计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2016.20150175
53(6):1186-1201,2016

异构系统双关键级分布式功能的动态调度

刘樑骄[1,2] 谢国琪[1,2] 李仁发[1,2] 杨 柳[3] 刘 彦[1,2]
[1](湖南大学信息科学与工程学院 长沙 410082)
[2](嵌入式与网络计算湖南省重点实验室(湖南大学) 长沙 410082)
[3](湖南省发展和改革委员会 长沙 410004)
(lljl984109@qq.com)

Dynamic Scheduling of Dual-Criticality Distributed Functionalities on Heterogeneous Systems

Liu Liangjiao[1,2], Xie Guoqi[1,2], Li Renfa[1,2], Yang Liu[3], and Liu Yan[1,2]
[1](*College of Computer Science and Electronic Engineering, Hunan University, Changsha* 410082)
[2](*Key Laboratory for Embedded and Network Computing of Hunan Province (Hunan University), Changsha* 410082)
[3](*Development and Reform Commission of Hunan Province, Changsha* 410004)

Abstract Heterogeneous distributed systems are mixed-criticality systems consisting of multiple functionalities with different criticality levels. A distributed functionality contains multiple precedence-constrained tasks. Mixed-criticality scheduling of heterogeneous distributed systems faces severe conflicts between performance and time constraints. Improving the overall performance of systems while still meeting the deadlines of higher-criticality functionalities, and making a reasonable tradeoff between performance and timing are major optimization problems. The F_DDHEFT(fairness of dynamic dual-criticality heterogeneous earliest finish time) algorithm is to improve the performance of systems. The C_DDHEFT (criticality of dynamic dual-criticality heterogeneous earliest finish time) algorithm is to meet the deadlines of higher-criticality functionalities. The D_DDHEFT (deadline-span of dynamic dual-criticality heterogeneous earliest finish time) algorithm is to allow the lower-criticality functionalities to be processed positively for better overall performance while still meeting the deadlines of higher-criticality functionalities, such that a reasonable tradeoff between performance and timing is made. Both example and extensive experimental evaluation demonstrate significant improvement of the D_DDHEFT algorithm.

Key words heterogeneous distributed embedded systems; dual-criticality; performance; real-time; deadline-span

摘 要 异构分布式嵌入式系统是由多种不同关键级功能组成的混合关键级系统,且每个功能又是由多个具有优先级约束的任务组成的分布式功能.异构分布式嵌入式系统的混合关键级调度在性能与时间

收稿日期:2015-02-27;**修回日期**:2015-09-06
基金项目:国家自然科学基金项目(61173036,61202102,61300039,61300037,61402170);国家"八六三"高技术研究发展计划基金项目(2012AA01A301-01);中国博士后科学基金项目(2016M592422)
This work was supported by the National Natural Science Foundation of China (61173036, 61202102, 61300039, 61300037, 61402170) and the National High Technology Research and Development Program of China (863 Program) (2012AA01A301-01), and the China Postdoctoral Science Foundation (2016M592422).
通信作者:谢国琪(xgqman@hnu.edu.cn)

约束上面临严重的冲突.如何提高系统总体性能,并仍然确保高关键级功能的实时性,在性能与实时性上取得合理的权衡则成为研究的主要优化问题.提出公平策略的动态双关键级任务调度算法 F_DDHEFT(fairness on dynamic dual-criticality heterogeneous earliest finish time)以提高系统的整体性能;提出关键级策略的动态双关键级任务调度算法 C_DDHEFT(criticality on dynamic dual-criticality heterogeneous earliest finish time)以满足高关键级功能的实时性;提出时限时距策略的动态双关键级任务调度算法 D_DDHEFT(deadline-span on dynamic dual-criticality heterogeneous earliest finish time),在满足高关键级功能实时性的基础上,提高系统的整体性能,最终在性能与时间约束上取得合理的权衡.实例分析和实验结果验证了 D_DDHEFT 算法的优越性.

关键词 异构分布式嵌入式系统;双关键级;性能;实时;时限时距

中图法分类号 TP316

新一代高端嵌入式系统是典型的异构分布式嵌入式系统.例如,汽车电子系统由 100 多个异构 ECU(electronic control unit)、各类传感器、执行器和网关等处理单元组成,并通过 CAN(controller area network),FlexRay,LIN(local interconnect network),MOST(media oriented system transport)和以太网等多种异构网络总线互联,系统体系结构的异构性日趋明显[1-3].当前,一辆豪华轿车至少包含 70 个 ECU 以及 2 500 个信号以完成各种复杂功能的执行[4].与此同时,具有优先级约束的分布式功能在任务数和复杂性上与日俱增.例如汽车电子系统中的线控刹车是典型的分布式主动安全功能,从感知数据开始到执行刹车指令结束,该功能所包含的计算任务存在明显的优先级约束[5].整个过程需分配给 5 个 ECU、1 个传感器和 2 个执行器等处理单元.不同的任务之间产生不同的通信信号并被打包成消息在网络总线上传输.

异构分布式嵌入式系统体系结构是一种典型的分布式集成体系结构,这种结构将多种安全与非安全关键功能集成到了同一个平台,并由此在学术界和工业界引入了关键级与混合关键级系统的概念,并成为复杂嵌入式系统的主要研究热点[6].“关键级”强调某个功能发生安全事故的严重性后果,以表示其重要程度,并经过了第三方机构的严格认证.高关键级功能相比低关键级功能意味着前者在时间需求上要求更加严格苛刻,错过高关键级功能的时隙将会造成严重的安全问题.关键级在汽车电子系统中用汽车安全完整性等级表示[7].道路车辆-功能安全标准规范“IS026262”对汽车电子系统中的功能从低到高划分成 A,B,C,D 四个 ASIL.其中,D 为最高关键级,其安全性需求最为苛刻;A 为最低等级,安全性需求最低.

任务调度是混合关键级系统的核心研究内容.通过高效的任务调度算法,可充分利用异构分布式嵌入式系统中的处理器和通信等资源以提高系统性能,但是面向具有不同关键级的分布式功能的调度却面临严峻的挑战.

首先,尽管混合关键级系统概念自诞生以来提出了众多的调度理论与算法,但是大多方法都是基于周期任务模型或随机任务模型,且都是从”任务级”的角度研究其调度问题[8-14].然而异构分布式嵌入式系统中的分布式功能所包含的任务存在明显的优先级约束,传统的”任务级”模型显然无法精确描述任务之间的优先级约束.如何从“功能级”的角度建立分布式的系统模型显得至关重要.近年来相关学者针对汽车电子系统提出了时序链[15]、功能链[16]和任务链[17]等功能模型以适应任务间的优先级约束问题,但仅限于简单的分布式功能.随着功能的日益复杂化与并行化,迫切需要一种能够更加精确反映功能特性的模型.在并行与分布式计算领域,有向无环图(directed acyclic graph,DAG)是一个能够描述具有复杂的任务优先级约束与蕴含通信开销的分布式功能的常用模型[18-19].

其次,异构分布式嵌入式系统在性能与时间约束上面临严重的冲突.若从系统的角度来看,降低整个系统的调度长度、提高系统性能是其主要目标.若从功能的角度来看,满足其时间约束、确保实时性是其主要需求.然而,对于具有资源约束的大规模异构分布式嵌入式系统,不可能满足所有分布式功能的时间约束,因此满足高关键级功能的时限、确保安全性则成为主要目标.在并行与分布式计算领域,通过公平调度多个分布式功能来降低系统的调度长度是一种可行的方法,但是应用于异构分布式嵌入式系统则会使得大量分布式功能(包括高关键级功能和

低关键级功能)的时间约束无法满足.

因此,如何提高系统的整体性能并确保高关键级功能的实时性则成为异构分布式嵌入式系统的混合关键级任务调度需要优化的主要问题.为此,本文旨在通过提出精确反映分布式功能的混合关键级系统模型,并在此基础上提出优化的任务调度算法,最小化系统性能与功能时限之间的冲突,在性能与实时性上取得合理的权衡.

1 相关研究

由于公平性是提高并行与分布式系统性能的重要方法,而满足高关键级分布式功能的时间约束则是混合关键级的主要研究内容,因此本节将公平性和混合关键级的相关研究分别论述.

1.1 公平性研究

针对单个分布式功能的 DAG 任务调度是研究多个分布式功能 DAG 调度的重要基础.异构系统单 DAG 任务调度一般包含 2 个步骤:1)根据优先级进行任务排序;2)将任务分配到合适的处理器上.异构分布式系统的 DAG 任务调度算法是一个典型的 NP 难问题,比较有名的算法有 CPOP[18],HEFT[18],HSV[19]等.同单 DAG 任务调度一样,异构系统多 DAG 调度也包含任务优先级排序和处理器分配 2 个步骤.文献[20]提出了将多个 DAG 合成一个复合 DAG 后,再利用诸如 HEFT 等经典的单 DAG 调度算法调度.很明显,合成方法没有考虑公平性问题.同样,先来先服务(first come first serve, FCFS)和及时服务(serve on time, SOT)也都不能体现公平性.

Zhao 等人[21]首次指出了在多 DAG 调度中的公平性问题,并提出了 slowdown 驱动的多 DAG 静态公平任务调度算法 Fairness.我们也提出了考虑通信开销的静态调度算法[1].多分布式功能 DAG 动态调度更加符合现实需求.动态调度即可在任意时刻提交新的功能到系统中,目前主要有 RANK_HYBD[22],OWM[23],FDWS[24]等算法.动态调度算法与静态调度算法的主要区别在于前者在有新功能提交时,要判断当前处理器是否空闲以及如何实现公平性等问题.如果当前处理器忙,OWM 会继续将任务保留在系统的公共就绪队列中等待处理器空闲,以等待下一轮的分配.OWM 的等待方式过于被动,FDWS 则通过为每个处理器提供一个处理器等待队列来实现任务的分配时隙并等待处理器空闲,以减少多次的就绪调度.

1.2 混合关键级研究

2007 年 Vestal[8]在嵌入式实时系统领域顶级国际会议 RTSS 上首次提出了混合关键级系统的概念以及相应的固定优先级任务调度策略.从此混合关键级系统相关的基础理论和调度问题迅速成为实时系统领域的热点问题.混合关键级系统包括周期任务模型和随机任务模型 2 个主要模型,调度策略则包括固定优先级抢占式策略、固定优先级非抢占式策略、动态优先级策略等.由此可见,混合关键级系统的调度问题实际上是传统实时系统在面临多种关键级任务时的扩展,因此改进和扩展的 RM 和 EDF 算法也成为混合关键级系统的主要调度方法[8-14].

早期的研究主要集中在单处理器调度,由于成本、功耗和性能等需求,基于多处理器或多核架构的混合关键级调度成为近年来的研究热点[25-26].分区调度和全局调度是这类架构的 2 种主要调度算法.分区调度即通过时间分区的方式将任务静态地分配到处理器核上,在任务分配给具体的处理器核以后,该任务只能在该处理器核上调度执行,被抢占或被挂起以后也不能再迁移到其他处理器核上调度执行.而全局调度策略可以将任务分配到任何处理器核上,并可在处理器之间迁移.需要指出的是,目前的混合关键级系统主要考虑双关键级系统,即包括高关键级和低关键级,或者安全关键级或非安全关键级 2 种级别.高关键体现为功能或任务的强实时性,低关键级体现为软实时性;而对于非关键级功能或任务,则无需关注其实时性需求.尽管双关键级仅考虑 2 个关键级问题,但是其调度问题仍然是一件非常复杂的工作,许多存在的问题需要综合权衡考虑,主要包括确保高关键级功能或任务的实时性、提高系统性能或资源利用率、处理器空闲时隙、系统关键级切换等问题.因此,本文也将双关键级系统作为研究对象,从"功能级"的角度研究异构分布式嵌入式系统的混合关键级调度问题.

近年来,具有优先级约束任务的混合关键级功能的调度研究也取得一定进展.将每个分布式功能抽象成 DAG 模型,文献[27-30]提出了一系列的异构分布式系统的混合关键级调度算法.文献[27]中通过采用时间与空间分区实现功能的隔离.每一个实时功能运行在独立的时间分区,其中高关键级功能使用静态循环调度算法,低关键级功能则使用固定优先级抢占策略以满足所有功能的时限.文献[28]对文献[27]的工作进行了扩展,考虑了不同关键

级功能的分区共享问题.上述2项工作的主要问题在于忽略了任务之间的通信开销.因此在文献[29]考虑在处理器之间通过TTEthernet协议实现通信,但仍对通信开销缺乏定量的分析.文献[30]则着重考虑满足硬实时功能的可调度性,并最大化软实时功能的服务质量.上述工作都基于如下前提:当且仅当系统有足够的时间和空间分区,混合关键级功能才能集成到同一个平台上来;其次,上述工作所采用的方法都是基于传统混合关键级系统的调度理论,包括使用改进的RM,EDF调度算法和分区调度机制等.尽管采用了DAG模型抽象了分布式功能,但并未充分利用异构分布式系统中DAG模型的经典理论与方法,无法体现异构分布式嵌入式系统的分布式与并行化特征.

本文的主要目标就是解决上述研究中面临的一些问题,提出面向异构分布式嵌入式系统的混合关键级模型,从"功能级"以及"并行与分布式计算"的角度来研究性能与时间约束权衡下的动态调度优化问题.

2 系统模型

以异构分布式汽车电子系统为例,它由多个异构ECU(不同供应商提供的ECU在设计方法或硬件实现上不同)、传感器和执行器等处理设备组成,本文将这些处理单元统称为处理器,并用集合$P=\{p_1,p_2,\cdots\}$来表示.用$CL=\{LC,HC\}$表示系统中的关键级层次,当前混合关键级系统的研究也主要集中在双关键级系统[8-14],因此本文也基于双关键级系统,将功能划分为低关键级(lower-criticality,LC)和高关键级(higher-criticality,HC)两种,且满足高关键级功能的实时性是必需的目标,否则会造成严重的安全问题.

以汽车电子系统中的线控刹车功能为例,它的实现是由多个相互具有数据依赖的任务组成.当驾驶者踩下刹车踏板时,首先同时对汽车当前的运行状态和周围的物理环境信息进行采集,接着将接触信号以电子形式传输至通过多种总线互连的ECU单元,ECU接收到信号并处理后,再传送给其他ECU单元执行并开启楔轴承机械装置,并以恰当的频率对刹车执行器发出指令启动刹车信号,然后传输给执行器执行刹车动作.整个计算执行和信号传送需要在毫秒级内完成[5].上述功能任务在不同处理器上执行并产生大量的通信开销,且这些功能任务又存在优先级约束的依赖关系,它的实现需要通过总线网络进行交互和协作才能完成.因此可以用一个有向无环图DAG来描述一个分布式功能[18-19],这也是并行与分布式计算领域分布式功能的常用模型.在本文用$Func=\{N,\boldsymbol{W},E,C,criticality,lower_bound,deadline,arrival_time,makespan\}$表示一个分布式功能.其中$N=\{n_1,n_2,\cdots\}$表示任务集合;由于处理器处理能力的异构性,使得不同任务在不同的处理器计算时间不尽相同,因此定义$\boldsymbol{W}$为一个$|N|\times|P|$的矩阵,$w_{i,k}$表示任务n_i在处理器p_k上的计算开销;$E=\{e_{1,2},e_{2,3},\cdots,e_{i,j},\cdots\}$描述为任务间的优先级约束与数据依赖关系集合;本文假设处理器之间采用CAN总线型互联,使用$C=\{c_{1,2},c_{2,3},\cdots,c_{i,j},\cdots\}$描述具有数据依赖的任务之间的通信开销集合,如果将这2个任务分配到同一个处理器上,则通信开销为0.通信开销的描述与经典DAG模型一样[18-19].由于处理器之间通过CAN,FlexRay,LIN,MOST等多种类型的网络和网关实现互联,不同总线的带宽不尽相同,因此任务之间的通信开销也不同,描述$C=\{c_{1,2},c_{2,3},\cdots,c_{i,j},\cdots\}$为具有数据依赖的任务之间的通信开销集合,如果将这2个任务分配到同一个处理器上,则通信开销为0.$pred(n_i)$表示任务n_i的直接前驱任务集合;$ind(n_i)$表示任务n_i的入度,也就是其直接前驱任务集合的个数,当前任务只有在它全部前驱任务的数据到达后才能执行.$succ(n_i)$表示任务n_i的直接后继任务集合;$outd(n_i)$表示任务n_i的出度,也就其直接后继任务集合的个数.没有前驱任务的任务为入口任务,记为n_{entry};没有后继任务的任务为出口任务,记为n_{exit}.$criticality\in CL$表示该功能的关键级;$lower_bound$表示该功能单独占用所有处理器资源时的调度长度,本文称之为下界;$deadline$表示该功能的时限.$criticality$,$lower\_bound$,$deadline$这3个参数都由第三方认证机构给出.$arrival_time$表示功能的到达时间;$makespan$表示功能的最终调度长度.

混合关键级系统包括多种安全关键和非安全关键功能,我们用$MS=\{\{Func_1,Func_2,\cdots\},system_criticality,makespan\}$表示.其中$system_criticality\in CL$表示当前系统所处的关键级,并且可以在$LC$和$HC$之间进行切换;$makespan$表示系统的整体调度长度.由于一个任务分配到不同处理器会产生一定的通信开销并使问题变得更加复杂.因此本文仅考虑非抢占式调度,而不包括抢占式调度和分区机制.

如图 1 为由 3 个分布式功能组成的双关键级系统,包含 $Func_A$, $Func_B$, $Func_C$. 表 1 为相应的参数值,其中 $Func_B$ 为高关键级功能,到达时刻为 20; $Func_A$ 和 $Func_C$ 为低关键级功能,到达时刻分别为 0 和 50. 图 1 中,任务 A_1 与任务 A_2 之间的连线表示只有 A_1 执行完成后 A_2 才有机会执行,连接任务 A_1 与 A_2 边的权值表示它们之间的通信开销为 18; 若任务 A_1 与 A_2 在分配在同一处理器上,则通信开销为 0. 表 1 中任务 A_1 与处理器 p_1 所结合处的数字 14 表示任务 A_1 在处理器 p_1 上的计算开销为 14.

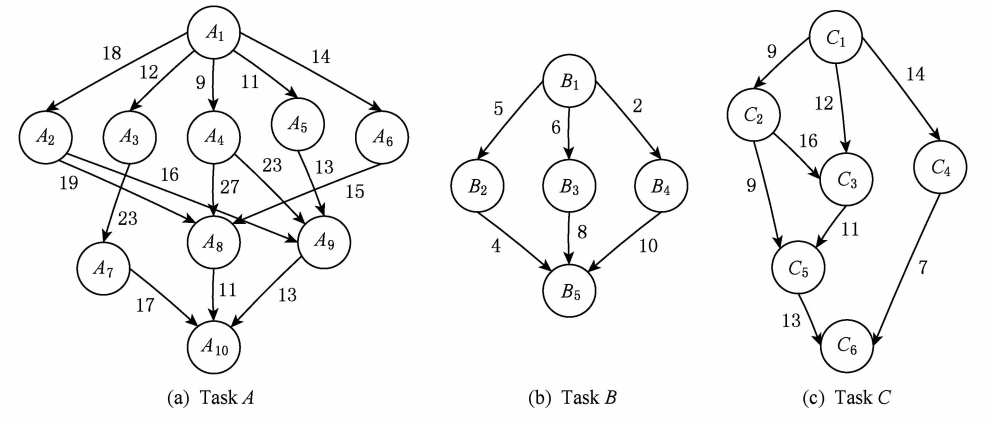


Fig. 1 Dual-criticality systems with three distributed functionalities.

图 1 包含 3 个分布式功能的双关键级系统

Table 1 Attributes of three Distributed Functionalities

表 1 3 个分布式功能的相关属性值

Functionality	Criticality	Arrival Time	Task	Processor			$rank_u(n_i)$
				p_1	p_2	p_3	
$Func_A$	LC	0	A_1	14	16	19	109
			A_2	13	19	18	78
			A_3	11	13	19	81
			A_4	13	8	17	81
			A_5	12	13	10	70
			A_6	13	16	9	64
			A_7	7	15	11	43
			A_8	5	11	14	36
			A_9	18	12	20	45
			A_{10}	21	7	16	15
$Func_B$	HC	20	B_1	4	5	6	42
			B_2	9	10	11	20
			B_3	18	17	16	31
			B_4	21	15	19	35
			B_5	7	6	5	6
$Func_C$	50	LC	C_1	8	11	19	110
			C_2	14	13	8	91
			C_3	9	12	16	63
			C_4	18	15	14	31
			C_5	18	16	20	39
			C_6	5	10	7	8

3 调度策略

3.1 认证分析

HEFT 算法是并行与分布式计算中众所周知的具有低复杂度和高性能的 DAG 调度算法[18]. 该算法使用向上排序值($rank_u$)的降序排列作为任务优先级排序标准,计算为

$$rank_u(n_i)=\overline{w_i}+\max_{n_j\in succ(n_i)}\{c_{i,j}+rank_u(n_j)\}. \quad (1)$$

使用最小最早完成时间作为处理器分配,计算为

$$EFT(n_i,p_k)=EST(n_i,p_k)+w_{i,k}, \quad (2)$$

其中,

$$\begin{cases} EST(n_{entry},p_k)=0, \\ EST(n_j,p_k)=\max\{avail[p_k], \\ \quad \max_{n_i\in pred(n_j)}\{AFT(n_i)+\overline{c_{i,j}}\}\}, \end{cases} \quad (3)$$

$avail[p_k]$表示处理器 p_k 的可用时间,$AFT(n_i)$表示任务 n_i 的实际完成时间. 很明显,对于分布式功能中的优先级约束的任务,最早完成时间策略体现了一种典型的贪婪策略.

混合关键级系统中的高关键级功能具有严格的硬实时需求,且需通过第三方认证机构进行严格的认证. 因此,第三方认证机构必须采用公认有说服力的标准算法对该功能进行严格的评估. 作为低复杂度和高性能调度算法的著名算法,HEFT 完全可以也应该作为异构分布式嵌入式系统中的功能认证算法. 第三方认证机构对高关键级功能进行认证时,会独占所有处理器进行调度,此时具有最小的调度长度,本文称之为下界(*lower_bound*). 根据下界值,第三方认证机构经过安全分析和风险评估后,会给出一个合理的时限时距(*deadline_span*). 需要指出的是,本文的主要工作是假设功能的高低关键级级别已经通过认证获得,并在此基础上进行调度. 而不是具体的安全分析和风险评估,也就是不对高低关键级关键进行具体的度量. 因此,本文的研究假设高低关键级级别已经度量. 综合 *lower_bound*,*deadline_span*,*arrival_time* 可得出该功能的绝对时限:

$$Func_m.deadline=Func_m.lower\_bound+Func_m.deadline\_span+Func_m.arrival\_time. \quad (4)$$

对于分布式功能的每个任务,也存在相应的下界为

$$lower\_bound(n_i)=AFT(n_i). \quad (5)$$

因此,每个任务的绝对时限为

$$deadline(n_i)=lower\_bound(n_i)+Func_m.deadline\_span+Func_m.arrival\_time. \quad (6)$$

表 2、表 3 分别列出了图 1 所示的高关键级功能 $Func_B$ 及其任务的下界值和绝对时限.

Table 2 Attributes of Higher Criticality $Func_B$

表 2 高关键级功能 $Func_B$ 的属性值

Attributes of Functions	$Func_B$
lower_bound	36
deadline_span	10
arrival_time	20
deadline	66

Table 3 Attributes of Tasks of $Func_B$

表 3 高关键级功能 $Func_B$ 的任务的属性值

Attributes of Tasks	Tasks of $Func_B$				
	B_1	B_2	B_3	B_4	B_5
$lower\_bound(n_i)$	4	20	22	21	36
$deadline(n_i)$	34	54	52	51	66

3.2 公平策略

首先提出系统的调度框架,如图 2 所示,该框架引入 3 个队列,分别为任务优先级队列 *task_priority_queue*、公共就绪队列 *common_ready_queue* 以及任务分配队列 *task_allocation_queue*.

1) 每一个分布式功能都拥有一个 *task_priority_queue*,用于对任务优先级进行排序;

2) 系统中仅有一个 *common_ready_queue*,用于存储待分配的就绪任务;

3) 每一个处理器都拥有一个 *task_allocation_queue*,用于存储分配到该处理器的相应任务.

公平策略的任务调度一共包括 5 个主要步骤:

Step1. 任务排序. 对一个分布式功能排序,并按 $rank_u$ 式(1)的降序排列.

Step2. 任务就绪. 基于轮转的公平策略,从每一个分布式功能的 *task_priority_queue* 中取出一个 $rank_u$ 最大的就绪任务(只有该任务的所有前驱任务分配完成后,该任务才能成为就绪任务),放入到 *common_ready_queue* 中,同样按 $rank_u$ 的降序排列.

Step3. 任务分配. 依次从 *common_ready_queue* 中取出每个就绪任务,在插入法的基础上将其分配到具有最小 EFT 的处理器 *task_allocation_queues* 上.

Step4. 任务执行. 若 *task_allocation_queue* 中任务的开始时间等于当前时间,对该任务进行调度.

Step5. 新功能到达. 若当前时刻有新的分布式

功能到达,则取消所有 *task_allocation_queue* 中未开始调度的任务,并将这些任务放回各自的 *task_priority_queue* 中. 重复 Step1,将新的分布式功能的任务与其他功能中未分配的任务一起进行新一轮的公平分配.

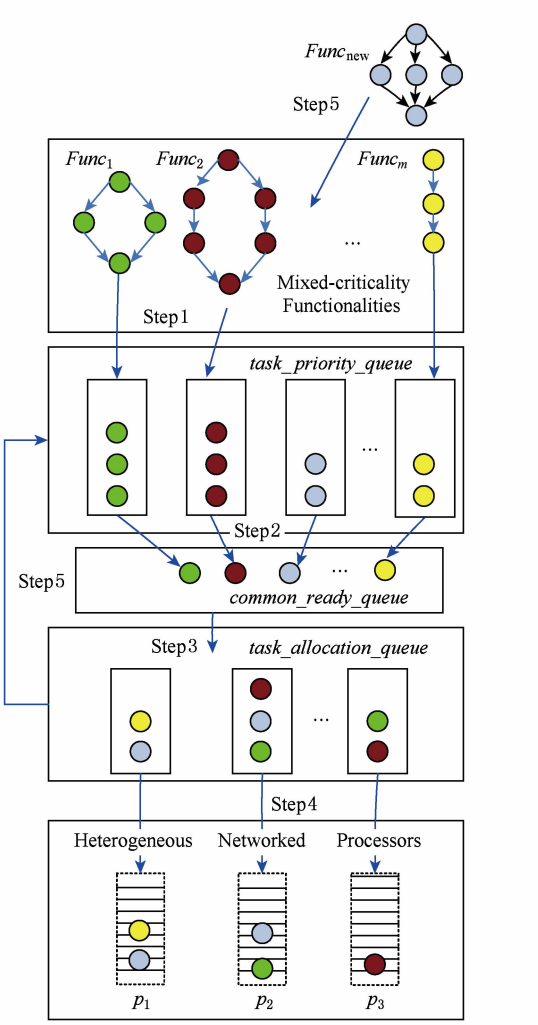


Fig. 2 Scheduling framework of systems.

图 2 系统调度框架

依据上述调度框架和公平策略,提出公平策略的动态双关键级任务调度算法 F_DDHEFT(fairness on dynamic dual-criticality heterogeneous earliest finish time),该算法步骤如算法 1 所示:

算法 1. F_DDHEFT 算法.

输入:处理器集;

输出:调度结果.

while (有功能需调度) do

if (有一个功能 $Func_{new}$ 在时刻 *current_time* 到达) then

计算 $Func_{new}$ 中所有任务的 $rank_u$ 值,并将这些任务放入到队列 *task_priority_queue*($Func_{new}$) 中;

$MS.add(Func_{new})$;

将任务分配队列中未执行的任务放回到各功能的任务优先级队列;

while (有任务可以分配)

for ($m=1$;$m \leqslant |MS|$;$m++$)

从 $task\_priority\_queue(Func_m)$ 中取出任务 n_i;

$common\_ready\_queue.put(n_i)$;

end for

while (!*common_ready_queue.empty*() do

从 *common_ready_queue* 中取出任务 n_i;

基于插入法将任务 n_i 分配到具有最小 $EFT(n_i, p_k)$ 的 $task\_allocation\_queue(p_k)$ 中;

end while

end while

end if

end while

F_DDHEFT 算法的时间复杂度为 $O(m \times n^2 \times p)$. 其中 m 代表功能数,n 表示包含最多任务数的功能拥有的任务数,p 为处理器数. 遍历所有功能的时间复杂度应为 $O(m)$;遍历任务次数的时间复杂度应为 $O(n)$;计算 n_i 的 EFT 需遍历其前驱和各处理器的时间复杂度应为 $O(n \times p)$. 因此 F_DDHEFT 算法的时间复杂度为 $O(m \times n^2 \times p)$.

本文提出的公平策略的最大改进在于当有新功能到达时,不像 OWM 或者 FDWS 算法那样会阻塞自己并等待其他任务完成. 本文的策略会取消那些"已选择处理器但未开始调度的任务",并与新的 DAG 一起进行公平分配. 基于图 1 提供的混合关键级实例,图 3 所示为 F_DDHEFT 算法的执行过程.

1) 如图 3(a)所示,低关键级功能 $Func_A$ 在时刻 0 到达,根据向上排序值 $rank_u$ 分配所有任务到相应的任务分配队列. 任务分配顺序依次为{A_1, A_3, A_4, A_2, A_5, A_6, A_9, A_7, A_8, A_{10}}.

2) 如图 3(b)所示,高关键级功能 $Func_B$ 在时刻 20 到达,从公平的角度出发,取消任务分配队列中未开始执行的任务,这些任务是{A_2, A_5, A_6, A_9, A_7, A_8, A_{10}}.

3) 如图 3(c)所示,$Func_B$ 中的任务{B_1, B_4, B_3, B_2, B_5}与 $Func_A$ 中取消的任务{A_2, A_5, A_6, A_9, A_7, A_8, A_{10}}一起公平调度,公平调度顺序为{A_2,

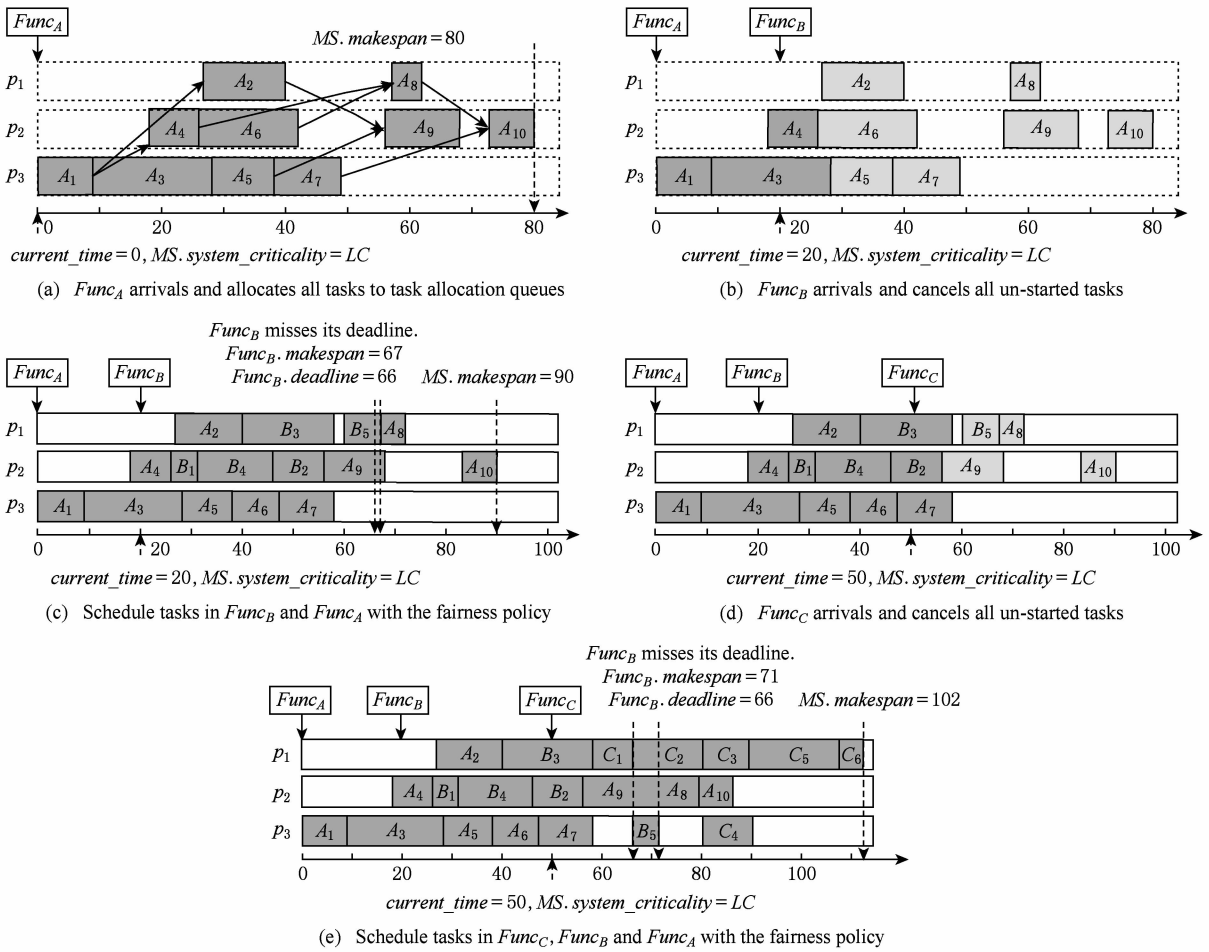


Fig. 3 Process of task scheduling based on the fairness policy.

图 3 基于公平策略的任务调度过程

B_1,A_5,B_4,A_6,B_3,A_9,B_2,A_7,B_5,A_8,A_{10}}. 需要指出的是,由于 $Func_B$ 是高关键级功能,其调度长度 $Func_B.makespan=67$,超过了 CA 所认证的绝对时限 $Func_B.deadline=66$(如表 2 所示). 因此 $Func_B$ 会对系统造成严重的安全问题.

4) 如图 3(d)所示,低关键级功能 $Func_C$ 在时刻 50 到达,同样从公平的角度出发,取消任务分配队列中未开始执行的任务,这些任务是{A_9,A_8,A_{10}}与{B_5}.

5) 如图 3(e)所示,$Func_C$ 中的任务{C_1,C_2,C_3,C_5,C_4,C_6}与 $Func_A$ 中取消的任务{A_9,A_8,A_{10}}和 $Func_B$ 中取消的任务{B_5}一起公平分配,分配顺序为{C_1,A_9,B_5,C_2,A_8,C_3,A_{10},C_5,C_4,C_6}. 此时,对于高关键级分布式功能 $Func_B$,其调度长度 $Func_B.makespan=71$,仍然超过了 CA 所认证的绝对时限 $Func_B.deadline=66$(如表 2 所示). 因此 $Func_B$ 会持续对系统造成严重的安全问题. 需要指出的是,此时系统的调度长度 $MS.makespan=102$,该项指标反映系统的整体性能,$MS.makespan$ 越低,则性能越好.

3.3 关键级策略

如引言所述,以及通过 3.2 节所展示的实例,公平任务调度易造成高关键级功能错过其绝对时限,这对安全高度关键的分布式嵌入式系统来说无法容忍. 因此,本节提出动态环境下满足高关键级功能绝对时限的任务调度算法以解决上述问题.

系统关键级是混合关键级系统中的一个重要概念. 系统关键级可以进行切换,比如从低关键级提升到高关键级,也可以从高关键级降回低关键级. 系统关键级的提升或下降实际上是系统的模式转换. 对于双关键级系统,如果系统处于低关键级,那么所有关键级功能都可以在该模式下执行;如果系统处于高关键级,那么仅有高关键级功能可以在该模式下执行. 因此,如图 3 所示的公平任务调度实例,为保证所有功能都有公平执行的机会,系统始终处于低关键级模式下.

由于系统中可能有多个高关键级功能,并且可在任意时刻提交新的高关键级功能到系统中,此时系统也不可能满足所有高关键级功能的实时性.因此根据实时调度理论的 EDF(earliest deadline first)策略,在高关键级模式下,系统会对所有高关键级功能按照"绝对时限"的降序排列,并优先分配 EDF 最小的高关键级功能中的所有任务.

本文还提出关键级策略的动态双关键级调度算法 C_DDHEFT(criticality on dynamic dual-criticality heterogeneous earliest finish time),其核心思想就是通过牺牲低关键级的公平执行机会,优先执行高关键级功能以满足时限.C_DDHEFT 算法的步骤如算法 2 所示:

算法 2. C_DDHEFT 算法.

输入:处理器集;

输出:调度结果.

```
while (有功能需调度) do
  if (有一个功能 Func_new 在时刻 current_time 到达) then
    计算 Func_new 中所有任务的 rank_u 值,并将这些任务放入到队列 task_priority_queue(Func_new) 中;
    MS.add(Func_new);
    if (Func_new).criticality=HC) then
      MS.system_criticality=HC;
      将任务分配队列中未执行的任务放回到各功能的任务优先级队列;
    else
      仅将低关键级功能中的任务分配队列中未执行的任务放回到各功能的任务优先级队列;
    end if
    if (MS.system_criticality=HC) then
      将高关键级功能按照 EDF 的降序排列;
      for (m=1;m≤|MS.higher_criticality.functionalities|;m++)
        从task_priority_queue(Func_m)中取出一个任务 n_i;
        基于插入法将任务 n_i 分配到具有最小 EFT(n_i, p_k) 的 task_allocation_queue(p_k)中;
      end for
    end if
    MS.system_criticality=LC;
    while (有任务可以分配)
      for (m=1; m≤|MS|; m++)
        从task_priority_queue(Func_m)中取出一个任务 n_i;
        common_ready_queue.put(n_i);
      end for
      while (!common_ready_queue.empty())
        do
        从common_ready_queue 中取出一个任务 n_i;
        基于插入法将任务 n_i 分配到具有最小 EFT(n_i, p_k) 的 task_allocation_queue(p_k)中;
      end while
    end while
  end if
end while
```

C_DDHEFT 算法的时间复杂度与 F_DDHEFT 算法一样,也为 $O(m\times n^2\times p)$.

C_DDHEFT 算法与 F_DDHEFT 算法的区别主要有 2 点:1)若新到达的功能是高关键级功能,那么系统会提升系统关键级,并取消所有任务分配队列中未开始执行的任务.①依据 EDF 原则对高关键级功能进行分配,待全部分配完成后将系统关键级降级为低关键级;②再与其他被取消分配的任务一起进行公平分配.2)若新到达的功能是低关键级功能,那么系统只会取消所有任务分配队列中未开始执行的低关键级功能中的任务(不包括高关键级功能中的任务),新到达的功能只与这些被取消分配的任务一起进行公平调度(换言之,低关键级功能不会影响高关键级功能中的任务分配).基于图 1 提供的混合关键级实例,图 4 为 C_DDHEFT 算法的任务调度过程.

1) 低关键级功能 $Func_A$ 在时刻 0 到达,系统关键级为 LC,根据向上排序值 $rank_u$ 分配所有任务到相应的任务分配队列.任务分配顺序依次为$\{A_1, A_3, A_4, A_2, A_5, A_6, A_9, A_7, A_8, A_{10}\}$.上述过程与图 3(a)完全相同,这里不再画图描述.

2) 如图 4(a)所示,高关键级功能 $Func_B$ 在时刻 20 到达,系统关键级提升到 HC,取消任务分配队列中未开始执行的任务,这些任务是$\{A_2, A_5, A_6, A_9, A_7, A_8, A_{10}\}$.

3) 如图 4(b)所示,$Func_B$ 中的任务$\{B_1, B_4, B_3, B_2, B_5\}$优先分配,分配完成后,系统关键级切换

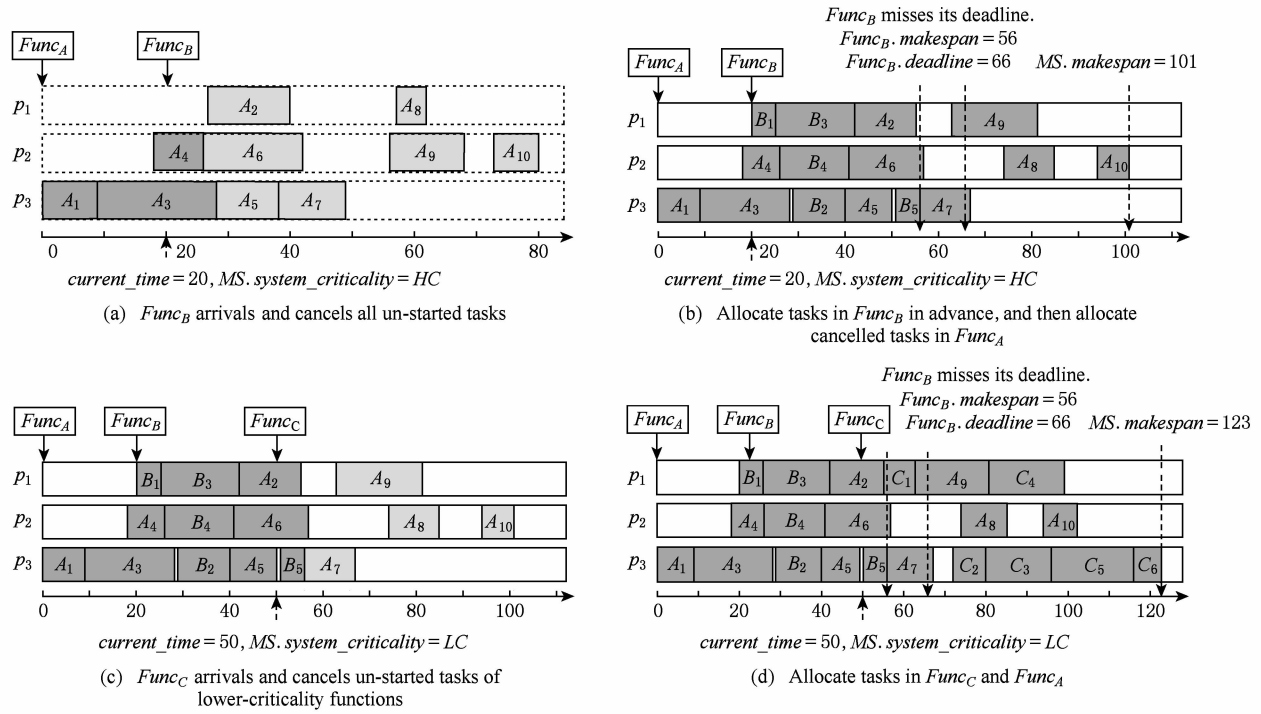


Fig. 4 Process of task scheduling based on the criticality policy.

图 4 基于关键级策略的任务调度过程

回 LC, $Func_A$ 中取消的任务{A_2,A_5,A_6,A_9,A_7,A_8,A_{10}}再进行分配,整个任务分配顺序依次为{B_1,B_4,B_3,B_2,B_5,A_2,A_5,A_6,A_9,A_7,A_8,A_{10}}. $Func_B$ 是高关键级分布式功能,此时其调度长度 $Func_B.makespan=56$,满足 CA 所认证的绝对时限 $Func_B.deadline=66$(如表 2 所示). 因此 $Func_B$ 不会对系统造成安全问题.

4) 如图 4(c)所示,低关键级功能 $Func_C$ 在时刻 50 到达,取消所有低关键级功能中未开始执行的任务,这些任务是{A_9,A_8,A_{10}}.

5) 如图 4(d)所示,$Func_C$ 中的任务{C_1,C_2,C_3,C_5,C_4,C_6}与 $Func_A$ 中取消的任务{A_9,A_8,A_{10}}一起公平分配,分配顺序为{C_1,A_9,C_2,A_8,C_3,A_{10},C_5,C_4,C_6}. 此时,对于高关键级分布式功能 $Func_B$,由于它并未受低关键级功能达到所造成的影响,因此它的调度长度仍然为 $Func_B.makespan=56$. 特别需要指出的是,此时系统的调度长度 $MS.makespan=123$,明显超过 3.2 节公平调度的调度长度 102. 因此尽管关键级的调度策略满足了 $Func_B$ 的实时性,但造成了系统整体性能过低.

3.4 时限时距策略

通过 3.2 节公平策略和 3.3 节关键级策略的实例分析可知:公平策略能够提高性能,但造成高关键级功能实时性得不到满足. 关键级策略虽能够满足高关键级功能的实时性,但造成系统的整体性能急剧下降. 异构分布式嵌入式系统的混合关键级任务调度在性能与时间约束上面临严重的冲突. 如何在动态环境下提高系统整体性能,并仍然确保高关键级功能的实时性,在性能与实时性之间取得合理的权衡则成为主要的优化问题.

提出时限时距的动态双关键级调度算法 D_DDHEFT(deadline-span on dynamic dual-criticality heterogeneous earliest finish time),其核心思想就是通过判断每个任务的时限是否满足来决定进一步的任务分配. D_DDHEFT 算法的步骤如算法 3 所示:

算法 3. D_DDHEFT 算法.

输入:处理器集;

输出:调度结果.

while (有功能需调度) do

  if (有一个功能 $Func_{new}$ 在时刻 $current\_time$ 到达) then

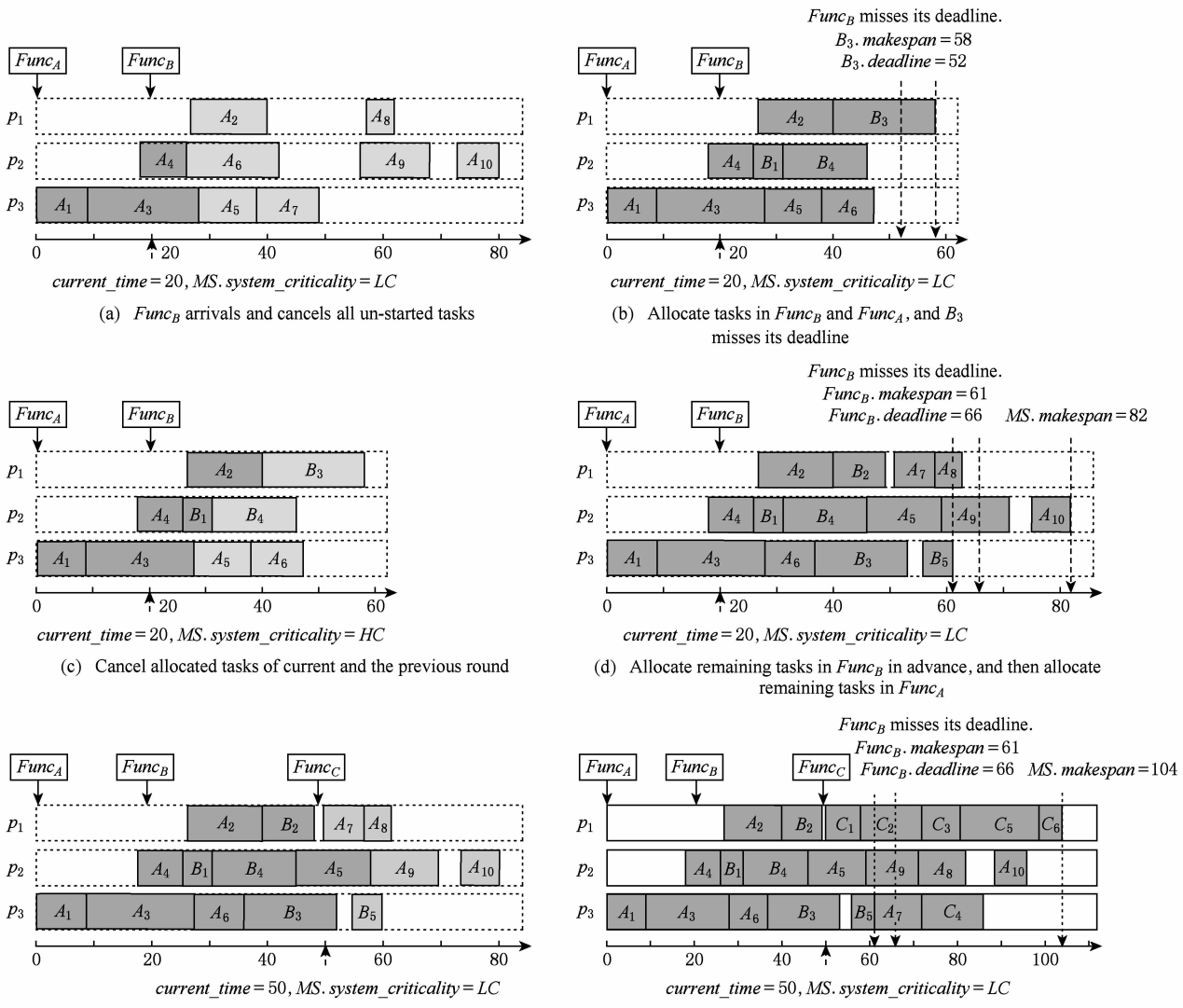
    计算 $Func_{new}$ 中所有任务的 $rank_u$ 值,并将这些任务放入到队列 $task\_priority\_queue(Func_{new})$ 中;

    $MS.add(Func_{new})$;

    将任务分配队列中未执行的任务放回到各功能的任务优先级队列;

    while (有任务可以分配)

```
for (m=1;m≤|MS|; m++)
    从task_priority_queue(Func_m)中取出一个任务 n_i;
    common_ready_queue.put(n_i);
end for
while (!common_ready_queue.empty()) do
    从common_ready_queue 中取出一个任务 n_i;
    基于插入法将任务 n_i 分配到具有最小 EFT(n_i, p_k) 的 task_allocation_queue(p_k)中;
    if (Func(n_i).criticality=HC && makespan(n_i) > deadline(n_i)) then
        取消本轮与前一轮的任务分配结果;
        将这些取消的任务放回到各功能的任务优先级队列;
        清空 common_ready_queue;
        MS.system_criticality=HC;
    end if
end while
if (MS.system_criticality=HC)
    将高关键级功能按照 EDF 的降序排列;
    for (m=1; m≤|MS.higher_criticality.functionalities|;m++)
        从task_priority_queue(Func_m)中取出一个任务 n_i;
        基于插入法将任务 n_i 分配到具有最小 EFT(n_i, p_k) 的 task_allocation_queue(p_k)中;
```



(a) $Func_B$ arrivals and cancels all un-started tasks

(b) Allocate tasks in $Func_B$ and $Func_A$, and B_3 misses its deadline

(c) Cancel allocated tasks of current and the previous round

(d) Allocate remaining tasks in $Func_B$ in advance, and then allocate remaining tasks in $Func_A$

(e) $Func_C$ arrivals and cancels all un-started tasks

(f) Allocate tasks in $Func_C$ and $Func_A$ and $Func_B$

Fig. 5 Process of task scheduling based on the deadline-span policy.

图 5 基于时限时距策略的任务调度过程

```
            end for
            MS.system_criticality=LC;
          end if
        end while
      end if
    end while
```

D_DDHEFT 算法的时间复杂度与 F_DDHEFT 和 C_DDHEFT 一样都为 $O(m\times n^2\times p)$. D_DDHEFT 并没有因为权衡优化而造成时间复杂度增长,由此表明关键级提升的优化不会造成时间复杂度的增长. 图 5 为 D_DDHEFT 算法的任务调度过程.

1) 低关键级功能 $Func_A$ 在时刻 0 到达,系统关键级为 LC,根据向上排序值 $rank_u$ 分配所有任务到相应的任务分配队列. 任务分配顺序依次为 $\{A_1,A_3,A_4,A_2,A_5,A_6,A_9,A_7,A_8,A_{10}\}$. 上述过程与图 4(a)完全相同,这里不再赘述.

2) 如图 5(a)所示,高关键级功能 $Func_B$ 在时刻 20 到达,此时系统关键级并不急于提升到 HC,而是 $Func_B$ 中的任务与 $Func_A$ 中的任务一起公平分配,直到 B_3 错过其时隙($makespan(B_3)=58$,大于 $deadline(B_3)=52$)(如表 2 所示),如图 5(b)所示.

此时,系统关键级提升到 HC,如图 5(c)所示,取消本轮与前一轮的公平任务分配(由于本轮分配尚未分配完成,若只取消本轮,继续分配还是本轮的分配结果,因此需要取消到前一轮),这些任务是 $\{A_5,B_4,A_6,B_3\}$.

3) 如图 5(d)所示,$Func_B$ 中的任务 $\{B_4,B_3,B_2,B_5\}$ 优先分配,分配完成后,系统关键级切回 LC, $Func_A$ 中取消的任务 $\{A_5,A_6,A_9,A_7,A_8,A_{10}\}$ 再分配,整个任务分配顺序为 $\{B_4,B_3,B_2,B_5,A_5,A_6,A_9,A_7,A_8,A_{10}\}$. $Func_B$ 是高关键级分布式功能,此时其调度长度 $Func_B.makespan=61$,满足 CA 所认证的绝对时限 $deadline=66$(如表 2 所示). 因此 $Func_B$ 不会对系统造成安全问题.

4) 如图 5(e)所示,低关键级功能 $Func_C$ 在时刻 50 到达,取消所有低关键级功能中未开始执行的任务,他们是 $Func_A$ 中的 $\{A_9,A_7,A_8,A_{10}\}$ 和 $Func_B$ 中的 $\{B_5\}$.

5) 如图 5(f)所示,$Func_C$ 中的任务 $\{C_1,C_2,C_3,C_5,C_4,C_6\}$ 与 $Func_A$ 中的 $\{A_9,A_7,A_8,A_{10}\}$,以及 $Func_B$ 中任务 $\{B_5\}$ 一起公平分配,分配顺序依次为 $\{C_1,A_9,B_5,C_2,A_7,C_3,A_8,C_5,A_{10},C_4,C_6\}$. 此时,对于高关键级分布式功能 $Func_B$,尽管其与其他功能中的任务一起分配,但其调度长度仍然为 $Func_B.makespan=61$. 特别需要指出的是,此时系统的调度长度 $MS.makespan=104$,明显低于 3.3 节关键级策略调度的调度长度 123. 不难发现,时限时距策略调度在满足 $Func_B$ 实时性的前提下,有效地提高了系统的整体性能.

4 实 验

4.1 评价指标

本文采用系统的调度长度比(schedule length ratio, SLR)[18]、功能之间的不公平性(unfairness)[19]以及高关键级功能的时隙错失率(deadlines missed radio, DMR)[31]作为评价指标.

实验的硬件环境为 1 台配置了奔腾双核处理器(3.2 GHz/2.0 GB RAM)的 Windows XP 机器,使用 Java 语言,根据不同参数生成各种不同特性的 DAG 功能样本. 主要包括任务个数 $n=\{30,40,50,60,70,80,90,100\}$. 产生随机 DAG 的参数设置为:任务个数 $n=\{30,40,50,60,70,80,90,100\}$,形状参数 $\alpha=\{0.5,1.0,2.0,4.0\}$,最大出度 $\beta=\{1,2,3,4,5\}$,最大入度 $\gamma=\{1,2,3,4,5\}$,通信开销与计算开销的比值 $CCR=\{0.1,0.5,1.0,5.0,10.0\}$,任务在不同处理器上执行时间的差异度 $\eta=\{0.1,0.5,1.0,2.0,4.0\}$. 假设 w_i 表示任务 n_i 的平均计算开销,那么任务 n_i 在处理器 p_k 上的计算开销为

$$w_i(1-\eta/4)\leqslant w_{i,k}\leqslant w_i(1+\eta/4). \tag{7}$$

以上参数主要针对单个功能,整个系统中的功能数为 $MSN=\{10,20,30,40,50\}$, 不同功能的到达时距(arrival interval) 为 $\{0,50,100,150,200\}$,每个高关键级功能的相对时限定为其下界的 110%.

4.2 公平性算法比较

本节的实验主要验证本文提出的算法与同类算法(RANK_HYBD[22], OWM[23], FDWS[24])采用的公平策略在性能上的比较.

实验 1. 探究 SLR 和 Unfairness 随功能到达时距变化的情况. 功能样本从样本空间取出. 每个功能的到达时距的变化范围是 0~200,并以 50 个时距递增. 到达时距的大小反映了系统中功能的动态负载情况,总到达的功能数固定为 50. 所有功能具有同样的关键级. 本文提出的 F_DDHEFT 与同类算法进行比较.

图 6 所示为采用 4 种算法的 SLR 和 Unfairness 随功能到达时距变化情况. 实验结果显示,到达间距越大,平均 SLR 和平均 Unfairness 越低. 表明动态

提交功能的间距越大,任务需要重新调度的几率减少,性能结果更好. F_DDHEFT 的平均 SLR 优于 RANK_HYBD,OWN,FDWS 的平均百分比分别达 30%,10%,5%以上;平均 Unfairness 优于 RANK_HYBD,OWN,FDWS 的平均百分比分别为 30%,30%,15%. 从数据分析可知,通过公平调度可以尽可能地提高调度性能,且 F_DDHEFT 相比同类算法性能更好.

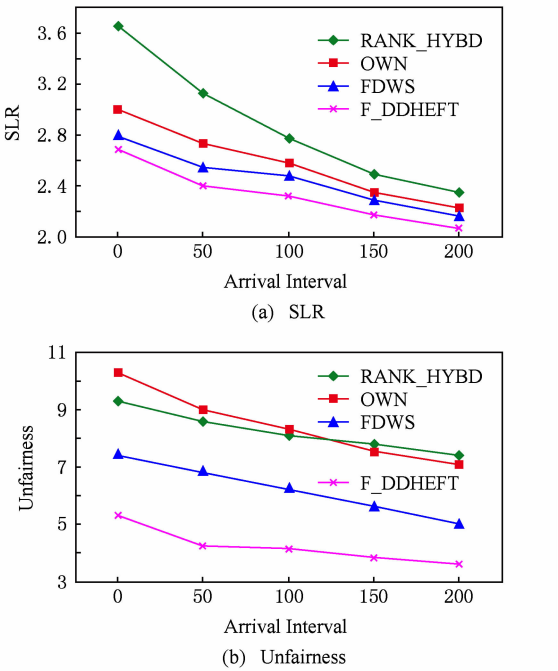


Fig. 6 SLR and Unfairness for varying deadline-spans.

图 6 SLR 和 Unfairness 随功能到达时距变化的情况

4.3 本文提出的算法比较

据我们所知,这是第 1 次针对性能与实时权衡的异构分布式嵌入式系统双关键级的动态研究,因此本节的实验主要探究和验证本文提出的 3 种算法在性能、实时及其权衡的优化.

实验 2. 探究 SLR,Unfairness,DMR 随系统中功能数变化的情况. 基于双关键级系统,系统总的到达时距固定为 0,即所有功能都在时刻 0 同时到达. 高关键级功能数的变化范围为 10～50,并以 10 递增. 功能数目反映了系统的工作负载情况. 通过对本文所提出的 3 种算法(F_DDHEFT,C_DDHEFT,D_DDHEFT)进行实验并比较相应结果.

图 7(a)和图 7(b)所示为 3 种算法的 SLR 和 Unfairness 随系统功能数目变化情况. 随着功能数目增加,SLR 都增大,不公平性都越来越高,这表明了系统中任务调度的公平性越来越低,系统的总体性能在逐渐下降. 可以发现 F_DDHEFT 拥有最好的性能,并且随着功能数目的增多,系统的 SLR 并未显著增长,始终控制在 1.9～2.6 范围内,不公平性也最低;C_DDHEFT 表现最差,而且随着系统负载不断加大,性能越低. 图 7(c)所示为采用 3 种算法的 DMR 随系统功能数目变化情况,随着功能数增加,DMR 都越来越高. F_DDHEFT 在 DMR 表现最差,DMR 始终控制在 90%以上;C_DDHEFT 在 DMR 表现最好.

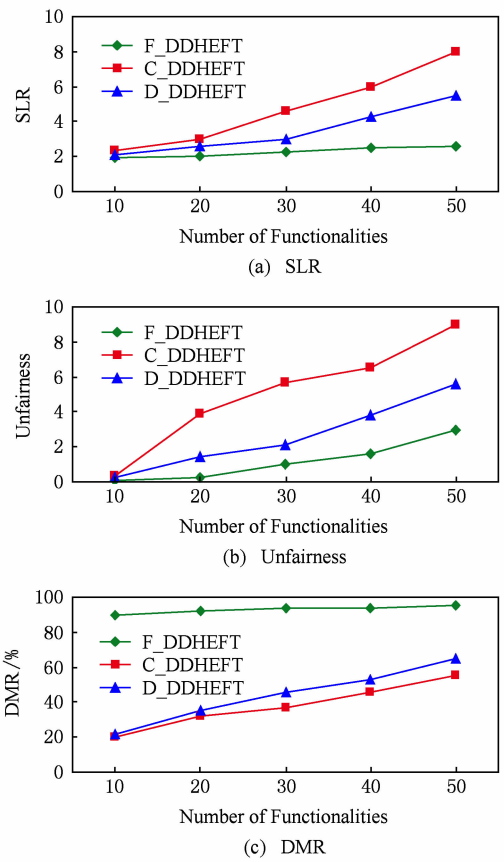


Fig. 7 SLR, Unfairness and DMR for varying numbers of functionalities.

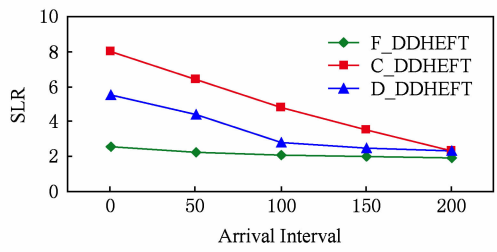
图 7 SLR,Unfairness,DMR 随系统中功能数变化的情况

上述实验可以发现,无论 SLR 和 Unfairness,还是 DMR, D_DDHEFT 处于 F_DDHEFT 和 C_DDHEFT 之间,并且在 DMR 上与 C_DDHEFT 较为接近, DMR 整体控制在 20%～65%. DDHEFT 在性能和实时性上体现了合理的权衡.

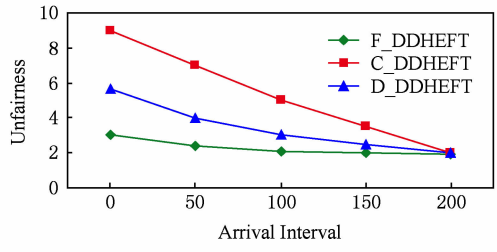
实验 3. 探究 SLR,Unfairness,DMR 随功能到达时距变化的情况. 功能样本从样本空间取出. 每个功能的到达时距的变化范围是 0～200,并以 50 个时距递增. 到达时距的大小反映了系统中功能的动态负载情况,总到达的功能数固定为 50. 基于双关键级系统,每个功能被认证为高关键级功能或低关键级功能,2 种功能数目各占一半. 通过对本文所提出的

3 种算法(F_DDHEFT,C_DDHEFT,D_DDHEFT)进行实验并比较相应结果.

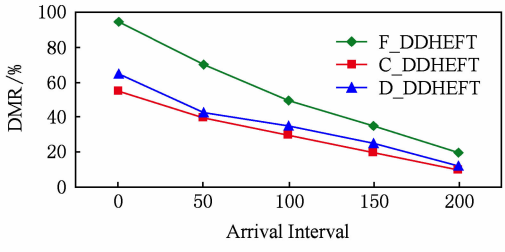
图 8 所示为采用 3 种算法的 SLR,Unfairness,DMR 随功能到达时距变化情况. 随着到达时距的增大,SLR 都减少,不公平性降低,DMR 降低,体现系统的各项指标都在提升. 当到达时距较小时,3 种算法的差距较为明显,一定程度上反映了 3 种算法的特点. 随着到达时距不断增长,3 种算法的差距逐渐减少,这实际上体现了 3 种算法最终趋近一种各自采用 HEFT 独立调度时的表现.



(a) SLR



(b) Unfairness



(c) DMR

Fig. 8 SLR,Unfairness and DMR for varying deadline-spans.

图 8 SLR,Unfairness,DMR 随功能到达时距变化的情况

实验 4. 探究 DMR 随高关键级功能数变化的情况. 功能样本同样从样本空间取出. 基于双关键级系统,系统总的功能数固定为 50,高关键级功能数的变化范围是 0～50,并以 10 递增. 系统总的到达时距固定为 0. 高关键级功能数的多少反映了系统的实时需求情况. 通过对本文所提出的 3 种算法(F_DDHEFT,C_DDHEFT,D_DDHEFT)进行实验并比较相应结果.

图 9 所示为采用 3 种算法的 DMR 随高关键级功能数变化的情况. F_DDHEFT 仍然表现最差,C_DDHEFT 表现最好,D_DDHEFT 与 C_DDHEFT 较为接近. 在高关键级功能数为 50 的时候,3 种算法的表现差距很小. 这是因为此时不存在低关键级功能,而 C_DDHEFT 和 D_DDHEFT 只能通过 EDF 来满足少量高关键级功能的实时性.

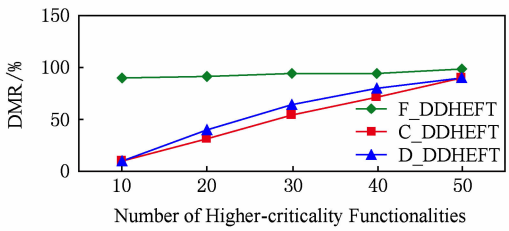
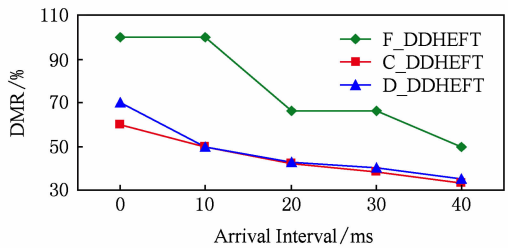


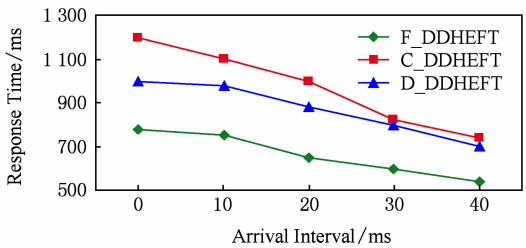
Fig. 9 DMR for varying numbers of functionalities.

图 9 DMR 随高关键级功能数目变化

实验 5. 汽车电子系统是典型的异构分布式嵌入式系统. 我们基于实际汽车电子环境,探讨端到端的响应时间与 DMR 随到达时距变化的情况. 系统使用的功能为线控刹车功能和安全气囊功能. 线控刹车功能为高关键级功能,它包括 10 个优先级约束的任务. 安全气囊为低关键级功能,它包括 8 个优先级约束的任务. 这 2 个功能共享 5 个 ECU. 所有的 ECU 有不同的计算能力. 实验环境下,到达时距的变化范围是 0～40 ms 并以 10 ms 的增量递增,到达功能数一共为 10 个,高关键级功能与低关键级功能各占一半. 图 10 所示为采用 3 种算法的 DMR 和 WCRT 随到达时距的变化情况. 可以发现实际环境下的实验结果与随机样本结果的整体趋势一样. 在满足实时性上,F_DDHEFT 仍然表现最差,C_DDHEFT



(a) DMR



(b) WCRT

Fig. 10 DMR and WCRT for varying deadline spans.

图 10 DMR,WCRT 随功能到达时距变化

表现最好,D_DDHEFT 与 C_DDHEFT 较为接近.在总体性能上,F_DDHEFT 仍然表现最好,C_DDHEFT 表现最差,D_DDHEFT 与 F_DDHEFT 较为接近.再次验证了 D_DDHEFT 在整体性能和实时性上的合理权衡.

5 结 论

本文面向异构分布式嵌入式系统,开展双关键级分布式功能的动态任务调度研究.以多 DAG 双关键级动态模型为基础,从"功能级"以及"并行与分布式计算"的角度,通过分析现有公平调度和混合关键级调度的研究进展,提出了面向不同目标的动态任务调度算法.包括以提高系统性能为目标的公平策略算法 F_DDHEFT,以满足更多高关键级功能实时性的关键级策略算法 C_DDHEFT,和在满足更多高关键级功能实时性的基础上提高系统性能的时限时距策略算法 D_DDHEFT,使得在系统整体性能和高关键级功能实时之间能够达到一种合理的权衡.最后通过实验将本文提出的 3 种算法进行全方位比较,实验结果验证了本文所提出的 D_DDHEFT 算法在满足更多高关键级功能实时性的条件下,能够有效地提高系统整体性能,且在汽车电子系统这一典型的异构分布式嵌入式系统中也有较好表现.

参 考 文 献

[1] Xie Guoqi, Li Renfa, Yang Fan, et al. Multiple-DAG offline task scheduling for heterogeneous networked automobile electronic system [J]. Journal on Communications, 2013, 34(12): 20-32 (in Chinese)
(谢国琪, 李仁发, 杨帆, 等. 异构网络化汽车电子系统中多 DAG 离线任务调度[J]. 通信学报, 2013, 34(12): 20-32)

[2] Xie Guoqi, Li Renfa, Liu Lin, et al. DAG Reliability model and fault-tolerant algorithm for heterogeneous distributed system [J]. Chinese Journal of Computers, 2013, 36(10): 2019-2032 (in Chinese)
(谢国琪, 李仁发, 刘琳, 等. 异构分布式系统 DAG 可靠性模型与容错算法[J]. 计算机学报, 2013, 36(10): 2019-2032)

[3] Seo S H, Kim J H, Hwang S H, et al. A reliable gateway for in-vehicle networks based on lin, can, and flexray [J]. ACM Trans on Embedded Computing Systems, 2012, 11(1): 1-24

[4] Buckl C, Camek A, Kainz G, et al. The software car: Building ICT architectures for future electric vehicles [C] // Proc of Electric Vehicle Conf. Piscataway, NJ: IEEE, 2012: 1-8

[5] Fürst S. Challenges in the design of automotive software [C] //Proc of the Conf on Design, Automation and Test in Europe. Piscataway, NJ: IEEE, 2010: 256-258

[6] Kelly O R, Aydin H, Zhao B. On partitioned scheduling of fixed-priority mixed-criticality task sets [C] //Proc of the 10th IEEE Int Conf on Trust, Security and Privacy in Computing and Communications. Piscataway, NJ: IEEE, 2011: 1051-1059

[7] Sinha P. Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives [J]. Reliability Engineering & System Safety, 2011, 96(10): 1349-1359

[8] Vestal S. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance [C] //Proc of IEEE Real-Time Systems Symp (RTSS 2007). Piscataway, NJ: IEEE, 2007: 239-243

[9] De Niz D, Lakshmanan K, Rajkumar R. On the scheduling of mixed-criticality real-time task sets [C] //Proc of IEEE Real-Time Systems Symp (RTSS 2009). Piscataway, NJ: IEEE, 2009: 291-300

[10] Lakshmanan K, De Niz D, Rajkumar R. Mixed-criticality task synchronization in zero-slack scheduling [C] //Proc of IEEE Real-Time and Embedded Technology and Applications Symp. Piscataway, NJ: IEEE, 2011: 47-56

[11] Baruah S, Vestal S. Schedulability analysis of sporadic tasks with multiple criticality specifications [C] //Proc of Euromicro Conf on Real-Time Systems. Piscataway, NJ: IEEE, 2008: 147-155

[12] Petters S M, Lawitzky M, Heffernan R, et al. Towards real multi-criticality scheduling [C] //Proc of IEEE Int Conf on Embedded and Real-Time Computing Systems and Applications. Piscataway, NJ: IEEE, 2009: 155-164

[13] 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

[14] Li H, Baruah S. An algorithm for scheduling certifiable mixed-criticality sporadic task systems [C] //Proc of IEEE Real-Time Systems Symp (RTSS 2010). Piscataway, NJ: IEEE, 2010: 183-192

[15] Zeller M, Prehofer C, Weiss G, et al. Towards self-adaptation in real-time, networked systems: Efficient solving of system constraints for automotive embedded systems [C] //Proc of Int Conf Self-Adaptive and Self-Organizing Systems. Piscataway, NJ: IEEE, 2010: 79-88

[16] Katoen J P, Noll T, Wu H, et al. Model-based energy optimization of automotive control systems [C] //Proc of the Conf on Design, Automation and Test in Europe. Piscataway, NJ: IEEE, 2013: 761-766

[17] Heinrich P, Prehofer C. Network-wide energy optimization for adaptive embedded systems [J]. ACM SIGBED Review, 2013, 10(1): 33-36

[18] Topcuoglu H, Hariri S, Wu M. Performance-effective and low-complexity task scheduling for heterogeneous computing [J]. IEEE Trans on Parallel and Distributed Systems, 2002, 13(3): 260-274

[19] Xie G, Li R, Xiao X, et al. A high-performance dag task scheduling algorithm for heterogeneous networked embedded systems [C] //Proc of IEEE Int Conf on Advanced Information Networking and Applications. Piscataway, NJ: IEEE, 2014: 1011-1016

[20] Hönig U, Schiffmann W. A meta-algorithm for scheduling multiple DAGs in homogeneous system environments [C] //Proc of the Parallel and Distributed Computing and Systems. Piscataway, NJ: IEEE, 2006: 147-152

[21] Zhao Henan, Sakellariou R. Scheduling multiple DAGs onto heterogeneous systems [C] //Proc of the IEEE Int Conf on Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2006: 189-194

[22] Yu Z F, Shi W S. A planner-guided scheduling strategy for multiple workflow applications [C] //Proc of the Int Parallel Processing Workshops. Piscataway, NJ: IEEE, 2008: 1-8

[23] Hsu C C, Huang K C, Wang F J. On line scheduling of workflow applications in grid environments [J]. Future Generation Computer Systems, 2011, 27(6): 860-870

[24] Arabnejad H, Barbosa J. Fairness resource sharing for dynamic workflow scheduling on heterogeneous systems [C] //Proc of the 10th IEEE Int Symp on Parallel and Distributed Processing with Applications. Piscataway, NJ: IEEE, 2012: 633-639

[25] Anderson J, Baruah S, Brandenburg B. Multicore operating-system support for mixed criticality [C] //Proc of the Workshop on Mixed Criticality: Roadmap to Evolving UAV Certification. Piscataway, NJ: IEEE, 2009: 1-11

[26] Mollison M S, Erickso J P, Anderson J H, et al. Mixed-criticality real-time scheduling for multicore systems [C] //Proc of IEEE Int Conf on Computer and Information Technology. Piscataway, NJ: IEEE, 2010: 1864-1871

[27] Tamas-Selicean D, Pop P. Optimization of time-partitions for mixed-criticality real-time distributed embedded systems [C] //Proc of Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW). Piscataway, NJ: IEEE, 2011: 1-10

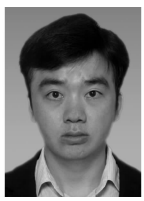
[28] Tamas-Selicean D, Pop P. Design optimization of mixed-criticality real-time applications on cost-constrained partitioned architectures [C] //Proc of IEEE Real-Time Systems Symp (RTSS 2011). Piscataway, NJ: IEEE, 2011: 24-33

[29] Tamas-Selicean D, Marinescu S, Pop P. Analysis and optimization of mixed-criticality applications on partitioned distributed architectures [C] //Proc of System Safety, incorporating the Cyber Security Conf. Piscataway, NJ: IEEE, 2012: 1-6

[30] Tamas-Selicean D, Pop P. Optimization of partitioned architectures to support soft real-time applications [C] //Proc of the 20th IEEE Pacific Rim Int Symp on Dependable Computing (PRDC 2014). Piscataway, NJ: IEEE, 2014: 24-33

[31] Kang W, Son S H, Stankovic J A, et al. I/O-aware deadline miss ratio management in real-time embedded databases [C] //Proc of IEEE Real-Time Systems Symp (RTSS 2007). Piscataway, NJ: IEEE, 2007: 277-287

Liu Liangjiao, born in 1984. PhD candidate. Student member of China Computer Federation. His main research interests include embedded computing, distributed computing, and automobile electronic systems.



Xie Guoqi, born in 1983. PhD, assistant professor. Member of China Computer Federation. His main research interests include embedded systems, distributed systems, real-time systems, in-vehicle networks, and cyber-physical systems.

Li Renfa, born in 1957. Professor, PhD, and PhD supervisor. Distinguished member of China Computer Federation. His main research interests include embedded computing, distributed computing, automobile electronic systems, and cyber-physical systems(CPS).

Yang Liu, born in 1983. PhD. His main research interests include embedded computing, cloud computing, and software engineering.

Liu Yan, born in 1979. PhD, assistant professor. Member of China Computer Federation. His main research interests include computer architecture, multicore scheduling, distributed computing systems.