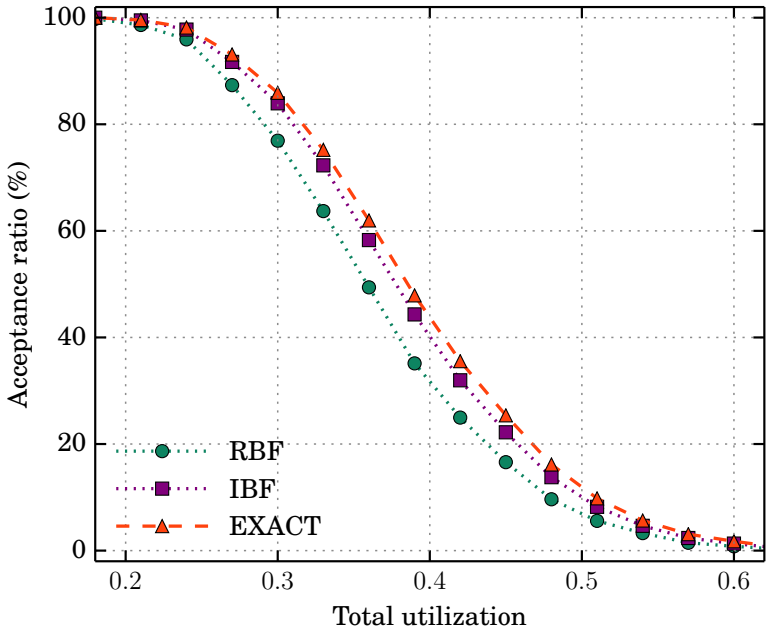


图 5.11 不同任务集利用率下的算法可调度比率

Fig. 5.11 Acceptance ratio with different total utilizations

图5.11展示了算法 *EXACT*, *RBF* 和 *IBF* 在不同任务集合总体利用率下的可调度性比率（接受率）。图5.11的x 轴表示任务集合的总体利用率，y 轴表示算法的接受率。图中的每个点表示在x 轴标示的总体利用率范围内的所有随机任务集合样本中，能够被该点对应算法运行时调度的任务集合数量占样本总量的百分比。图5.11中结果来源的实验使用了两种类型的顶点，具体的类型参数设置如下：

类型 1: $e(v) \in [1,6]$ and $p(v,u) \in [20,100]$。

类型 2: $e(v) \in [7,9]$ and $p(v,u) \in [101,300]$。

图5.12展示了随机任务集合随其包含任务数量变化而导致的不同算法下接受率变化趋势。对于该次实验，仅使用了一种顶点类型$p(v,u) \in [50,300]$ 和$e(v) \in [1,x]$。但需要注意的是，对于每个任务，通过调节x 的取值，能够调节每个任务的利用率范围，进而控制任务集合中的任务数量。本次实验评测的任务数量，及对应的随机任务顶点最差执行时间上界x 取值设置如 表5.1所示。本次实验对于每个目标任务数量取值，都

表 5.1 生成不同数量任务的任务集合对应的不同 $e(v)$ 值上界随机参数

Table 5.1 Different upper bounds of $e(v)$ to generate task sets with different number of tasks

number of tasks	2	3	4	5	6	7	8
x	33	20	15	12	10	8	7

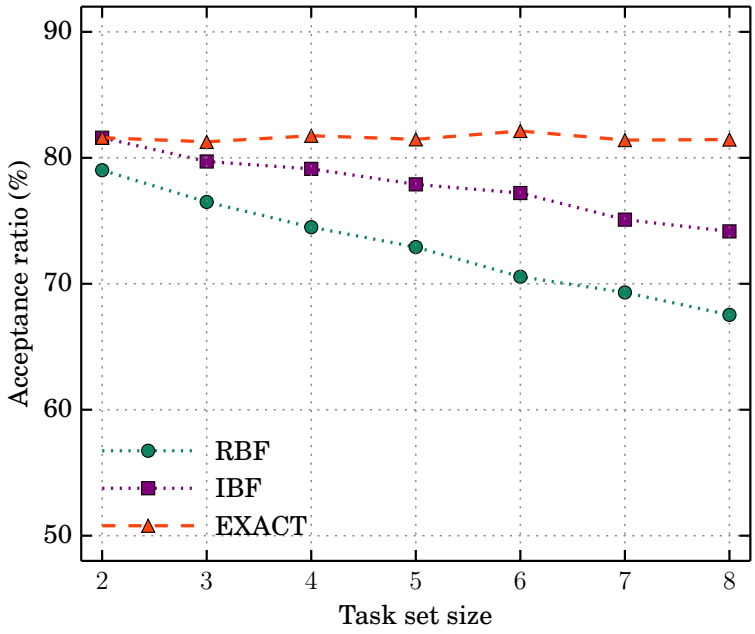


图 5.12 不同任务集任务数量下的算法可调度比率

Fig. 5.12 Acceptance ratio with different number of tasks

会从总体利用率在特定范围内的随机任务集合中，选择符合任务数量约束的样本进行评测。实验参数选择的目的是将*EXACT* 算法对于不同任务数量的随机任务集合样本，都能够保持80% 左右的稳定接受率。观察 图5.12可以发现当任务集合包含更少数量任务时，*IBF* 相对与*RBF* 的优势更为突出。特别的，对于只有两个任务的系统，*IBF* 与精确方法*EXACT* 的判断结果是等效的。这些评价实验的结果与 小节5.3 中使用加速比进行理论分析的结果是一致的。

本小节将对 *RBF*，*IBF* 和*EXACT* 三种响应时间分析方法的时间效率进行实验比较分析。实验使用基于 Python 的算法实现程序，并在一台配置 Intel Core i7-2600 CPU（3.40GH）处理器的台式电脑上执行实验程序。实验结果表明*IBF* 与*RBF* 在进行响应时间分析时的时间效率是十分接近的，并远远好于精确算法*EXACT* 的效率。在所有的实验中，RBF 与 IBF 方法的执行时间通常不超过 1 秒，而最多也不超过 4 秒。但是精确算法 EXACT 的执行时间十分不稳定。对于有些任务集合其效率很高（与 RBF 和 IBF 接近），但对于很多的任务集合其执行时间为 RBF 和 IBF 方法的上千倍。而其中还有相当一部分任务集合，EXACT 算在数小时内仍然不能结束。综上，RBF 和 IBF 分析方法相对 EXACT 方法在时间效率时的提升是极其显著的。

5.5 小结

本章使用两种近似的响应时间分析方法来判定实时任务有向图模型的可调度性，并通过分析各自的加速比来对两种方法进行了有效性的比较。第一种近似分析方法

RBF 的精确加速比为 2，即便对于仅仅包含两个任务的系统也是如此。第二种近似分析方法 IBF 的加速比为 $1+\frac{\sqrt{k^2-k}}{k}$，其中 k 为分析某特定任务时优先级较之更高的干涉任务的数量。特别的，当 k=1 时，IBF 方法的加速为 1，也即是说该方法对于仅包含两个任务的系统的分析结果是精确的。另一方面，当 k 值趋近于正无穷时，IBF 方法的加速比收敛于 2.

本章还通过基于随机生成任务集合的模拟实验评价了 RBF 与 IBF 两种方法的精确性和运行时效率。实验结果表明，本章提出的近似分析方法能够得到非常高的运行时效率，并且仅损失了很小的精度。同时也验证了关于加速比的理论分析结果。

本章的分析都是建立在所有作业都具有限制性截止期的假设之上。在未来工作中将扩展此分析方法和加速比性质到允许顶点的截止期大于边上标注的最小释放时间间隔的系统中。

第 6 章 基于 DRT 模型的优化可调度性算法

如上一章的介绍，传统的周期性重复执行任务 [7] 的模型并不能很好的描述复杂的现代实时嵌入式系统。近年来更具表达能力的基于图的实时任务模型 [10–12, 19, 121] 研究成为了热点之一。这些基于图的模型能够精确的描述复杂计算模型的时间约束，比如有限自动机模型（FSM，Finite State Machines）[139–141]。而 FSM 已经被包含到常见的建模与代码合成工具（比如 Simulink Stateflow [142]）中。

实时嵌入式系统设计者在设计阶段必须保证系统的可调度性，即在任何的系统行为下都能满足运行时的时间约束。而复杂结构的实时系统通常会在某些短时间区间内，会产生释放工作量大于平均情况的工作量突发行为。因此即便长期的资源供给能够满足所有释放工作量的需求，但在工作量突发行为下系统仍然可能违反时间约束。

为了提高基于静态优先级的实时任务有向图系统的可调度性，本章提出了针对任务有向图的整形算法。通过该方法能使得整形后的有向图释放的工作量更加的均匀，从而在一定程度上避免工作量突发的问题，或降低其影响。该整形算法的核心步骤是为图中特定顶点添加一个作业释放时间延迟变量，从而调整与图中相关顶点和边上的参数。通过相关参数的调整，整形后的新图能够保留原图中每个顶点的原始时间约束信息。因此如果系统能够满足新图中规定的时间约束，亦能满足原始图中的时间约束。以达到按调整后有向图约束的行为满足原始有向图中时间约束的目的。

延迟某个顶点上作业的释放时间后，可能使得一些特定路径上的工作量释放更加均匀，但同时也会导致另一些路径上的工作量突发问题更为严重。然而由一个 DRT 任务（根据图中的遍历路径）可能释放的不同运行时作业序列数量随时间长度呈指数级增长。因此通过枚举所有可能路径的方式来评价整形操作对于系统可调度性的影响，从时间复杂度的角度来看是不可行的。

本章提出了一种高效率的整形算法，尝试对不可调度的 DRT 任务系统进行提升可调度性的整形操作。本章提出的方法通过对 DRT 任务有向图中的每个顶点依次进行整形操作，并调整与被整形顶点相关联的边上参数取值。通过发掘任务图中的一些性质和进行合理的抽象，使得算法能够快速并且有效的完成整形操作。

本章的主要工作基于 Digraph Real-Time (DRT) task model [19] 进行描述。由于 DRT 模型为大多数已有基于图的实时任务模型（如 GMF [12]，RRT [10]，和 ncRRT [11]）的泛化模型，因此本章的所有结论同样可以直接应用到这些约束性更强的模型中。

6.1 问题模型

本章使用和上一章中类似的实时任务有向图模型。一个实时任务集合 τ 由 N 个相互独立的实时任务 $\{T_1, T_2, \cdots, T_n\}$ 构成。一个实时任务 T 被表述为一个有向图 $G(T) = (V(T), E(T))$，其中 $V(T)$ 表示图中所有顶点构成的集合，$E(T)$ 表示所有边的集合。每个顶点 $v \in V(T)$ 由二元组 $\langle e(v), d(V) \rangle$ 来标注。每条有向边 $(u,v) \in E(T)$ 均由 $p(u,v) \in \mathbb{N}$ 来标注。需要注意的是，在本章研究的模型中，参数 $e(v), d(V), p(u,v)$ 的取值均被定义为非负整数。

本章使用 $Prod(v)$ 和 $Succ(v)$ 分别表示顶点 v 的所有前驱顶点和所有后继顶点构成的集合。另外还定义 $Prod^{-}(v) \triangleq Prod(v) \setminus \{v\}$ 和 $Succ^{-}(v) \triangleq Succ(v) \setminus \{v\}$ 分别表示前驱顶点集合和后继顶点集合与顶点 v 自身构成单元素集合的差集。

6.2 任务有向图整形的基本思想

接下来本节将介绍本章提出的 DRT 任务有向图整形方法的基本思想。即是说，如何通过整形操作来减少整形任务对较低优先级任务的干涉，从而达到提升系统可调度性的目的。

首先考虑如 图6.2-(a) 所示的 DRT 任务，和一个对应路径 $\pi = (v_2, v_3, v_2, v_3, \cdots)$ 的特定作业序列（如 图6.2-(c) 所示）。如果令顶点 v_3 总是延迟 1 个时间单位释放新的作业（比如将 v_3 新释放作业休眠 1 个时间单位再进行调度），那么调整后作业序列的行为如图6.2-(d) 所示。调整后作业序列的累积工作量变得更为“均匀”，而这对于低优先级任务的可调度性可能提供潜在的益处。

上述对于如图6.2-(a) 所示 DRT 任务的调整，等效于将该任务整形为如图6.2-(b) 所示的任务。通过对顶点 v_3 的调整，使得边 (v_2, v_3) 标示的释放间隔增加了一个时间单

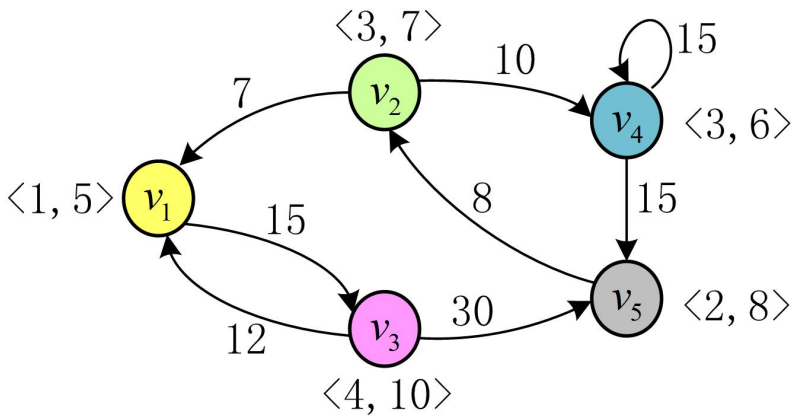
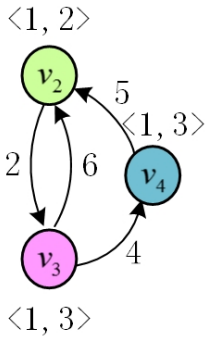
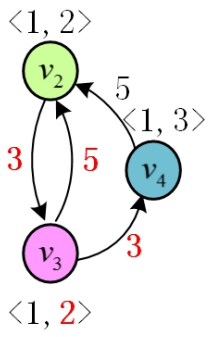


图 6.1 包含 5 种不同类型作业的 DRT 任务实例

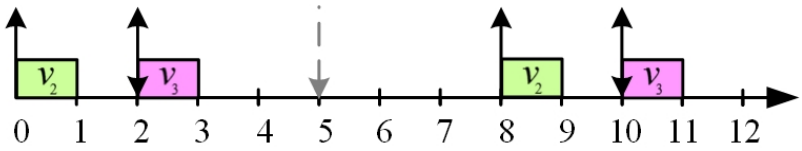
Fig. 6.1 An example task containing five different jobs



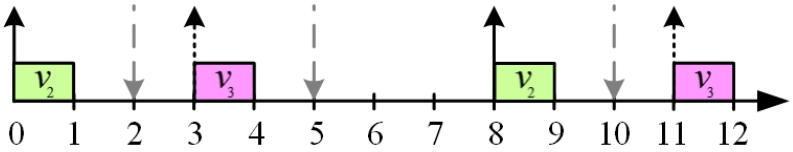
(a) 原始 DRT 任务
(a) original DRT task



(b) 整形后 DRT 任务
(b) new DRT task



(c) 原始 DRT 任务释放的作业序列
(c) job sequence from the original task



(d) 整形后 DRT 任务释放的作业序列
(d) job sequence from the new task

图 6.2 DRT 任务整形操作实例
Fig. 6.2 Illustration of DRT task transformation

位。由于 v_3 仍然需要满足其原始的相对截止期约束，因此该顶点被延迟释放的作业与其绝对截止期之间的距离应当被减小一个时间单位。而对于所有从以 v_3 为起始点的边所标注的释放时间间隔都应当被减小一个时间单位，以满足原始的作业释放间隔约束条件。

接下来，本小节将介绍上述 DRT 系统任务有向图整形的形式化描述。首先，为每个有向图中的顶点 v 引入一个新的非负参数释放延迟 $\delta(v)$。对于每个被整形的顶点 v，每条以该顶点为终止点的边上标记的释放时间间隔将被增加 $\delta(v)$，而该顶点对应的相对截止期 $d(v)$ 和每条以该顶点为起始点的边上标记的释放时间间隔将被减小 $\delta(v)$。特别地，对于同一顶点 v 为起始点和终止点的边，在分别经过增加和减小 $\delta(v)$ 操作后，其标记的释放时间间隔维持原有值不变。为了区分释放时间间隔和相对截止期这两个参数在整形操作前后的变化，本节引入如下的标记定义

定义 6.1：定义整形后的释放时间间隔为

$$pp(u,v) \triangleq p(u,v) - \delta(u) + \delta(v) \tag{6.1}$$

来表示整形后从顶点 u 释放作业到顶点 v 释放作业的最小时间间隔。定义 整形后的相对截止期为

$$dd(v) \triangleq d(v) - \delta(v) \tag{6.2}$$

来表示因顶点 v 整形后的释放时间延迟而调整的相对截止期。

有向图整形操作的目标是为每个任务的每个顶点 v 分配一个合适的 $\delta(v)$ 值，使得初始状态不能被调度的任务结合变得可调度。

接下来，本小节将说明为什么即便对于近包含三个简单 DRT 任务（$\mathscr{P}(T_1) < \mathscr{P}(T_2) < \mathscr{P}(T_3)$）的小任务集合，整形问题也不是一个简单可解的问题。

（1）考虑如 图6.2-(a) 所示的原始状态任务集合与一组作业序列。

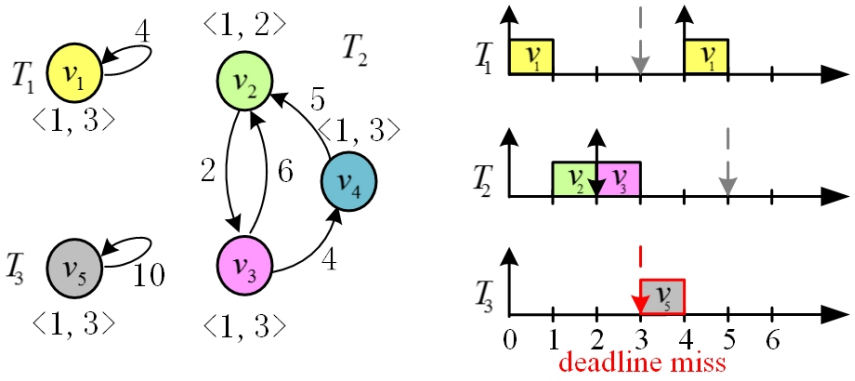
所有任务均在 0 时刻同时释放各种的第一个运行时作业，分别对应顶点 v_1, v_2, v_5。然后 v_3 在 2 时刻释放一个运行时作业。在区间 $[0,3)$ 内累积的工作量综合为 $e(v_1)+e(v_2)+e(v_5)+e(v_3)=4$，大于顶点 v_5 的相对截止期 $d(v_5)=3$，v_5 释放作业将在 3 时刻错失截止期。因此，如图6.2所示的 DRT 任务集合 τ 在其给定的优先级次序下，不可能被静态优先级调度算法调度。

（2）将任务 T_2 按照如 图6.2-(b) 所示进行整形。

尝试设置 $\delta(v_3)=1$，即将顶点 v_3 释放的所有作业的释放时间延迟 1 个时间单位。顶点 v_3 整形操作相关联的有向图参数需要遵从定义6.1的定义进行更新计算。相关联的参数包括顶点 v_3 的相对截止期，以及与顶点 v_3 相连接的所有边上标记的释放时间间隔。通过该整形操作，由路径 $(v_2, v_3, \cdots)$ 释放的工作量分布的更为“均匀”。该路径在 $[0,3)$ 区间内释放的总工作量为 $e(v_1)+e(v_2)+e(v_5)=3$，因此顶点 v_5 释放的作业可以在其截止期 3 之前完成执行。另一方面，尽管顶点 v_3 对应的相对截止期被减小了 1 个时间单位，但是由于 $e(v_1)+e(v_3)=2=dd(v_3)$，因此该顶点释放的作业依然可以被成功调度。

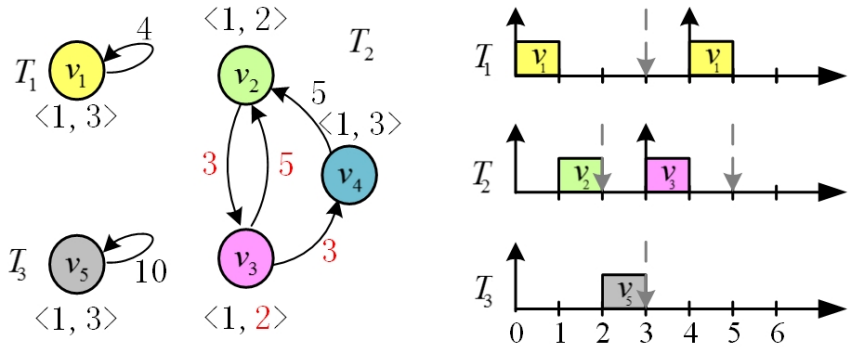
（3）将任务 T_2 按照如 图6.2-(c) 所示进行整形。

如果设置 $\delta(v_3)=2$，将导致 $pp(v_2,v_3)=pp(v_3,v_2)$。由顶点 v_2 和 v_3 释放的工作量比之前的情况更为“光滑”。但是，由此过于激进的整形导致顶点 v_3 自身的相对截止期过短（$dd(v_3)=3-2=1$），并导致顶点 v_3 自身释放作业变为不可调度。另外，设置 $\delta(v_3)=2$ 不仅造成 v_3 释放作业不可调度，同时也导致了任务 T_3 的顶点 v_5 释放作业将错失截止期。如图6.2-(c) 所示，设置 $\delta(v_3)=2$ 尽管能够使顶点 v_2 和 v_3 释放的工作量



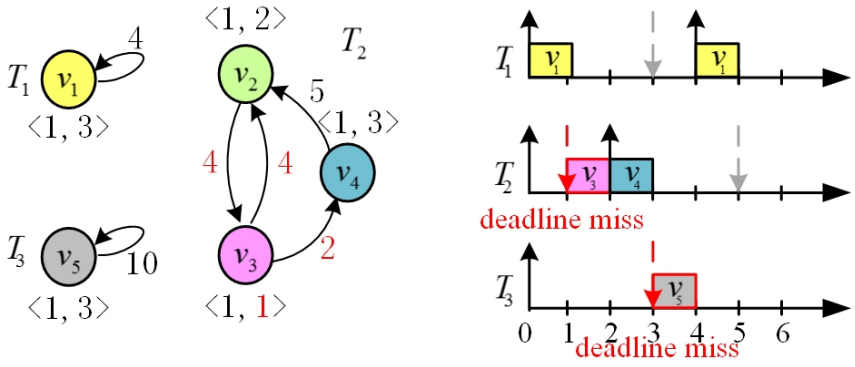
(a) $\delta(v_3)=0$（原始任务），v_5 错失截止期

(a) $\delta(v_3)=0$ (original task), v_5 misses its deadline



(b) $\delta(v_3)=1$，没有作业错失截止期

(b) $\delta(v_3)=1$, no deadline miss



(c) $\delta(v_3)=2$，v_3 和 v_5 错失截止期

(c) $\delta(v_3)=2$, v_3 and v_5 miss their deadlines

图 6.3　设置 $\delta(v_3)$ 为不同取值时任务的可调度性实例

Fig. 6.3　Schedulability with different values of $\delta(v_3)$

分布的更为“光滑”，但同时也使得另一条路径 $(v_3, v_4, v_2, \cdots)$ 释放工作量的分布更为集中，并导致顶点 v_5 释放的作业变为不可调度。

上述的例子演示了延迟某个顶点 v 的释放时间可能会对系统的可调度性产生正面和负面的双重影响：

- 正面的影响，主要表现在使得某些通过任务有向图的路径释放的工作量分布更为平均，并能够潜在的提高低优先级任务的可调度性。
- 负面的影响，表现在使得另一些通过有向图的路径释放的工作量变得更为集中，并对低优先级任务的调度产生不利的影响。
- 负面的影响，还表现在缩短了顶点 v 自身的相对截止期，并使得其释放的作业更能被调度。

对于较为复杂的任务有向图，很难判断将某一顶点设置特定大小释放延迟是否有利于提升系统的可调度，抑或降低系统的可调度性。而评估对于多个顶点分别设置不同释放延迟 $\delta(v)$ 对系统可调度性的影响时，问题变得更为复杂。一般来讲，判断是否存在一组每个顶点 $\delta(v)$ 的参数设置，使得系统可调度的问题的时间复杂度是强反 NP 难的。一个简单的方法是通过枚举所有顶点以及顶点的 $\delta(v)$ 不同取值的所有可能组合，并检查每一种参数组合设置下的系统可调度性。显然这种简单方面在计算复杂度层面上是不可行的。本章研究的目标是设计一种高效的技术，能够有效地对任务有向图进行整形，进而提高系统的可调度性。需要注意的是，本章提出的技术并不保证找到对每个顶点设置最优 $\delta(v)$ 取值，而是侧重于快速地找到高质量的结果，使得原本不可调度的任务集合中显著的一部分能够变得可调度。

6.3 高效整形算法

6.3.1 算法概述

整形过程中需要反复的分析系统的可调度性（显式的或隐式的），来保证整形过程是向着有利于系统可调度性的方向推进。由于精确的 DRT 任务集合固定优先级可调度性判定方法的时间复杂度已经非常不可接受，因此在整形过程中反复使用精确可调度性判定方法的时间开销是更加不可接受的。

取而代之的是，本章使用抽象化的需求上界函数 $rbf_T(t)$，来更有效率地执行充分的可调度性判定。小节6.3.2将介绍需求上界函数的定义，以及高效的计算方法。直观上来看，需求上界函数 $rbf_T(t)$ 限定住了 DRT 任务 T 在任意时间间隔 t 内对低优先级任务总称干涉的最大值。

本章提出的算法以优先级从高到低的顺序依次为每个任务分别进行整形操作，也

即是说，拥有最小取值的任务第一个进行整形。$\mathscr{P}$ 整形操作的目标为，对于每个整形任务，尽可能多的减小其对低优先级任务的干涉，并同时避免自身变得不可调度。如果整形过程返回 **true**，则表示整形后的任务集合是可调度的。但是，当整形过程返回 **false**时并不表示整形后的任务集合不可调度，而仅表示其没有通过基于需求上界函数的可调度性判定充分非必要条件的可调度性检验。因此在此情况下，本章提出的算法需要再次通过文献 [138] 中提出的可调度性精确分析方法来最终判定整形后的任务集合是否可调度。

图6.4展示了本章提出整形算法的伪代码描述。在原始的 DRT 任务集合 τ 中，每个任务顶点的释放延迟 $\delta(v)$ 初始时均被设置为 0。整形操作从 DRT 任务的优先级从高到低的顺序依次执行。对于当前整形任务的每个顶点 v，算法首先计算一个 v 顶点自身的释放延迟上界 Δ_{slf}。算法可以保证，当顶点 v 实际被分配的释放延迟 $\delta(v)$ 不大于 Δ_{slf} 时，该顶点能够保持其原有的可调度性。Δ_{slf} 通过算法 Slf_Bound 来计算。算法 Slf_Bound 需要利用顶点 v 自身的信息，还需要知道所有更高优先级任务各种的需求函数。在小节6.3.3将相信介绍算法 Slf_Bound。如果某顶点计算得到的 Slf_Bound 是一个负值，则表明顶点 v 自身可能不能被调度（比较悲观的判断），而算法不会对这样的顶点进行整形操作。

如果某顶点计算得到的 Slf_Bound 是一个大于 0 的值 Δ_{slf}，那么算法可以确保在将顶点 v 的释放延迟设置任意不大于 Δ_{slf} 时间单位的值时，顶点 v 释放的作业仍能够满足自身的截止期约束。然后，算法将通过过程 Itf_Bound 来计算顶点参数 $\delta(v)$ 的另一个上界 Δ_{itf}，来尽可能的降低对低优先级任务的干涉。小节6.3.4将详细介绍过程 Itf_Bound。最终，算法将选择上述顶点 v 的两个上界中较小的一个赋予 $\delta(v)$，并进一步调整与顶点 v 相关的其他参数。在计算完任务 T 所有所有顶点的释放延迟之后，算法会计算该任务的干涉函数，以用于低优先级任务整形操作的计算。最后，如果算法能够成功的为所有任务的所有顶点分配一个非负的释放间隔 $\delta(V)$ 值，那么整形后的 DRT 任务集合能够保证被调度。

接下来，本章首先介绍需求上界函数 $rbf_T(t)$ 的形式化定义。并分别在小节6.3.2中介绍其高效的计算方法，在小节6.3.3介绍过程 Slf_Bound，在6.3.4介绍过程 Itf_Bound。

6.3.2 需求上界函数 rbf_T

本小节首先定义通过一个 DRT 任务有向图的特定路径的需求函数:

定义 6.2 (特定路径的需求函数)：给定一个 DRT 任务 T，对于一条通过有向图

Transform(τ)

1: $result \leftarrow$ **true**
2: $\forall v \in \tau : \delta(v) \leftarrow 0$
3: **for each** $T \in \tau$ in increasing order of $\mathscr{P}(T)$ **do**
4: **for each** $v \in G(T)$ **do**
5: $\Delta_{slf} = Slf\_Bound(v, \{rbf_{T'} | \mathscr{P}(T') < \mathscr{P}(T)\})$
6: **if** $\Delta_{slf} \geq 0$ **then**
7: $\Delta_{itf} = Itf\_Bound(v, \Delta_{slf})$
8: $\delta(v) \leftarrow \min(\Delta_{slf}, \Delta_{itf})$
9: **else**
10: $failure \leftarrow$ **false**
11: **end if**
12: **end for**
13: Compute rbf_T
14: **end for**
15: **return** *result*

图 6.4 整形算法的伪代码描述

Fig. 6.4 Pseudo-code of the Transformation Algorithm.

$G(T)$ 的路径 $\pi = (v_0, \cdots, v_l)$，定义其需求函数 rf_π 为:

$$rf_\pi(t) \triangleq \max\left\{e(\pi') \mid \pi' \text{ is prefix of } \pi \text{ and } p(\pi') < t\right\}$$

$$\text{where } e(\pi) \triangleq \sum_{i=0}^{l} e(v_i) \text{ and } p(\pi) \triangleq \sum_{i=0}^{l-1} pp(v_i, v_{i+1})。$$

$rf_\pi(t)$ 为对于自变量 t 非递减的阶梯函数。其中每个水平分段为左开右闭，特别定义 $rf_\pi(0) = 0$。

定义 6.3 (需求上界函数)：给定一个路径集合 $S = \{\pi_1, \cdots, \pi_n\}$，定义该集合路径的需求上界函数 rbf_S 为：

$$rbf_S(t) \triangleq \max_{\pi_i \in S}\left\{rf_{\pi_i}(t)\right\}.$$

特别的，给定一个 DRT 任务 T，定义该任务对应有向图中所有可能路径的需求上界函数 rbf_T 为：

$$rbf_T(t) \triangleq \max_{\pi \in G(T)}\left\{rf_\pi(t)\right\}.$$

需求上界函数同样也是阶梯函数，并且拥有和需求函数类似的性质。

由于有向图 $G(T)$ 随路径长度增加其路径数量呈指数级爆炸性增长，通过枚举每条具体路径来计算需求上界函数的方法在计算复杂度层面上是不可行的。文献 [19] 中提出了一种动态规划的技术，能够在伪多项式时间内解决该问题。此技术的主要思想是在路径遍历过程中，使用抽象方法来压缩需要记录的已遍历路径信息。伪多项式时

间复杂度的计算 $rbf_T(t)$ 的算法伪代码如图6.5所示。其中，$\langle e,r,u\rangle$: 是一个用于抽象表达已遍历路径信息的三元组：u 表示当前遍历路径的最后一个顶点，r 表示当前路径上最后顶点 u 的最近一次释放时间，e 表示在时刻 r 之前当前路径上所累积的工作量之和。RF_k 表示已遍历过的包含 k 个顶点的路径 π 的集合，并且满足 $\text{len}(\pi) \le t$，其中 $\text{len}(\pi)$ for a path $\pi = \{v_1, v_2, \cdots, v_n\}$ 被定义为：$\text{len}(\pi) = \sum_{i=1}^{n-1} p(v_i, v_{i+1})$。

```
CalculateRBF(T,t)
 1: RF_0 ← {⟨0,0,v_i⟩|v_i vertex of G(T)}
 2: for k = 1 to t do
 3:     RF_k ← ∅
 4:     for each ⟨e,r,u⟩ ∈ RF_{k−1} do
 5:         for each edges(u,v) in G(T) do
 6:             e' ← e + e(u)
 7:             r' ← r + pp(u,v)
 8:             if r' ≤ t then
 9:                 RF_k ← RF_k ∪ {⟨e',r',v⟩}
10:             end if
11:         end for
12:     end for
13: end for
14: return max{e + e(v)|⟨e,r,v⟩ ∈ ⋃_{k≤t} RF_k}
```

图 6.5 计算需求上界函数 $rbf_T(t)$ 算法伪代码

Fig. 6.5 An algorithm for computing $rbf_T(t)$

6.3.3 $Slf\_Bound()$ 过程

$Slf\_Bound()$ 过程的主要思想是根据已整形高优先级任务集合的信息，快速计算出顶点 v 的一个响应时间上界 R_v。然后通过 $d(v) - R_v$ 来计算出保证顶点 v 不错失自身截止的一个安全的释放延迟上界。算法的正确性参考下面的引理。

引理 6.1：给定 DRT 任务 T 在设置释放延迟 $\delta(v)$ 后仍然可调度的一个充分条件是满足 $\delta(v) \le d(v) - R_v$，其中

$$R_v \triangleq \min\left\{ t \,\middle|\, e(v) + \sum_{\mathscr{P}(T') < \mathscr{P}(T)} rbf_{T'}(t) \le t \right\} \tag{6.3}$$

证明：该引理通过反证法来证明。假设存在由顶点 v 在 t 时刻释放的一个作业，在延迟 $\delta(v)$ 时间单位释放后于 t_d 时刻错失其截止期。因此 $t_d - t_r = dd(v) = d(v) - \delta(v)$。令 $H(T)$ 表示比任务 T 优先级更高的任务组成的集合。在使用静态优先级算法调度 DRT 任务集合时，文献 [123] 证明了所有任务同步释放首个作业的情况将导致关键时刻（*critical instant*）[7]。也即是说，如果存在一个作业序列，其中的一个作业 v 会错

失截止期，那么一定可以构造一个所有 $H(T)$ 中的任务都同时在 t_r 时刻释放作业的序列造成该作业错失截止期。因此，不是一般性地假设在上述方法构造作业序列中，集合 $H(T)$ 中的每个任务在 t_r 时刻释放第一个作业，并使用 $\pi^{T'}$ 来表示作业序列中由任务 $T' \in H(T)$ 释放的所有作业构成的子序列对应的通过有向图 $G(T')$ 的并在区间 $[t_r, t_d]$ 内路径。

因为顶点 v 释放的作业错失了截止期，因此对于任意时刻 $t_1 \in [t_r, t_d]$，必然有属于集合 $H(T)$ 或任务 T 自身释放的作业处于执行完成的状态。所以可以得到对于任意 $t \in [0, d(v) - \delta(v)]$，满足

$$e(v) + \sum_{T' \in H(T)} rf_{\pi^{T'}}(t) > t$$

根据需求上界函数的定义（定义6.3），可知 $\forall t : rf_{\pi^{T'}}(t) \leq rbf_{T'}(t)$。因此上式可以重新表达为

$$e(v) + \sum_{T' \in H(T)} rbf_{T'}(t) > t$$

又因为 $\delta(v) \leq d(v) - R_v$，上述不等式可推出

$$e(v) + \sum_{T' \in H(T)} rbf_{\pi^{T'}}(R_v) > R_v$$

上式与式 (6.3)相矛盾。引理得证。 □

因为只需要检查 $[e(v), d(v)]$ 区间内响应时间，因此 R_v 的计算可以在伪多项式时间内完成。如果 R_v 超出了 $[e(v), d(v)]$ 的范围，那么 *Slf_Bound* 将返回一个负值。这表示顶点 v 释放的作业即便设置 $\delta(v) = 0$ 也不能够被调度（充分非必要条件检测），而图6.4所示的算法也不会对这类顶点进行后续的释放延迟时间计算。文献 [143] 中提出的使用固定点（fixed-point）迭代技术来计算标准实时任务响应时间的方法可以应用到提升计算 R_v 效率的后续工作中。

通过引理6.1可以容易的总结出如下的定理：

定理 6.2：给定 DRT 任务集合 τ，如果如 6.4 所示的任务有向图整形算法返回 **true**，那么整形后的 DRT 任务集合 τ 一定是可调度的。

由于整形算法中并没有使用精确的可调度性分析方法，因此当整形算法返回 **false**时，并不表示整形后的 DRT 任务集合一定不可调度。在这种情况下，算法将对所有任务整形完毕后的任务集合使用文献 [138] 中介绍的精确分析方法来最终判定其可调度性。

6.3.4 $Itf\_Bound()$ 过程

如果将 $\delta(v)$ 设置任何不大于上节中介绍的 $Slf\_Bound()$ 过程计算得到上界值，则能够保证顶点 v 保持其自身的原始可调度性。但是，正如小节6.3.3中的讨论，对于 $\delta(v)$ 如何取值才能使得更低优先级任务的调度更加有利的问题，仍然不明确。在本小节，将介绍 $Itf\_Bound()$ 过程，从有利于低优先级任务调度的角度计算 $\delta(v)$ 的合适取值。$Itf\_Bound()$ 返回的上界值并不一定是最优的，但是在大多数情况下能够显著地提高低优先级任务的可调度性。

大致而言，$Itf\_Bound()$ 过程的目标是寻找一个合适的 $\delta(v)$ 取值，使之能够尽可能多地减小需求上界函数 $rbf_T(t)$。确切地讲，该过程之考虑在与低优先级任务的调度相关的时间 t 值范围内考虑 $rbf_T(t)$ 返回值的变化。计算这样的相关时间 t 的上界的一个直接方法是搜索所有低优先级任务顶点的最长相对截止期。而事实上，并不需要考虑所有低优先级顶点的相对截止期，而仅需考虑那些调度更困难的顶点即可。接下来，本节将基于顶点支配的概念来给出调度困难的形式化定义：

定义 6.4 (顶点支配)：对于 DRT 任务 T 对应有向图 $G(T)$ 中的两个顶点 v 和 v'，如果 v 支配 v'（记作 $v \succcurlyeq v'$），那么对于下列两个条件必须满足其中至少一个：

$$e(v) \geq e(v') \wedge dd(v) - e(v) \leq dd(v') - e(v') \tag{6.4}$$

$$\lceil e(v')/e(v) \rceil \leq \lfloor dd(v')/dd(v) \rfloor \tag{6.5}$$

特别的，顶点 v 绝对支配顶点 v'（记作 $v \succ v'$）的充分必要条件为 $v \succcurlyeq v' \wedge v' \not\succcurlyeq v$。

根据顶点的支配关系，下面形式化地定义什么样的顶点为难于调度的：

定义 6.5 (关键顶点)：给定顶点 v，如果不存在能够绝对支配该顶点的其它顶点，则定义这样的顶点 v 为关键顶点。给定由关键顶点构成的集合 $S(\tau)$，如果对于任意 τ 中的顶点 u，均存在 $v \in S(\tau)$ 满足 $v \succcurlyeq u$，则称该集合为任务集合 τ 的关键集合，记作 $CS(\tau)$。

关键顶点具有如下的性质：

引理 6.3：一个 DRT 任务集合 τ 是静态优先级算法可调度的充分必要条件为，τ 的所有关键顶点都是可调度的。

证明：如果 τ 是静态优先级算法可调度，那么 τ 中所有的顶点（包含所有的关键顶点）必然都是可调度的。因此引理的必要性显然成立。

对于每个 DRT 任务 $T \in \tau$，根据定义6.4可以证明对于任意顶点 $v \in G(T)$，如果满足 $v \notin CS(\tau)$，则比如存在 $u \in CS(\tau)$ 和 $v_1, \cdots, v_n \notin CS(\tau)$ 满足 $u \succ v_1 \succ \cdots \succ v_n \succ v$。

考虑命题：给定一个 DRT 任务 $T \in \tau$ 和两个顶点 $v,v' \in G(T)$，满足 $v \succcurlyeq v'$，如果顶点 v 可调度那么顶点 v' 一定也是可调度的。根据上面的讨论可以推出，如果该命题成立，那引理的充分性即是成立的。下面将通过反证法来证明该命题的正确性。

因为 $v \succcurlyeq v'$，定义6.4中列出的条件只要有一个必须被满足。假设顶点 v 可调度，但是顶点 v' 不可调度。

令 $\beta(l)$ 表示处理器在长度为 l 的时间间隔内可以分配给任务 T 执行的最小累积时间。显然对于任意 $t_1 \leq t_2$ 满足

$$0 \leq \beta(t_2) - \beta(t_1) \leq t_2 - t_1 \tag{6.6}$$

为了简化表达，令 $d = dd(v), d' = dd(v'), e = e(v)$ and $e' = e(v')$。

首先考虑定义6.4中式 (6.4) 被满足的情况。因为顶点 v 可调度但是顶点 v' 不可调度，可推出 $\beta(d') < e'$ 和 $\beta(d) \geq e$。因为 $e \geq e'$，可推出 $\beta(d') < \beta(d) \Rightarrow d' < d$。又因为 $\beta(d') < e'$ 和 $\beta(d) \geq e$，可推出

$$\beta(d) - \beta(d') > e - e'$$

根据 6.6 可推出

$$d - d' > e - e'$$

上式与式 (6.4)相矛盾。

接着考虑式 (6.5)成立的情况。首先由 $d' \geq d \cdot \lfloor d'/d \rfloor$ 可推出

$$\beta(d') \geq \beta\left(d \cdot \lfloor d'/d \rfloor\right) \tag{6.7}$$

一个长度为 $d \cdot \lfloor d'/d \rfloor$ 的时间间隔可以被分成 $\lfloor d'/d \rfloor$ 数量的长度为 d 的分段。由 $\beta(l)$ 的定义可推出

$$\beta\left(d \cdot \lfloor d'/d \rfloor\right) \geq \beta(d) \cdot \lfloor d'/d \rfloor \tag{6.8}$$

与 (6.7) 和 (6.8) 联立可推出

$$\beta(d') \geq \beta(d) \cdot \lfloor d'/d \rfloor$$

又由 式 (6.5) 可推出

$$\beta(d') \geq \beta(d) \cdot \lceil e'/e \rceil \geq \beta(d) \cdot e'/e$$

由 v 可调度可推出 $\beta(d) \geq e$。将其代入上式可得 $\beta(d') > e$，这与 v' 不可调度相矛盾。

综上，每种情况都会导致矛盾的结论。由此定理得证。 □

以为任务集合的可调度性完全取决于关键顶点，因此为当前整形的高优先级任务 T 的某顶点 v 选择 $\delta(v)$ 取值时，只需要在长度为比任务 T 优先级更低任务的所有关键顶点中最大相对截止期的区间内考虑减小 $itf_T(t)$ 取值。该长度被定义为关键窗口长

度。

定义 6.6 (关键窗口长度)：任务 T 的关键窗口长度被定义为：

$$\rho_T \triangleq \max\{dd(v)|v \in CS(\tau), \mathscr{P}(v) > \mathscr{P}(T)\}. \tag{6.9}$$

需要注意的是，当需求上界函数 $rbf_T(t)$ 在区间 $(0, \leq \rho_T]$ 内单调递减时，$rbf_T(t)$ 仍然可能对于某些 $t > \rho_T$ 时的取值增大。但是，根据引理6.3可知只要能够保证在关键顶点集合 $CS(\tau)$ 内的顶点全部可调度，上述行为就不会对不包含在 $CS(\tau)$ 中低优先级顶点的可调度性造成影响。

接下来将介绍另一个关于路径之间支配关系的重要概念：

定义 6.7 (路径支配)：给定两条通过 DRT 任务 T 对应有向图 $G(T)$ 的路径 π 和 π'，定义在区间 $[0,x]$ 内的关系 π 支配 π'（记作 $\pi \succcurlyeq_x \pi'$）成立的充分必要条件为：

$$\forall t \in [0,x] : rf_\pi(t) \geq rf_{\pi'}(t)$$

定义两条路径 π 和 π' 是不可比较的充分必要条件为 $\pi \succcurlyeq \pi'$ 和 $\pi' \succcurlyeq \pi$ 均不成立。

正如小节6.2中的讨论，延迟 DRT 任务 T 对应有向图 $G(T)$ 中某顶点 v 释放作业的释放时间，可能会减小某条通过 $G(T)$ 的路径 π 对于低优先级任务的干涉，但同时也可能增加其他通过 $G(T)$ 的路径 π' 的干涉工作量。但是，如果保证在顶点的整形操作完成后能够满足 $\pi \succcurlyeq_{\rho_T} \pi'$，那么需求函数 $rf_{\pi'}(t)$ 部分取值的增加不会导致在区间 $[0,\rho_T]$ 内任务 $[0,\rho_T]$ 的最差干涉函数 $rf_{\pi'}(T)$ 的增加。因此整形操作不会对任意低优先级任务的可调度性造成不利的影响。

特别说明的是，对于任意非 v 为起始点的路径 π，顶点 v 释放延迟的增加不会造成其需求函数 $rf_\pi(t)$ 在任意时刻 t 取值的增加（可能会减小），却会造成以顶点 v 为起始点路径在特定 t 取值的增加（不会减小）。因此，整形操作时总是在满足长度为 ρ_T 的时间间隔内 v 起始点路径增加的工作量能够被非 v 起始点路径支配的条件下，选择尽可能大的 $\delta(v)$ 取值，以尽可能多的减小非 v 起始点路径的干涉工作量。接下来，本小节将详细介绍如何寻找满足上述条件的释放延迟 $\delta(v)$ 的上界。

定义 6.8 (上升点)：给定阶梯函数 f，定义其上升点 p 为满足条件 $f(p) < f(p^+)$ 的点。其中 $p^+ \triangleq p + \varepsilon$，$\varepsilon$ 为一个任意接近于0 的正实数。

定义 6.9 (支配点)：给定两个阶梯函数 f 和 g，对于函数 f 上任意的上升点 p，定义函数 g 上的上升点 q 是 p 的支配点的充分必要条件为：

$$0 \leq q \leq p \wedge g(q) < f(p^+) \wedge g(q^+) \geq f(p^+). \tag{6.10}$$

如果对于函数 f 上的每个上升点（在时间区域 $[0,\rho)$ 内），在函数 g 上都存在一个支配

点，则定义在时间区域 $[0,\rho)$ 内函数 g 支配函数 f。记作：$f' \succcurlyeq_\rho f$。

引理 6.4：给定一条具体路径 π 和路径集合 $S=\{\pi_1,\cdots,\pi_n\}$，那么关系 $rf_\pi \succcurlyeq_\rho rbf_S$ 成立的充分必要条件为：$\forall \pi_i \in S, \pi \succcurlyeq_\rho \pi_i$.

证明：必要性：由 $\forall \pi_i \in S, \pi \succcurlyeq_\rho \pi_i$ 可推出 $\forall t \in [0,\rho] \mid rf_\pi(t) \geq rbf_S(t)$。因此对于函数 rbf_S 上的每个上升点均满足 $rf_\pi(p^+) \geq rbf_S(p^+)$。根据定义可知 $rf_\pi(0)=0<rbf_S(p^+)$，所以必然存在某个 $p' \in [0,p]$ 满足 式 (6.10)，也即是说 p' 支配 p。

充分性：接下来通过反证法来证明引理的充分性。假设存在某条路径 $\pi_k \in S$ 满足 $\pi \not\succcurlyeq_\rho \pi_k$。从而必然存在 $t_0 \in (0,\rho)$ 满足

$$rf_{\pi_k}(t_0) > rf_\pi(t_0) \geq 0$$

根据 需求上界函数的定义，可推出

$$rbf_S(t_0) \geq rf_{\pi_k}(t_0) > rf_\pi(t_0)$$

根据函数 rbf_S 的单调性可知，必然存在 $p \leq t_0$ 满足

$$rbf_S(p) < rbf_S(t_0) \wedge rbf_S(p^+) = rbf_S(t_0)$$

也即是说，p 是在区间 $(0,\rho]$ 内函数 rbf_S 上的一个上升点。又因为 $\forall t \leq p \leq t_0$，可推出

$$rf_\pi(t) \leq rf_\pi(t_0) < rbf_S(p^+)$$

也即是说，点 p 不可能被函数 rf_π 上的任何一个上升点所支配。这和假设是相矛盾的。

综上，引理得证。 □

增加特定顶点 v 的 释放延迟 有利于减小非 v 起始点路径的需求函数在区间 $[0,\rho+\Delta]$ 内的累积干涉工作量，但同样也可能增加 v 起始点路径的干涉工作量。算法将通过寻找 v 起始点路径的支配路径的方法来限制 v 起始点路径干涉工作量增加对 DRT 系统可调度性的影响。

接下来，本小节将集中讨论为每个候选整形顶点 $v \in G(T)$ 计算释放时间延迟 $\delta(v)$ 上界的方法。

如果想覆盖所有的可能性，整形算法在计算顶点 v 的释放延迟是，需要为每条 v 起始点路径搜索所有的非 v 起始点路径来寻找支配路径。但是由于无论 v 起始点路径还是非 v 起始点路径的数量都是随着 ρ_T 呈指数级增长的，因此上述方法的时间复杂度是不可承受的。

为解决上述问题，本节定义 *v-started request bound function* 来评价任意 v 起始点路径的累积需求上界: $\forall t \geq 0$

$$rbf_T^v(t) \triangleq \max\left\{rf_\pi(t) \mid \pi \in G(T) \wedge \pi\text{的起始点为}v\right\} \tag{6.11}$$

另一方面，对于每个顶点 $u \in G(T) \wedge u \neq v$，使用一种贪婪算法来生成一条具体的路径 π 来检验该顶点为起始点的路径在区间 $[0,\rho]$ 内的支配关系。首先设置初始路径 π 为 (u)。然后从上次迭代的路径 π 末尾顶点 v 的后继顶点中选择具有最大最差执行时间的顶点 v'，并将其添加到路径 π 的结尾。充分上述步骤，直至当前迭代路径的末尾顶点没有后继顶点，或者满足 $p(\pi) \geq \rho$。该贪婪算的详细伪代码描述如图6.6所示。

GenerateGreedyPath(u, ρ)

1: $\pi \leftarrow (u)$
2: $v \leftarrow u$ {use v indicating the last vertex of π}
3: **while** $p(\pi) < \rho \wedge Succ(v) \neq \emptyset$ **do**
4: 　from $Succ(v)$ find v' with the maximal $e(v')$
5: 　append v' to the end of π
6: 　$v \leftarrow v'$
7: **end while**
8: **return** π

图 6.6 生成 u 起始点路径的贪心算法

Fig. 6.6 Greedy algorithm for generating u-started path

基于上面介绍的 v-start rbf 和贪婪算法，接下来将介绍如何计算释放时间延迟。

引理 6.5：给定一个 v 起始点路径的需求上界函数 rbf_T^v，且该函数被一条 u 起始点路径 $\pi\,(u \neq v)$ 的需求函数 rf_π 在区间 $\rho+\delta$ 内支配。并且对于函数 rbf_T^v 上任意的上升点 p，令 p_d 表示函数 rf_π 上的支配 p 的支配顶点。如果将参数 $\delta(v)$ 增加 δ 个时间单位，且满足 $\delta \leq dd(v) - e(v) < dd(v)$，$\delta \leq \rho$ 和

$$\delta \leq \min\left\{ \frac{(p-p_d)}{2} \;\middle|\; p \text{ 是 } rbf_T^v \text{ 上的一个上升点} \right\}$$

$$\text{其中 } p \in (0,\rho+\delta) \wedge rbf_T^v(p^+) > rf_\pi(0^+) = e(u)$$

那么在将参数 $\delta(v)$ 增加 δ 后，所有 v 起始点路径都仍将被路径 π 在区间 $(0,\rho]$ 内支配。

证明：因为 $rf_\pi \succcurlyeq_{\rho+\delta} rbf_T^v$，所以对于每条 v 起始点路径 π_i 均满足 $\pi \succcurlyeq_{\rho+\delta} \pi_i$。因此有 $\forall t \in [0,\rho+\delta] \mid rf_\pi(t) \geq rf_{\pi_i}(t)$。作为将参数 $\delta(v)$ 增加 δ 的结果，对于任意顶点 $u \in Pred^-(v)$，其相关参数 $pp(u,v)$ 将同样被增加 δ。而对于任意 $v' \in Succ^-(v)$，其对应参数 $pp(v,v')$ 将被减小 δ。由于 $\delta \leq dd(v) - e(v) \leq pp(v) - e(v)$，整形后的参数 $pp(v,v')$ 将不会小于 $e(v) > 0$，因此不会造成整形顶点与其后继顶点的交叠。如果存在环形边 (v,v)，参数 $pp(v,v)$ 将保持其原有取值不变。

根据 需求函数 的定义可知，对于每条 v 起始点路径 π_i，在将参数 $\delta(v)$ 增加 δ 的整形操作后，其需求函数 rf'_{π_i} 满足 $\forall t \in [0,\rho] \mid rf'_{\pi_i}(t) \leq rf_{\pi_i}(t+\delta)$。同时对于每条非 v 起始点路径 π，整形后的需求函数 rf'_π 满足 $\forall t \in [0,\rho] \mid rf'_\pi(t+\delta) \geq rf_\pi(t)$。

根据上面的讨论可知，对于任意时间 $t \in (0,\rho)$，必然在函数 rbf_T^v 上存在一个上升点 $p \in [0,t+\delta)$，

$$rbf_T^v(p^+) = rbf_T^v(t+\delta) \geq rf_{\pi_i}(t+\delta) \geq rf'_{\pi_i}(t) \tag{6.12}$$

如果 $rbf_T^v(p^+) \leq rf_\pi(0^+)$，那么易知

$$rf'_{\pi_i}(t) \leq rf_\pi(0^+) \leq rf'_\pi(t).$$

然后考虑 $rbf_T^v(p^+) > rf_\pi(0^+)$ 时的情况。因为 $\pi \succcurlyeq_{\rho+\delta} \pi_i$，所以在函数 rf_π 上必然存在一个上升点 p_d，满足

$$\begin{aligned} rbf_T^v(p^+) &\leq rf_\pi(p_d^+) = rf_\pi(p^+ - (p-p_d)) \\ &\leq rf_\pi(t+\delta-(p-p_d)) \\ &\leq rf'_\pi(t+2\cdot\delta-(p-p_d)). \end{aligned}$$

由 $2\cdot\delta \leq p-p_d$ 可推出

$$rbf_T^v(p^+) \leq rf'_\pi(t) \tag{6.13}$$

综合上面的讨论可推出，对于每个 $t \in (0,\rho)$ 均满足 $rf'_{\pi_i}(t) \leq rf'_\pi(t)$。根据 需求上界函数的定义，可以推出 $rf'_\pi \succcurlyeq_\rho rbf'^v_T$。根据 引理6.4，该引理得证。 □

对于任意顶点 $u \in G(T)\setminus\{v\}$，根据 引理6.5 可以计算一个安全的释放延迟上界以保证原来的 前驱顶点支配。因为更大的 $\delta(v)$ 取值，能使得非 v 起始点路径的需求函数变得更有利于低优先级任务的调度，所以算法将从所有计算得到的候选取值中选取最大值赋予参数 $\delta(v)$。

基于上述讨论，计算释放延迟上界算法的伪代码如图6.7所示。其中第 3 行，算法首先使用与 图6.5列出算法类似的方法（不同之处是在第 1 行设置 RF_0 为 $\{\langle 0,0,v\rangle\}$）生成需求上界函数 rbf_T^v，来评价任意 v 起始点路径（指数级规模）的累积工作量上界。在第 4 行，算法记录函数 rbf_T^v 在区间 $[0,\rho)$ 内的所有上升点。从第 5 行至第 13 行的循环体依次遍历每个候选整形顶点 u（$\neq v$）来计算最大的释放延迟时间。算法第 6 行使用 图6.6 介绍的方法为每个顶点 u（$\neq v$）构造一条以 u 为起始点的候选支配路径 π。如果在第 8 行的判定条件满足 $rf_\pi \succcurlyeq_{\pi+\Delta(v)} rbf_T^v$，那么第 9 行和第 10 行将根据引理6.5来计算释放延迟的取值。然后第 11 行将已计算得到结果的最大值记录到变量 *ret* 中。最后，第 14 行返回三个参数取值中的最小值以满足引理6.5的要求。

6.3.5 整形算法的性质

- 时间复杂度

给定一个 DRT 任务集合 τ，根据如 图6.5 所示的算法和 式 (6.3)，图6.4所示算法第

CalDelayBound(v, ρ_T, Δ_{slf})

1: $\rho \leftarrow \rho_T + \Delta_{slf}$
2: $ret \leftarrow 0$
3: Generate rbf_T^v for the v-started paths up to ρ
4: $RS \leftarrow \{p \mid rbf_T^v(p) < rbf_T^v(p^+)\}$
5: **for each** $u \in G(T) \setminus \{v\}$ **do**
6: $\quad \pi \leftarrow$ GenerateGreedyPath(u, ρ)
7: $\quad$ generate the *request function* rf_π for π
8: $\quad$ **if** $\forall t \in [0, \rho] \mid rbf_T^v(t) \le rf_\pi(t)$ **then**
9: $\quad\quad DS \leftarrow \{\langle p, p_d\rangle \mid p \in RS, rbf_T^v(p^+) > rf_\pi(0^+)\}$
10: $\quad\quad tem \leftarrow \min\{(p - p_d)/2 \mid \langle p, p_d\rangle \in DS\}$
11: $\quad\quad ret \leftarrow \max(ret, tem)$
12: $\quad$ **end if**
13: **end for**
14: **return** $\min\{ret, \rho_T, \Delta_{slf}\}$

图 6.7 计算释放延迟时间算法

Fig. 6.7 Algorithm for calculating release delay bound.

4 行可以在伪多项式时间内计算 $\Delta_{slf}(v)$ 的取值。如图6.7所示，在算法 *CalDelayBound* 中需要执行的迭代次数随任务集合 τ 中顶点数量呈线性增长，并且 图6.7 中所调用的所有子过程的时间复杂度也是伪多项式的。因此，计算 $\Delta_{itf}(v)$ 算法的整体时间复杂度是伪多项式级别的。

另外，图6.4中嵌套循环的次数也不会超过任务集合 τ 中顶点的数量，所以图6.4中给出的完整整形算法的时间复杂度是伪多项式级别的。

• 严格提升可调度性

定理 6.6：给定 DRT 任务集合 τ 以及 DRT 任务 $T \in \tau$，并且 T 是在优先级次序 $\mathscr{P}$ 设置下为静态优先级可调度的。则在如图6.4所示算法的任意整形操作之后，该顶点仍然是静态优先级可调度的。

证明：考虑通过如图6.4所示算法进行整形的任意一个高优先级任务 $T' \in \tau | \mathscr{P}(T') < \mathscr{P}(T)$。

对于任意 $v' \in G(T')$，参数 Δ_{itf} 取值的上界可以通过 图6.7求得。根据 引理6.5可推出，顶点 v' 在整形操作中设置 Δ_{itf} 后不会导致任务 T' 在区间 $[0, \rho_{T'}]$ 内干涉工作量的增加。因此，对于任意 $v \in CS(\tau) \cap G(T)$，其需求函数在不大于 $d(v) \le \rho_{T'}$（根据 式(6.9)求得）时的取值不会因顶点 v' 的整形操作而增加。从而，所有 $G(T)$ 中的关键顶点将保持它们的可调度性，并且根据引理6.3可推出任务 T 在任意高优先级顶点的整形操作后仍将是可调度的。

最后考虑整形操作对于任务 T 自身可调度性的影响。根据 引理6.1可推出，对于任意整形顶点 v，在释放延迟设置不大于通过式 (6.3)计算得到的 Δ_{slf} 后，顶点 v 不会损失其原有的可调度性。

综上，该定理得证。 □

6.4 实验评价

本节实验的主要目的是评价本章提出的 DRT 任务有向图整形算法在提高 DRT 任务结合可调度性方面的有效性。实验方法是比较随机生成的任务集合在使用整形算法前后可调度样本数量的变化程度。

6.4.1 生成随机任务集合

与上一章的定义类似，本节定义 DRT 任务 T 的利用率为有向图 $G(T)$ 的所有简单环中累积执行时间与总释放间隔比值的最大值。进而定义所有任务利用率之和为任务集合的利用率。显然 DRT 任务可调度的一个必要条件为任务集合的利用率不能大于 1。

随机任务集合的生成过程与小节5.4.1 中的方法类似。生成一个随机 DRT 任务时，首先生成一个任务包含顶点个数的随机数，并创建该数量的顶点。各个顶点之间由满足各自出度约束的边相连接。对于每个随机顶点 v, 所有从该顶点起始的边对于的$e(v)$ 和 $p(v,u)$ 的取值都从它们分别对于的范围内随机选择，并服从均匀分布。而$d(v)$ 被设置为所有外出边(v,u) 中$p(v,u)$ 的最小取值。

生成随机任务集合的过程如下：首先，构建由两个随机生成的任务组成的任务集合，并进行实验评价。接下来向上一步生成的任务集合中添加一个新生成的随机任务并进行实验评价，知道任务集合的总体利用率超过1 时结束迭代。然后重新构建一个有两个随机任务组成的集合，并按照上述方法继续迭代。不断重复上述过程，直至生成并评价了足够数量的任务集合时算法结束。

为了评价整形算法对于不同类型任务结合的性能表现，本节设计了 light 任务，medium 任务和 heavy 任务三种不同随机参数设置的任务类型。具体的参数设置如表6.1 所示。这些不同类型之间的区别在于随机任务包含顶点个数的范围，顶点出度的范围，以及顶点的执行时间范围。本节分别采用不同的参数设置进行时间，并对实验结果进行比较。

表 6.1 不同随机任务类型对应的随机参数

Table 6.1 Different parameters to generate task sets with different type of tasks

类型	顶点数量	出度	p	e
Light	[7,15]	[1,3]	[50,300]	[1,4]
Medium	[7,15]	[1,4]	[50,300]	[1,6]
Heavy	[7,15]	[1,5]	[50,300]	[1,8]

6.4.2 实验结果分析

实验过程中对于随机任务集合中的任务分配静态优先级时，会按照每个任务 T 对应有向图 $G(T)$ 中所有顶点标注相对截止期的最小值对所有任务排序。其中具有更小相对截止期的任务将分配更高的优先级。如果有多个任务具有相同的最小相对截止期取值，那么实验将为这些任务随机分配优先级。

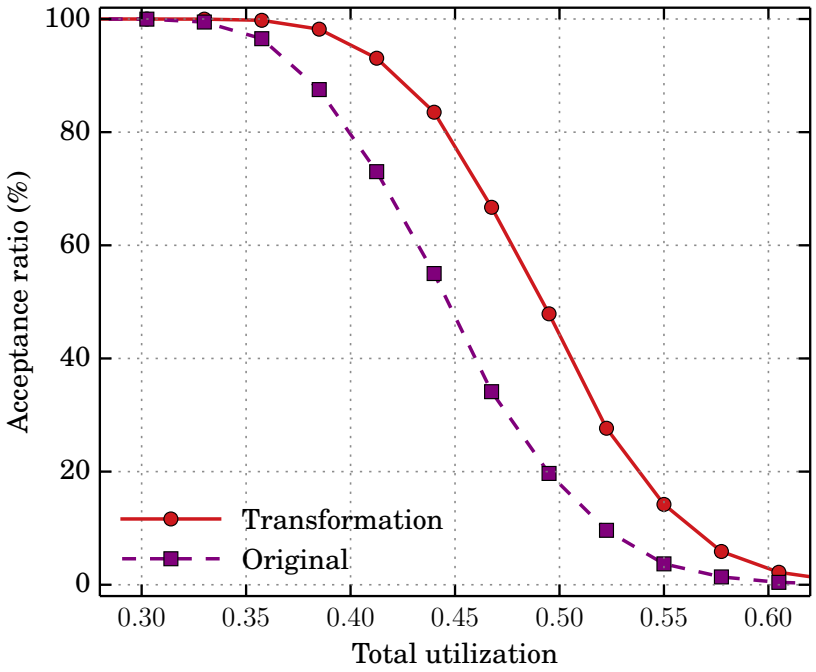
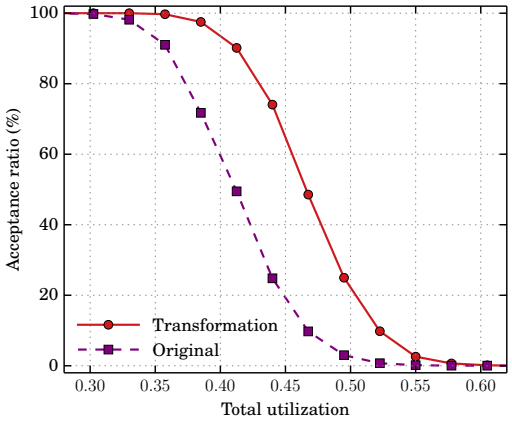


图 6.8 接受率提升效果

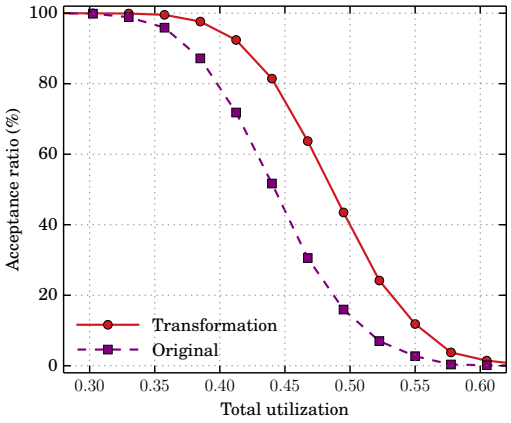
Fig. 6.8 Improvement of acceptance ratio.

本章提出 DRT 任务有向图整形算法的有效性的主要评价指标为接受率：随机生成任务集合中能够被调度的样本数量与总体样本数量的比值。本章的实验的接受率比较对象包括：

- *Transformation*：使用本章提出整形算法调整后的随机任务集合接受率。
- *Original*：未经过整形操作的原始随机任务集合使用文献 [138] 提出算法判定的接受率。

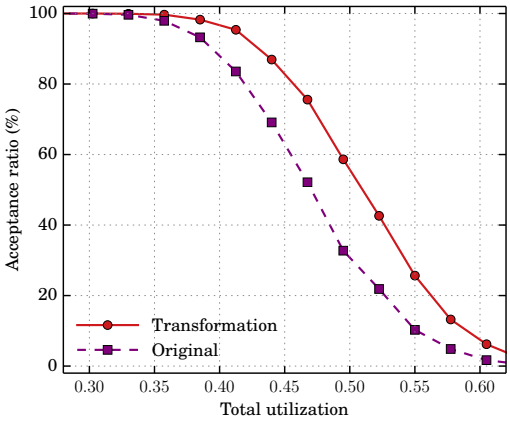


(a) subfloat



(b) ddd

(a) Improvement of LIGHT type of tasks.　(b) Improvement of MEDIUM type of tasks.



(c) subfloat

(c) Improvement of HEAVY type of tasks.

图 6.9 不同类型任务结合的接受率比较

Fig. 6.9 Comparison of acceptance ratio between different task types.

使用混合任务类型（每次生成随机任务时，随机选择一个类型的随机参数）的随机实验结果如图6.8所示。采用不同任务类型的随机实验结果如图6.9的各个子图所示。对于每个实验结果图中的每个点，均包含了至少 5000 组随机生成任务集合的数据。从实验结果中可以发现，通过本章提出的整形算法，随机任务集合的接受率在对不同任务类型的实验结果中均有显著地提高。而对于低利用率的随机任务集合（小于 0.3），无论是否采用整形操作接受率均为 100%，因此整形算法没有对接受率的提升空间。随着随机任务集合利用率的增加，高利用率随机任务集合变得很难被调度，这也造成接受率的提升幅度的下降。

本节实验还对本章提出整形算法的时间开销进行了评价。实验程序使用 Python 语言实现，并在一台安装 64 位 Linux 操作系统的桌面电脑（Intel Core i7-2600）上执

行。本章提出的整形算法和文献 [138] 提出的原始方法都会执行一次精确的可调度性分析程序。不同之处在于，整形算法会在可调度性判定之前调用一次整形程序。但实验结果表明，相对于精确可调度性分析过程，整形过程引入的额外时间开销非常低。一般情况下整形过程的额外开销不超过可调度性分析开销的 5%。由此可以说明，任务有向图整形算法是非常高效的，并可以应用到大规模任务系统中。

6.5 小结

本章提出了用于提高 DRT 任务系统可调度性的任务有向图整形算法。整形操作的目标是为每个整形任务的顶点设置一个合适作业释放时间延迟参数。但是，一般来讲延迟作业的释放时间会对不同的路径分别造成有利于和不利于系统可调度性的影响。整形算法解决的主要问题是如何快速地为每个任务的每个顶点选择合适的释放延迟，以尽可能大的提升不可调度任务集合的可调度可能性。本章提出了高效率的技术来解决上述问题，既能保证不影响被整形任务的可调度性，又可以保证低优先级的关键顶点受到的干涉工作量不会增加。这一性质能够高效地引导整形过程快速地得到高质量的结果。基于随机生成任务集合的实验表明本章提出的技术执行非常快速，并且能够显著地提升 DRT 任务系统的可调度性。

第 7 章 结 论

众所周知，传统实时调度很难直接应用到混合关键性系统中，针对该问题本文提出了线性时间复杂度的 OCBP 调度算法，并研究了混合关键性系统中的多核（处理器）划分调度问题。此外，此外在实时系统模型方面传统的周期实时系统模型已无法精确描述复杂系统，而基于有向图的 D R T 模型具备强大的系统描述能力，但时间复杂度大，相关技术还不成熟。由此本文提出了近似的响应时间分析方法，给出了基于加速比的量化评价，同时建立了高效的有向图整形算法并提升了 DRT 系统在固定优先级调度时的可调度性能。

随着处理器技术的不断发展，尤其是多核处理器平台在嵌入式系统中的广泛应用，在单一硬件平台中集成众多不同关键性级别应用的混合关键性系统成为现代嵌入式实时系统发展的趋势。然而传统实时调度研究的成果很难直接应用到混合关键性系统中，近年来围绕混合关键性系统的研究也成为实时调度领域的研究热点。本文围绕该问题提出了线性时间复杂度的 OCBP 族调度算法，并研究了混合关键性系统中的多核（处理器）划分调度问题。由于现代嵌入式系统设计的日益复杂，传统基于周期的实时系统模型很难满足精确描述复杂系统的需求。基于有向图的 DRT 模型具备强大的系统描述能力，但是对于该模型的分析却面临着时间复杂度过大，相关技术不成熟等问题。本文围绕高效的 DRT 系统分析方法问题进行了研究，提出了近似的响应时间分析方法并给出了基于加速比的量化评价，同时本文还提出了高效的有向图整形算法以提升 DRT 系统在固定优先级调度时的可调度性能。

7.1 本文主要贡献与结论

7.1.1 基于混合关键系统

（1）提出了基于 OCBP 策略具有线性运行时时间复杂度的固定作业优先级单处理器混合关键性实时调度算法 LPA。之前基于 OCBP 算法的 LB、PLRS 等算法虽然成功将 OCBP 算法扩展到偶发任务模型，但其较高运行时复杂度限制了在实际系统中的应用。本文提出的 LPA 算法，在运行时尽可能晚的对各个任务的优先级进行调整，从而避免繁重且不必需的运行时优先级调整计算，只是当作业被释放时进行轻微优先级调整，进而有效的改善了系统复杂度（线性时间复杂度）。本文还提出了更精确的混合关键性系统忙碌周期上界的计算方法，使用该方法不但可以改善运行时的空间效率，还可以一定程度上提升系统的可调度性。

（2）提出了分别对高关键性任务和低关键性任务采用不同策略的混合划分调度算法 MPVD（Mixed-criticality Partitioning with Virtual Deadlines）。该算法首先使用最差适应（worst-fit (WF)）划分策略来分配高关键性任务，然后使用首次适应（first-fit）划分策略来分配低关键性任务，并在运行时采用 EY-VD 算法进行调度。通过混合划分策略，能够使高关键性的任务被均匀地分配到不同处理器（核心）中，以使得 EY-VD 算法能够有更多的空间来平衡不同关键性级别下的工作量，并提升系统的可调度性。MPVD 算法的性能会随着处理器（核心）数量的增加而出现明显的下降。为了解决该问题，本章提出了两个优化算法来进一步提升算法的性能。首先考虑到由于高关键性任务在所有处理器中的均匀分配，可能导致无法为利用率较高的低关键性任务适配到拥有足够剩余资源的处理器，造成充足的处理器（核心）剩余资源总量无法被充分利用。针对该问题，本章提出了为利用率较高的低关键性任务预留资源的策略。另外，本章还提出了一种优化的虚拟截止期调整算法来进一步提升 MPVD 算法的性能。

（3）提出了为不同的系统关键性模式采用不同的任务集合划分方案的 OCOP 混合关键性多处理器划分调度策略。本文首先基于传统划分调度策略，提出了多处理器混合关键性系统中的划分调度算法 MC-PEDF。为了解决混合关键性系统在不同系统关键性级别下任务工作量分布差异较大的问题，本文放松了传统划分策略禁止运行时作业迁移的限制，提出了 OCOP 划分策略。OCOP 允许在系统关键性模式切换时为任务重新分配处理器，从而显著提升了系统在不同关键性模式中的处理器资源利用率。最后本文本文还提出了基于 OCOP 策略改造的新划分调度算法 MC-MP-EDF。实验结果表明 MC-PEDF 和 MC-MP-EDF 算法在可调度性上优于先前的多处理器混合关键性实时调度算法，而采用 OCOP 划分调度策略的 MC-MP-EDF 算法则具有更好的可调度性能。

7.1.2 基于实时任务有向图模型

（1）提出了两种分析 DRT 任务系统响应时间的近似分析方法 RBF 和 IBF，并通过加速比分析，量化评价了该两种近似方法的性能。其中基于加速比的的性能评价被广泛应用于众多调度问题的近似算法分析中。本章的主要成果可总结如下：

- RBF 近似响应时间分析方法的精确加速比为 2（即便是仅包含两个任务的系统）。
- IBF 近似响应时间分析方法的加速比为 $1+\frac{\sqrt{k^2-k}}{k}$，其中 k 为干涉任务（优先级高于当前分析任务者）的数量。

因为当 $k=1$ 时有 $1+\frac{\sqrt{k^2-k}}{k}=1$，所以对于只有两个任务的系统 IBF 方法能够得到精确解。另外由于 $1+\frac{\sqrt{k^2-k}}{k}$ 为以 k 为自变量的单调递增函数，因此 IBF 方法的分析精度随着干涉任务数量的增加而降低。而当 k 趋于无穷时，IBF 的加速比也趋近于 2（与 RBF 一致）。这两种方法均为伪多项式时间复杂度，并可以很高效地处理大规模的任务系统。

（2）提出了一种高效率的 DRT 任务有向图整形算法。本文提出的方法通过对 DRT 任务对应有向图中的每个顶点依次进行整形操作，并调整与被整形顶点相关联的边上参数取值。通过发掘任务图中的一些性质和进行合理的抽象，算法能够快速并且有效的完成整形操作。通过对一些关键顶点快速的设置合适的作业释放时间延迟参数，能够使得整形后的 DRT 任务释放的工作量更加均匀。从而使得一些不可调度的 DRT 任务系统变得可调度。实验结果表明本文提出的整形算能够显著提升 DRT 系统的可调度性能。

7.2 进一步的工作

随着嵌入式芯片架构的不断发展和嵌入式系统功能的日益复杂，使得现代嵌入式实时系统所面临的调度问题更为复杂。基于已经获得的研究成果，在未来工作中，将在以下几个方面进一步深入研究：

7.2.1 基于混合关键系统

（1）本文研究的基于 OCBP 的实时调度算法 LPA 采用的是单处理模型。而现代嵌入式系统越来越多的采用多核心的处理器架构。因此，未来应继续研究将 OCBP 族的算法扩展到多核/处理器平台的方法。

（2）本文研究的多核/处理器划分调度算法采用的是同质处理器模型，即每个处理器（核心）的架构和计算速度都是相同的。但是当今的多核处理器发展发展方向是更多的采用异构的架构，比如将 GPU 以及不同架构和主频的处理器核心集成到同一芯片中。因此未来将继续展开异构多核（处理器）平台上的混合关键性系统实时调度问题。

7.2.2 基于实时任务有向图模型

（1）本文研究的 DRT 系统的近似响应时间分析方法 RBF 和 IBF 在对抢占工作量的描述上均有不足之处，而且本文并没有证明 IBF 近似分析方法加速比的精确性。未来工作中将进一步分析 IBF 近似分析方法的精确加速比，并在深入研究 DRT 系统抢占工作量性质的基础上提出精确度更高的新型近似分析方法。

（2）本文研究的 DRT 系统中任务都是相互独立的，即所有任务释放的作业之间不存在共享资源的问题。但是 DRT 系统的一个主要优点是对实际系统的描述能力强，而增加对实际系统中广泛存在的共享资源问题的描述将使得 DRT 系统的描述能力更强，但也同时也会大大增加研究的难度。未来工作中将进一步研究考虑共享资源的 DRT 系统分析问题。

参考文献

1. Krishna C. Real-Time Systems [M], Wiley Online Library, 1999.
2. Liu J. Real-time systems [M], America: Prentice Hall, 2000.
3. Buttazzo G C. Hard Real-Time Computing Systems [M], Springer, 2011.
4. Stankovic J A, Ramamritham K. Tutorial on Hard real-time systems [J], 1988.
5. AUTOSTAR [EB/OL], 2015. http://www.autosar.org/.
6. ARINC [EB/OL], 2015. http://www.arinc.com/.
7. Liu C L, Layland J W. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment [J], J. ACM, 1973, 20(1): 46–61.
8. Baruah S, Mok A, Rosier L. Preemptively scheduling hard-real-time sporadic tasks on one processor [A], Proceedings of the 11th IEEE Real-Time Systems Symposium (RTSS) [C], 1990, 182–190.
9. Baruah S. Feasibility analysis of recurring branching tasks [A], Proceedings of Real-Time Systems, 1998. Proceedings. 10th Euromicro Workshop on [C]. IEEE, 1998, 138–145.
10. Baruah S. Dynamic-and static-priority scheduling of recurring real-time tasks [J], Real-Time Systems, 2003, 24(1): 93–128.
11. Baruah S. The Non-cyclic Recurring Real-Time Task Model [A], Proceedings of the IEEE 31st Real-Time Systems Symposium (RTSS) [C], 2010, 173–182.
12. Moyo N T, Nicollet E, Lafaye F, et al. On schedulability analysis of non-cyclic generalized multiframe tasks [A], Proceedings of the 22nd Euromicro Conference on Real-Time Systems (ECRTS) [C]. IEEE, 2010, 271–278.
13. Davis R, Feld T, Pollex V, et al. Schedulability tests for tasks with Variable Rate-dependent Behaviour under fixed priority scheduling [A], Proceedings of Real-Time and Embedded Technology and Applications Symposium (RTAS), 2014 IEEE 20th [C], 2014, 51 – 62.
14. Palencia J C, Harbour M G. Schedulability Analysis for Tasks with Static and Dynamic Offsets [A], Proceedings of 2013 IEEE 34th Real-Time Systems Symposium [C], 2001, 26.
15. Kalyanasundaram B, Pruhs K. Speed is as powerful as clairvoyance [scheduling problems] [A], Proceedings of the 36th Annual Symposium on Foundations of Computer Science [C], 1995, 214–221.

16. Maki-Turja J, Nolin M. Fast and Tight Response-Times for Tasks with Offsets [A], Proceedings of the 17th Euromicro Conference on Real-Time Systems [C], 2005, 127 – 136.
17. Searcoid M. Metric spaces [M], Springer undergraduate mathematics series, Springer-Verlag, 2006.
18. Stigge M. Real-time workload models: Expressiveness vs analysis efficiency [D], Uppsala University, 2014.
19. Stigge M, Ekberg P, Guan N, et al. The digraph real-time task model [A], Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) [C]. IEEE, 2011, 71–80.
20. Vestal S. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance [A], Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS) [C], 2007, 239–243.
21. Audsley N, Burns A, Richardson M, et al. Applying new scheduling theory to static priority preemptive scheduling [J], Software Engineering Journal, 1993, 8(5): 284–292.
22. Leung J Y T, Whitehead J. On the complexity of fixed-priority scheduling of periodic real-time tasks [J], Performance Evaluation (Netherlands), 1982, 2(4): 237–250.
23. Audsley N. On priority assignment in fixed priority scheduling [J], Information Processing Letters, 2001, 79(1): 39–44.
24. Baruah S, Bonifaci V, D`Angelo G, et al. Scheduling Real-Time Mixed-Criticality Jobs [J], IEEE Transactions on Computers, 2012, 61(8): 1140–1152.
25. Burns A. System mode changes - general and criticality-based [A], Proceedings of 2nd Workshop on Mixed Criticality Systems (WMC), RTSS [C], 2014, 3–8.
26. Graydon P, Bate I. Safety assurance driven problem formulation for mixedcriticality scheduling [A], Proceedings of WMC, RTSS [C], 2013, 19–24.
27. Burns A, Quiggle T. Effective use of abort in programming mode changes [J], Ada Letter, 1990.
28. Emberson P, Bate I. Minimising task migrations and priority changes in mode transitions [A], Proceedings of the 13th IEEE Real-Time And Embedded Technology And Applications Symposium (RTAS 07) [C], 2007, 158–167.
29. Pedro P, Burns A. Schedulability analysis for mode changes in flexible realtime systems [A], Proceedings of 10th Euromicro Workshop on Real-Time Systems [C], 1998, 172–179. IEEE Computer Society.

30. Real J, Crespo A. Mode change protocols for real-time systems: A survey and a new protocol [J], Journal of Real-Time Systems, 2004, 26(2): 161–197.
31. Sha L, Rajkumar R, Lehoczky J, et al. Mode change protocols for priority-driven preemptive scheduling [J], Journal of Real-Time Systems, 1989, 1(3): 244 264.
32. Tindell K, Alonso A. A very simple protocol for mode changes in priority preemptive systems [R], Technical report, Technical report, Universidad Politecnica de Madrid, 1996.
33. Tindell K, Burns A, Wellings A J. Mode changes in priority preemptive scheduled systems [A], Proceedings of Real Time Systems Symposium, pages 100-109, Phoenix, Arizona [C], 1992.
34. Baruah S. Mixed criticality schedulability analysis is highly intractable [R], Technical report, Technical report, University of North Carolina at Chapel Hill, 2009.
35. Baruah S, Bonifaci V, D`Angelo G, et al. Scheduling real-time mixed-criticality jobs [A], Proceedings of the 35th International Symposium on the Mathematical Foundations of Computer Science, volume 6281 of Lecture Notes in Computer Science [C], 2010, 90–101. Springer.
36. Baruah S, Li H, Stougie L. Towards the Design of Certifiable Mixed-criticality Systems [A], Proceedings of the 16th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) [C], 2010, 13–22.
37. Baruah S, Guo Z. Mixed criticality scheduling upon unreliable processors [R], Technical report, Technical report, University of North Carolina at Chapel Hill, 2013.
38. Gu C, Guan N, Deng1 Q, et al. Improving ocbp-based scheduling for mixedcriticality sporadic task systems [A], Proceedings of RTCSA [C], 2013.
39. Li H, Baruah S. An Algorithm for Scheduling Certifiable Mixed-Criticality Sporadic Task Systems [A], Proceedings of the 31st IEEE Real-Time Systems Symposium (RTSS) [C], 2010, 183–192.
40. Li H, Baruah S. Load-based schedulability analysis of certifiable mixed-criticality systems [A], Proceedings of the tenth ACM international conference on Embedded software [C]. ACM, 2010, 99–108.
41. Park T, Kim S. Dynamic scheduling algorithm and its schedulability analysis for certifiable dual-criticality systems [A], Proceedings of ACM EMSOFT [C], 2011, 253–262.
42. Socci D, Poplavko P, Bensalem S, et al. Mixed critical earliest deadline first [R], Technical report, Technical Report TR-2012-22, Verimag Research Report, 2012.

43. Socci D, Poplavko P, Bensalem S, et al. Mixed critical earliest deadline first [A], Proceedings of Euromicro Conference on Real-Time Systems (ECRTS) [C], 2013.
44. Dorin F, Richard P, Richard M, et al. Schedulability and sensitivity analysis of multiple criticality tasks with fixed-priorities [J], Real-Time Systems, 2010, 46(3): 305–331.
45. Baruah S K, Bonifaci V, D`Angelo G, et al. Mixed-criticality scheduling of sporadic task systems, Proceedings of Algorithms-ESA [C]. 2011: .
46. Audsley N C. Optimal priority assignment and feasibility of static priority tasks with arbitrary start times [M], Citeseer, 1991.
47. Guan N, Ekberg P, Stigge M, et al. Effective and Efficient Scheduling of Certifiable Mixed-Criticality Sporadic Task Systems [A], Proceedings of the 32nd IEEE Real-Time Systems Symposium (RTSS) [C], 2011, 13–23.
48. Zhao Q, Gu Z, Zeng H. Integration of resource synchronization and preemption-thresholds into EDF-based mixed-criticality scheduling algorithm [A], Proceedings of RTCSA [C], 2013.
49. Zhao Q, Gu Z, Zeng H. PT-AMC: Integrating preemption thresholds into mixed-criticality scheduling [A], Proceedings of DATE [C], 2013, 141–146.
50. Saksena M, Wang Y. Scaleable real-time systems design using preemption thresholds [A], Proceedings of Proceeding 21st IEEE Real-Time Systems Symposium. [C], 2000, 25–34.
51. Burns A, Davis R. Adaptive mixed criticality scheduling with deferred preemption [A], Proceedings of IEEE Real-Time Systems Symposium [C], 2014, 21–30. IEEE.
52. Burns A. Preemptive priority based scheduling: An appropriate engineering approach [A], Proceedings of S.H. Son, editor, Advances in Real-Time Systems [C], 1994, 225–248. Prentice–Hall.
53. Davis R, Bertogna M. Optimal fixed priority scheduling with deferred preemption [A], Proceedings of IEEE Real-Time Systems Symposium [C], 2012, 39–50.
54. Baruah S, Burns A, Davis R. Response-Time Analysis for Mixed Criticality Systems [A], Proceedings of the 32nd IEEE Real-Time Systems Symposium (RTSS) [C], 2011, 34–43.
55. Fleming T, Burns A. Extending mixed criticality scheduling [A], Proceedings of WMC, RTSS [C], 2013, 7–12.
56. Huang H M, Gill C, Implementation C L, et al. ACM Trans [J], Embedded Systems, 2014, 13: 126:1– 126:25.

57. Baruah S, Burns A, Davis R. An extended fixed priority scheme for mixed criticality systems [A], Proceedings of ReTiMiCS, RTCSA [C], 2013, 18–24.
58. Baruah S, Chattopadhyay B. Response-time analysis of mixed criticality systems with pessimistic frequency specification [A], Proceedings of RTCSA [C], 2013.
59. Burns A, Davis R. Mixed criticality on controller area network [A], Proceedings of Euromicro Conference on Real-Time Systems (ECRTS) [C], 2013, 125–134.
60. Niz D, Lakshmanan K, Rajkumar R. On the Scheduling of Mixed-Criticality Real-Time Task Sets [A], Proceedings of the 30th IEEE Real-Time Systems Symposium (RTSS) [C], 2009, 291–300.
61. Lakshmanan K, Niz D, Rajkumar R, et al. Resource Allocation in Distributed Mixed-Criticality Cyber-Physical Systems [A], Proceedings of the 30th IEEE International Conference on Distributed Computing Systems (ICDCS) [C], 2010, 169–178.
62. Lakshmanan K, Niz D, Rajkumar R. Mixed-Criticality Task Synchronization in Zero-Slack Scheduling [A], Proceedings of the 17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) [C], 2011, 47–56.
63. Huang H M, Gill C, Lu C. Implementation and evaluation of mixed criticality scheduling approaches for periodic tasks [A], Proceedings of the IEEE Real-Time Technology and Applications Symposium (RTAS) [C], 2012, 23–32.
64. Niz D, Phan L. Partitioned scheduling of multi-modal mixed-criticality real-time systems on multiprocessor platforms [A], Proceedings of Real-Time and Embedded Technology and Applications Symposium (RTAS) [C], April 2014, 111–122.
65. Niz D, Wrage L, Rowe A, et al. Utility-based resource overbooking for cyber-physical systems [A], Proceedings of RTCSA [C], 2013.
66. Neukirchner M, Axer P, Michaels T, et al. Monitoring of workload arrival functions for mixed-criticality systems [A], Proceedings of IEEE 34th Real-Time Systems Symposium [C], 2013, 88–96.
67. Neukirchner M, Quinton S, Lampka K. Multi-mode monitoring for mixedcriticality real-time systems [A], Proceedings of Int’l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS) [C], 2013.
68. Baruah S, Vestal S. Schedulability Analysis of Sporadic Tasks with Multiple Criticality Specifications [A], Proceedings of the 20th Euromicro Conference on Real-Time Systems (ECRTS) [C], 2008, 147–155.

69. Ekberg P, Yi W. Outstanding Paper Award: Bounding and Shaping the Demand of Mixed-Criticality Sporadic Tasks [A], Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS) [C], 2012, 135–144.
70. Ekberg P, Yi W. Bounding and shaping the demand of generalized mixedcriticality sporadic task systems [J], Journal of Real-Time Systems, 2014, 50: 48–86.
71. Guan N, Yi W. Improveing the scheduling of certifiable mixed criticality sopradic task systems [R], Technical report, Technical report, University of Uppsala, 2012.
72. Easwaran A. Demand-based scheduling of mixed-criticality sporadic tasks on one processor [A], Proceedings of IEEE 34th Real-Time Systems Symposium [C], 2013, 78–87.
73. C Yao L Z, Huagang X. Efficient schedulability analysis for mixed-criticality systems under deadline-based scheduling [J], Chinese Journal of Aeronautics, 2014.
74. Zhang F, Burns A. Schedulability analysis for real-time systems with EDF scheduling [J], IEEE Transaction on Computers, 2008, 58(9): 1250–1258.
75. Lipari G, Buttazzo G. Resource reservation for mixed criticality systems [A], Proceedings of Proceeding ofWorkshop on Real-Time Systems: the past, the present, and the future, pages 60-74, York, UK [C], 2013.
76. Su H, Zhu D. An elastic mixed-criticality task model and its scheduling algorithm [A], Proceedings of the Conference on Design, Automation and Test in Europe, DATE [C], 2013, 147–152.
77. Su H, Zhu D, Mosse D. Scheduling algorithms for elastic mixed-criticality tasks in multicore systems [A], Proceedings of RTCSA [C], 2013.
78. Buttazzo G, Lipari G, Abeni L. Elastic task model for adaptive rate control [A], Proceedings of IEEE Real-Time Systems Symposium [C], 1998, 286–295.
79. Anderson J, Baruah S, Brandenburg B. Multicore operating-system support for mixed criticality [A], Proceedings of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification, San Francisco [C], 2009.
80. Mollison M, Erickson J, Anderson J, et al. Mixed criticality real-time scheduling for multicore systems [A], Proceedings of the 7th IEEE International Conference on Embedded Software and Systems [C], 2010, 1864–1871.
81. Herman J, Kenna C, Mollison M, et al. RTOS support for multicore mixed-criticality systems [A], Proceedings of the 18th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) [C], 2012.

82. Mollison M, Erickson J, Anderson J, et al. Mixed-Criticality Real-Time Scheduling for Multicore Systems [A], Proceedings of the 10th IEEE International Conference on Computer and Information Technology (CIT) [C], 2010, 1864–1871.
83. Bommert M. Schedule-aware distributed of parallel load in a mixed criticality environment [A], Proceedings of JRWRTC, RTNS [C], 2013, 21–24.
84. Liu G, Lu Y, Wang S, et al. Partitioned multiprocessor scheduling of mixedcriticality parallel jobs [A], Proceedings of IEEE Embedded and Real-Time Computing Systems and Applications (RTCSA) [C], 2014.
85. Baruah S. Implementing mixed criticality synchronous reactive systems upon multiprocessor platforms [R], Technical report, Technical report, University of North Carolina at Chapel Hill, 2013.
86. Tamas-Selicean D, Pop P. Design optimisation of mixed criticality real-time applications on cost-constrained partitioned architectures [A], Proceedings of Real-Time Systems Symposium (RTSS) [C], 2011, 24–33.
87. Tamas-Selicean D, Pop P. Optimization of time-partitions for mixed criticality real-time distributed embedded systems [A], Proceedings of 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops [C], 2011, 2–10.
88. Tamas-Selicean D, Pop P. Task mapping and partition allocation for mixed criticality real-time systems [A], Proceedings of IEEE Pacific Rim Int. Sym. on Dependable Computing [C], 2011, 282–283.
89. Zhang X, Zhan J, Jiang W, et al. Design optimization of securitysensitive mixed-criticality real-time embedded systems [A], Proceedings of ReTiMiCS, RTCSA [C], 2013, 12–17.
90. David E. Goldberg: Genetic Algorithms in search, optimization, and machine learning [J], Compact Mri for Tb of the Distal Radius, 1989.
91. Alonso A, Salazar E, Miguel M. A toolset for the development of mixedcriticality partitioned systems [A], Proceedings of HiPEAC Workshop [C], 2014.
92. Kelly O R, Aydin H, Zhao B. On partitioned scheduling of fixed-priority mixed-criticality task sets [A], Proceedings of TrustCom [C], 2011.
93. Rodriguez P, George L, Abdeddaim Y, et al. Multi-criteria evaluation of partitioned EDF-VD for mixed criticality systems upon identical processors [A], Proceedings of WMC, RTSS [C], 2013, 49–54.

94. Gratia R, Robert T, Pautet L. Adaptation of RUN to mixed-criticality systems [A], Proceedings of 8th Junior Researcher Workshop on Real-Time Computing, RTNS [C], 2014.
95. Regnier P, Lima G, Massa E, et al. RUN: Optimal multiprocessor real-time scheduling via reduction to uniprocessor [A], Proceedings of Real-Time Systems Symposium (RTSS) [C], 2011, 104–115. IEEE.
96. Bletsas K, Petters S. Using NPS-F for mixed criticality systems [A], Proceedings of WiP, RTSS, page 25 [C], 2012.
97. Bletsas K, Petters S. Using NPS-F for mixed criticality multicore ystems [R], Technical report, Cister-tr-130303, CISTER, 2013.
98. Axer P, Sebastian M, Ernst R. Reliability analysis for mpsocs with mixedcritical, hard real-time constraints [A], Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis, CODES+ISSS ’11 [C], 2011, 149–158. ACM.
99. Lee J, Phan K M, Gu X, et al. MC-Fluid: Fluid model-based mixed-criticality scheduling on multiprocessors [A], Proceedings of IEEE Real Time Systems Symposium [C], 2014, 41–52. IEEE.
100. Holman P, Anderson J. Adapting Pfair scheduling for symmetric multiprocessors [J], Journal of Embedded Computing, 2005, 1(4): 543–564.
101. Li H, Baruah S. Outstanding Paper Award: Global Mixed-Criticality Scheduling on Multiprocessors [A], Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS) [C], 2012, 166–175.
102. Baruah S. Optimal utilization bounds for fixed priority scheduling of periodic task systems on identical multiprocessors [J], IEEE Transactions on Software Engineering, 2004, 53(6): 781 – 784.
103. Baruah S. Optimal utilization bounds for the fixed-priority scheduling of periodic task systems on identical multiprocessors [J], IEEE Transactions on Computers, 2004, 53(6).
104. Baruah S, Chattopadhyay B, Li H, et al. Mixed-criticality scheduling on multiprocessors [J], Real-Time Systems, 2013.
105. Pathan R. Schedulability Analysis of Mixed-Criticality Systems on Multiprocessors [A], Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS) [C], 2012, 309–320.

106. Kritikakou A, Baldellon O, Pagetti C, et al. Monitoring on-line timing information to support mixed-critical workloads [A], Proceedings of WiP, RTSS [C], 2013, 9–10.
107. Kritikakou A, Pagetti C, Rochange C, et al. Distributed run-time WCET controller for concurrent critical tasks in mixedcritical systems [A], Proceedings of RTNS [C], 2014.
108. Socci D, Poplavko P, Bensalem S, et al. Multiprocessor scheduling of precedence-constrained mixed-critical jobs [R], Technical report, Technical Report TR-2014-11, Verimag, Research Report, 2014.
109. Cazorla F, Quiones E, Vardanega T, et al. PROARTIS: Probabilistically analyzable real-time systems [J], ACM Trans. Embedded Comput. Syst., 2013, 12(2): 94.
110. Altmeyer S, Cucu-Grosjean L, Davis R. Static probabilistic timing analysis for real-time systems using random replacement caches [J], Real-Time Systems, 2015, 51(1): 77–123.
111. Cucu-Grosjean L, Santinelli L, Houston M, et al. Measurement-based probabilistic timing analysis for multi-path programs [A], Proceedings of 24th Euromicro Conference on Real-Time Systems (ECRTS) [C], 2012, 91–101.
112. Davis R, Santinelli L, Altmeyer S, et al. Analysis of probabilistic cache related pre-emption delays [A], Proceedings of ECRTS [C], 2013, 129–138.
113. Edgar S, Burns A. Statistical analysis of WCET for scheduling [A], Proceedings of 22nd IEEE Real-Time Systems Symposium [C], 2001.
114. Baruah S. Certification-cognizant scheduling of tasks with pessimistic frequency specification [A], Proceedings of 7th IEEE International Symposium on Industrial Embedded Systems (SIES'12) [C], 2012, 31–38.
115. Baruah S. Response-time analysis of mixed criticality systems with pessimistic frequency specification [R], Technical report, Technical report, University of North Carolina at Chapel Hill, 2013.
116. Baruah S, Burns A. Implementing mixed criticality systems in Ada [A], Proceedings of Reliable Software Technologies - Ada-Europe 2011 [C], 2011, 174–188. Springer.
117. Burns A, Baruah S. Timing faults and mixed criticality systems [J], Lecture Notes in Computer Science, 2011. 147–166.
118. Fersman E, Krcal P, Pettersson P, et al. Task automata: Schedulability, decidability and undecidability [J], Information & Computation, 2007, 205(8): 1149–1172.
119. Mok A K L. Fundamental design problems of distributed systems for the hard-real-time environment [M], volume 1, MIT Thesis, May, 1983.

120. Mok A K, Chen D. A multiframe model for real-time tasks [J], IEEE Transactions on Software Engineering, 1997, 23(10): 635–645.

121. Baruah S, Chen D, Gorinsky S, et al. Generalized multiframe tasks [J], Real-Time Systems, 1999, 17(1): 5–22.

122. Chakraborty S, Erlebach T, Thiele L. On the Complexity of Scheduling Conditional Real-Time Code [J], Proc International Workshop on Algorithms & Data Structures Lecture Notes in Computer Science, 2003. 38–49.

123. Stigge M, Yi W. Hardness Results for Static Priority Real-Time Scheduling [A], Proceedings of the 24th Euromicro Conference on Real-Time Systems (ECRTS) [C]. IEEE, 2012, 189–198.

124. Takada H, Sakamura K. Schedulability of generalized multiframe task sets under static priority assignment [A], Proceedings of 2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications [C], 1997, 80.

125. Stigge M, Guan N, Yi W. Refinement-based exact response-time analysis [A], Proceedings of the 26th EUROMICRO Conference on Real- Time Systems (ECRTS) [C], 2014.

126. Gringeri S, Shuaib K, Egorov R, et al. Traffic shaping, bandwidth allocation, and quality assessment for mpeg video distribution over broadband networks [J], IEEE Networks, 1998, 12(6): 94–107.

127. Rexford J, Bonomi F, Greenberg A, et al. Scalable architectures for integrated traffic shaping and link scheduling in high-speed ATM switches [J], IEEE Journal on Selected Areas of Communication, 1997, 15(5): 938–950.

128. Wandeler E, Maxiaguine A, Thiele L. Performance analysis of greedy shapers in real-time systems [A], Proceedings of Design, Automation and Test in Europe, 2006. DATE'06. Proceedings [C], volume 1. IEEE, 2006, 6–pp.

129. Wandeler E, Maxiaguine A, Thiele L. On the use of greedy shapers in real-time embedded systems [J], ACM Transactions on Embedded Computing Systems (TECS), 2012, 11(1): 1.

130. Le Boudec J Y, Thiran P. Network calculus: a theory of deterministic queuing systems for the internet [M], volume 2050, Springer, 2001.

131. Richter K, Jersak M, Ernst R. A formal approach to MpSoC performance verification [J], IEEE Computer, 2003, 36(4): 60–67.

132. Phan L, Lee I. Improving schedulability of fixed-priority real-time systems using shapers [A], Proceedings of the IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS) [C], 2013, 217–226.
133. Thiele L, Chakraborty S, Naedele M. Real-time calculus for scheduling hard real-time systems [A], Proceedings of the 2000 IEEE International Symposium on Circuits and Systems (ISCAS) [C]. IEEE, 2000, 101–104.
134. Bastoni A, Brandenburg B B, Anderson J H. An empirical comparison of global, partitioned, and clustered multiprocessor EDF schedulers [A], Proceedings of RTSS [C], 2010.
135. Carpenter J, Funk S, Holman P, et al. A categorization of real-time multiprocessor scheduling problems and algorithms [J], Proceedings of the IEEE, 2004.
136. Johnson D S. Near-optimal bin packing algorithms [D], Boston: Massachusetts Institute of Technology, 1973.
137. Stigge M, Ekberg P, Guan N, et al. On the tractability of digraph-based task models [A], Proceedings of 24th Euromicro Conference on Real-Time Systems (ECRTS) [C], 2011.
138. Stigge M, Yi W. Combinatorial Abstraction Refinement for Feasibility Analysis [A], Proceedings of Procedings of the 34th IEEE Real-Time Systems Symposium (RTSS) [C], 2013, 340–349.
139. Gill A. Finite-State Machines [J], IEEE Transactions on Computers, 1970, 19(11).
140. Natale M D, Zeng H. Task implementation of synchronous finite state machines [A], Proceedings of DATE [C], 2012, 206–211.
141. Zeng H, Natale M D. Schedulability Analysis of Periodic Tasks Implementing Synchronous Finite State Machines [A], Proceedings of ECRTS [C], 2012, 353–362.
142. Bernacki M. Simulink Stateflow [EB/OL], http://www.mathworks.cn/products/stateflow/.
143. Joseph M, Pandya P. Finding response times in a real-time system [J], The Computer Journal, 1986, 29(5): 390–395.

攻博期间发表的论文

1. **Myself**, Nan Guan, **Supervisor** and Wang Yi. Improving OCBP-based Scheduling for Mixed-Criticality Sporadic Task Systems. 2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), p 247-56, 2013. (EI:20141817683493)
2. **Myself**, Nan Guan, **Supervisor** and Wang Yi. Partitioned mixed-criticality scheduling on multiprocessor platforms. 2014 Design, Automation & Test in Europe. Conference & Exhibition (DATE), p 6 pp., 2014. (EI:20142817930667)
3. **Myself**, 于金铭, 关楠, 王义, **Supervisor**. 多处理器混合关键性系统中划分调度策略的研究. 软件学报, 2014. (EI:20141017430645)
4. Nan Guan, **Myself**, Martin Stigge, **Supervisor** and Wang Yi. Approximate Response Time Analysis of Real-Time Task Graphs. IEEE Real-Time Systems Symposium (RTSS), 2014.
5. Nan Guan, Meiling Han, **Myself**, **Supervisor** and Wang Yi. Bounding Carry-in Interference to Improve Fixed-Priority Global Multiprocessor Scheduling Analysis. 2015 IEEE 21th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2015. (Accepted)

攻博期间参与的项目

1. 国家自然科学基金（项目编号：0973017），多核系统中实时调度策略的设计与分析技术的研究，2009.12 – 2012.12，主要研究人员.
2. 国家支撑计划（项目编号：2012BAK24B0104），基于物联网的地铁施工安全风险识别与可视化预警，2010.7 – 2014.7，主要研究人员.
3. 中央高校基本科研业务费重大创新（项目编号：N110804003），面向采掘业和巷道施工安全作业的物联网关键技术研究，2012.1 – 2014.12，主要研究人员.