计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2015.20140776
52(11):2608-2621,2015

通信竞争的混合关键级系统多 DAG 动态调度策略

刘樑骄[1,2] 谢国琪[1,2] 李仁发[1,2] 杨 柳[3] 谢 勇[4]
[1](湖南大学信息科学与工程学院 长沙 410082)
[2](湖南大学嵌入式与网络计算湖南省重点实验室 长沙 410082)
[3](湖南省发展和改革委员会 长沙 410004)
[4](厦门理工学院计算机与信息工程学院 福建厦门 361024)
(llj1984109@qq.com)

Multiple DAGs Dynamic Scheduling for Mixed-Criticality Systems with Communication Contention

Liu Liangjiao[1,2], Xie Guoqi[1,2], Li Renfa[1,2], Yang Liu[3], and Xie Yong[4]
[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)
[4](*College of Computer and Information Engineering, Xiamen University of Technology, Xiamen, Fujian* 361024)

Abstract The scheduling of mixed-criticality systems with communication contention based on multiple DAGs model is the requirement of automobile electronic systems with heterogeneity and distribution. Firstly, the more accurate "upward rank value" and "earliest finish time" with communication contention on time are implemented to adapt to the heterogeneity of both computing and networking, and the synchronization of task and message in this paper. Then, an algorithm called fairness on multiple DAGs dynamic task and message scheduling (F_MDDTMS) is proposed to reduce scheduling length. An algorithm called criticality on multiple DAGs dynamic task and message scheduling (C_MDDTMS) is proposed to ensure the real-time of higher criticality applications. An algorithm called mixed criticality on multiple DAGs dynamic task and message scheduling (MC_MDDTMS) is also proposed based on the joint of F_MDDTMS algorithm and C_MDDTMS algorithm to ensure hard real-time of higher criticality applications and the activity of lower criticality applications of mixed-criticality systems. Example analysis and experimental results show that the proposed algorithms are excellent on scheduling length, unfairness, worst-case response time (WCRT) and real-time.

Key words communication contention; mixed-criticality systems; multiple DAGs; dynamic scheduling; real-time

摘 要 以多 DAG 模型研究通信竞争的混合关键级系统(mixed-criticality systems)的调度问题是适应现代汽车电子系统异构化和分布式的需要.首先实现通信竞争环境下"向上排序值(upward rank

收稿日期:2014-08-24;**修回日期**:2014-12-16
基金项目:国家自然科学基金项目(61173036,61202102,61300039,61300037,61502405);国家"八六三"高技术研究发展计划基金项目(2012AA01A301-01)
通信作者:谢国琪(xgqman@gmail.com)

value)"和"最早完成时间(earliest finish time)"中时间的精确分析,以适应系统中计算与网络均异构,且任务与消息的同步特征.接着提出公平策略的多 DAG 动态任务与消息调度 F_MDDTMS 算法,以降低系统的调度长度;提出关键级策略的多 DAG 动态任务与消息调度 C_MDDTMS 算法,以确保高关键级应用的实时性;结合 F_MDDTMS 算法和 C_MDDTMS 算法,提出混合关键级策略的多 DAG 动态任务与消息调度 MC_MDDTMS 算法,既确保混合关键级系统中高关键级应用的实时性,又使得低关键级应用得到积极的处理.实例分析和实验结果验证了提出的算法在调度长度、不公平性、最差响应时间和实时性上的优越性.

关键词 通信竞争;混合关键级系统;多 DAG;动态调度;实时性

中图法分类号 TP316

为了提高汽车的安全性和驾驶性能等,作为高端嵌入式系统的现代汽车电子系统由异构 ECU(electronic control unit)、传感器、执行器和网关等 100 多个处理设备组成,并通过 LIN(local interconnect network),CAN(controller area network),FlexRay,MOST(media oriented system transport),Ethernet 等多种异构网络总线互联.系统中的应用数量骤增,体系结构日益复杂,异构性和分布式特征日趋明显[1-2].现代汽车电子由动力控制子系统、底盘控制子系统、安全控制子系统和车身控制子系统等多个子系统组成.每个子系统又可细分为多个子功能或子应用(如安全控制子系统包括防抱死制动、线控刹车等多个子应用),且各子应用共享多种总线.这些子应用为满足不同的需求,分别会采用不同的设计方法,其组成部件也由不同级别的汽车供应商提供[3],造成不同应用在实时性需求上各不相同,并产生不同的安全关键级别.因此现代汽车电子系统是典型的混合关键级系统(mixed-criticality systems)[4],以支持多种安全关键和非安全关键应用的共存.道路车辆-功能安全(road vehicles-functional safety)标准规范"ISO 26262"对汽车电子系统中的应用从低到高划分成 A,B,C,D 这 4 个汽车安全完整性等级(automotive safety integrity level, ASIL)[5],其中 D 为最高等级,其安全性要求最为苛刻.如车身子系统中应用的响应时间可以在 10～100 ms 之间,但是动力控制子系统中应用的响应时间却只限制在 10～100 μs 之间.汽车电子系统中的主动安全和线控等分布式应用增加了车内 ECU 的集成度以及应用内不同任务之间的优先级约束[6-8].近年来相关学者提出了时序链[6]、功能链[7]和任务链[8]等模型以适应任务间的优先级约束问题,但仅限于简单的汽车电子应用.随着应用的日益复杂化与并行化,迫切需要 1 种能够更加精确反映应用功能特性的计算模型.汽车电子系统是典型的异构分布式嵌入式系统,在并行与分布式领域,任务间的优先级约束常用有向无环图(directed acyclic graph, DAG)表示[9-11].需要指出的是,本文的"关键级"是从应用功能的角度强调该应用发生安全事故的严重性后果,以表示应用的重要程度,并且经过了第三方机构的严格认证;而"优先级"则强调某个应用内任务之间的数据依赖关系,是任务在时序上的优先级关系.因此本文中的"关键级"与"优先级"面对的对象不同,它们有着本质区别.

调度是混合关键级系统的核心研究内容,但传统的调度往往被认为是任务调度或处理机调度,并假设 DAG 任务调度中的通信是完全连接的,即在任何时刻都可用[12].汽车电子系统是以网络为特征的嵌入式系统,计算和网络具有同等重要的地位,单纯考虑计算单元的任务调度并不能完整地体现调度目标[13-14].为了获得更现实的描述,通信竞争被引进,也即通信并不是在任何时刻都可用,通信资源存在竞争问题,必须将任务调度与通信消息调度同等对待[15].尽管在分布式领域已有相关成果考虑通信竞争问题,但缺乏对任务与消息的同步调度,使得结果不够精确,无法适应计算与网络都异构且深度融合的混合关键级汽车电子系统.

为此,本文以通信竞争的混合关键级汽车电子系统为例,建立动态多 DAG 模型,以 HS-CAN/LS-CAN 异构网络作为实验环境,研究其若干调度策略.DAG 的公平性是降低多 DAG 调度长度的有效方法,但是现有公平策略的动态调度在处理新 DAG 达到时,会因处理器忙碌而出现阻塞,并造成性能急剧下降,这对时间严格苛刻的混合关键级汽车电子系统而言是无法容忍的.实时性是汽车电子系统

安全可靠运行的前提,传统方法都从“任务级”的角度来分析,但汽车电子系统是复杂化和分布式的混合关键级系统,需从“应用级”的角度来确保端到端的最差响应时间(worst-case response time, WCRT),而且,就我们所知,这是首次以多 DAG 模型从“应用级”的角度研究通信竞争的混合关键级系统的调度问题.

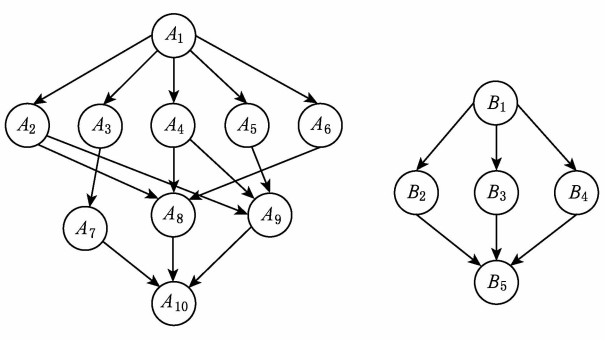
1 系统模型

1.1 多 DAG 系统模型

以主动安全系统中的线控刹车应用为例,从感知数据开始到执行刹车指令结束,所包含的计算任务存在明显的优先级约束,整个应用分配给 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打包成消息在网络总线上传输,消息传输过程跨越 2 种异构总线网络. 线控刹车应用从感知、处理和执行命令的整个过程以降低端到端的执行和传输时间作为主要目标[16].

本文用 DAG 来描述 1 个应用,用 $D=\{N,\boldsymbol{W},E,C\}$ 表示,其中 $N=\{n_1,n_2,\cdots,n_i\}$ 表示任务集合. $pred(n_i)$ 表示任务 n_i 的直接前驱任务集合,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. 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};应用端到端,也即 DAG 中的入口任务到出口任务. $P=(p_1,p_2,\cdots,p_k)$ 表示异构处理器集合. 由于处理器处理能力的异构性,使得不同任务在不同处理器上的计算开销不尽相同,因此描述 $\boldsymbol{W}$ 是 1 个 $|N|\times|P|$ 的矩阵,$w_{i,k}$ 表示任务 n_i 在 p_k 上的计算开销. $E=\{e_{1,2},e_{2,3},\cdots,e_{i,j}\}$ 表示具有优先级约束的任务间的通信消息. 任务间的实际通信开销取决于实际的通信路径,因此在模型中不能给出实际的通信开销值,而只能以平均通信开销表示. 当前几乎所有文献都将任务间的平均通信开销作为通信开销值[1-2,17-18],即 $C=\{c_{1,2},c_{2,3},\cdots,c_{i,j}\}$. 为了获得更加精确的通信开销值的表示,本文考虑将通信开销关联到具体的处理器,即在不同的处理器上有不同的通信开销值,也即 $C=\{c_{1,2}^{p_1},c_{1,2}^{p_2},\cdots,c_{1,2}^{p_k},c_{2,3}^{p_1},c_{2,3}^{p_2},\cdots,c_{2,3}^{p_k},\cdots,c_{i,j}^{p_1},c_{i,j}^{p_2},\cdots,c_{i,j}^{p_k}\}$ 表示消息在不同源处理上 $(p_1,p_2,\cdots,p_k)$ 的平均通信开销集合. 如果连接消息的 2 个任务分配到同一处理器上,则通信开销为 0.

单 DAG 模型只能抽象汽车电子系统中的单个应用,若要描述整个分布式汽车电子系统,则需用多 DAG(multiple DAGs)[2,19] 来表示. 相互之间具有优先级约束的任务集表示 1 个 DAG 应用,各 DAG 共享处理器和通信总线以实现协同调度. 如图 1 为由 2 个 DAG 组成的 1 个多 DAG 系统,包含 DAG-*A* 和 DAG-*B*,表 1 为相应的多 DAG 计算开销矩阵,表 2 为多 DAG 通信开销矩阵. 例如图 1 中,A_1 与 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;表 1 中 A_1 与 p_1 所结合表示的数字 14 表示任务 A_1 在处理器 p_1 上的计算开销为 14. 表 2 中 $e_{1,2}$ 与 p_1 所结合表示的数字 18 表示连接任务 A_1 与 A_2 的消息 $e_{1,2}$ 在处理器 p_1 上的通信开销为 18;若 A_1 与 A_2 分配在同一处理器上,则通信开销为 0.



(a) DAG-*A* (b) DAG-*B*

Fig. 1 Instance of multiple DAGs model.

图 1 多 DAG 模型实例

Table 1 Computation Cost of Tasks in DAG-*A* and DAG-*B* on Different Processors

表 1 DAG-*A* 和 DAG-*B* 中各任务在处理器上的计算开销

Processor	DAG-*A*										DAG-*B*				
	A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8	A_9	A_{10}	B_1	B_2	B_3	B_4	B_5
p_1	14	13	11	13	12	13	7	5	18	21	4	9	18	21	7
p_2	8	19	13	8	13	16	15	11	12	7	5	10	17	15	6
p_3	16	18	19	17	10	9	11	14	20	16	6	11	16	19	5

Table 2 Communication Cost of Messages in DAG-*A* and DAG-*B* on Different Processors

表 2 DAG-*A* 和 DAG-*B* 中各消息在处理器上的通信开销

Processor	DAG-*A*															DAG-*B*					
	$e_{1,2}$	$e_{1,3}$	$e_{1,4}$	$e_{1,5}$	$e_{1,6}$	$e_{2,8}$	$e_{2,9}$	$e_{3,7}$	$e_{4,8}$	$e_{4,9}$	$e_{5,9}$	$e_{6,8}$	$e_{7,10}$	$e_{8,10}$	$e_{9,10}$	$e_{1,2}$	$e_{1,3}$	$e_{1,4}$	$e_{2,5}$	$e_{3,5}$	$e_{4,5}$
p_1	18	12	9	11	14	19	16	23	27	23	13	15	17	13	11	5	6	2	4	8	10
p_2	12	8	6	7	9	13	11	15	18	15	9	10	11	9	7	3	4	1	3	5	7
p_3	12	8	6	7	9	13	11	15	18	15	9	10	11	9	7	3	4	1	3	5	7

1.2 异构网络模型

为了满足不同应用在网络方面的特定需求,现代汽车上采用了多种类型的网络技术.汽车网络不再是完全的总线拓扑结构,而是混合了多种网络类型的混合拓扑结构,包括总线型、星型、环型、树型、网型甚至任意网络类型(arbitrary types of network)[17].比如 CAN 和 FlexRay 一般采用总线型,但 MOST 在汽车网络中一般采用环形拓扑.因此,与业界研究和使用较多的单一总线型、树型、星型和任意类型网络等传统网络拓扑结构存在一定布局规则不同[17-18],汽车电子系统的网络拓扑结构不存在特定的布局规则,而且随着应用的不断复杂化,上述混合网络的拓扑结构将在很长时间内长期共存.由此表明,汽车电子系统中各链路在网络中的使用频率不同,而且同一消息在不同的链路上传输具有不同的通信速率.

异构网络可用无向图 $U=\langle P,G,L\rangle$ 表示,其中 P 表示处理单元集合,也就是 1.1 节描述的异构处理单元集合;G 表示网关集合,并用 $G=\{g_1,g_2,\cdots,g_h\}$ 表示.设任务 n_i 传递给 n_j 的消息表示为 $e_{i,j}^{p_{\text{src}},p_{\text{dest}}}$,其中 p_{src} 表示任务 n_i 所分配的处理器,也就是消息的源处理器,n_i 称作源任务;p_{dest} 表示任务 n_j 预分配的目的处理器,n_j 称作目的任务.那么消息 $e_{i,j}^{p_{\text{src}},p_{\text{dest}}}$ 在 p_{src} 和 p_{dest} 之间可能存在多条通信路径,每条通信路径可认为是处理器之间的 1 条路由(router).每条通信路径或路由可能经过多个处理器或网关等物理设备,系统中的链路集合用 $L=\{l_1,l_2,\cdots,l_v\}$ 表示.p_{src} 和 p_{dest} 之间的通信路径集 $R^{p_{\text{src}},p_{\text{dest}}}=\{r_1^{p_{\text{src}},p_{\text{dest}}},r_2^{p_{\text{src}},p_{\text{dest}}},\cdots,r_z^{p_{\text{src}},p_{\text{dest}}}\}$;每条通信路径又通过处理器或网关由多条链路连接而成,而且每条链路具有不同的通信速率.

图 2 所示为简单的汽车电子系统网络拓扑结构实例 U_1,p_1 与 p_2 之间一共有 2 条通信路径,用集合表示为 $R^{p_1,p_2}=\{r_1^{p_1,p_2},r_2^{p_1,p_2}\}$,其中 2 条通信路径所组成的链路集合为 $r_1^{p_1,p_2}=\{l_1^1,l_2^1\}$,$r_2^{p_1,p_2}=\{l_1^2,l_4^2,l_3^2\}$,即第 1 条通信路径由 l_1 和 l_2 这 2 条链路组成,第 2 条通信路径由 l_1,l_4,l_3 这 3 条链路组成.消息的通信路径查找可用深度优先搜索(deep first search, DFS)或广度优先搜索(breadth first search, BFS)等查找算法实现.

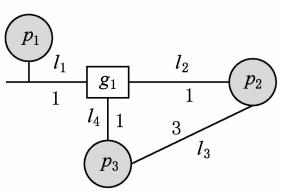


Fig. 2 U_1 Instance of heterogeneous network topology for automobile electronic systems.

图 2 汽车电子系统的异构网络拓扑结构实例 U_1

在通信竞争环境下,文献[17]的网络结构为星型网络拓扑,2 个处理器之间只存在 1 条通信路径;文献[18]提出的任意处理机网络认为任意处理器之间存在 $p\times(p-1)/2$ 条通信路径.上述 2 种结构都过于理想化,不符合汽车电子系统的现实异构网络环境.本文提出的异构网络拓扑结构针对汽车电子系统网络结构的混合特征,基于严格的通信路径查找,认为处理器之间的通信路径个数属于 $[1,p\times(p-1)/2]$ 范围,这也是本文与其他网络拓扑结构的最大区别.

2 相关研究

2007 年 Vestal[4] 在嵌入式实时系统领域顶级国际会议 RTSS(Real-Time Systems Symposium)上首次提出了混合关键级系统的概念及其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.Dorin 等人[20]证明了文献[4]的策略在严格时限约束的独立任务系统中是最优的.Baruah 等人[21]基于一个简单的任务调度模型(有限数量的具有固定释放时间的任务集)对混合关键级任务调度进行了一系列的分析,证明了混合关键级任务调度问题是 NP 难问题.上述混合关键级系统的研究均基于传统的独立任务模型.然而,汽车电子系统从同步的

角度考虑端到端的 WCRT 优化使得传统实时系统中单纯的独立任务模型已不能精确描述任务之间的优先级约束.因此,以分布式为特征的 DAG 模型在汽车电子系统中成为研究热点.

经典的 DAG 调度算法有 HEFT(heterogeneous earliest finish time)[9], CPOP(critical path on a processor)[9], CEFT(constrained earliest finish time)[10]等几十种,还包括考虑通信竞争的 DAG 调度算法[12,15,17-18],但解决面向汽车电子系统的同步问题的首要目标是提出能够适应系统的同步模型. Klobedanz 等人[22]基于 DAG 模型研究了 FlexRay 的静态段配置容错调度,但该算法仅针对单一网络总线 FlexRay,且限于同一功能子系统内的调度,不适应汽车电子系统的异构化、网络化和复杂化需求.谢勇等人[23]提出融合任务和消息的 DAG 调度模型来对任务和消息之间的同步关系进行建模,并研究其消息调度问题,但未同时考虑任务和消息的同步调度问题. Zeng 等人[13]将端到端的计算抽象成数据流图的 DAG,研究基于 CAN 总线的端到端同步实时调度.接着,Zeng 等人[14]又基于 DAG 模型,研究了时间触发的 FlexRay 静态段的同步调度优化问题. Schmidt 等人[24]提出基于带权有向图的 DAG 模型,研究交换以太网的端到端最长路径与最差响应时间问题.上述算法虽然在基于 DAG 模型研究端到端同步问题上取得了一定的进展,但并没有充分发挥和利用 DAG 在并行与分布式计算方面的优势,使得计算结果不够精确,无法适应汽车电子系统的并行化趋势.

当前基于多 DAG 的混合调度主要应用于科学计算的工作流系统.早期的多 DAG 混合调度以静态调度为主,即调度多个同时发生的 DAG,如将多个 DAG 合成 1 个复合 DAG 的算法[25]、公平算法 Fairness[17].近几年,关于多 DAG 的研究集中在动态调度上,即在任意时刻可提交新的 DAG 到多 DAG 系统中.例如 RANK_HYBD(rank hylorid prioritization algorithm)[26], OWM(online workflow management)[27], FDWS(fairness dynamic workflow scheduling)[28]等,但这些算法都面向工作流系统,没有考虑通信竞争问题,且在动态环境下因处理器忙碌而易出现阻塞并造成不公平性的现象,无法适应汽车电子系统对时间的苛刻要求及任务与消息的同步调度.文献[2]作为我们最新的研究成果,基于现有多 DAG 调度的研究成果,首次将汽车电子系统抽象为多 DAG 静态调度模型,但针对的是静态调度且只考虑了通信开销.本文不仅考虑通信开销,还引入通信竞争.本文相比文献[2]的先进之处主要在于:1)从 DAG 应用功能端到端的角度研究计算任务与通信消息的同步调度问题;2)考虑动态环境下,新 DAG 达到时的多 DAG 调度策略.由于汽车电子系统是计算与网络均异构且深度融合的系统,因此对通信竞争的混合关键级汽车电子系统的多 DAG 调度的研究提出了严峻的挑战.

3 通信竞争下的计算精确化

众所周知,DAG 调度包含任务优先级排序与处理器分配 2 个核心步骤.任务优先级排序一般基于向上排序值(upward rank value)[9-10,25];处理器分配则是基于最早完成时间(earliest finish time)[9-10,25].

1) 向上排序值的精确化

一般地,向上排序值 $rank_u(n_i)$的计算公式为

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

其中,$\overline{w_i}$为任务 n_i 的平均计算开销,$\overline{c_{i,j}}$为任务 n_i 与 n_j 的平均通信开销.但是,在通信竞争环境下,由于需要同时考虑消息调度及任务调度,因此传统的计算公式 $rank_u(n_i)$已不再适应.我们用 $rank_u(n_i,p_k)$表示新的向上排序值,其计算公式为

$$\begin{cases}rank_u(n_i,p_k)=w_{i,k}+\\ \quad\max\limits_{n_j\in succ(n_i)}\{rank_u(n_j,p_k)+c_{i,j}^{p_k}\},\\ rank_u(n_{exit},p_k)=w_{exit,k}.\end{cases}\quad(2)$$

$rank_u(n_i,p_k)$与 $rank_u(n_i)$的区别在于前者将计算开销 $w_{i,k}$和通信开销 $c_{i,j}^{p_k}$都关联到了具体的处理器,体现了计算与通信均异构的特点.接着,我们将 $rank_u(n_i,p_k)$均转化为 $hrank_u(n_i)$,其计算结果相比 $rank_u(n_i)$能得出更加精确的排序,计算公式为

$$hrank_u(n_i)=\frac{1}{|P|}\times\sum_{p_k\in P}rank_u(n_i,p_k).\quad(3)$$

2) 最早完成时间的精确化

一般地,最早开始时间的计算公式 $EST(n_j,p_k)$的计算方法为

$$\begin{cases}EST(n_{entry},p_k)=0,\\ EST(n_j,p_k)=\max\{avail[p_k],\\ \quad\max\limits_{n_i\in pred(n_j)}\{AFT(n_i)+\overline{c_{i,j}}\}\}.\end{cases}\quad(4)$$

其中,$avail[p_k]$表示处理器 p_k 的最早可用时间,$AFT(n_i)$表示任务 n_i 的实际完成时间.

同样地,在通信竞争环境下,基于任务与消息的同步性,任务在目的处理器 p_{dest}的最早开始时间还与其消息传输的通信路径 $r_z^{p_{src},p_{dest}}$ 密切相关,即

$$\begin{cases} EST(n_{\text{entry}}, p_{\text{dest}}, r_z^{p_{\text{src}},p_{\text{dest}}}) = 0, \\ EST(n_j, p_{\text{dest}}, r_z^{p_{\text{src}},p_{\text{dest}}}) = \max\{avail[p_{\text{dest}}], \\ \quad \max_{n_i \in pred(n_j), proc(n_i)=p_{\text{src}}, p_{\text{src}} \in P} \{MFT(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, r_z^{p_{\text{src}},p_{\text{dest}}})\}\}, \end{cases} \tag{5}$$

其中,$MFT(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, r_z^{p_{\text{src}},p_{\text{dest}}})$表示消息 $e_{i,j}^{p_{\text{src}},p_{\text{dest}}}$ 在通信路径 $r_z^{p_{\text{src}},p_{\text{dest}}}$ 的完成时间(只有消息传输完成后,任务才能开始执行),也就是在通信路径 $r_z^{p_{\text{src}},p_{\text{dest}}}$ 中最后 1 条链路 l_{end}^z 的完成时间,即

$$\begin{cases} EST(n_{\text{entry}}, p_{\text{dest}}, r_z^{p_{\text{src}},p_{\text{dest}}}) = 0, \\ EST(n_j, p_{\text{dest}}, r_z^{p_{\text{src}},p_{\text{dest}}}) = \max\{avail[p_{\text{dest}}], \\ \quad \max_{n_i \in pred(n_j), proc(n_i)=p_{\text{src}}, p_{\text{src}} \in P} \{LFT(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, \\ \quad l_{\text{end}}^z, r_z^{p_{\text{src}},p_{\text{dest}}})\}\}. \end{cases} \tag{6}$$

$LFT(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, l_{\text{end}}^z, r_z^{p_{\text{src}},p_{\text{dest}}})$中,$l_{\text{end}}^z$ 表示通信路径 $r_z^{p_{\text{src}},p_{\text{dest}}}$ 中的最后 1 条链路.关于 LFT 的详细定义和计算方法将在定义 2 给出.

由式(6)可知 EST 与 LFT 紧密关联,任务启动某消息后,此消息不一定会立即传输,因为在通信竞争条件下,相关的通信链路正被其他消息占用,因此需要等待链路空闲时才能传输.同样,通信竞争条件下,任务在通信路径上的最早完成时间的计算公式应为

$$EFT(n_j, p_{\text{dest}}, r_z^{p_{\text{src}},p_{\text{dest}}}) = EST(n_j, p_{\text{dest}}, r_z^{p_{\text{src}},p_{\text{dest}}}) + w_{j,\text{dest}}. \tag{7}$$

式(4)~(7)表明:在融合异构网络的通信竞争条件下,任务的最早开始时间融合了消息的传输时间;而消息的传输时间,又受限于通信路径中链路的实际完成时间.为此,我们首先需要计算链路的开始时间.

定义 1. 链路的开始时间(link start time, LST).$LST(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, l_{x+1}^z, r_z^{p_{\text{src}},p_{\text{dest}}})$表示消息 $e_{i,j}^{p_{\text{src}},p_{\text{dest}}}$ 在第 z 条通信路径 $r_z^{p_{\text{src}},p_{\text{dest}}}$ 上传输时第 $x+1$ 条链路的开始时间,递归计算公式为

$$\begin{cases} LST(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, l_1^z, r_z^{p_{\text{src}},p_{\text{dest}}}) = \\ \quad \max\{AFT(n_i^{p_{\text{src}}}), avail(l_1^z)\}, \\ LST(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, l_{x+1}^z, r_z^{p_{\text{src}},p_{\text{dest}}}) = \\ \quad \max\{LST(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, l_x^z, r_z^{p_{\text{src}},p_{\text{dest}}}), avail(l_{x+1}^z)\}. \end{cases} \tag{8}$$

定义 2. 链路的完成时间(link finish time, LFT).$LFT(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, l_{x+1}^z, r_z^{p_{\text{src}},p_{\text{dest}}})$表示在通信路径 $r_z^{p_{\text{src}},p_{\text{dest}}}$ 上第 $x+1$ 条链路的最早完成时间,其值为链路的最早开始时间与链路的通信时间之和.

为了简化链路的最早完成时间的计算公式,首先得出消息在链路上的通信时间(message link communication time, MLCT),其计算公式为

$$MLCT(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, l_x^z) = \frac{w(e_{i,j}^{p_{\text{src}},p_{\text{dest}}})}{speed(l_x^z)}, \tag{9}$$

其中,$speed(l_x^z)$表示链路 l_x^z 的通信速率.结合式(8)(9),可得出链路的完成时间为

$$\begin{cases} LFT(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, l_1^z, r_z^{p_{\text{src}},p_{\text{dest}}}) = \\ \quad LST(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, l_1^z, r_z^{p_{\text{src}},p_{\text{dest}}}) + \\ \quad MCLT(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, l_1^z), \\ LFT(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, l_{x+1}^z, r_z^{p_{\text{src}},p_{\text{dest}}}) = \\ \quad \max\{LFT(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, l_x^z, r_z^{p_{\text{src}},p_{\text{dest}}}), \\ \quad LST(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, l_{x+1}^z, r_z^{p_{\text{src}},p_{\text{dest}}}) + \\ \quad MCLT(e_{i,j}^{p_{\text{src}},p_{\text{dest}}}, l_{x+1}^z)\}. \end{cases} \tag{10}$$

通过上述分析,通信竞争环境下,在计算任务的最早开始时间上,本文考虑了任务执行与消息传输的同步特征,即从应用级的角度融合了计算任务和通信消息,处理器和通信路径的分配也可同步获得.相比文献[12,15,17-18]将任务的最早完成时间和链路的完成时间分开计算与处理而言,本文提出的方法能够产生更加精确的结果,这也是在考虑通信竞争条件下本文相比其他方法在计算方法上的最大改进.

4 调度算法

4.1 动态调度模型

动态调度模型是分析动态调度算法的基础,动态调度与静态调度的最大区别在于前者可以在任意时刻提交新的 DAG 到系统中.如图 3 所示,假设 DAG-A 按某种调度算法已处于调度执行过程中,在时刻 35 有 1 个新的 DAG-B 到达,那么 DAG-A 中的所有任务可以分为 4 部分:1)已经调度完毕的任务组(A_1,A_2,A_4);2)正在调度的任务组(A_3,A_6);3)已分配处理器但未开始调度的任务组(A_5,A_7,A_9);4)未选择处理器且未调度的任务组(A_8,A_{10}).由于新到达 DAG 应用的不确定性,本文约定每个 DAG 应用都偶发性到达,也即偶发性到达的动态多 DAG 模型;同时约定正在执行的任务不可剥夺,也即采用非抢占式的任务调度策略.

动态通信竞争环境下的最早开始时间 EST 已不能由式(6)表示,假设 1 个新的 DAG 应用 D_m 在时刻 t_a 到达,那么动态环境下 D_m 中的 EST,EFT 分别调整为

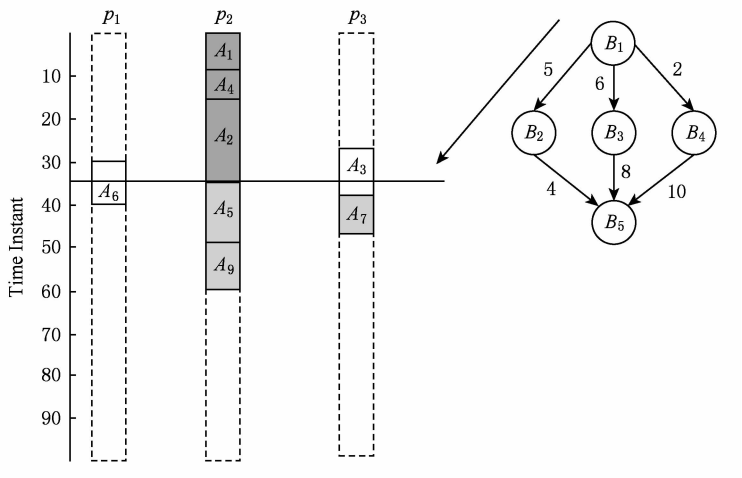


Fig. 3 Example of multiple DAGs dynamic scheduling.

图 3 多 DAG 动态调度示例

$$EST_{\mathrm{d}}(n_j^{p_{\mathrm{dest}}}, p_{\mathrm{dest}}, r_z^{p_{\mathrm{src}},p_{\mathrm{dest}}}) = \max\{EST(n_j^{p_{\mathrm{dest}}}, p_{\mathrm{dest}}, r_z^{p_{\mathrm{src}},p_{\mathrm{dest}}}), t_a\} \quad (11)$$

和

$$EFT_{\mathrm{d}}(n_j^{p_{\mathrm{dest}}}, p_{\mathrm{dest}}, r_z^{p_{\mathrm{src}},p_{\mathrm{dest}}}) = EFT_{\mathrm{d}}(n_j^{p_{\mathrm{dest}}}, p_{\mathrm{dest}}, r_z^{p_{\mathrm{src}},p_{\mathrm{dest}}}) + w_{j,\mathrm{dest}}. \quad (12)$$

4.2 公平调度策略

和许多其他多 DAG 调度一样,通过公平调度来降低整个系统的调度长度和提高性能是行之有效的方法.多 DAG 动态调度一般包括任务排序、任务就绪、处理器分配、新 DAG 到来和任务运行这 5 个步骤.我们引入 3 种需要使用的优先级队列加以说明.

1) DAG 任务优先级队列 Task_Priority_Queue.每个 DAG 都拥有 1 个,此队列将 DAG 中的任务按某种方式排列.

2) 多 DAG 公共就绪队列 Commom_Ready_Queue.所有 DAG 共享 1 个,用于存储就绪任务,同样地,任务也按某种方式排列.

3) 处理器任务队列 Process_Task_Queue.每个处理器都拥有 1 个,用于存储分配了处理器的任务.值得注意的是,该队列中包含"已分配处理器但未开始调度的任务",这是动态环境下才引入的队列,也是与静态调度的主要区别.

根据上述分析,在通信竞争环境下,提出公平策略的多 DAG 动态任务与消息调度(fairness on multiple DAGs dynamic task and message scheduling, F_MDDTMS)算法.

算法 1. F_MDDTMS 算法.

① 初始化:依据式(2)(3)分别计算每个 DAG 中任务的向上排序值 $rank_{\mathrm{u}}(n_j, p_k)$ 和 $hrank_{\mathrm{u}}(n_j)$,并按 $hrank_{\mathrm{u}}(n_i)$ 的递减顺序将任务放入各 DAG 的任务优先级队列 Task_Priority_Queue;

② while 有 DAG 可以调度 do

③ 检查每个处理器任务队列 Process_Task_Queue 中是否存在任务的最早开始时间 EST_{d} 等于当前时间,如果有则进行调度,并标记此任务为正在调度的任务;

④ 检查每个 Process_Task_Queue 中是否存在任务的最早完成时间 EST_{d} 小于等于当前时间,如果有则标记此任务为已调度完成的任务;

⑤ 如果有新 DAG 到达,计算新 DAG 中任务的 $rank_{\mathrm{u}}(n_j, p_k)$, $hrank_{\mathrm{u}}(n_j)$,并按 $hrank_{\mathrm{u}}(n_j)$ 的递减顺序将任务放入该 DAG 的 Task_Priority_Queue,同时将所有 Process_Task_Queue 中未调度的任务也放回相应的 Task_Priority_Queue;

⑥ 分别从各 DAG 的 Task_Priority_Queue 中选择 $hrank_{\mathrm{u}}(n_j)$ 最大的就绪任务,放入多 DAG 公共就绪队列 Commom_Ready_Queue;

⑦ while Commom_Ready_Queue 中有任务可以调度 do

⑧ 从 Commom_Ready_Queue 中选择 $hrank_{\mathrm{u}}(n_j)$ 最大的任务 n_{s};

⑨ 计算 n_{s} 在每个处理器上的 EST_{d};

⑩ 在插入法的基础上,将 n_{s} 分配到 EST_{d} 最小的处理器上,实现任务和消息的同步分配;

⑪ if 当前处理器 p_k 空闲

⑫ 将 n_s 分配到 p_k 上进行调度,标记 n_s 为正在调度的任务;

⑬ else

⑭ 将 n_s 放入 p_k 的 Process_Task_Queue,标记为已分配处理器但未开始调度任务;

⑮ end if

⑯ end while

⑰ end while

⑱ return.

F_MDDTMS 算法的时间复杂度为 $O(n^2\times p^3\times l\times d)$. 其中,$n$ 为拥有最多任务数的 DAG 中的任务数,d 为 DAG 数,p 为处理器数,l 为链路数. 相比 OWN 与 FDWS 等多 DAG 动态任务调度算法,本文提出的公平策略的最大改进在于当有新 DAG 调度的时候,不像 OWN 算法或者 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 一起进行公平调度.

基于图 1 提供的多 DAG 实例(DAG-B 在时刻 35 到达,且其截止时限为 75),图 4(a)为该 F_MDDTMS 算法的 Gannt 图,调度长度为 85,t_a 为到达时间(arrival time). 多 DAG 调度长度即在多 DAG 中调度长度最大 DAG 的调度长度.

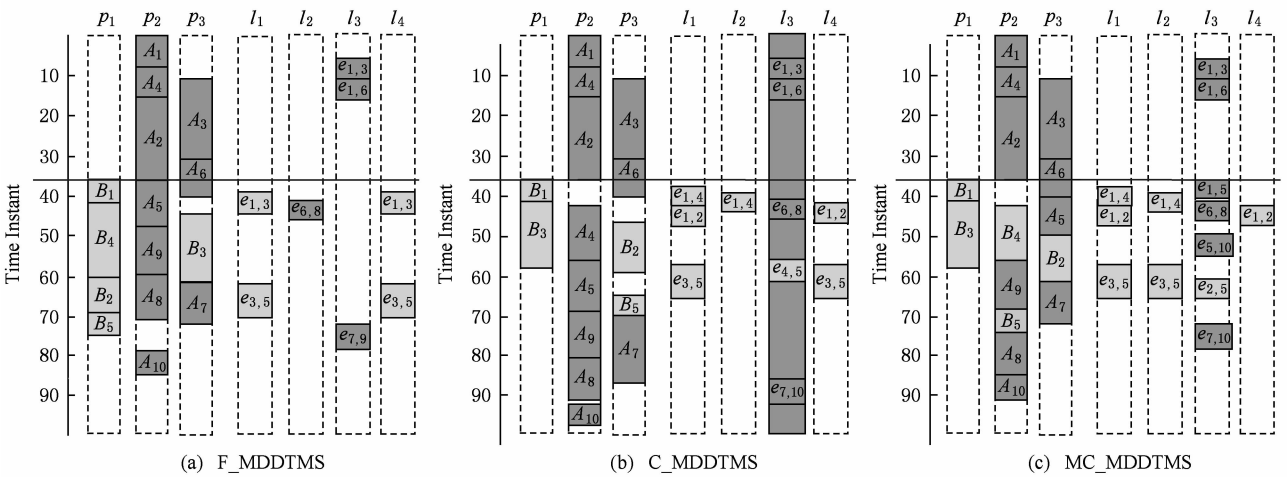


Fig. 4 Gannt chart of scheduling for instance of multiple DAGs using three algorithms($t_a=35$, $deadline_b=75$).

图 4 多 DAG 实例采用 3 种算法的调度 Gannt 图($t_a=35$, $deadline_b=75$)

4.3 混合关键级调度策略

混合关键级系统的核心是将以前相互独立的子应用集成到统一平台上运行,一方面既需要保证高安全关键应用能够通过 CA 的认证,满足其实时性;另一方面又要尽可能地提高系统的调度性能. 若从确保强实时的角度来看,则需将资源尽可能多地分配给高安全关键应用,并满足严格的截止时限即可;若从提高系统性能的角度来看,那么所有应用总的调度长度越短愈好. 因此如何在强实时性和高性能上取得符合需求的平衡才是混合关键级系统调度的核心所在. 基于公平策略的多 DAG 任务调度算法无论在静态和动态环境下都具有较好的性能,但显然无法适应混合关键级系统中多种关键级应用共存的环境. 如当汽车发生碰撞事故时,安全气囊要在恰当的时间打开,太早可能对驾驶员或乘客造成未提前感知的伤害,太迟则不能发挥其正常的保护作用. 因此,既要满足高关键级应用的实时性需求,但又不能因过度分配资源以追求高安全关键级应用的正确性而忽视低安全关键级应用的执行.

1)关键级策略的多 DAG 任务与消息调度

根据混合关键级系统的多关键级特性,为了在强实时和高性能上取得平衡,本文采用混合关键级调度策略. 在提出混合关键级调度策略前,先提出关键级策略. 该策略的核心思想是:在任务分配上,高关键级 DAG 应用中的所有任务都要优先于低关键级 DAG 应用中的所有任务,即只有当高关键级 DAG 应用中的所有任务分配处理器后,低关键级 DAG 应用中的任务才能分配处理器.

在关键级策略中,DAG 的入口任务 n_{entry} 的开始时间也得到了一定的推迟. 因此,其所有后继任务的最早完成时间的计算要作相应调整:

$$EFT'_{\text{d}}(n_j,p_{\text{dest}},r_z^{p_{\text{src}},p_{\text{dest}}})=EFT_{\text{d}}(n_j,p_{\text{dest}},r_z^{p_{\text{src}},p_{\text{dest}}})-EST_{\text{d}}(n_{\text{entry}},p_{\text{dest}},r_z^{p_{\text{src}},p_{\text{dest}}}). \tag{13}$$

接下来,提出关键级策略的多 DAG 动态任务与消息调度(criticality on multiple DAGs dynamic task and message scheduling, C_MDDTMS)算法.

算法 2. C_MDDTMS 算法.

① 初始化:分别计算每个 DAG 中任务的向上排序值 $rank_u(n_j, p_k)$ 和 $hrank_u(n_j)$,并按 $hrank_u(n_i)$的递减顺序将任务放入各 DAG 的任务优先级队列 Task_Priority_Queue;

② 根据各 DAG 认证的关键级,按关键级的递减顺序将各 DAG 放入多 DAG 关键级队列 Commom_Criticality_Queue;

③ while 有 DAG 可以调度 do

④ 检查每个处理器任务队列 Process_Task_Queue 中是否存在任务的最早开始时间 EST_d 等于当前时间,如果有则进行调度,并标记此任务为正在调度的任务;

⑤ 检查每个 Process_Task_Queue 中是否存在任务的最早完成时间 EST_d 小于等于当前时间; 如果有则标记此任务为已调度完成的任务;

⑥ 如果有新 DAG 到达,计算新 DAG 中各任务的 $rank_{ucc}(n_j, p_k)$和 $hrank_{ucc}(n_j)$,并将任务放入该 DAG 的 Task_Priority_Queue,根据此 DAG 的关键级,放入 Commom_Criticality_Queue;同时将所有 Process_Task_Queue 中未调度的任务也放回相应 DAG 的 Task_Priority_Queue,从 Commom_Criticality_Queue 中取出未调度完成中关键级最高的 DAG,命名此 DAG 的任务队列为 Q_s;

⑦ while Q_s 中有任务可以调度 do

⑧ 从 Q_s 中选择 $hrank_u(n_j)$最大的就绪任务 n_s;

⑨ 在插入法的基础上,将 n_s 分配到 EFT'_d 最小的处理器上和通信路径上,实现对任务与消息的同步分配;

⑩ if 当前处理器 p_k 空闲

⑪ 将 n_s 分配到 p_k 上进行调度,标记 n_s 为正在调度的任务;

⑫ else

⑬ 将 n_s 放入 p_k 的 Process_Task_Queue,标记为已分配处理器但未开始调度任务;

⑭ end if

⑮ end while

⑯ end while

⑰ return.

C_MDDTMS 算法的时间复杂度为 $O(n^2 \times p^3 \times l \times d)$. 其中,$n$ 为拥有最多任务数的 DAG 中的任务数,d 为 DAG 数,p 为处理器数,l 为链路数. 基于图 1 提供的多 DAG 实例(DAG-B 在时刻 35 到达,且其截止时限为 75),图 4(b)为 C_MDDTMS 算法的 Gannt 图,调度长度为 100.

2) 混合关键级策略的动态 DAG 动态任务与消息调度

基于关键级策略的 C_MDDTMS 算法虽然能够满足实时性,但其付出的代价是使低关键级 DAG 应用未得到积极处理,从而加大了整个多 DAG 的调度长度. 在任务分配上,实时系统并不要求高关键级 DAG 应用中的所有任务都要优先低关键级 DAG 应用中的所有任务,而是只要在截止时限内调度完成高关键级 DAG 应用中的所有任务即可,并且还能从整体降低多 DAG 的调度长度,这样在保证实时性的前提下又有效降低了调度长度,达到高性能与强实时的平衡,这也就是混合关键级系统的核心调度目标. 为此,我们提出混合关键级策略的多 DAG 动态任务与消息调度(mixed criticality on multiple DAGs dynamic task and message scheduling, MC_MDDTMS) 算法.

算法 3. MC_MDDTMS 算法.

① 初始化:分别计算每个 DAG 中任务的向上排序值 $rank_u(n_j, p_k)$ 和 $hrank_u(n_j)$,并按 $hrank_u(n_i)$的递减顺序将任务放入各 DAG 的任务优先级队列 Task_Priority_Queue;

② 根据各 DAG 认证的关键级,按关键级的递减顺序将各 DAG 放入多 DAG 关键级队列 Commom_Criticality_Queue;

③ while 有 DAG 可以调度 do

④ 从 Commom_Criticality_Queue 中取出未调度完成且关键级最高的 DAG 为 D_m,其截止时限为 $deadline(D_m)$;

⑤ 用 F_MDDTMS 算法单独调度 D_m,计算其调度长度为 $makespan(D_m)$;

⑥ if $makespan(D_m) \leqslant deadline(D_m)$

⑦ while D_m 中有任务可以调度 do

⑧ 用 F_MDDTMS 算法预调度多 DAG 系统,并更新 D_m 的调度长度 $makespan(D_m)$;

⑨ 从 D_m 中取出 $hrank_u(n_j)$ 最小的就绪任务 n_s;
⑩ if $makespan(D_m) \leqslant deadline(D_m)$
⑪ 用 F_MDDTMS 调度算法调度任务 n_s;
⑫ else
⑬ 用 C_MDDTMS 调度算法调度任务 n_s;
⑭ end if
⑮ end while
⑯ else
⑰ 该多 DAG 系统不可调度;
⑱ end if
⑲ end while
⑳ return.

MC_MDDTMS 算法的时间复杂度为 $O(n^3 \times p^3 \times l \times d^2)$. 其中,$n$ 为拥有最多任务数的 DAG 中的任务数,d 为 DAG 数,p 为处理器数,l 为链路数. 基于图 1 提供的多 DAG 实例(DAG-B 在时刻 35 到达,且其截止时限为 75),图 4(c)为 MC_MDDTMS 算法的 Gannt 图,调度长度为 92.

不公平性(unfairness)是评价多 DAG 调度的核心指标[17],不公平性的具体计算方法为首先计算某个 DAG 的 $slowdown$ 值:

$$slowdown(D_m) = \frac{makespan_{own}(D_m)}{makespan_{multi}(D_m)}, \quad (14)$$

其中,$makespan_{own}(D_m)$表示 D_m 单独调度的调度长度,$makespan_{multi}(D_m)$表示 D_m 在多 DAG 中调度的调度长度. 基于 $slowdown(D_m)$计算多 DAG 系统调度的不公平性,其计算公式为

$$unfariness(MD) = \sum_{D_m \in MD} | slowdown(D_m) - \frac{1}{|MD|} \sum_{D_m \in MD} slowdown(D_m) |. \quad (15)$$

表 3 为 F_MDDTMS,C_MDDTMS,MC_MDDTMS 这 3 种算法调度结果比较. F_MDDTMS 算法在公平性和调度性能上都最好,但不能满足 DAG-B 的截止期限为 75 的要求;基于关键基策略的 C_MDDTMS 算法在公平性和多 DAG 整体性能上最差,但能满足 DAG-B 的截止期限要求;MC_MDDTMS 算法综合考虑 F_MDDTMS 算法和 C_MDDTMS 算法的特点,采用混合关键级策略,在确保高关键级 DAG 应用实时性的基础上又具有较好的性能,在强实时和高性能上取得了较好的平衡.

Table 3 Comparison of Dynamic Scheduling for Instance of Multiple DAGs Using Three Algorithms($t_a=35$, $deadline_b=75$)

表 3 多 DAG 实例采用 3 种策略算法的动态调度结果比较($t_a=35$, $deadline_b=75$)

Algorithm	Priority		Scheduling Length			Unfairness	Time Complexity
	Before Time Instant of 35	After Time Instant of 35	*makespan* (DAG-*A*)	*makespan* (DAG-*B*)	*makespan* (*MD*)		
F_MDDTMS	A_1, A_4, A_2, A_3, A_6	$B_1, A_5, B_4, A_9, B_3, A_7, B_2, A_8, B_5, A_{10}$	85	76	85	0.105 9	$O(n^2 \times p^3 \times l \times d)$
C_MDDTMS	A_1, A_4, A_2, A_3, A_6	$B_1, B_4, B_3, B_2, B_5, A_5, A_9, A_7, A_8, A_{10}$	100	70	100	0.3	$O(n^2 \times p^3 \times l \times d)$
MC_MDDTMS	A_1, A_4, A_2, A_3, A_6	$B_1, B_4, A_5, B_3, A_9, B_2, A_7, B_5, A_8, A_{10}$	92	72	92	0.217 4	$O(n^3 \times p^3 \times l \times d^2)$

5 实 验

5.1 评价指标

调度长度比(schedule length ratio, SLR)是评价 DAG 调度性能的核心指标[9-10];端到端最差响应时间 WCRT 是评价分布式汽车电子系统性能的核心指标[1-2],因此本文选用上述指标和不公平性作为实验评价标准.

对于多个 DAG,SLR 即多 DAG 算法实际调度长度与 DAG 关键路径任务执行时间的最小值之比. 调度算法产生的 SLR 越小,说明算法越高效,SLR 计算公式为

$$SLR(MD) = \frac{makespan(MD)}{\min_{p_k \in P}\{\sum_{D_m \in MD}(\sum_{n_i \in CPL(D_m)} w_{i,k})\}}. \quad (16)$$

其中,CPL 表示关键路径长度(critical path length).

实验的硬件环境为 1 台奔腾双核处理器(3.2 GHz/2.0 GB RAM)的 Windows XP 机器,所使用的软件工具为编程语言 Java,DAG 任务图生成工具 TGFF 3.5(task graphs for free)[1-2]. 根据不同参数生成各种特性不同的加权 DAG. 多 DAG 系统的 DAG 数 $d \in \{2,4,10,20,40,60,80,100\}$. 产生随机

DAG 的参数设置任务个数 $n\in\{30,40,50,60,70,80,90,100\}$，DAG 形状参数 $\alpha\in\{0.5,1.0,2.0,4.0\}$，最大出度 $\beta\in\{1,2,3,4,5\}$，最大入度 $\gamma\in\{1,2,3,4,5\}$，通信开销与计算开销的比值 $CCR\in\{0.1,0.5,1.0,5.0,10.0\}$，任务在不同处理器上执行时间的差异度 $\eta\in\{0.1,0.5,1.0,2.0,4.0\}$，处理器链路个数 $\mu\in\{1,5,9,13\}$，DAG 到达时间间隔(arrival interval, AI) $AI\in\{0,50,100,150,200,250,300,350,400\}$，增加 D_m 的截止时限长度统一为 $makespan(D_m)$的 90%.

若 $CCR<1$，计算开销占主导地位；若 $CCR>1$ 且越大时，说明通信开销占据比例越大，网络异构性越明显. η 越大，说明任务的计算开销在不同处理器上的差异越大，计算异构性越明显. 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 上的计算开销可以通过公式产生而得，即

$$w_i(1-\eta/4)\leqslant w_{i,k}\leqslant w_i(1+\eta/4). \quad (17)$$

通过生成不同特征的大量随机 DAG，并模拟 15 个异构多处理器组成的异构网络计算系统的运行，处理器之间异构网络的拓扑结构按处理器链路个数 μ 生成.

5.2 实验结果及分析

实验 1. 分别考虑计算异构性 η 和网络异构性 CCR 对 SLR 的影响. 对比算法为 EFCS[18] 和 LS(dls)[17]，EFCS 为消息调度时的最早通信完成查找算法；LS(dls)为通信竞争条件下基于同构计算的 DLS 算法. LS(dls)算法的网络结构为星型拓扑，2 个处理器之间只存在 1 条通信路径，EFCS 算法基于任意处理机网络(arbitrary processor network, APN)，任意处理器之间存在 $p\times(p-1)/2$ 条通信路径. 上述 2 种算法代表了当前通信竞争的 DAG 调度主要采用的网络拓扑结构. 由于当前只有基于通信竞争的单 DAG 调度，因此，本次实验 F_MDDTMS 算法是在单 DAG 调度情况下产生的结果，平均 SLR 随计算 η 和 CCR 分别如图 5(a)和 5(b)所示. 当 η 变大，说明 DAG 的计算异构性越明显，F_MDDTMS 相比其他算法在向上排序值的计算上更加考虑计算的异构特性，其平均 SLR 相对更低，实验结果验证了 F_MDDTMS 算法的优越性. 当 CCR 越大时，DAG 的网络异构性越明显，F_MDDTMS 算法优于其他算法达 15%以上，表明 F_MDDTMS 算法在精化向上排序值的通信开销上优势明显，而且在计算异构性和网络异构性加强的情况下，算法更加优越.

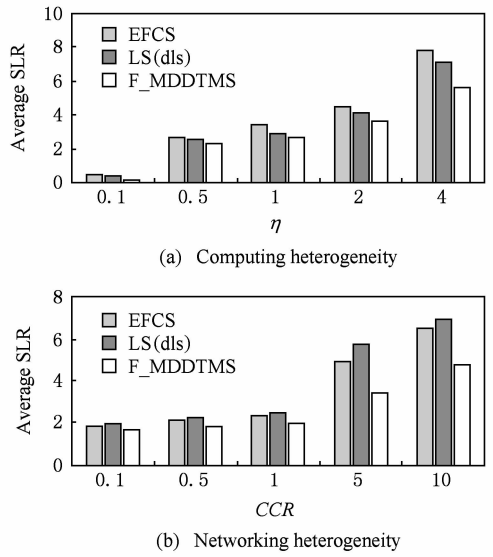


Fig. 5 Average SLR for varying heterogeneity in DAG task and message scheduling.

图 5 DAG 任务与消息调度中平均调度长度比异构性变化

实验 2. 针对多个 DAG 样本，考虑本文提出的 3 种策略(公平策略、关键级策略与混合关键级策略)的多 DAG 动态调度的情况比较. 实验如图 6 所示，整体来看，F_MDDTMS 算法的性能最高，C_MDDTMS 算法最差. 如果 F_MDDTMS 算法能够满足实时性，那么 MC_MDDTMS 算法与 F_MDDTMS 算法的调度结果是一致的. 但结果显

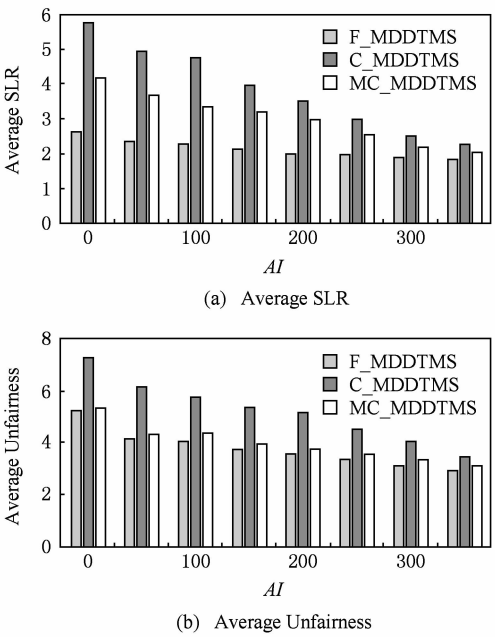


Fig. 6 Performance for varying arrival intervals in multiple DAGs dynamic scheduling.

图 6 多 DAG 动态调度中 3 种策略的性能指标随到达时距变化

示 MC_MDDTMS 算法与 F_MDDTMS 算法并不相同,说明使用 F_MDDTMS 算法时部分 DAG 不能满足实时性;但随着到达间距越大,MC_MDDTMS 算法和 F_MDDTMS 算法的性能值越接近. 由图 6(b)可知,MC_MDDTMS 算法和 F_MDDTMS 算法在不公平性上比较接近,并且相比 C_MDDTMS 算法的不公平性优势很明显. 可见 MC_MDDTMS 算法较好地兼顾了实时性与性能的平衡.

实验 3. 在汽车电子真实消息集环境下分析 WCRT 随消息数变化的情况. 本节在汽车厂商提供的大规模真实消息集的基础上,采用 CAN 网络的真实消息集[1-2],该实验集得到了学术界和工业界的广泛认可. 该实验集包括 65 个消息,并且被分配到 14 个 ECU 之中,同时也支持 500 Kbps 和 250 Kbps 这 2 种网络带宽. 本文将 CAN 网络的带宽分别设为 500 Kbps 和 250 Kbps,组成 HS-CAN/LS-CAN 异构网络,该异构网络代表了当前此类异构网络的最新真实消息集. 同时,本实验将消息按份数复制,以实现多个 DAG 应用的任务与消息同步调度,查看 3 种策略在真实环境中的实验结果,如图 7 所示. F_MDDTMS 算法的 WCRT 最短,优于其他策略算法 20%以上;F_MDDTMS 算法和 MC_MDDTMS 算法的不公平性也都优于 C_MDDTMS 算法. 在真实环境下,MC_MDDTMS 算法也较好地兼顾了实时性与性能的平衡.

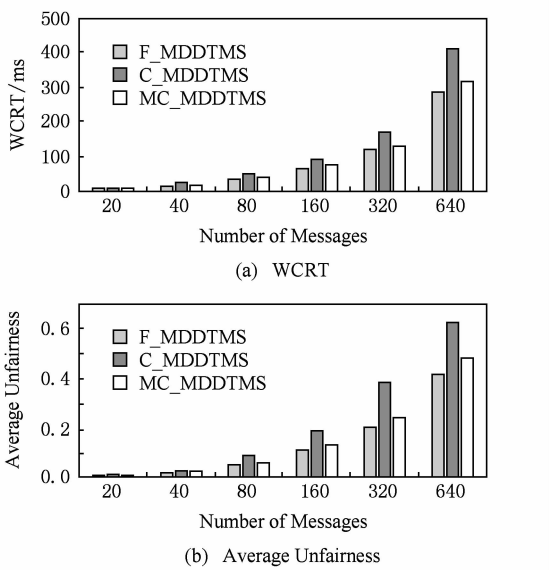


Fig. 7 Performance for varying number of messages in multiple DAGs dynamic scheduling.

图 7 多 DAG 动态调度中 3 种策略的性能指标随消息数变化

以上 3 次实验结果表明,本文所提出的相关多 DAG 动态调度算法在调度长度、最差响应时间和不公平性显示了较好的优越性;在真实消息集环境下,也具有更好的结果,因而能够较好地适应通信竞争的混合关键级汽车电子系统的多 DAG 动态调度需求.

6 结 论

本文面向通信竞争的混合关键级汽车电子系统的调度问题进行研究,通过建立多 DAG 异构计算模型和异构网络模型,分析了现有混合关键级系统和 DAG 调度算法;在通信竞争环境下,实现了向上排序值和最早完成时间的精确化,以适应系统中计算与网络均异构且任务与消息的同步特征. 基于动态模型,提出了公平策略的多 DAG 动态任务与消息调度算法 F_MDDTMS,以降低多 DAG 的调度长度;接着提出了关键级策略的多 DAG 动态任务与消息调度算法 C_MDDTMS,以确保高关键级 DAG 应用的实时性;最后综合 F_MDDTMS 和 C_MDDTMS,提出混合关键级策略的多 DAG 动态任务与消息调度算法 MC_MDDTMS,既确保混合关键级系统中高关键级 DAG 应用的实时性,又使得低关键级 DAG 应用得到积极的处理. 最后利用仿真实验与相关算法进行比较,得出所提出的算法在动态环境下、在满足高关键级应用实时性的基础上,能够产生较好的调度性能,能够满足通信竞争的混合关键级汽车电子系统动态调度的强实时和高性能需求.

参 考 文 献

[1] Xie Guoqi, Li Renfa, Xiao Xiongren, et al. A high-performance DAG task scheduling algorithm for heterogeneous networked embedded systems [C] //Proc of the 28th IEEE Int Conf on Advanced Information Networking and Applications. Piscataway, NJ: IEEE, 2014: 1011-1016

[2] Xie Guoqi, Li Renfa, Yang Fan, et al. Multiple DAG off-line task scheduling for heterogeneous networked automobile electronic systems [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)

[3] Di N M. Design and development of component-based embedded systems for automotive applications [C] //Proc of the 13th Ada-Europe Int Conf on Reliable Software Technologies. Berlin: Springer, 2008: 15-29

[4] Vestal S. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance [C] //Proc of the 28th IEEE Real-Time Systems Symp. Piscataway, NJ: IEEE, 2007: 239-243

[5] 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

[6] 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 the 5th IEEE Int Conf on Self-Adaptive and Self-Organizing Systems. Piscataway, NJ: IEEE, 2011: 9-88

[7] 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

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

[9] 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

[10] Khan M A. Scheduling for heterogeneous systems using constrained critical paths [J]. Parallel Computing, 2012, 38(4): 175-193

[11] Ilavarasan E, Thambidurai P. Low complexity performance effective task scheduling algorithm for heterogeneous computing environments [J]. Journal of Computer Sciences Distributed Systems, 2007, 3(2): 94-103

[12] Sinnen O, Sousa L A, Sandnes F E. Toward a realistic task scheduling model [J]. IEEE Trans on Parallel and Distributed Systems, 2006, 17(3): 263-275

[13] Zeng H, Di Natale M, Giusto P, et al. Stochastic analysis of CAN-based real-time automotive systems [J]. IEEE Trans on Industrial Informatics, 2009, 5(4): 388-401

[14] Zeng H, Di Natale M, Ghosal A, et al. Schedule optimization of time-triggered systems communicating over the FlexRay static segment [J]. IEEE Trans on Industrial Informatics, 2011, 7(1): 1-17

[15] Sinnen O, Sousa L A. Communication contention in task scheduling [J]. IEEE Trans on Parallel and Distributed Systems, 2005, 16(6): 503-515

[16] Leu K L, Chen Y Y, Wey C L, et al. A Bayesian network reliability modeling for FlexRay systems [J]. World Academy of Science, Engineering and Technology, 2010, 4(5): 42-47

[17] Sinnen O, To A, Kaur M. Contention-aware scheduling with task duplication [J]. Journal of Parallel and Distributed Computing, 2011, 71(1): 77-86

[18] Tang Xiaoyong, Li Kenli, Padua D. Communication contention in APN list scheduling algorithm [J]. Science in China: Information Sciences, 2009, 52(1): 59-69

[19] Zhao Henan, Sakellariou R. Scheduling multiple DAGs onto heterogeneous systems [C] //Proc of the 20th IEEE Int Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2006: 189-194

[20] 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

[21] Baruah S, Li H, Stougie L. Towards the design of certifiable mixed-criticality systems [C] //Proc of the 16th Real-Time and Embedded Technology and Applications Symp. Piscataway, NJ: IEEE, 2010: 13-22

[22] Klobedanz K, Koenig A, Mueller W. A reconfiguration approach for fault-tolerant FlexRay networks [C] //Proc of the Design, Automation & Test in Europe Conference & Exhibition. Piscataway, NJ: IEEE, 2011: 1-6

[23] Xie Yong, Li Renfa, Ruan Huabin, et al. Optimal configuration algorithm for static segment of FlexRay [J]. Journal on Communications, 2012, 33(11): 33-40 (in Chinese)
(谢勇, 李仁发, 阮华斌, 等. 最优的 FlexRay 静态段配置算法[J]. 通信学报, 2012, 33(11): 33-40)

[24] Schmidt K, Schmidt E G. A longest-path problem for evaluating the worst-case packet delay of switched Ethernet [C] //Proc of the 5th IEEE Int Symp on Industrial Embedded Systems. Piscataway, NJ: IEEE, 2010: 205-208

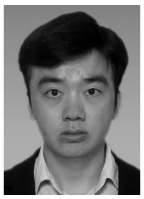
[25] Hong U, Schiffmann W. A meta-algorithm for scheduling multiple DAGs in homogeneous system environments [C] //Proc of the 18th IASTED Int Conf on Parallel and Distributed Computing Systems. Piscataway, NJ: IASTED, 2006: 147-152

[26] 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

[27] 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

[28] 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

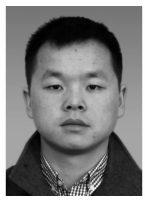
Liu Liangjiao, born in 1984. PhD candidate. Student member of China Computer Federation. His main research interests include embedded computing, parallel computing, and automobile electronic systems.



Xie Guoqi, born in 1983. PhD, postdoctoral researcher. Member of China Computer Federation. His main research interests include embedded computing, parallel computing, and software engineering.

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

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



Xie Yong, born in 1985. PhD, assistant professor. Member of China Computer Federation. His main research interests include automotive embedded systems, and cyber-physical systems(CPS).