面向工业智能体互联网的“通信–控制”协同原子化重构机制

陈乐，马彰超，董芃，张荣辉，牛子儒，王健全
（北京科技大学自动化学院，北京 100083）

摘　要：针对工业智能体互联网中智能体协同关系与通信拓扑动态、原子化调整挑战，提出一种支持“通信–控制”协同的意图驱动控制系统架构及原子化重构机制。首先，设计了将智能体控制逻辑与通信拓扑统一封装为可动态演化的形式化协同状态模型。进一步，提出了双缓冲原子化协同状态转移机制。基于该机制，通过后台重构与原子化切换实现了智能体内部行为与外部连接的微秒级、无中断同步演化。特别地，在控制器内构建了支持动态增删的协议驱动与数据缓存池，通过“增量预热、存量延后”策略，保障多协议网络拓扑重构的数据一致性与时序确定性。实验结果表明，所提架构可根据视觉协同意图，在17.4 μs内完成协同状态切换，实现同步误差<4.1 ms的动态协同分拣任务，证明了该架构可有效支撑智能体网络高效协同与按需重构。

关键词：工业智能体互联网；基于意图的网络；原子化逻辑状态转移；通信–控制协同重构
中图分类号：TP2；TN91
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2026053

Communication-control collaborative atomic reconfiguration mechanism for industrial Internet of agents

Chen Le, Ma Zhangchao, Dong Peng, Zhang Ronghui, Niu Ziru, Wang Jianquan
School of Automation and Electrical Engineering, University of Science and Technology Beijing, Beijing 100083, China

Abstract: To address the challenges of dynamic and atomic adjustment of cooperative relationships and communication topologies in the industrial Internet of intelligent agent, an intent-driven architecture supporting “communication control” collaboration was proposed. Firstly, a formalized, dynamically evolvable cooperative state model was designed to unify agent control logic and communication. Further, a double-buffer atomic transition mechanism enabled agents’ internal behaviors and external connections to evolve synchronously in microseconds without interruption. Additionally, a dynamic protocol-driven data cache pool within the controller used an “incremental preheating and stock delay” strategy to guarantee data consistency and timing determinacy during network reconfiguration. Experimental results show the architecture completes cooperative state switching within 17.4 μs and achieves a dynamic sorting task with a synchronization error under 4.1 ms, effectively supporting efficient collaboration and on-demand reconfiguration of agent networks.

Keywords: industrial intelligent agent Internet, intent-based network, atomic logical state transition, communication-control collaborative reconfiguration

收稿日期：2025–12–24；**修回日期：**2026–02–14
通信作者：马彰超，mazhangchao@ustb.edu.cn
基金项目：国家自然科学基金联合基金资助项目（No.U25A20433）；国家自然科学基金资助项目（No.62471069）；山西省重大专项基金资助项目（No.202301020101001）
Foundation Items: The Joint Funds of the National Natural Science Foundation of China (No.U25A20433), The National Natural Science Foundation of China (No.62471069), Shanxi Province Major Projects (No.202301020101001)

0　引言

随着工业 4.0 与智能制造的深入推进，制造业正从刚性的大规模生产向柔性的、高度定制化的生产模式转型[1]。工业多智能体系统（industrial multi-agent system, IMAS）作为实现产线柔性的关键使能技术，通过部署多个自主或半自主的智能体来执行复杂的生产任务，为制造业的数字化、智能化升级提供了新的路径[2-3]。然而，随着 IMAS 在工业场景中的规模化应用，其对控制系统灵活性和协同性的需求日益剧增。智能体需要根据瞬息万变的生产指令（如订单变更、工艺切换）动态调整其协同行为、控制逻辑与网络连接，这为传统的、逻辑固化的工业控制系统带来了严峻的挑战[4]。一方面，现有的工业网络（如 EtherCAT、Profinet、时间敏感网络（time-sensitive networking, TSN））为了保证高实时性和确定性，网络拓扑和通信配置（如时隙、虚拟局域网（virtual local area network, VLAN））通常是静态预配置的[5]。这种静态网络无法支持智能体之间“机会性”和“突发性”的动态协同需求。另一方面，传统的 IMAS 研究[6]侧重于高层决策算法（如合同网、强化学习），它们假设智能体间的通信是“全连接”或“按需即时可用”的，完全忽略了底层网络拓扑重构和控制逻辑重构的物理约束和实时代价[7]。这一鸿沟导致了当前工业系统的“刚性协同”，而且协同模式一经设计便难以更改。

意图驱动网络（intent-based networking, IBN）为 IMAS 所面临的柔性协同难题提供了一种可行的解决方案[8]。通过将上层声明式的“业务意图”与底层的“网络配置”相解耦，IBN 架构能够动态感知系统状态，并将高层意图自动转译为底层的通信策略[9]。将意图驱动机制引入 IMAS，不仅能够实现通信资源的自动化编排，还能将复杂的网络管理逻辑从端侧智能体中剥离至中心控制器，从而大幅降低了底层设备的计算与配置负担，并提升了系统的可扩展性[10]。然而，在实时工业协同场景中，不恰当的意图驱动机制可能导致智能体的控制行为与其通信拓扑之间产生危险的“失配”状态，或引入不可容忍的重构时延。因此，有必要对面向“通信–控制”协同的意图执行机制展开深入研究。

目前，已有若干工作围绕意图驱动、多智能体协同以及动态可重构控制系统（dynamic reconfigurable control system, DRCS）展开了研究。在 IBN 领域，文献[11-13]主要将其应用于工业物联网或边缘计算中，通过意图实现网络服务质量（quality of service, QoS）保障或计算资源编排，但其执行对象是网络设备配置，而非深入控制器内部的实时控制逻辑。在 IMAS 领域，文献[14-16]利用合同网、博弈论、强化学习等方法，在高层任务分配、资源调度和路径规划方面取得了显著成果，但这些研究大多假设智能体已具备执行任务所需的全部能力，其核心在于“选择”而非“实时改变”智能体的底层行为与连接。在 DRCS 领域，文献[17-20]以 IEC 61499 标准为基础，实现了控制逻辑的运行时重构，但其重构过程通常是分步执行的，且大多忽略了通信拓扑的同步重构，缺乏原子性保证，且引入的时延和抖动难以确定，无法满足高精度协同任务的实时性与安全性需求。

现有研究主要围绕意图的网络编排、智能体的高层任务决策和控制逻辑的非原子化重构展开，却忽略了工业协同场景对“通信–控制”协同重构的原子性与实时确定性的特殊需求。高保真的意图执行是意图驱动的根基。在工业实时协同控制过程中，如果逻辑与通信的切换无法保证原子性，将导致系统因两者不一致而陷入“半配置”的危险状态。如果切换时延不可控，将导致多智能体时序失配，无法有效提供高精度的协同控制。然而，若仅注重执行的安全性而采用“停机更新”策略，则完全违背了柔性制造对高效率和连续运行的根本要求。

针对这一挑战，本文结合意图驱动与多智能体协同作业的实际特性，提出了一种支持“通信–控制”协同原子化重构的意图驱动协同控制架构，以实现高层意图向底层实时控制执行的安全、原子化映射。基于该架构，针对工业协同控制的实时性与原子性限制，建立了“协同状态”形式化模型，将智能体的内部控制逻辑与外部通信拓扑统一定义为可动态演化的整体，并建立了协同状态在转移过程中必须满足状态切换的原子化、有界时延转移以及控制过程连续性的核心设计目标。针对该目标，本文设计了一种基于双缓冲运行时内核的原子化协同状态转移（atomic collaborative-state transition, ACST）机制，并引入基于状态感知的意图驱动机制，以适

应动态的工业协同场景。物理实验结果表明，所提架构和机制在响应传感器触发的协同意图时，能够以微秒级的切换时延（<17.4 μs）原子化地重构其控制逻辑与通信连接，并实现高精度的动态协同任务（同步误差<4.1 ms），表现出卓越的实时性与高保真度。本文的主要工作如下。

1) 提出了一种支持“通信–控制”协同原子化重构的协同控制架构。基于该架构建立了“协同状态”形式化模型，将高层意图转译、协同决策与底层协同状态演化相结合，提出了实现该状态原子化、有界时延转移的优化目标。

2) 为高效实现所提目标，提出了一种基于双缓冲运行时内核的ACST机制，通过后台重构与原子化指针交换，在理论上保证了协同状态转移的原子性、实时性和连续性，以适应高确定性的工业协同场景。

3) 物理实验结果表明，所提架构和机制能够在保障物理控制流无中断的前提下，以微秒级的切换时延原子化地重构智能体间的协同关系，实现高精度的协同控制。

1 相关研究工作

本文工作处于意图驱动网络、工业多智能体系统与动态可重构控制系统3个领域的交叉点。本节将分别从这3个领域的核心理念、最新进展及其在工业应用中的局限性展开详细综述。

1.1 意图驱动网络

意图驱动网络旨在通过高级策略抽象来简化复杂系统的管理与自动化。在数据中心、软件定义网络和网络功能虚拟化等信息技术（information technology, IT）领域，IBN已展示出巨大成功[21-22]，它允许网络管理员声明“What”（期望结果）而非“How”（具体配置），显著提升了运维效率。近年来，随着工业互联网和5G技术的发展，意图驱动的理念逐步被引入工业协同场景。例如，有研究将其应用于工业物联网中，实现端到端的差异化QoS保障[23]；或用于工业边缘计算中，实现计算与网络资源的动态编排[24]。这些研究解决了操作技术（operational technology, OT）网络层的灵活性问题，如为关键控制流动态分配高优先级5G切片或TSN时隙。

然而，这些研究的共同点在于，它们主要集中在IT与OT融合的“网络与服务”层面，其意图驱动的执行对象是路由器、交换机或边缘服务器的配置。在这些工作中，底层的工业控制器，如可编程逻辑控制器（programmable logic controller, PLC）通常被视为黑盒式的“数据端点”，意图并未触及控制器内部的“控制逻辑”本身。更关键的是，现有工作通常独立关注网络资源或计算资源的优化，而未能将网络拓扑的动态调整与智能体内部控制逻辑的动态重构进行协同考虑。如何将抽象意图转译为PLC内部的功能块（function block, FB）组合与数据流连接，并在保证硬实时确定性的前提下实现“通信–控制”的原子化更新，是当前意图驱动范式深入OT控制领域所面临的重大技术挑战。

1.2 工业多智能体系统

在IMAS领域，研究重点在于实现多个自主或半自主智能体之间的协同与决策[25]。这是一个成熟且广泛的研究领域，为实现复杂的协同行为，学术界已提出多种协调机制，如经典的合同网协议[26]、黑板模型[27]，以及近年来基于多智能体强化学习的优化算法[28]。这些机制在解决“任务分配”“资源调度”和“路径规划”等高层决策问题上表现出色，能够有效提升系统在物流、仓储和装配等场景的整体效率与鲁棒性。然而，IMAS领域的研究大多基于一个隐性假设：即智能体已经具备执行所有可能任务所需的能力（即控制逻辑），且智能体之间的通信链路是预先存在且随时可用的[29-30]。该领域的研究核心在于选择和协调这些既有能力，而非实时生成或改变这些能力本身[31]，也未解决智能体如何动态建立或断开与其他智能体的通信连接以适应新的协同关系的问题。例如，在协同搬运任务中，现有IMAS研究侧重于决策两个智能体A和B去执行任务，但并未解决智能体A和B如何从“独立运行”逻辑动态切换到“同步控制”逻辑，以及如何为这种同步控制动态建立专用的通信链路的问题。这导致在需要深度逻辑耦合的工业协同场景下，现有IMAS方法存在理论与实践的脱节。

1.3 动态可重构控制系统

在DRCS领域，其研究目标是解决控制逻辑的动态变更问题。其中，IEC 61499标准是该领域公认的基石，它提供了一种事件驱动、面向分布式的“功能块”模型，允许在运行时创建、删除和重定向FB实例。基于此，大量研究探讨了基于代理[32]

或服务[33]的重构管理。然而，现有动态重构方法存在两个根本性局限：一是缺乏原子性保证，IEC 61499标准的重构过程（如删除一个FB，再创建一个新的FB）是分步执行的，这使系统在重构过程中可能处于一个短暂的、未定义的“半配置”状态，这在安全关键的工业应用中是不可接受的；二是缺乏实时确定性，重构过程的时延和抖动通常是不可控或无上界的，这使它们无法满足多个智能体之间必须在同一控制周期内（或极小时间窗内）同时完成逻辑切换的高精度协同需求[34]。此外，这些动态重构研究的关注点仅限于单个控制器内部的控制逻辑重构，少有工作深入探讨如何在保证原子性的前提下，同步重构多个智能体之间的通信拓扑与协同关系，使它们难以直接应用于需要动态网络配置的智能体互联网场景。

针对上述现有问题，IBN 缺乏对 OT 控制器实时执行机制和动态通信重构能力，IMAS 缺乏改变智能体底层行为与动态建立通信连接的运行时重构能力，DRCS 缺乏满足协同控制所需的原子性与实时确定性以及“通信-控制”协同保证。这 3 个领域各自的局限性，共同指向了一个未被解决的核心问题，即如何将高层抽象意图安全、原子化地转译为多个工业智能体底层的、实时的、协同的行为逻辑与网络连接关系。在此基础上，本文提出了以“协同状态”模型为形式化依据，以意图驱动和协同决策为高层指引，以ACST机制为底层执行保障的新架构，旨在打通从抽象意图到原子化、实时化、动态网络协同化控制状态演化的完整技术链条。

2　意图驱动的通信-控制协同重构架构

2.1　架构总体设计

随着工业智能体系统在复杂制造场景中的广泛部署，控制系统的柔性化、分布化与网络化趋势日益显著。在传统控制体系中，控制逻辑与通信拓扑通常被分离设计与独立配置，导致系统在任务切换或拓扑变化时无法实现“通信-控制”的同步重构，进而影响系统的实时性与稳定性。为此，本文提出了一种意图驱动的“通信-控制”协同重构架构，通过在控制与通信之间建立动态协同关系，实现面向意图的自适应系统重构与高确定性响应。意图驱动的“通信-控制”协同重构架构旨在实现从高层生产意图到底层实时控制执行的安全和原子化映射，通过对“通信拓扑”和“控制逻辑”的统一抽象与协同调度，解决工业多智能体系统中通信和控制解耦与重构不同步的问题。其总体目标是在保证物理过程连续性和时序确定性的前提下，使系统能够根据业务意图动态调整智能体间的协同关系、执行逻辑与通信路径，实现柔性制造场景下的即时协同与自适应控制。

该架构以“意图”为系统运行的驱动核心，将上层生产意图抽象为形式化的协同指令，并通过层次化的状态建模和运行时重构机制，在底层实现控制逻辑与通信拓扑的协同演化。整体设计遵循“感知-决策-执行”闭环原则，自上而下分为意图决策层、控制执行层和感知交互层 3 个层级。不同于传统的“控制逻辑独立演化”模式，该架构提出了“通信-控制”协同重构的形式，将控制逻辑、数据流、通信链路等要素统一纳入一个可演化的“协同状态”中进行管理与转移。通过意图驱动的协同重构指令解析与原子化状态切换机制，系统能够在微秒级时延内完成控制与通信的同步切换，从而实现跨控制器和跨网络的高精度协同。

本文提出的意图驱动的“通信-控制”协同重构架构如图 1 所示，该架构自上而下可划分为 3 个层级：意图决策层、控制执行层和感知交互层。

意图决策层承担高层业务意图的解析与协同策略的生成，其核心功能是根据制造任务、资源约束和环境状态，生成“协同意图模型”。该层相当于控制系统的“认知中枢”，通过形式化的意图描述语言，将抽象的业务目标转译为一组可执行的协同约束（如目标状态、同步精度、协作拓扑等），并形成结构化的“协同重构指令”。

控制执行层是架构的核心运行层，其对应各智能体的运行时控制内核，内部实现了面向多任务并行与实时重构的运行时控制内核。其关键思想是将控制逻辑与通信拓扑统一纳入“协同状态模型”进行管理，并通过原子化状态切换机制在不中断执行的情况下完成新旧任务之间的无缝切换。控制执行层通过双缓冲结构将执行与重构逻辑分离，使系统能够在维持实时性的前提下完成自演化。

感知交互层则构成整个系统的外部接口，包括传感器、执行器和工业网络接入模块。该层的主要职责是实时采集物理世界状态信息并反馈至意图决策层，同时根据控制执行层下发的指令驱动物理过

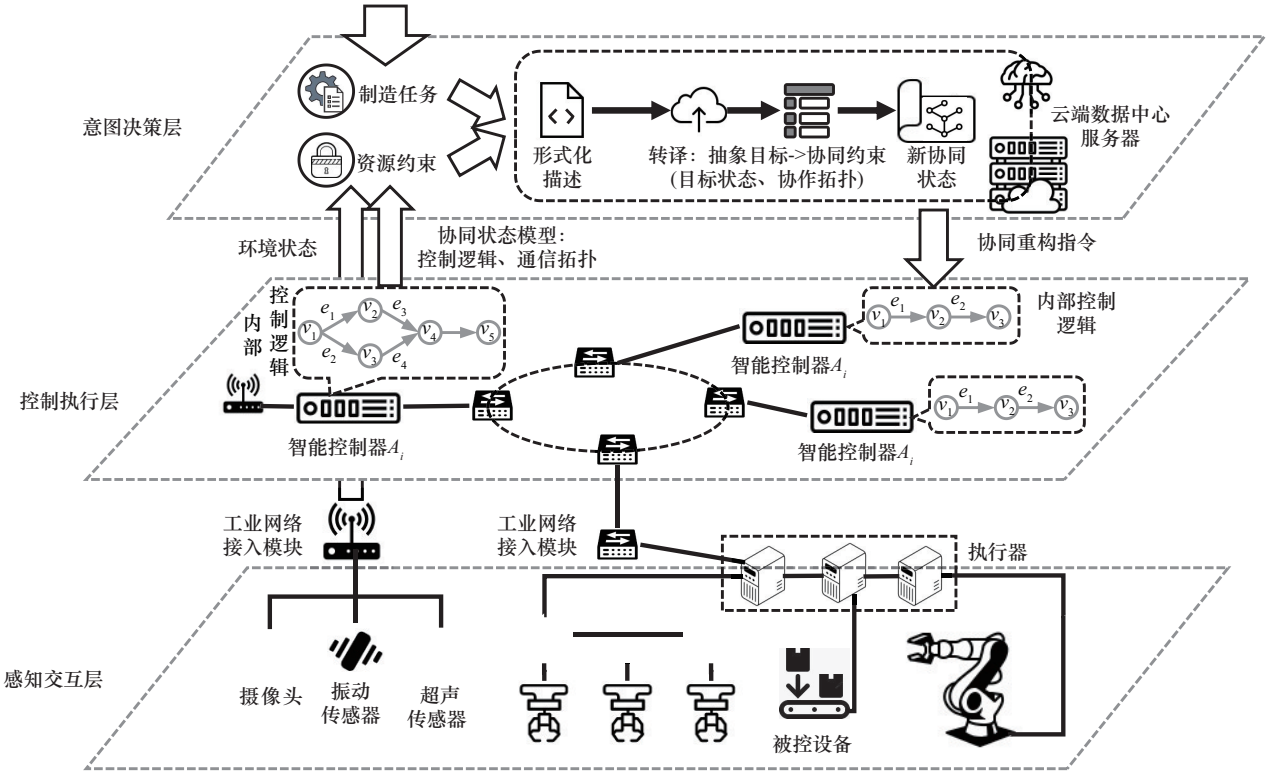

图 1　意图驱动的“通信–控制”协同重构架构

程运行，其信息流的时效性直接决定了系统的“感知–决策–执行”闭环速度。

该架构实现了从“认知意图”到“执行行为”的纵向映射，同时通过协同状态接口实现横向信息一致。系统不再将控制与通信视为两个独立的域，而是通过统一状态模型在逻辑层上将它们绑定为一个动态协同整体。高层意图的变化直接对应系统协同状态的转移，控制逻辑与通信拓扑在同一时间窗口内完成一致性更新，从而消除了控制重构与通信重配置之间的时间错位问题。这种设计思路本质上是一种“结构可变、时序可控、状态可验证”的自适应控制框架，它既继承了传统控制系统的确定性和周期性，又具备了网络化智能系统的灵活性与自治性，为复杂制造系统提供了一种统一的体系化解决思路。

2.2　架构核心组件与功能定义

意图驱动的“通信–控制”协同重构架构通过多模块协同运行实现从高层意图到底层执行的闭环映射，如图 2 所示，其核心由意图管理模块、协同状态建模器、运行时重构内核、通信编排引擎等组成。各模块既独立分工，又通过统一的协同状态接口保持数据与行为一致性。

意图管理模块是系统的决策与认知中心，其功能在于理解、抽象并规划任务目标。它通过对生产任务、工艺指令和现场状态的语义解析，提取系统需要达到的期望状态与约束条件。意图在此不再是静态配置文件，而是可被动态触发和演化的高层逻辑。意图管理模块通过内置基于规则的预定义策略选择，将意图转化为一组可执行的协同计划，其中包括控制逻辑更新的目标函数、通信拓扑的约束边界以及任务调度的时间要求等。随后，系统会对生成计划进行约束校验，验证其在当前资源与网络条件下是否满足实时性与安全性约束。这一过程相当于在“通信–控制”协同空间中寻找一条从现有状态到目标状态的可达路径，确保系统的演化在逻辑上正确、时间上可控。

协同状态建模器则是系统的结构化支撑模块，用于统一表示控制逻辑与通信拓扑的动态映射关系。它将系统抽象为一个有向图结构，其中每个节点对应一个逻辑功能单元，边则表示信号依赖、任务触发或数据传递关系。通过这种建模方式，控制逻辑与通信拓扑不再被分别管理，而是在同一数据结构中被描述为“协同状态”。模型中既包含控制执行序列和任务依赖关系，也包含网络链路、交互

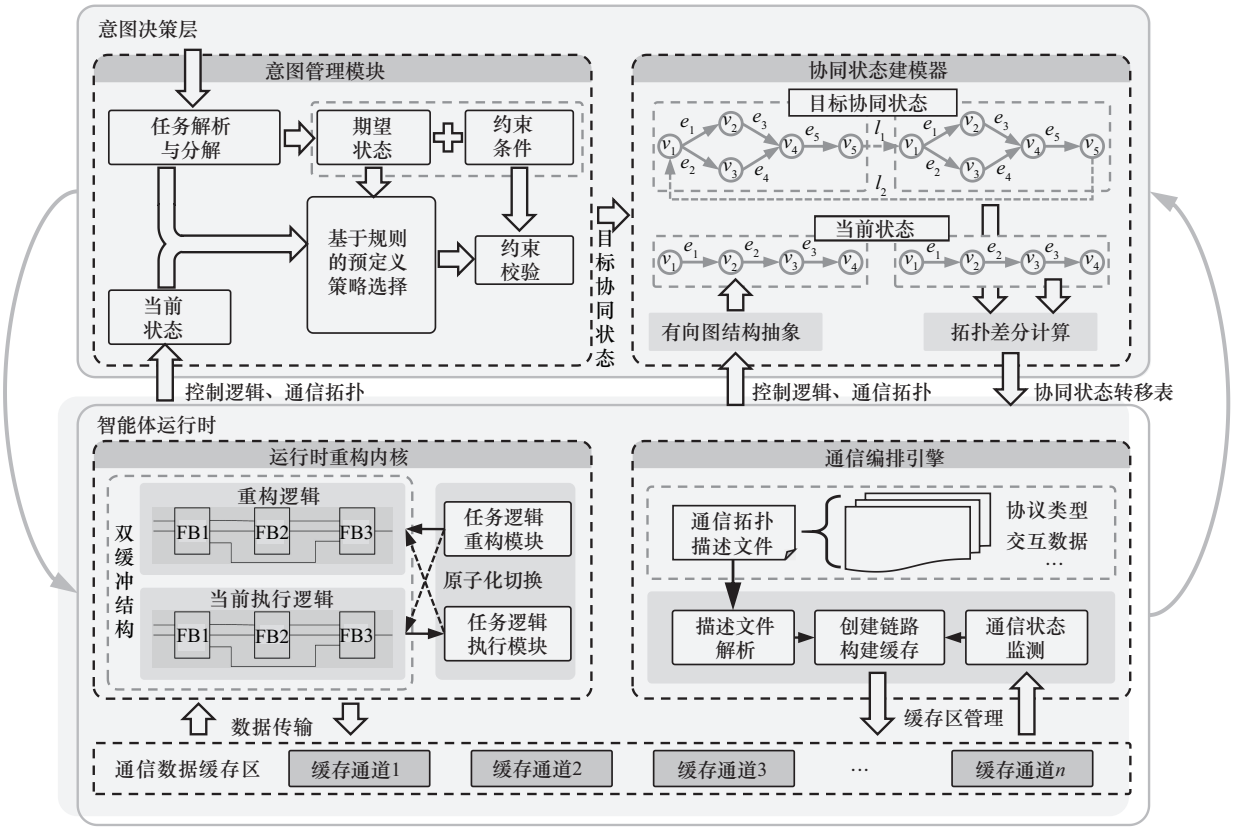

图2　意图驱动的“通信–控制”协同重构架构关键模块

数据与同步周期等信息。每当新的意图被解析后，协同状态建模器会根据拓扑差分计算结果生成协同状态转移表，以确保系统在拓扑和时序上的完整性。该模型为运行时的原子化切换提供了语义支撑，使通信与控制的协同演化具有形式化基础。

运行时重构内核是系统的执行核心，负责将协同状态的更新在时间域上具体落实。它采用双缓冲结构，在保证当前控制任务连续运行的同时，在后台完成新的控制逻辑和通信拓扑的加载与验证。当后台重构完成且达到同步条件时，系统通过原子指针交换操作完成前后台切换，实现瞬时状态转移而不影响执行周期。内核中嵌入多线程调度与周期监测机制，确保原子化切换操作在较小时间窗口内完成，满足工业控制系统对实时性的要求。同时，内核具备状态回滚与冗余保护机制，当检测到切换异常或通信不同步时，可立即恢复至先前稳定状态，从而保证系统安全可控。

通信编排引擎负责实现网络层的同步演化与链路重构。其核心思想是将通信通道视为可配置的逻辑资源，采用控制面与数据面分离的机制实现网络配置的可编程化。通信编排引擎根据协同状态模型中的拓扑描述自动生成对应的通信链路与通信数据缓存方案，并在系统切换时与运行时内核保持严格同步，确保数据流在拓扑更新期间无丢失和无时延累积。此外，该模块持续监测链路时延、抖动等指标，并通过反馈通道将性能数据回传至意图管理模块，可用于后续的意图调整与系统优化。通过该机制，通信网络的行为被纳入控制体系的统一调度中，真正实现了控制与通信的协同一体化管理。

这 4 个核心组件通过统一的协同状态接口互联，在架构层面形成了自上而下的决策链与自下而上的反馈链。意图在顶层被解析与规划，经由协同状态建模器转化为可执行结构，再通过运行时内核与通信编排引擎共同完成实际的逻辑和网络重构。整个过程保持信息流、时间流与控制流的高度一致，使系统在面对动态工况时能够实现快速、稳定且可验证的自适应重构。

2.3　意图驱动的闭环工作流

意图驱动的“通信–控制”协同重构架构并非静态结构，而是一个随任务与环境动态演化的闭环系统。系统的运行体现为“感知触发–意图驱动–闭环执行”的过程。需要说明的是，虽然物理层的数

据变化（状态感知）往往是系统演化的触发点，但系统的核心驱动力始终是高层意图。感知交互层的作用在于发现当前状态与预设意图之间的偏差或识别新的业务场景，从而激活意图管理模块生成新的协同策略。系统并非被动响应传感器数据，而是主动追求与高层意图的一致性，其意图驱动的闭环工作流如图3所示。

系统运行首先从状态感知开始。感知交互层以高采样率采集过程数据、网络运行状态和设备资源信息，并通过边缘节点进行时间戳标定与预处理，保证感知数据的同步性与可靠性。这些状态向量被封装为统一格式并传输至意图决策层，使系统具备对当前运行环境的全局认知。当外部任务变化、设备状态异常或性能偏离目标时，系统触发意图生成阶段。意图管理模块根据当前状态与目标任务，生成新协同意图并确定其优先级。

在协同规划阶段，协同状态建模器对当前协同状态与目标协同状态进行差分计算，识别需要调整的逻辑单元和通信连接关系，并计算出一条可行的状态迁移路径。状态迁移结果以协同状态转移表的形式下发至控制执行层，为运行时重构内核提供结构依据。运行时重构内核随后在后台缓冲区加载新的控制逻辑与通信拓扑，并调用通信编排引擎生成对应的通信链路与通信数据缓存方案。后台构建过程与前台执行相互独立，系统的实时任务不受影响，保证了连续运行的确定性。

当后台准备完成并到达预设切换点时，系统进入原子化重构阶段。此时运行时重构内核执行一次原子指针交换操作，在单个控制周期内完成前后台的任务替换。与此同时，通信编排引擎同步更新数据分发关系与缓冲区映射，实现控制逻辑与通信拓扑的同时迁移。整个切换过程持续时间远短于系统控制周期，因此对可观测的输出扰动较小。切换后系统进入新的协同状态运行阶段，感知交互层开始采集新状态下的执行数据与通信性能指标。意图决策层对比目标意图与实际反馈，若偏差超出阈值，则触发局部回滚或参数修正机制，实现系统级的自校正。通过这一系列循环，系统形成了一个能够持续自感知、自调整和自重构的动态演化体系。

这种闭环工作流的最大特征在于意图贯穿始终。系统的每一次状态更新都以意图为驱动力，以协同状态模型为支撑，以运行时内核为执行核心，以通信编排引擎为保障环节，从而在整个生命周期内实现控制逻辑与通信拓扑的统一演化。该机制保证了工业多智能体系统在复杂网络环境下的高可靠性运行，也为未来的意图驱动工业网络提供了可验证的工程化实现路径。

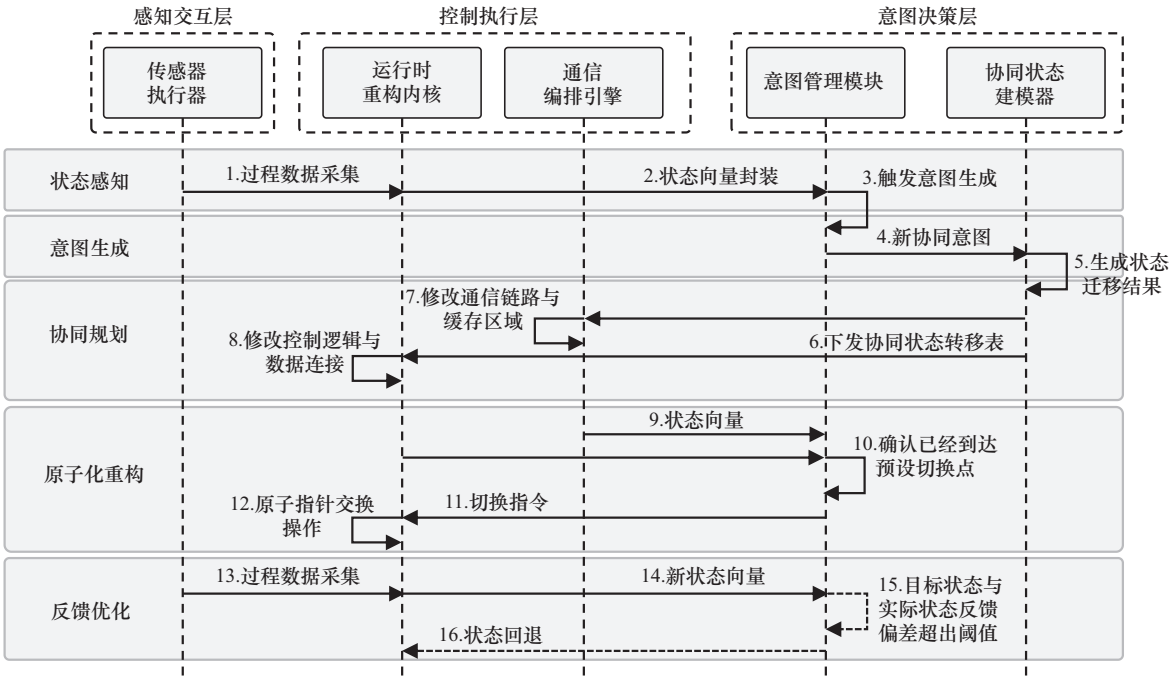

图3　意图驱动的闭环工作流

3　“协同状态”的形式化建模与决策

在前述意图驱动“通信–控制”协同重构架构的基础上，为实现该连接的严谨性与可靠性，本节为该架构的闭环工作流建立了一个贯穿全局的形式化模型。但缺乏一个严谨的数学模型，“意图”将停留在一个模糊的业务概念层面，无法为工业OT领域所要求的安全性、实时性和确定性提供任何保障。本节所建立的形式化模型，旨在从理论层面解决以下两个根本问题。

1) 意图管理器的协同决策过程，即如何将一个抽象的、声明式的意图，转译为一组精确、无歧义的协同策略。

2) 工业智能体的策略执行标的，即智能体的控制逻辑与通信拓扑应如何被严谨地定义与描述，使其本身成为可被机器计算、分析和重构的对象。

3.1　“协同状态”模型定义

在本文架构中，一个工业智能体（如PLC）被定义为一个可动态演化的“白盒”实体，其内部行为和外部连接均可根据高层策略进行重构，而非传统PLC的静态“黑盒”执行器。这种设计理念要求本文必须能够精确描述智能体在任意时刻的完整功能状态。为此，本文将其定义为“协同状态”，记作 S_{collab}。这是一个通信与控制协同单元，由内部的“逻辑状态” S_{logic} 和外部的“通信状态” S_{comm} 共同表征。

$$S_{\text{collab}} = \left\{ S_{\text{logic}}, S_{\text{comm}} \right\} \tag{1}$$

其中，S_{logic} 表示系统的逻辑状态集，S_{comm} 表示系统的通信状态集。前者描述控制系统的内部逻辑关系与任务执行流程，后者反映系统间的数据交换与时序依赖关系。

逻辑状态模型。逻辑状态 S_{logic} 表征智能体内部的控制程序结构与数据处理流，其基本构成元素如下。

1) FB实例 v。控制逻辑的基础单元被定义为FB。设工业智能体 A_i 的能力库为其可用的FB类型集合 $C_{\text{fb}} = \{ T_{\text{fb1}}, T_{\text{fb2}}, \cdots \}$。一个FB实例 v 可被定义为

$$v = (\text{id}, T_{\text{fb}}, P_{\text{in}}, P_{\text{out}}, M) \tag{2}$$

其中，id为实例的唯一标识符，$T_{\text{fb}} \in C_{\text{fb}}$ 为其类型，P_{in} 和 P_{out} 分别为输入和输出端口的集合，M 为其内部状态机或内存变量。

2) 逻辑状态图 G_{logic}。智能体 A_i 的当前逻辑状态 $S_{\text{logic\_}i}$ 定义为一个有向无环图（directed acyclic graph, DAG），可表示为

$$S_{\text{logic\_}i} = G_{\text{logic}}(V_i, E_i) \tag{3}$$

其中，$V_i = \{ v_1, v_2, \cdots, v_n \}$ 是当前在 A_i 上实例化的FB实例集合。E_i 是有向边的集合，表征FB实例间的执行顺序以及内部数据流连接。$E_i \subseteq V_i \times V_i$，每条边 $e = (v_a, v_b)$ 表示 v_a 的一个输出端口 $P_{\text{out}} \in v_a$ 连接至 v_b 的一个输入端口 $P_{\text{in}} \in v_b$。

图 G_{logic} 如图4所示，精确地描述了智能体在一个控制周期内的计算逻辑和执行顺序。选用DAG模型是因为它天然符合工业PLC“输入–执行–输出”的周期性扫描和数据流计算模式。该定义将传统固

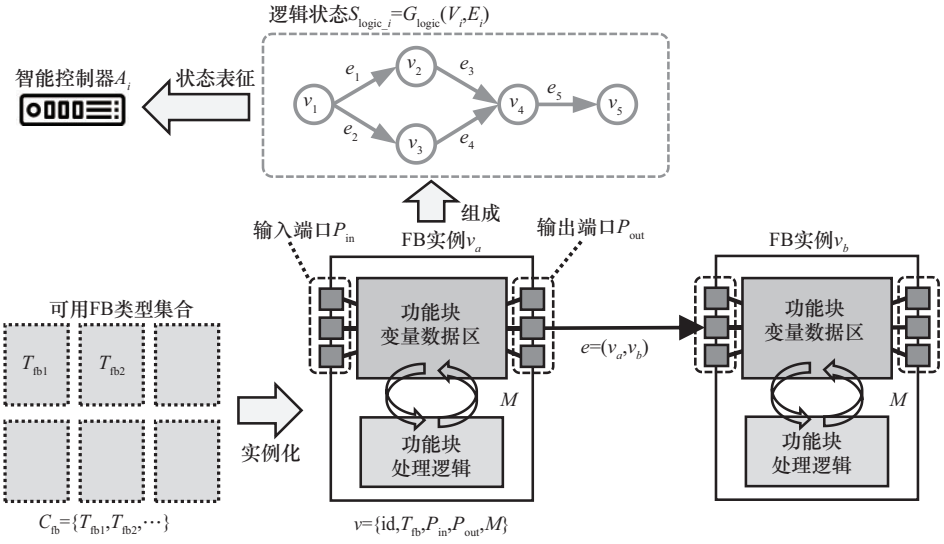

图4　逻辑状态图模型

化在设备内存中、由人力编写的静态“程序代码”转变为一个可被数学描述、分析、验证和动态操作的“状态”对象。这种转变的意义在于，控制逻辑本身成为可计算和可编排的资源，是实现自动化意图转译的必要步骤。

通信状态模型。通信状态S_{comm}表征智能体之间的网络连接关系。在工业智能体互联网系统中，智能体间的协同关系是动态建立的，因此其通信拓扑也必须是动态演化的，而非传统静态现场总线。

1) 智能体集合A。设整个工业智能体互联网系统由N个智能体组成，$A=\{A_1,A_2,\cdots,A_N\}$。

2) 通信链路l。一个动态的、按需建立的逻辑通信链路l被定义为一个多元组，表示为

$$l=(A_s,A_d,T_{\text{proto}},\omega) \tag{4}$$

其中，$A_s\in A$为源智能体，$A_d\in A$为目标智能体或一个智能体组。T_{proto}为通信协议类型，如$T_{\text{proto}}\in\{\text{OPC UA, Modbus TCP, EtherCAT}\}$，协议的选择直接影响通信的性能和模式。$\omega$为该通信链路所需的吞吐配额/缓冲区容量，表示该链路在逻辑层分配的数据缓冲区大小。这直接决定了系统在处理突发数据时的抗阻塞能力。

3) 通信状态图G_{comm}模型如图5所示。整个系统的全局通信状态S_{comm}被形式化地定义为一个有向图，表示为

$$S_{\text{comm}}=G_{\text{comm}}(A,L) \tag{5}$$

其中，A为节点集，$L=\{l_1,l_2,\cdots,l_m\}$为当前所有已激活的逻辑通信链路的集合。将S_{comm}建模为图G_{comm}，意味着系统的网络拓扑不再是固定的物理布线，而是由意图管理器动态规划和维护的一个逻辑变量。

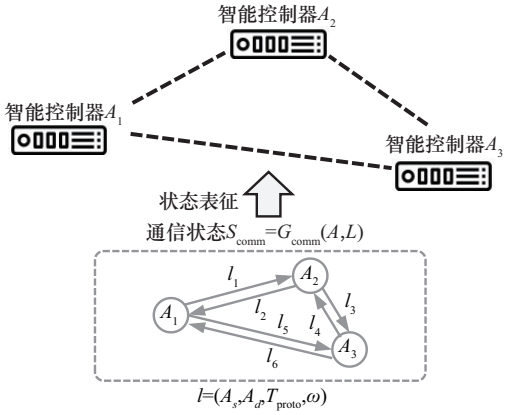

图5　通信状态图模型

协同状态模型。基于上述定义，智能体A_i的协同状态S_{collab_i}被定义为其内部逻辑状态和其参与的外部通信状态的协同单元，表示为

$$S_{\text{collab}\_i}=(S_{\text{logic}\_i},S_{\text{comm}\_i}) \tag{6}$$

其中，$S_{\text{comm}_i}\subseteq S_{\text{comm}}$是$A_i$作为通信端点的所有链路$l$的集合，$S_{\text{logic}_i}$定义了智能体“如何计算”，$S_{\text{comm}_i}$定义了其计算所需的“数据来源”和“结果去向”。在协同任务中，这两者必须严格匹配，共同构成智能体在某一时刻的完整功能视图。

相应地，全局系统状态S_{sys}被定义为所有智能体的协同状态的集合如图6所示，表示为

$$S_{\text{sys}}=\{S_{\text{collab}\_i}|A_i\in A\} \tag{7}$$

值得注意的是，当多个智能体（如A_i,A_j）通过通信链路（$l_i,l_j\in L$）进行强耦合协同（如S_{logic_i}依赖S_{logic_j}的输出，反之亦然）时，系统的全局协同状态S_{sys}在拓扑结构上将形成有向有环图。

这种“代数环”在瞬时计算中通常是禁止的。然而，在本文架构中，该环路通过离散控制周期的时序延迟被自然解耦。

由于S_{logic}的执行是周期性的（基于控制周期T_{cycle}），智能体A_i在k时刻读取的来自A_j的数据，实际上是A_j在$k-1$或更早时刻写入通信缓存区的数据。

本文所指的“统一模型”并非意味着控制逻辑与通信设施在物理实现上的消亡或融合（二者在物理上仍分别运行于CPU与网卡/交换机），而是指在逻辑管理维度的统一。通过式(1)，本文将传统上割裂配置的两个域在数学描述上通过元组耦合。这种建模方法强制规定，一个合法的系统状态必须同时包含匹配的逻辑与通信子状态，从而在理论层面否定了独立修改其中一者而保留另一者的合法性，为实现原子化切换提供了形式化基础。

这种机制将空间上的“闭环反馈”转化为时间轴上的“开环递推”，从而在保证局部逻辑S_{logic}为DAG满足单周期可计算性的同时，实现全局的闭环协同控制。这一机制揭示了通信数据缓存区在离散协同系统中的关键作用，即作为时序隔离界面，将空间上的代数环解耦为时间上的因果链。这为4.2节通信状态重构阶段关于通信数据缓存区预分配与管理策略的设计提供了坚实的理论依据。

3.2　意图管理器的决策模型

意图管理器的核心功能是执行“协同决策”，

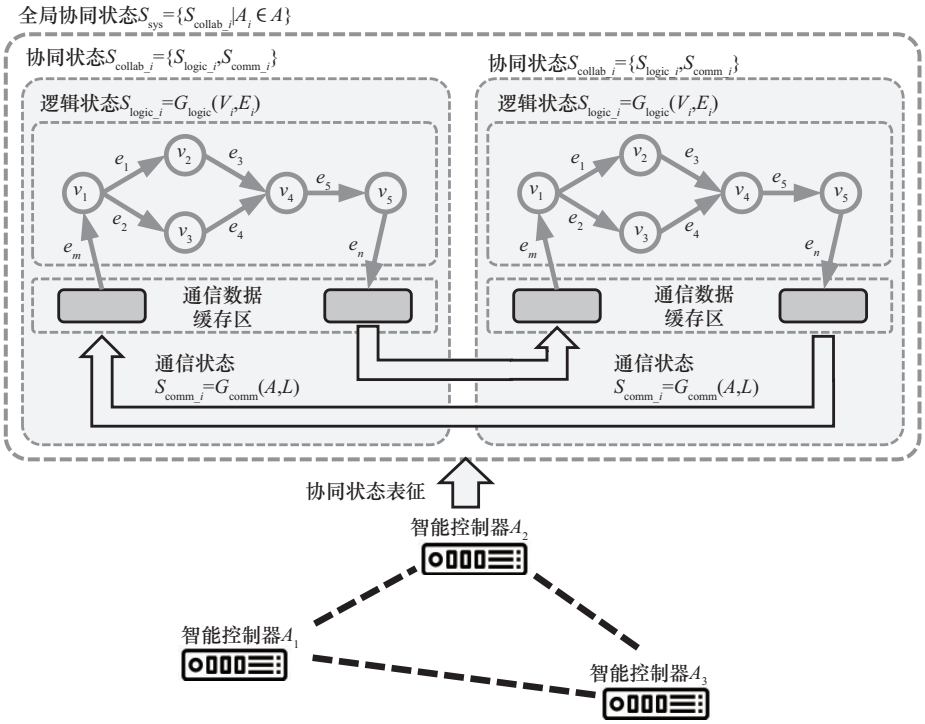

图6　协同状态图模型

即根据高层意图I和当前系统状态S_{sys}，计算出一个最优的目标协同状态S'_{collab}。该决策过程Π本质上是一个复杂的约束优化问题，它需要在一个庞大的、由逻辑和通信组合而成的状态空间中搜索最优解。

设S_{sys}为当前系统状态，I为新到达的意图。决策过程Π的目标是求解S'_{sys}（或等价地求解该状态转移所需的重构操作集R）。

$$\Pi(I, S_{sys}) \rightarrow S'_{sys} \tag{8}$$

该求解过程旨在最大化一个系统效用函数$U(S'_{sys})$，同时满足一系列严苛的约束C。

$$\max_{S'_{sys}} U(S'_{sys}|I)$$
$$\text{s.t. } C(S'_{sys}|I) \geqslant 0 \tag{9}$$

其中，$U(S'_{sys}|I)$为效用函数，该函数量化了目标状态S'_{sys}对意图I的满足程度。它是一个面向业务的多目标函数，其具体形式取决于意图的类型。例如，对于“高效协同”类型意图，U可定义为一个复合函数，旨在最小化任务完成时间$T_{complete}$并最大化安全性saft的加权和。

$$U(S'_{sys}|I) = \alpha \frac{1}{T_{complete}(S'_{sys})} + \beta \text{saft}(S'_{sys}) \tag{10}$$

根据不同类型的意图，U还可以包含最小化能耗、最大化设备利用率或平衡负载等其他维度，从而为意图管理器提供灵活的决策依据。

约束集合为$C(S'_{sys}|I)$，其将意图决策的约束集合映射到3个正交的状态空间中，意图管理器的任务即是在这3个状态空间的交集中寻找满足意图I的最优协同状态S'_{sys}可行解。

物理运动空间约束C_{phys}：该空间描述机械系统的几何与运动学限制，约束项包括但不限于安全边界约束和动力学可行性约束。安全边界约束可表示为

$$\forall t, \forall A_i, A_j \in A, \left\| \text{pos}_i(t) - \text{pos}_j(t) \right\|_2 \geqslant d_{saft} \tag{11}$$

其中，$\text{pos}_i(t)$为在状态S'_{sys}下规划出的未来轨迹，d_{saft}为空间距离约束值。

动力学可行性约束包括最大速度、最大加速度、最大扭矩等物理执行器的动力学限制。

控制逻辑空间约束C_{logic}：该空间描述控制器内部FB的运算与组合逻辑，约束项如下。

S'_{sys}中的逻辑状态S'_{logic}必须包含能实现意图I所需的特定算法或功能模块。此约束首先要检查智能体A_i的能力库C_{fb_i}是否支持所需的FB。例如，某种协同意图I_{task}蕴含

$$\exists v \in V'_i, v.T_{fb} = T_{required} \text{ 和 } T_{required} \in C_{fb\_i} \tag{12}$$

如果智能体A_i的能力库不支持该FB，则该决

策S'_{sys}无效。

通信拓扑逻辑空间约束C_{comm}：该空间描述底层网络架构对协同行为的支持能力，约束项如下。

S'_{sys}中的通信状态S'_{comm}必须满足S'_{logic}的数据交互需求。这体现了“通信–控制”的耦合关系。例如，若逻辑中包含一个需要高频状态同步的FB，意图管理器必须为其规划一条满足通信需求的链路。

$$\exists l \in L', l = (A_i, A_j, T_{\text{p2p}}, \omega) \tag{13}$$

此约束在S'_{logic}和S'_{comm}之间建立了不可分割的依赖关系。决策算法Π必须同时求解出S'_{logic}和S'_{comm}这一对满足约束的组合，任何只考虑其一的决策都将导致系统失效。

值得注意的是，上述所述的协同状态求解本质上是一个图约束满足问题，在一般情况下属于NP-Hard问题。然而，在工业智能体场景中，搜索空间受限于两个物理事实：一是智能体的可用功能块库C_{fb}通常较小（$< 10^2$量级）；二是单体控制器的最大连接数$L_{\max}$有限。

为实现在动态工业环境下对上述NP-Hard问题的快速、确定性求解，本文采用了一种基于启发式的协同映射算法。该算法将重构决策分解为3个序贯步骤：获选意图特征解构、逻辑匹配与通信拓扑规划。其逻辑流程如算法1所示。

算法1 基于规则的启发式协同状态映射

输入 意图I，当前全局系统状态S_{sys}，约束集合C

输出 目标状态S'_{sys}与原子重构指令集R

1) 意图特征解构：将意图I拆解为逻辑功能块需求FB_{req}（数量V）与连接约束L_{req}

2) 逻辑匹配（复杂度$O(|V||C_{\text{fb}}|)$）

3) for FB in FB_{req}

4) 遍历智能体能力库C_{fb}，筛选支持该功能类型且计算资源满足要求的候选节点

5) 采用贪心策略，将对应FB映射至当前负载最低候选节点

6) 通信拓扑规划（复杂度$O(|L|^2)$）

7) 根据L_{req}规划智能体间的通信路径，剔除时延不满足通信约束的链路

8) for每对需要通信的节点 do

9) 基于剔除后的结果，执行最短路径搜索以建立逻辑链路

10) 生成指令：计算当前状态S_{sys}与目标状态S'_{sys}的差异，生成原子重构指令集R并返回

因此，该优化问题的实际搜索空间被限定在有限域内。本文采用基于规则的启发式协同映射算法，其最坏时间复杂度可控制在$O(|V||C_{\text{fb}}| + |L|^2)$。对于节点数$N < 20$的典型工业协同场景，决策求解时间保持在毫秒级，远小于非实时的意图更新周期（通常为秒级），因此满足工业应用的可解性与实时性边界要求。

3.3 意图解析与协同约束生成

上述决策模型Π代表了一个通用的数学描述。在实际架构中，意图管理器通过一个“意图解析”的工程步骤来求解该模型。意图解析是将一个高层、声明式的意图I（可能来自人类指令或由感知器触发）翻译为一组具体的、机器可读的约束C_I的过程。

以一个通用的“紧耦合协同任务”意图为例，如图7所示，其意图解析与协同约束生成过程如下。

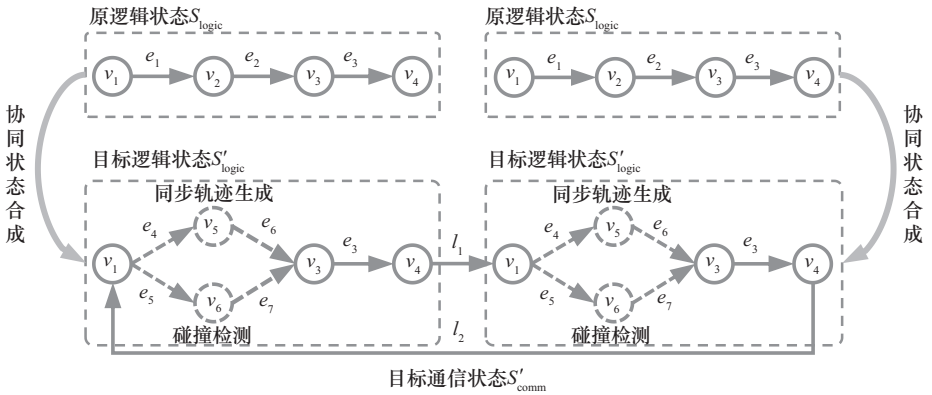

图7 协同分拣意图的目标逻辑状态与通信状态

1) 意图解析($I \rightarrow C$):意图管理器的解析器查询预定义的“意图知识库”。该库将高层业务意图I映射为一组标准化的协同约束$C_I = \{ C_{\text{phys}}, C_{\text{logic}}, C_{\text{comm}} \}$。

C_{phys}表示物理约束,包括智能体(或其控制的被控设备)之间需要保持相对距离d_{rel}或者互斥安全距离d_{safe}等空间约束。

C_{logic}表示功能约束,明确执行该任务所需要的算法FB,如必须加载特定的协同控制算法块FB_{sync}和安全监控模块FB_{safty}。

C_{comm}表示网络约束,定义支持上述功能所需的通信质量。例如,必须为协同控制算法建立智能体间的高速数据通道,要求端到端时延和抖动低于特定阈值,以匹配控制算法的数据刷新率。

2) 协同状态合成($C \rightarrow S'_{\text{collab}}$):基于$C_I$和当前$S_{\text{sys}}$,决策模块开始合成目标协同状态$S'_{\text{collab}}$。

S'_{logic}合成:执行图组合操作,生成新的逻辑图G'_{logic}。此图在原有逻辑基础上,动态插入了C_{logic},并规划了它们与现有功能块以及输入输出端口之间的数据流连接。

S'_{comm}合成:通信编排引擎执行通信链路和数据缓存区分配,生成新的通信图G'_{comm}。此图根据C_{comm}在参与协同的智能体节点之间建立了一条新的、专用的逻辑链路,用于承载高速实时数据交换。

3) 策略生成($S'_{\text{collab}} \rightarrow R$):意图管理器将目标协同状态$S'_{\text{collab}}$与当前状态$S_{\text{collab}}$进行差分比较,生成一组重构指令$R$。

$$R = \text{diff}\,(S_{\text{collab}}, S'_{\text{collab}}) \tag{14}$$

R是一组描述变化量的指令集,而非完整的状态快照,这使通信载荷极大降低。例如,$R = \{$ Create (FB5, Type = "FB_Sync"), Create (Connection, src="FB5.out",$\cdots$), Create (Comm_Link, Target="Agent1", $\cdots$),$\cdots \}$,此R即“意图通信”的最终载荷,被发送给工业智能体执行。

3.4　协同状态转移的一致性与原子性挑战

上述决策模型虽然清晰地定义了目标协同状态S'_{collab},但也暴露了执行过程中一个严峻的技术难题,即“通信–控制”失配。

在实际系统中,从当前状态$S_{\text{collab}} = (S_{\text{logic}}, S_{\text{comm}})$转移到目标协同状态$S'_{\text{collab}} = (S'_{\text{logic}}, S'_{\text{comm}})$是一个涉及多步骤(创建FB、建立连接和配置网络)的复杂过程。控制逻辑的重构和通信链路的建立在物理上是两个独立的操作,它们之间的时间不同步会导致灾难性的“失配”状态。

危害1:逻辑先于通信($S'_{\text{logic}}, S_{\text{comm}}$)。当智能体$A_i$的逻辑已切换到$S'_{\text{logic}}$(如启动了$\text{FB}_{\text{sync}}$),但通信链路$S'_{\text{comm}}$尚未建立或激活。此时,$\text{FB}_{\text{sync}}$所需的$A_j$的实时位置数据无法通过通信链路获取(数据缺失或陈旧)。这将导致FB_{sync}的算法读入无效值或陈旧数据,进而产生错误的控制输出(如速度突变、急停、误判等),引发物理系统的剧烈抖动甚至碰撞。

危害2:通信先于逻辑($S_{\text{logic}}, S'_{\text{comm}}$)。当通信链路$S'_{\text{comm}}$已被激活,$A_j$开始向$A_i$高频发送同步数据。但$A_i$的逻辑仍处于旧的$S_{\text{logic}}$(如未加载$\text{FB}_{\text{sync}}$,也未开放对应的数据端口)。这导致$A_i$的网络协议栈(如OPC UA服务器)接收到大量无法处理的、非预期的订阅数据。这可能造成网络协议栈缓冲区溢出、CPU因处理无效中断而负载飙升,甚至导致实时控制周期的“过载”,使整个智能体控制失效,触发安全停机。

因此,为保障系统安全与协同的有效性,“协同状态转移”过程必须定义3个严苛的工业约束,即原子性[35]、实时性[36]和连续性[37](atomicity, real-time, continuity, ARC)约束。

A-原子性:转移必须是原子的。系统绝不能陷入上述的($S'_{\text{logic}}, S_{\text{comm}}$)或($S_{\text{logic}}, S'_{\text{comm}}$)两种失配状态。系统状态必须从($S_{\text{logic}}, S_{\text{comm}}$)瞬时、完整地跳变为($S'_{\text{logic}}, S'_{\text{comm}}$)。

R-实时性:在多智能体协同系统中,所有参与者(如A_i和A_j)的原子化切换操作必须在一个极小且有上界的时间窗口Δt_{sync}内同时完成,以保证时序同步。

C-连续性:在准备新的S'_{collab}的整个过程中(此过程可能耗时较长,如后台加载FB、建立TCP连接),旧的S_{collab}必须持续运行,保障物理控制流的无中断。

如何设计一个能同时满足严苛ARC约束条件的机制,是实现意图驱动工业智能体互联网(IoA)架构的核心技术挑战。第4节将为此提出本文的解决方案——ACST机制。

4　原子化协同状态转移机制

第3节的形式化模型揭示了实现意图驱动IoA

的难题：如何在保障工业控制ARC约束的前提下，安全地执行“协同状态”$S_{\text{collab}}=(S_{\text{logic}},S_{\text{comm}})$的转移。为解决这一难题，本文提出了一种ACST机制。该机制是所提架构的关键使能技术，其设计核心在于通过内存空间的隔离与原子化指针操作，从根本上杜绝“通信–控制”失配的风险。

4.1　ACST机制设计

ACST机制的实现，依赖于在工业智能体的运行时内核中，为“协同状态”S_{collab}引入双缓冲内存管理模型。该内核不再维护单一的、被频繁修改的全局状态，而是在内存中实例化两个完全独立的协同状态副本。

1) 活动协同状态$S_{\text{collab_active}}$：该实例是当前唯一对物理世界生效的状态。在t_k时刻的控制周期中，实时任务执行器仅从此实例中读取其逻辑状态$S_{\text{logic_active}}$和通信状态$S_{\text{comm_active}}$，以完成该周期的计算与I/O。对于重构管理模块而言，该实例在实时控制流中是只读的，从而保证了其在执行过程中的绝对稳定性。

2) 备用协同状态$S_{\text{collab_stanby}}$：该实例是“影子”状态或“后台”状态。它在功能上与$S_{\text{collab_active}}$完全等价，但在常规运行中处于休眠状态，不参与任何实时控制。当接收到来自意图管理器的重构指令R时，运行时管理模块将锁定该实例，并对其进行写入操作，在后台安全地构建新的目标状态S'_{collab}。

为管理这两个状态实例，ACST机制在运行时内核中维护一个活动状态指针ptr_activate。

$$\text{ptr\_activate}\in\{\&S_{\text{collab\_active}},\&S_{\text{collab\_stanby}}\}\quad(15)$$

系统的实时控制循环被严格定义为仅依赖于该指针所指向的实例。

$$\text{ControlLoop}(t_k)\in\text{Execute}(\text{ptr\_activate}\rightarrow S_{\text{logic}},\text{ptr\_activate}\rightarrow S_{\text{comm}})\quad(16)$$

如图8所示，ptr_activate指针的指向是ACST机制的核心。实时控制流与重构管理流通过该指针实现了彻底隔离。实时控制流只“读”ptr_activate所指向的内容，而重构管理流只“写”ptr_activate未指向的实例。这种设计将“协同状态转移”这一复杂且耗时的过程，简化成了一个单一的、瞬时的“原子化切换”（ptr_activate指针的重定向）操作。

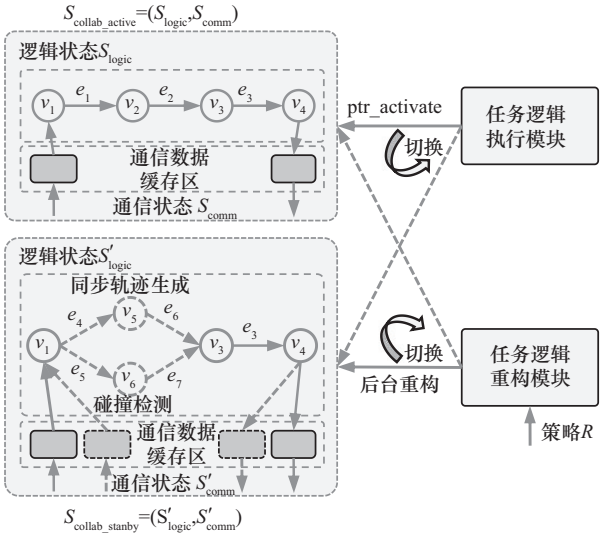

图8　ACST机制的双缓冲协同状态模型

4.2　协同重构的工作流程

基于双缓冲模型，ACST机制将协同状态转移流程细化为3个阶段。针对控制逻辑与通信拓扑在分布式环境下不同的控制周期与不同启动时刻的异构特性，本节提出了一种“离散对齐、末值容错”的转移策略，以保障在非同步物理瞬间的逻辑一致性。

ACST机制的三阶段工作流程如图9所示，该流程始于工业智能体的运行时管理模块接收到来自意图管理器的原子重构指令集$R=\text{diff}(S_{\text{collab}},S'_{\text{collab}})$。

阶段1　后台预配置

在此阶段，实时控制流继续在当前激活的$S_{\text{collab_active}}$上运行，重构管理模块锁定备用集$S_{\text{collab_stanby}}$在后台执行差异化重构。对于控制逻辑（如功能块增删或连线修改），系统直接在$S_{\text{logic_stanby}}$中实例化新对象并重定向指针连接；由于该过程与实时运行环境完全隔离，不会对当前控制流产生干扰。

针对通信状态的重构，则根据链路操作类型采取不同策略。对于新建链路，采用“增量预热”策略，即在后台提前创建对应套接字或开启协议驱动，并分配缓冲区Buf_{new}启动数据传输。此时新逻辑端口虽已绑定至Buf_{new}，但在切换前流入的数据仅被视为无效数据，其目的在于消除连接建立时的网络握手时延。对于链路删除，采用“存量延后”策略，即仅在备用逻辑中解除模块与旧缓冲区Buf_{old}的关联，而暂不执行物理层断开操作。这确保了在重构准备期间，当前逻辑$S_{\text{logic_stanby}}$仍能正常

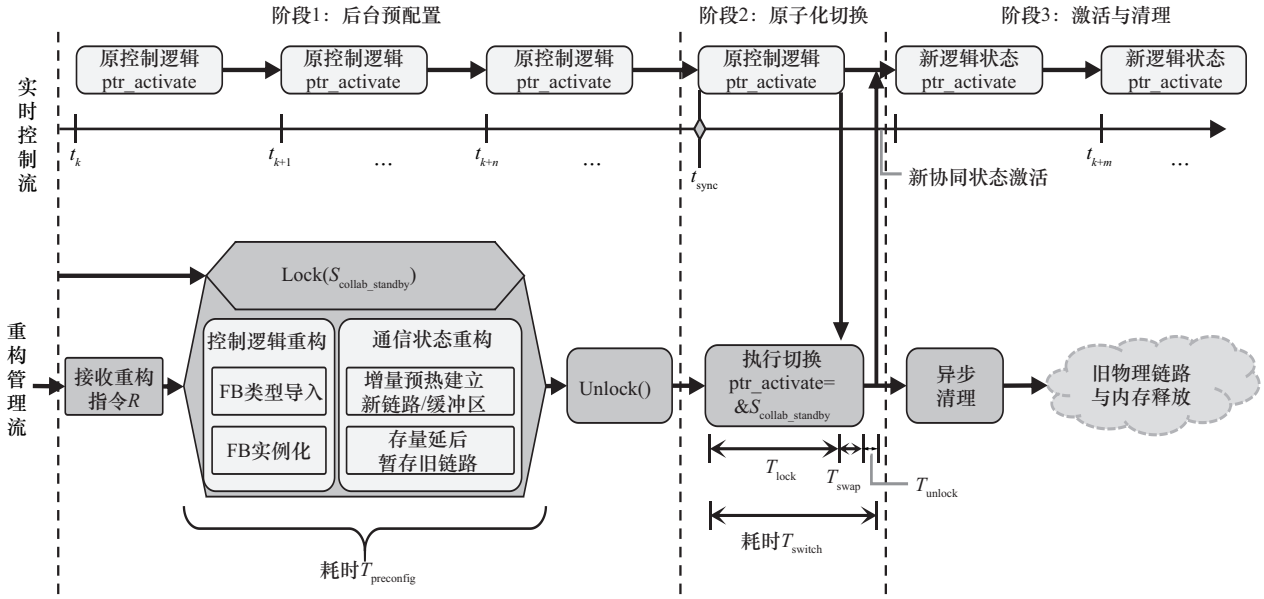

图9　ACST机制的三阶段工作流程

读写旧缓冲区 $\mathrm{Buf}_{\mathrm{old}}$，有效防止了因“通信先于逻辑断开”导致的运行时错误。

阶段2　原子化切换

当 $S_{\mathrm{collab\_stanby}}$ 完成所有逻辑构建及网络预热后，系统进入切换阶段，其核心是执行 ptr_activate 指针的原子交换。考虑到分布式系统中各工业智能体必然存在控制周期 T_{cycle} 与运行起始相位的差异，强制要求所有节点在同一物理时刻切换会破坏各自控制周期的完整性。因此，ACST 机制放弃了物理时间的绝对同步，转而追求“逻辑”的对齐。该阶段采用了基于时间触发的“开环”切换策略，而非传统的“闭环握手”确认。这是基于工业硬实时特性的慎重选择：在极短的控制周期内，试图在分布式节点间完成“请求−ACK−确认”的多次网络握手会引入巨大的、不可预测的通信时延与抖动。因此，本文采用分阶段混合策略：在阶段 1 利用闭环机制完成所有资源的预检查与锁定；而在最关键的阶段 2，则可以利用工业网络的高精度时钟同步特性，依赖全局时间基准进行开环瞬时动作，以确保确定性。

在准备阶段，意图管理器向各协同节点下发重构指令 R，待所有工业智能体完成后台预配置并反馈 Ready 信号后，意图管理器根据网络最大时延计算未来的全局生效时刻 T_{trigger}（$T_{\mathrm{trigger}} = T_{\mathrm{now}} + \Delta t_{\mathrm{margin}}$）并广播给所有协同节点。各工业智能体收到重构指令后并不立即执行，而是将切换请求注册至实时调度器的“前置检查点”。得益于工业网络的高精度时钟同步特性，各智能体在每个控制周期的起始入口处检测本地时钟。

$$\mathrm{Action} = \begin{cases} \mathrm{swap}(\mathrm{ptr\_activate}, \mathrm{ptr\_stanby}), & t \geqslant C \\ \mathrm{wait}, & t < T_{\mathrm{trigger}} \end{cases} \tag{17}$$

该机制将“硬性的时间同步”转化为“逻辑周期的对齐”。虽然各节点的物理切换动作存在由周期相位差引起的微小时间偏移，但从控制逻辑的视角看，全系统均在 T_{trigger} 之后的第一个逻辑步完成了状态迁移。这种策略在保障分布式协同一致性的同时，严格维护了单机控制周期的完整性。

为了清晰地阐述原子化切换过程中的指针操作与资源锁定逻辑，本文描述了 ACST 原子化切换与双缓冲管理算法，如算法 2 所示。

算法 2　ACST 原子化切换与双缓冲管理

输入　目标状态 S_{new}，全局生效时刻 T_{trigger}

输出　状态转移执行结果

阶段 1：后台预配置（非实时线程）函数

1) 锁定备用缓冲区：Lock (ptr_standby)

2) 清空备用缓冲区：Clean (ptr_standby)

3) 加载新控制逻辑：Load_Logic (ptr_standby, S_{logic})

4) 建立连接/缓冲区：Preheat_Comm (ptr_standby, S_{comm})

5) Flag_Ready=True

6) Unlock (ptr_standby)

阶段 2：原子化切换（实时控制线程）函数

7) 控制循环中断：Control_Cycle_Interrupt (Current_Time)

8) if Control_Cycle_Interrupt (Current_Time) //判断是否到达周期中断时间

9) Execute_Control (ptr_active) //执行当前控制逻辑

10) if (Flag_Ready == True and Current_Time >= $T_{trigger}$) //在周期结束时检查前置条件

11) std: swap (ptr_active, ptr_standby) //原子化指针交换

12) Flag_Ready=False

阶段 3：资源释放/清理（非实时控制线程）函数

13) 等待新控制逻辑稳定运行

14) if 系统无异常中断

15) 释放/清理内存

该机制通过互斥锁防止后台重构与前台执行的资源竞争。此外，双缓冲机制本质上是“以空间换时间”。相比传统单例运行，该机制使控制逻辑部分的堆内存占用增加约 100%。考虑到现代工业控制器（如 IPC 或高端服务器）通常配备 GB 级内存，而控制逻辑状态通常仅占用 MB 级内存，这种资源开销在可接受范围内。

阶段 3 激活与清理

在切换后的首个控制周期，新协同状态正式被激活，系统进入资源确立与清理阶段。得益于阶段 1 的“预热”操作，新逻辑在激活瞬间即可从 Buf_{new} 中读取到最新的有效数据，实现了通信链路的“零等待”上线。同时，由于 ptr_activate 已指向新逻辑，旧缓冲区 Buf_{old} 及其对应的物理链路变为不再被索引的“孤立资源”。重构管理模块随即在非实时线程中执行“清理”操作，安全关闭旧物理链路并释放内存。这种异步清理机制在消除资源占用的同时，确保了实时控制过程不受任何副作用影响，从而实现了控制与通信资源的平滑演进。

4.3 同步鲁棒性保障与重构异常回滚机制

考虑到实际工业场景中存在的时钟漂移与网络丢包风险，该架构在 ACST 机制的基础上构建了容错保障体系，以确保重构过程在“最坏情况”下的确定性。

首先，针对同步误差容错与量化约束的分析。系统的同步精度是保障原子化切换的前提。定义全局时钟同步误差为 $\Delta\epsilon$，ACST 控制指针切换耗时为 T_{swap}。为确保所有智能体在同一控制周期 T_{cycle} 内完成状态转移，系统必须满足时序约束 $\Delta\epsilon + T_{swap} < T_{guard} < T_{cycle} - T_{task}$。其中，$T_{task}$ 为协同控制算法的实际执行时间，T_{guard} 为预留的切换保护带宽。通过 PTP 协议将 $\Delta\epsilon$ 抑制在微秒级，结合 ACST 机制极低的切换开销，该约束在常规控制频率下具有极大的鲁棒性裕量，从而有效抵御了时钟漂移对控制周期完整性的冲击。

其次，针对广播时延波动导致的“半配置”风险，本文设计了基于版本校验的“就绪确认–全局触发”流程。在该流程中，当中心决策单元发出重构指令后，各智能体在后台缓冲区 S_{collab_stanby} 完成配置并进行逻辑校验，随后反馈就绪（Ready）信号。若中心决策单元在预设的观察窗口内未收到全网节点的反馈，或检测到同步精度超过阈值 $\Delta\epsilon_{max}$，则判定当前重构环境存在风险。一旦识别到重构风险，中心决策单元就会放弃下发全局切换脉冲。由于 ACST 机制实现了控制域与配置域的物理隔离，此时各节点的实时控制流 ControlLoop 依然安全地锁定在原活跃缓冲区 S_{collab_active} 运行。系统通过保持 ptr_activate 指针指向不变，实现了逻辑层面的“零代价”平滑回退，确保了在极端重构失效场景下，工业生产过程的连续性不受任何瞬态干扰。

最后，为了应对广播时延的非确定性并将其转化为确定性量值，重构触发信号采用“绝对时间戳触发”模式。中心决策单元将切换指令的生效时刻设定为未来某一绝对时间点 $T_{trigger}$。只要指令在 $T_{trigger}$ 之前到达各节点，网络抖动即被转化为确定的等待时长，从而保障了大规模集群重构动作在时间尺度上的严格一致性。

4.4 ACST 机制的 ARC 属性分析

ACST 机制的设计旨在严格满足前文中提出的工业 ARC 约束，本节将从理论上分析该机制如何保证以下 3 个核心属性。

1) 原子性分析

系统状态必须从 S_{collab} 瞬时、完整地跳变为 S'_{collab}，绝不能陷入 (S'_{logic}, S_{comm}) 或 (S_{logic}, S'_{comm}) 的“失配”中间态。ACST 机制通过状态副本和后台重构策略，确保了转移的原子性。在耗时最长的“后台预配置”阶段，所有对控制逻辑和通信链路的修

改均发生在“隐形”的$S_{\text{collab_stanby}}$实例上。在此期间，实时系统可见的$S_{\text{collab_active}}$保持不变。“协同状态”的真正转移被ACST机制归结为单次、不可分割的ptr_stanby指针交换操作T_{switch}。在t_{sync}周期的瞬间，ptr_stanby的指向从旧状态S_{collab}切换到新状态S'_{collab}。因此，实时控制流在前一个周期完整地执行S_{collab}，在后一个周期完整地执行S'_{collab}。在任何时刻，实时控制流所执行的S_{logic}和S_{comm}均是来自同一个且逻辑上一致的S_{collab}实例。两种“失配”状态在设计上被消除，从而确保了转移的原子性。

2) 实时性分析

状态转移的开销必须极小，以满足多智能体（如A_i, A_j）的同步切换需求。ACST机制通过异步化设计，将重构的总时间$T_{\text{reconfig}} = T_{\text{preconfig}} + T_{\text{switch}}$进行解耦，其中，后台开销$T_{\text{preconfig}}$可能长达毫秒甚至秒级，在非实时线程中执行，完全不占用实时控制周期的CPU时间。实时切换开销T_{switch}是唯一影响实时性能的开销，它是在实时控制周期T_{cycle}内，为执行指针切换所必须的同步与操作时间。本文定义T_{switch}为ACST机制在实时周期内引入的最大同步开销。该开销主要由等待锁、执行指针交换和释放锁组成。

$$T_{\text{switch}} = T_{\text{lock}} + T_{\text{swap}} + T_{\text{unlock}} \tag{18}$$

由于T_{swap}是一个$O(1)$的CPU指令，且同步锁（如轻量级自旋锁）的开销极低，故T_{switch}的时间上界是极小且可被精确测定的。这一微秒级的切换时延$T_{\text{switch}} \ll T_{\text{cycle}}$，证明了ACST机制卓越的实时性。对于多智能体同步，云端可编程逻辑控制器（cloud-based programmable logic controller, CPLC）只需广播一个“在t_{sync}时刻开始切换”的指令，所有本地可编程逻辑控制器（local-based programmable logic controller, LPLC）可在T_{switch}这一极小的时间内完成切换，从而保证了高精度的时序同步。

3) 连续性分析

在准备新状态S'_{collab}的整个过程中，旧状态S_{collab}必须持续运行，保障物理控制流的无中断。此属性由ACST机制的ptr_stanby指针设计天然保证。在耗时最长的$T_{\text{preconfig}}$阶段（后台预配置），ptr_stanby指针始终指向$S_{\text{collab_active}}$。实时控制流ControlLoop(t_k)的执行路径完全独立于重构管理模块的活动。实时控制流不需要通过执行任何if-else来判断系统是否处于重构中，也不需要等待任何资源。它在t_k周期、t_{k+1}周期均满负荷、确定性地运行。而对于物理过程的控制从未被暂停、阻塞或中断。ACST机制以“零打扰”的方式在后台完成了新状态的准备，完美满足了工业控制对连续性的严苛要求。

5　案例研究:基于双天车分拣的协同状态演化

为从物理层面端到端地验证本文所提意图驱动架构、“协同状态”模型及ACST机制的有效性与性能，本节搭建了一个双天车协同分拣物理实验平台。首先，介绍平台的物理组成与网络拓扑，并将其组件映射到第2节的理论架构。随后，通过复现一个完整的“状态感知–意图决策–原子化执行”闭环，定性地验证架构的功能可行性。最后，通过高精度时序测量，定量地分析ACST机制的ARC（原子性、实时性和连续性）性能，并验证其对意图执行高保真度的保障能力。

5.1　物理实验平台搭建

实验平台是一个集成了感知、决策、控制和执行的完整IoA物理缩影，其物理拓扑与架构组件如图10所示。实验平台主要由4类组件构成。

1) 边缘计算单元：由一台移动边缘计算（mobile edge computing, MEC）服务器和一台双目相机组成。MEC与双目相机连接并负责运行视觉识别算法（以下简称CV-Camera），实时检测分拣台上物体的类型、位置等。

2) 中心决策单元：由基于5G核心网（5GC）和基带处理单元（baseband unit, BBU）构建的5G专网计算基础设施充当。该单元利用其高性能计算能力，承载全局的、非实时的意图解析与协同决策任务。

3) 实时控制单元：由两台独立的工业个人计算机（personal computer, PC）充当，分别搭载了具备实时补丁的操作系统。每台工业PC通过EtherCAT总线精确控制3个伺服电机，模拟一台三轴（X, Y, Z）天车的运动机构。

4) 网络基础设施：包含两种异构网络层级。第一层为基于标准TCP/IP协议的以太网，用于连接MEC、中心决策单元和实时控制单元，承载非实时的策略下发与状态交互。第二层为硬实时的EtherCAT现场总线，专门用于实时控制单元与伺

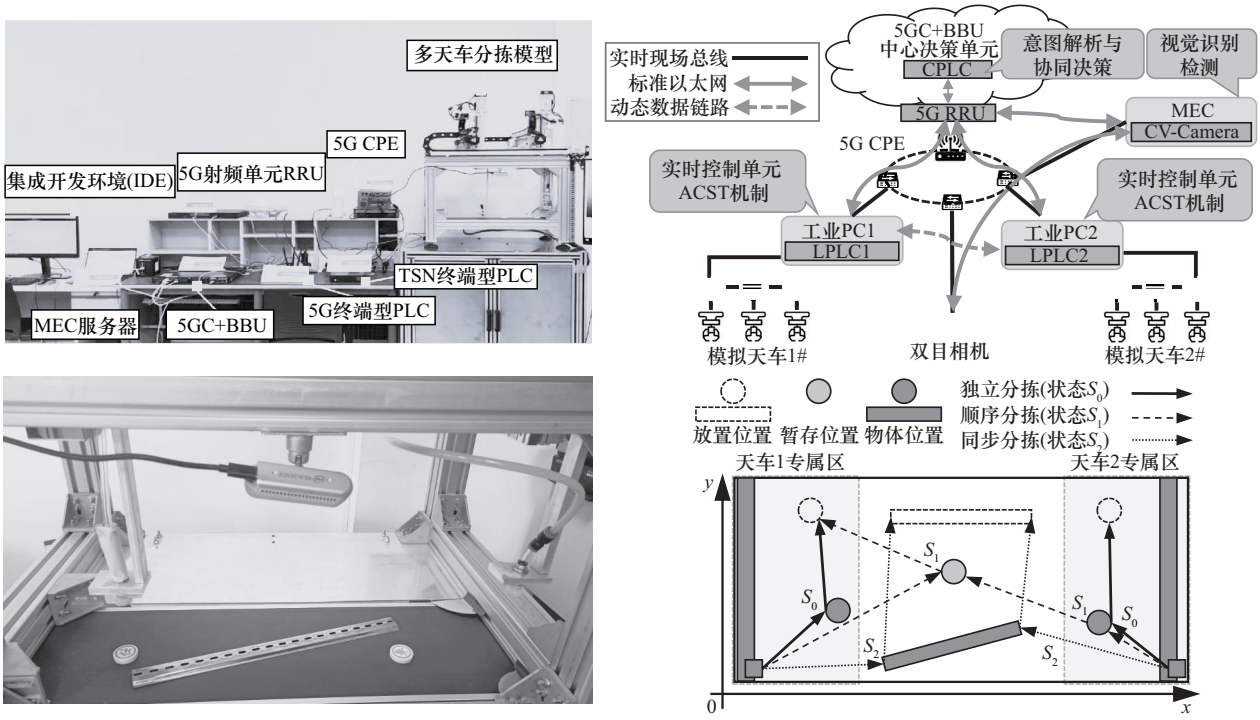

图10　实验平台物理拓扑与架构组件

服电机之间的高频控制信号传输。

为验证所提理论，本文将实验平台组件严格映射到第2节所提的意图驱动架构中。

意图管理器作为“中心决策单元”，其上运行的应用软件被称为CPLC，负责执行意图解析与协同决策算法Π。

工业智能体分别为两台“实时控制单元”。每台工业PC上运行的虚拟化PLC实例标记为LPLC1与LPLC2，均集成了第4节设计的ACST机制。它们是“协同状态”S_{collab}的最终承载者和执行者，负责执行具体的控制逻辑与通信任务。

感知器作为“感知与边缘计算单元”，负责感知当前待拾取的环境状态，是整个系统中“B-triggers-A”闭环工作流的触发源。

策略载荷：CPLC下发的协同策略（即重构指令R），在物理上被编码为可扩展标记语言格式的“重配置文件”，通过OPC UA协议经由以太网下发给LPLC1和LPLC2。

5.2　协同状态演化闭环验证

本节旨在定性地验证所提架构的完整功能闭环，即证明该架构能够响应物理世界的状态变化，自动完成“意图生成–协同决策–状态转移-物理执行”的完整流程，并根据不同的意图，演化出不同的物理行为。

1) 初始状态：独立分拣（状态S_0）

$$S_{\text{collab\_0}} = (S_{\text{logic\_indep}}, S_{\text{comm\_iso}}) \tag{19}$$

$S_{\text{logic\_indep}}$：LPLC1和LPLC2的内部逻辑仅包含独立的位置闭环控制（如FB1）。

$S_{\text{comm\_iso}}$：LPLC之间无通信连接，它们仅与CPLC保持低频心跳/状态上报链路。

物理行为：在此状态下，两个天车智能体完全独立，只能在各自的专属区域内执行简单的分拣任务，无法感知对方或进行协同。

2) 场景1：顺序分拣意图（演化至状态S_1）

步骤1　意图生成。感知层传感器检测到一个需要跨区域（从LPLC1到LPLC2区域）搬运的物体。

步骤2　协同决策。CPLC接收到该意图I_{seq}，执行决策模型Π。由于存在物理防撞约束，CPLC将其解析为一个两阶段的顺序任务，即LPLC1先将物体运至中继暂存位置，LPLC2再从中继暂存位置取走。

步骤3　策略下发。CPLC生成策略P_1，该策略要求LPLC1和LPLC2加载用于任务交接和避障的FB_{avoid}逻辑，并建立一个用于状态同步的通信链路$G_{\text{p2p_low}}$。

步骤 4　原子化转移。LPLC1 和 LPLC2 调用 ACST 机制，原子化地从 S_0 切换到 $S_1 = (S_{\text{logic\_seq}}, S_{\text{comm\_low}})$。

步骤 5　物理演化。如图 11 所示，系统正确地执行了两阶段的顺序分拣动作。LPLC1 首先进行移动，LPLC2 在 LPLC1 释放任务信号（通过 $G_{\text{p2p\_low}}$ 链路）后才启动，验证了新的 $S_{\text{collab\_1}}$ 被成功执行。

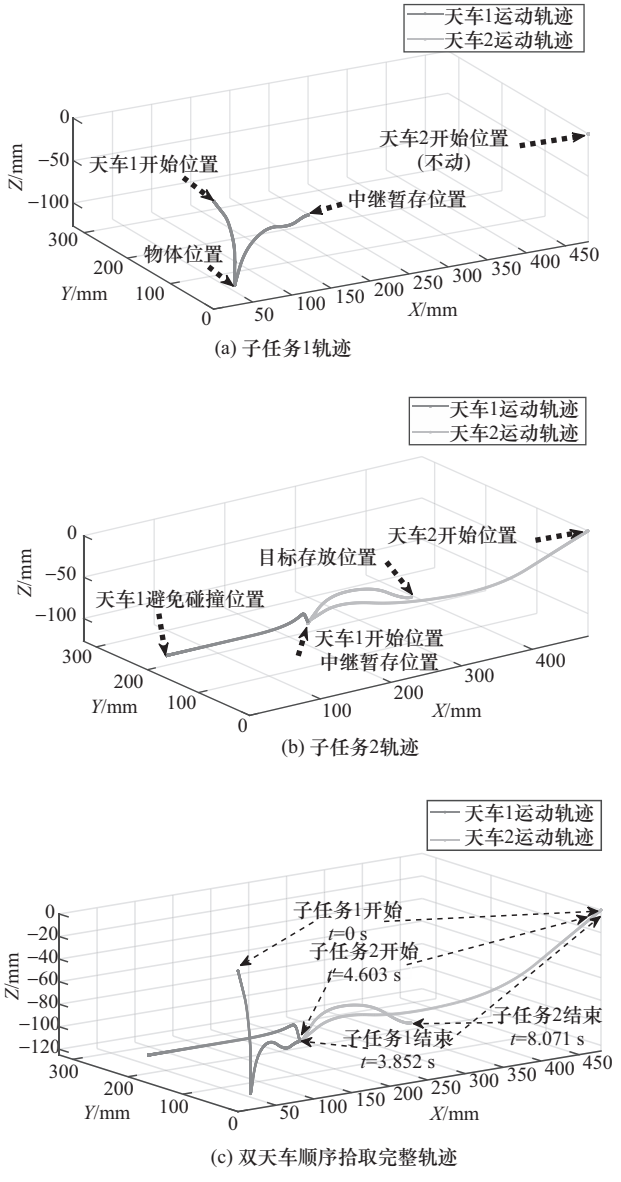

图 11　协同状态 S_1 下的顺序分拣物理轨迹

3) 场景 2：同步分拣意图（演化至状态 S_2）

步骤 1　意图生成。感知器检测到一个尺寸过大、必须双机协同搬运的物体。

步骤 2　协同决策。CPLC 接收到该意图 I_{sync}，执行决策模型 Π。CPLC 将其解析为一个高精度的同步任务。

步骤 3　策略下发。CPLC 生成策略 P_2。该策略要求 LPLC1 和 LPLC2 同时加载高频同步轨迹生成模块 FB_{sync} 和碰撞检测模块 FB_{avoid}。更关键的是，它要求同时建立一个高频和低时延的通信链路 $G_{\text{p2p\_high}}$，以满足 FB_{sync} 对伙伴实时位置的数据需求。

步骤 4　原子化转移。LPLC1 和 LPLC2 调用 ACST 机制，原子化地从 S_0（或 S_1）切换到 $S_2 = (S_{\text{logic\_sync}}, S_{\text{comm\_high}})$。

步骤 5　物理演化。如图 12 所示，系统正确地执行了高精度的同步轨迹。两个天车在 $G_{\text{p2p\_high}}$ 链路上高频交换数据，实时计算并跟踪同步轨迹，验证了 S_2 这一更复杂的协同状态被成功执行。

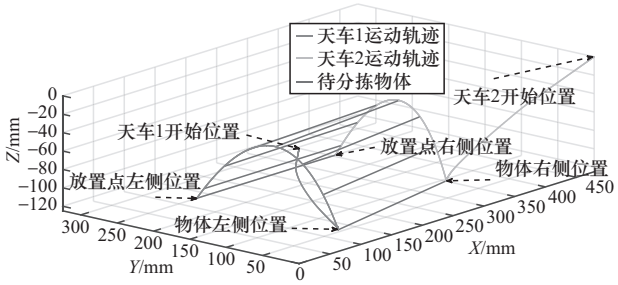

图 12　协同状态 S_2 下双天车的同步分拣物理轨迹

该实验定性地证明了所提架构的完整功能。系统能够根据不同的意图 I，正确地决策、生成并原子化地切换到完全不同的目标协同状态 S'_{collab}，进而演化出截然不同的物理行为，这证实了协同状态作为意图执行标的的有效性。

5.3　ACST 机制的切换时延与确定性分析

本节旨在定量验证第 4 节所提 ACST 机制本身的性能，即其是否真正满足 ARC 约束。

1) 连续性验证

ACST 机制的连续性由其双缓冲结构设计在理论上保证。在“后台预配置”阶段，所有重构操作均在 $S_{\text{collab\_stanby}}$ 上进行，而实时控制流 ControlLoop 始终在 $S_{\text{collab\_sctive}}$ 上无中断运行。在本次实验中，即使在执行最复杂的 S_2（同步协同）状态重构时，天车的实时控制周期也从未发生中断或过载，这证实了 ACST 机制的“零打扰”连续性。

2) 实时性与原子性验证

实时性和原子性是 ACST 机制最关键的指标，由阶段 2 中原子化切换的开销 T_{swap} 决定。该值是 ACST 机制在实时周期内引入的唯一开销，相比于

控制周期必须足够小。

为精确测量T_{swap}，本文在LPLC的运行时内核中设置高精度时间戳，测量从“请求切换标志$flag_{switch_req}$置位”到“ptr_active指针交换完毕”所消耗的实际CPU时间。本文执行了多组不同的重构任务，包括FB功能实例化、任务构造、创建数据连接以及最终的原子化切换。

实验平台为RT-Linux，4核CPU，实验结果如表1所示。

表1 ACST机制关键操作的性能开销

重构动作	对应ACST阶段	执行耗时/μs
配置文件解析/个	阶段1（后台）	86.247~136.161
FB功能注入/个	阶段1（后台）	275.696~1 442.789
FB功能实例化/个	阶段1（后台）	22.676~84.068
任务构造/个	阶段1（后台）	13.261~28.858
创建数据连接（平均）/个	阶段1（后台）	97.247
创建触发器/个	阶段1（后台）	97.766~109.643
通信协议创建（OPC UA等）/个	阶段1（后台）	36 647.733~56 485.533
原子化切换T_{swap}	阶段2（实时）	4.715~17.389

表1的实验数据清晰地验证了ACST机制的ARC属性。

后台开销：如“通信协议创建”（涉及网络I/O）这样耗时极长（36 647.733~56 485.533 μs）的操作，其开销完全被ACST机制隐藏在“阶段1：后台预配置”中，没有影响实时控制流，验证了连续性。

实时切换开销：ACST机制在实时周期内引入的唯一开销——“原子化切换T_{swap}”，其耗时为4.715~17.389 μs。这是一个18 μs以内的耗时值，远小于4 ms的当前运动控制周期，证明了ACST机制满足实时性约束。

原子性保证：由于T_{swap}（<18 μs）远小于一个控制周期T_{cycle}（4 000 μs），ptr_active指针的切换被确保在一个周期的起始点瞬时完成。实时控制流不会观察到“半配置”的中间态，从而在物理执行层面上保证了原子性。

为进一步验证ACST机制在工业实时场景下的优越性，本文将实验测得的性能指标与现有的IEC 61499动态重构机制进行了对比。表2展示了3种重构机制的性能差异。

表2 不同重构机制的性能与特征对比

重构机制	核心原理	实时性干扰（阻塞时间）	原子性保障	适用场景
停机重构	停止运行-更新-重启	高（>秒级中断）	绝对原子性	允许停机的非连续生产
IEC 61499动态重构[38]	单缓冲区修改+资源锁定	中（150~1 500 μs）	分布执行（弱原子性）	软实时或弱耦合系统
ACST	双缓冲区后台构建+指针交换	低（<18 μs）	单指令切换（强原子性）	强耦合硬实时协同

从表2可以看出，现有IEC 61499动态重构机制虽然避免了停机，但由于需要在实时控制流中插入重构操作（如挂起功能块、重新连接链路），不可避免地引入了数百微秒甚至毫秒级的阻塞时延。这种时延对于高频（如1 ms周期）的运动控制协同往往是不可接受的。相比之下，ACST机制通过内存隔离策略，将重构带来的计算与I/O开销（表1中的36 647.733~56 485.533 μs）完全移出实时路径，仅保留微秒级的指针交换操作（<18 μs），时间复杂度为常数级$O(1)$。这不仅将实时干扰降低了一个数量级，更重要的是消除了重构耗时随任务复杂度增加的不确定性，从而满足了工业ARC约束中的硬实时性要求。

5.4 ACST机制下多智能体协同的同步精度验证

本节将进一步验证该机制在实际IoA应用中，能否确保意图被高保真地执行，即验证系统级的协同性能。

在5.2节的“同步分拣”场景中，CPLC的意图I_{sync}要求两个智能体LPLC1和LPLC2同时切换到$S_2 = (S_{logic_sync}, S_{comm_high})$状态，并开始执行同步轨迹。如果ACST机制有效，两个智能体在物理世界的启动和停止时间应高度一致。如果ACST机制无效（如切换非原子化或时延抖动大），则两者之间将出现明显的时序失配。

本文通过在LPLC的实时内核中记录高精度时间戳，捕获两个智能体在执行S_2（同步分拣）任务时，各自的物理启动时刻t_{start}和结束时刻t_{end}。定义系统级的启动同步误差为$\Delta t_{start} = |t_{start,LPLC1} - t_{start,LPLC2}|$，结束同步误差为$\Delta t_{end} = |t_{end,LPLC1} - t_{end,LPLC2}|$。

本文连续执行了 4 次由意图触发的同步分拣任务，实验数据如表 3 所示。

表 3 数据表明，在 ACST 机制的支持下，两个 LPLC 的启动同步误差 Δt_{start} 始终为 0.064~1.594 ms，结束同步误差 Δt_{end} 为 0.106~4.096 ms。

高保真度验证：所有的同步误差均被控制在极小的范围内。这证明了 CPLC 的“同步”意图被两个 LPLC 高保真地复现到了物理世界。

ACST 机制有效性验证：如此高的同步精度，直接归功于 ACST 机制的原子性和实时性。它确保了 LPLC1 和 LPLC2 在收到策略后，能够同时、原子化地切换到 $S_2 = (S_{\text{logic\_sync}}, S_{\text{comm\_high}})$ 状态。如果切换是非原子化的，或者切换时延抖动巨大，则两个智能体之间必然会产生巨大的时序失配，导致同步任务失败。

表 3　多次同步分拣任务的系统级同步误差

实验次数	任务描述	启动时刻/ms	结束时刻/ms	同步误差 Δt/ms
1	LPLC1（左侧）	0.033	2 761.560	启动：0.064
	LPLC2（右侧）	0.097	2 757.678	结束：3.882
2	LPLC1（左侧）	0.215	2 563.556	启动：0.197
	LPLC2（右侧）	0.018	2 563.662	结束：0.106
3	LPLC1（左侧）	0.071	2 561.563	启动：1.594
	LPLC2（右侧）	1.665	2 565.659	结束：4.096
4	LPLC1（左侧）	0.571	2 564.656	启动：0.099
	LPLC2（右侧）	0.670	2 564.709	结束：0.144

小于 18 μs 耗时的切换机制时延（T_{swap}）和 5.4 节的毫秒级（<4.1 ms）系统同步误差（Δt）共同构成了完整的证据链，证明了 ACST 机制是解决 IoA 中“通信–控制”协同重构问题的一种有效且高性能的使能技术。

5.5　端到端重构时延的确定性理论分析

本节建立时延模型以分析系统在网络波动等最坏情况下的表现。端到端重构总时延 T_{e2e} 可分解为

$$T_{\text{e2e}} = T_{\text{dist}} + T_{\text{pre}} + T_{\text{sync}} + T_{\text{swap}} \tag{20}$$

其中，T_{dist} 为意图解析与重构指令在网络中的分发时延；T_{pre} 为智能体接收重构指令后，在后台进行的资源加载与预配置耗时；T_{sync} 为所有节点就绪后，等待全局同步触发信号的缓冲时间；T_{swap} 为 ACST 机制执行原子化指针切换的耗时。

最坏情况下的确定性分析：在理想网络环境下，T_{dist} 通常稳定在毫秒级。然而，在工业现场遭遇高负载干扰时（最坏情况），T_{dist} 可能会因 TCP 重传或交换机排队而显著增加，甚至出现长尾抖动。在传统的直接重构方法中，网络抖动会直接耦合进控制回路，导致控制周期 T_{cycle} 超时。然而，本文架构通过 ACST 机制实现了控制域与配置域的时序解耦。

非实时域的隔离（$T_{\text{dist}}+T_{\text{pre}}$）：重构指令分发与后台预配置过程完全在非实时线程中执行。根据双缓冲原理，此时实时控制流仍锁定在旧状态 $S_{\text{collab\_active}}$ 上运行。即使网络拥塞导致 T_{dist} 激增至秒级，仅会导致 T_{e2e} 变大，而绝不会阻塞当前正在运行的控制循环，也不会引入控制周期的抖动。

实时域的确定性（T_{swap}）：一旦所有节点完成 T_{pre}，系统进入同步等待期 T_{sync}。此时，最终的切换动作 T_{swap} 仅依赖本地内存指针操作，其耗时是确定性的常数时间。

理论分析表明，ACST 机制将网络层的不确定性屏蔽在实时控制层之外。即使在网络丢包率上升的极端情况下，系统仍能保障底层控制过程的连续性与确定性，这符合工业控制系统“安全优先”的设计准则。

6　结束语

本文面向工业 IoA 的柔性动态协同需求，提出意图驱动的“通信–控制”协同重构架构。通过“协同状态”形式化模型和基于双缓冲的 ACST 机制，实现了高层意图到底层控制逻辑与通信拓扑的原子化同步演化。在最小化状态切换时延与抖动的同时规避了“通信–控制”失配的风险。物理实验结果表明，本文架构可满足工业场景对高保真和高实时动态协同（同步误差小于 4.1 ms）的严苛要求。

展望未来，意图驱动的工业 IoA 架构仍需在多方面开展进一步研究。首先，可以探索分层分布式的意图管理框架，结合局部自治与全局协调以应对大规模异构集群的扩展性挑战。并借助软件定义驱动的动态映射技术提升多协议适配能力。其次，可以进一步探索 ACST 机制与 TSN、5G-

URLLC等技术的融合，通过跨层时间对齐实现跨地域高精度协同，为软件定义工业智能体提供确定性通信保障。最后，针对工业现场的极端工况构建极端工况应力测试床，完善“两阶段确认”回退机制以增强传输失效或同步超差等异常下的安全性与确定性。

参考文献：

[1] Didden J B H C, Dang Q V, Adan I J B F. Decentralized learning multi-agent system for online machine shop scheduling problem[J]. Journal of Manufacturing Systems, 2023, 67: 338-360.

[2] 张平, 刘会永, 李文璟, 等. 工业智能网: 工业互联网的深化与升级[J]. 通信学报, 2018, 39(12): 134-140.
Zhang P, Liu H Y, Li W J, et al. Industrial intelligent network: deepening and upgrading of industrial Internet[J]. Journal on Communications, 2018, 39(12): 134-140.

[3] 胡玉姣, 黄韬, 贾庆民, 等. 通算存智一体协同的未来网络模型[J]. 通信学报, 2024, 45(5): 12-28.
Hu Y J, Huang T, Jia Q M, et al. Future network model for integrated collaboration of communication, computing, caching and intelligence[J]. Journal on Communications, 2024, 45(5): 12-28.

[4] Xu W T, Gu J H, Zhang W Q, et al. Multi-agent reinforcement learning for flexible shop scheduling problem: a survey[J]. Frontiers in Industrial Engineering, 2025, 3: 1611512.

[5] Sasiain J, Franco D, Atutxa A, et al. Toward the integration and convergence between 5G and TSN technologies and architectures for industrial communications: a survey[J]. IEEE Communications Surveys & Tutorials, 2025, 27(1): 259-321.

[6] Höse K, Amaral A, Götze U, et al. Manufacturing flexibility through industry 4.0 technological concepts: impact and assessment[J]. Global Journal of Flexible Systems Management, 2023, 24(2): 271-289.

[7] Liu L S, Xu Z J, Qu X B. A reconfigurable architecture for industrial control systems: overview and challenges[J]. Machines, 2024, 12(11): 793.

[8] Zeydan E, Mangues J, Arslan S S, et al. Reconfigurable production lines for industrial 5.0 automation: an intent-based approach[J]. IEEE Open Journal of the Computer Society, 2025, 6: 1341-1352.

[9] 杨静雅, 唐晓刚, 周一青, 等. 意图抽象与知识联合驱动的6G内生智能网络架构[J]. 通信学报, 2023, 44(2): 12-26.
Yang J Y, Tang X G, Zhou Y Q, et al. 6G native intelligence network architecture enabled by intent abstraction and knowledge[J]. Journal on Communications, 2023, 44(2): 12-26.

[10] 张汝云, 肖戈扬, 单麒赫, 等. 多模态网络下多智能体协同控制的通信拓扑重构方法[J]. 通信学报, 2022, 43(4): 50-59.
Zhang R Y, Xiao G Y, Shan Q H, et al. Communication topology reconstruction method for multi-agent cooperative control in polymorphic networks[J]. Journal on Communications, 2022, 43(4): 50-59.

[11] Leivadeas A, Falkner M. A survey on intent-based networking[J]. IEEE Communications Surveys & Tutorials, 2023, 25(1): 625-655.

[12] Mehmood K, Kralevska K, Palma D. Intent-driven autonomous network and service management in future cellular networks: a structured literature review[J]. Computer Networks, 2023, 220: 109477.

[13] Filinis N, Tzanettis I, Spatharakis D, et al. Intent-driven orchestration of serverless applications in the computing continuum[J]. Future Generation Computer Systems, 2024, 154: 72-86.

[14] 王义涛, 王俊森, 石章松, 等. 基于改进快速搜索树和合同网的多智能体目标分配算法[J]. 兵工学报, 2025, 46(5): 23-34.
Wang Y T, Wang J S, Shi Z S, et al. Task allocation for multi-agent system based on extended rapidly-exploring random tree and contract net[J]. Acta Armamentarii, 2025, 46(5): 23-34.

[15] Liu Y, Xia Z C, Gui W H. Multiobjective distributed optimization via a predefined-time multiagent approach[J]. IEEE Transactions on Automatic Control, 2023, 68(11): 6998-7005.

[16] Chang H G, Liu Y M, Sheng Z G. Distributed multi-agent reinforcement learning for collaborative path planning and scheduling in blockchain-based cognitive Internet of vehicles[J]. IEEE Transactions on Vehicular Technology, 2024, 73(5): 6301-6317.

[17] Strasser T, Zoitl A, Christensen J H, et al. Design and execution issues in IEC 61499 distributed automation and control systems[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2011, 41(1): 41-51.

[18] Lyu G L, Brennan R W. Towards IEC 61499-based distributed intelligent automation: a literature review[J]. IEEE Transactions on Industrial Informatics, 2021, 17(4): 2295-2306.

[19] Faqrizal I, Salaün G, Falcone Y. Adaptive industrial control systems via IEC 61499 and runtime enforcement[J]. ACM Transactions on Autonomous and Adaptive Systems, 2024, 19(4): 1-31.

[20] Prenzel L, Zoitl A, Provost J. IEC 61499 runtime environments: a state of the art comparison[C]//Computer Aided Systems Theory-EUROCAST 2019. Berlin: Springer, 2020: 453-460.

[21] Martini B, Gharbaoui M, Castoldi P. Intent-based network slicing for SDN vertical services with assurance: context, design and preliminary experiments[J]. Future Generation Computer Systems, 2023, 142: 101-116.

[22] Chowdhury M. Accelerator: an intent-based intelligent resource-slicing scheme for SFC-based 6G application execution over SDN-and NFV-empowered zero-touch network[J]. Frontiers in Communications and Networks, 2024, 5: 1385656.

[23] Barzegar S, Ruiz M, Velasco L. Autonomous flow routing for near real-time quality of service assurance[J]. IEEE Transactions on Network and Service Management, 2024, 21(2): 2504-2514.

[24] Betalo M L, Leng S, Abishu H N, et al. Multi-agent deep reinforcement learning-based task scheduling and resource sharing for O-RAN-empowered multi-UAV-assisted wireless sensor networks[J]. IEEE Transactions on Vehicular Technology, 2023, 73(7): 9247-9261.

[25] Ning Z P, Xie L H. A survey on multi-agent reinforcement learning and its application[J]. Journal of Automation and Intelligence, 2024, 3(2):

73-91.

[26] Li Y X, Liu Q H, Li X Y, et al. Manufacturing resource-based self-organizing scheduling using multi-agent system and deep reinforcement learning[J]. Journal of Manufacturing Systems, 2025, 79: 179-198.

[27] Straub J. A modern blackboard architecture implementation with external command execution capability[J]. Software Impacts, 2022, 11: 100183.

[28] Mason F, Chiariotti F, Zanella A, et al. Multi-agent reinforcement learning for coordinating communication and control[J]. IEEE Transactions on Cognitive Communications and Networking, 2024, 10(4): 1566-1581.

[29] Gielis J, Shankar A, Prorok A. A critical review of communications in multi-robot systems[J]. Current Robotics Reports, 2022, 3(4): 213-225.

[30] Zhu C X, Dastani M, Wang S H. A survey of multi-agent deep reinforcement learning with communication[J]. Autonomous Agents and Multi-Agent Systems, 2024, 38: 4.

[31] Tan C, Dong X Y, Li Y J, et al. Leader-following consensus problem of networked multi-agent systems under switching topologies and communication constraints[J]. IET Control Theory & Applications, 2020, 14(20): 3686-3696.

[32] Dai W B, Vyatkin V, Christensen J H, et al. Bridging service-oriented architecture and IEC 61499 for flexibility and interoperability[J]. IEEE Transactions on Industrial Informatics, 2015, 11(3): 771-781.

[33] Lepuschitz W, Zoitl A, Vallée M, et al. Toward self-reconfiguration of manufacturing systems using automation agents[J]. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 2011, 41(1): 52-69.

[34] Seitz M, Gehlhoff F, Salazar L A C, et al. Automation platform independent multi-agent system for robust networks of production resources in industry 4.0[J]. Journal of Intelligent Manufacturing, 2021, 32(7): 2023-2041.

[35] Ahrens E, Bozga M, Iosif R, et al. Reasoning about distributed reconfigurable systems[J]. Proceedings of the ACM on Programming Languages, 2022, 6(OOPSLA2): 145-174.

[36] Li J, Xiong H G, Li Q, et al. Run-time reconfiguration strategy and implementation of time-triggered networks[J]. Electronics, 2022, 11(9): 1477.

[37] Gundall M, Stegmann J, Reichardt M, et al. Downtime optimized live migration of industrial real-time control services[C]//Proceedings of the 2022 IEEE 31st International Symposium on Industrial Electronics (ISIE). Piscataway: IEEE Press, 2022: 253-260.

[38] Prenzel L, Hofmann S, Steinhorst S. Real-time dynamic reconfiguration for IEC 61499[C]//Proceedings of the 2022 IEEE 5th International Conference on Industrial Cyber-Physical Systems (ICPS). Piscataway: IEEE Press, 2022: 1-6.

[作者简介]

陈乐（2000– ），男，广西贵港人，北京科技大学博士生，主要研究方向为工业互联网、网络化控制系统。

马彰超（1984– ），男，山西晋城人，博士，北京科技大学副教授、硕士生导师，主要研究方向为网算控一体化的智能开放工业控制系统。

董芃（1998– ），男，天津人，北京科技大学博士生，主要研究方向为工业互联网、信息物理系统。

张荣辉（1987– ），男，河南驻马店人，博士，北京科技大学副教授、硕士生导师，主要研究方向为通信感知计算控制一体化。

牛子儒（2002– ），男，内蒙古通辽人，北京科技大学硕士生，主要研究方向为工业互联网、任务编排。

王健全（1974– ），男，山西平遥人，博士，北京科技大学教授、博士生导师，主要研究方向为工业泛在网络、工业互联网、工业互联网安全、网络协同与智能制造等。