计 算 机 研 究 与 发 展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202220741
60(5): 1086−1098, 2023

ROS2 多线程执行器上 DAG 任务的优先级分配方法

纪　东[1,2]　魏阳杰[1,2]　李宇溪[1]　王　义[1,2]

[1]（东北大学计算机科学与工程学院　沈阳　110819）

[2]（医学影像智能计算教育部重点实验室（东北大学）沈阳　110819）

（jidong@cse.neu.edu.cn）

Priority Assignment Method of DAG Task on ROS2 Multithreaded Executor

Ji Dong[1,2], Wei Yangjie[1,2], Li Yuxi[1], and Wang Yi[1,2]

[1]（*School of Computer Science and Engineering*, *Northeastern University*, *Shenyang* 110819）

[2]（*Key Laboratory of Intelligent Computing in Medical Image*(*Northeastern University*), *Ministry of Education*, *Shenyang* 110819）

Abstract　With the growing popularity of the robot operating system (ROS), these systems are becoming increasingly complex, and the computing platforms they run on are transforming into multi-core platforms. In ROS, the order of task execution is determined by the underlying task scheduling strategy and the priorities assigned to the tasks. Minimizing the execution time of all tasks is a crucial goal in task scheduling for parallel systems. To address this challenge, we propose a reinforcement learning-based task priority assignment method, inspired by recent achievements in using reinforcement learning for handling various combinatorial optimization problems and considering the scheduling mechanisms and execution constraints of ROS2 multi-threaded executors. This method extracts the temporal and structural features of the task set, which is represented in the form of a directed acyclic graph (DAG), and efficiently learns the ROS2 scheduling policy through a combination of policy gradient and Monte Carlo tree search (MCTS) methods, providing a reasonable priority setting scheme. The goal of minimizing the completion time of DAG parallel tasks is achieved through this method. The proposed method is evaluated by simulating randomly generated task graphs in a simulation platform environment. The results show that it outperforms the benchmark methods. As an off-line analysis method, the proposed method can be easily extended to more complex ROS and can find a near-optimal solution in an acceptable amount of time.

Key words　robot operating system (ROS); reinforcement learning (RL); DAG task; priority assignment; Monte Carlo tree search (MCTS)

摘　要　随着机器人操作系统（robot operating system, ROS）的日益普及，系统也变得更加复杂，这类系统的计算平台正逐渐转变为多核心平台. 在 ROS 中，任务执行的顺序取决于底层任务调度策略和分配给任务的优先级，而最大限度地缩短所有任务的执行时间是并行系统任务调度的一个重要目标. 受强化学习在解决各种组合优化问题的最新研究成果的启发，在考虑 ROS2 多线程执行器的调度机制和执行约束的前提下，提出了一种基于强化学习的任务优先级分配方法，该方法提取了基于有向无环图形式表示的任务集的时间和结构特征，通过策略梯度和蒙特卡洛树搜索（Monte Carlo tree search, MCTS）方法有效地学习 ROS2 调度策略并给出合理的优先级设置方案，最终达到最小化并行任务的最大完工时间的目的. 通过模拟平台环境下随机生成的任务图以评估所提方法，结果表明所提方法明显优于基准方法. 作为一种离线分析方法，所提方法可以很容易地扩展到复杂的 ROS 中，在可接受的时间内找到接近最优的解决方案.

收稿日期：2022-08-22；**修回日期**：2023-02-17

基金项目：国家自然科学基金项目(61973059)；中央高校基本科研业务费专项资金 (N2116006)

This work was supported by the National Natural Science Foundation of China (61973059) and the Fundamental Research Funds for the Central Universities (N2116006).

通信作者：李宇溪(liyuxi@cse.neu.edu.cn)

关键词 机器人操作系统；强化学习；DAG 任务；优先级分配；蒙特卡洛树搜索

中图法分类号 TP399

随着机器人和自动驾驶行业的发展，为了能够实现产品的快速迭代，代码复用性和模块化的需求变得日益强烈. 目前行业内已经提出在许多软件平台（中间件）引入了模块化和可适应的功能，从而使构建相关系统变得更加容易. 其中，机器人操作系统（robot operating system, ROS）在机器人行业中扮演着至关重要的角色，它以软件模块化和可组合性来加速机器人软件开发的过程. 2017 年第 2 代 ROS2 发布[1]，ROS2 提供了强大的实时能力，以支持实时机器人软件. 例如，ROS2 采用数据分发服务（data distribution service, DDS）来进行实时数据的端到端交换，并且可以部署在实时操作系统之上. 与 ROS2 实时性能相关的核心组件是执行器，它协调底层操作系统（operating system, OS）线程中工作负载的执行. ROS2 有 2 个内置执行器：一个是按顺序执行工作负载的单线程执行器，另一个是跨多个线程分配工作负载执行的多线程执行器. 近些年，有几项工作[2-6]研究了 ROS2 单线程执行器的建模和实时性能. 结果表明，ROS2 执行器的调度行为与以往实时调度研究中的调度行为有很大不同. 而随着现代机器人应用程序变得越来越复杂，通常需要在多线程执行器上实现它们，以充分挖掘多核处理器的计算能力，这就需要新的分析技术.

本文旨在研究 ROS2 多线程执行器上运行的以有向无环图（directed acyclic graph, DAG）为特征的并行任务的执行行为，通过配置任务内结点的调度顺序，降低 ROS2 上应用程序执行时间. DAG 调度问题是 2 个众所周知的 NP 难问题的组合：最小完工时间问题[7]和装箱问题[8]. 其目标是找到一个最短的最长完工时间的调度顺序. 然而，在调度并行 DAG 任务时，在某个时间点，该任务的许多顶点都符合执行条件，并且数量多于可用核数，因此需要提出一种优先级分配方法来指定这些顶点的执行顺序. 但现有的调度算法，如文献 [9−12] 在这种情况下没有指定相关执行顺序. 有的研究者通过求解一个具有分支定界[13-14]的整数规划问题来找到执行顺序，但是无法解决大多数实际规模的问题. 而启发式方法（如最高级别优先、最长工作时间、关键路径和随机优先级[15]）为任务分配优先级，属于迭代型优化算法. 当问题规模很大时，大量的迭代搜索仍然会导致较大的计算耗时. 此外，当问题发生变化时，通常需要重新搜索解决方案或调整启发式规则以获得更好的结果，但这会增加计算成本.

最近，工业界和学术界对寻找使用机器学习的自适应数据驱动调度算法的兴趣激增. 本文基于自适应学习的思路，针对 ROS2 多线程执行器和特殊的调度方式，提出了基于强化学习与蒙特卡洛树搜索（Monte Carlo tree search, MCTS）的优先级分配方法，通过在模拟平台进行反复交互，逐步学习 DAG 的拓扑结构和执行状态，以及 DAG 中每个顶点的运行时间和硬件资源使用情况. 在此基础上，本方法利用策略价值网络在搜索树中逐步探索 DAG 中任务执行先后顺序，目的是最小化最大完工时间.

本文的主要贡献包括 3 个方面：

1）对 ROS2 上任务的执行方式进行了全面和深入的分析和研究，尤其对于多线程执行器的调度机制进行了详细建模.

2）针对 ROS2 上特殊的调度机制，本文开发了一套用于模拟 ROS2 执行行为的模拟器，用于验证和追踪不同调度方法下程序的执行状态，并可提供详尽的模拟结果.

3）提出了基于强化学习和蒙特卡洛树搜索的 ROS2 多线程执行器以 DAG 表征的任务优先级配置方法. 并对随机生成的测试用例进行了分析，结果表明提出的优先级配置方案相比基准测试方法能够找到更优的方案，进而缩短了任务的总执行时间.

1 相关工作

为了提高 ROS[16-17]的实时性能，研究者们已经开展了大量工作. Casini 等人[2]对 ROS2 执行器进行了形式化建模和分析，特别是对单线程执行器的调度行为进行了建模，为后续分析奠定了基础. Tang 等人[3]改进了响应时间分析技术并提出了优先级分配方法以优化响应时间. 文献 [5] 通过考虑实际执行时间的差异并进一步探索 ROS2 中的调度特性来改进文献 [2] 的问题. 文献 [18] 为 ROS2 框架提出了一种基于链式结构的新的优先级驱动的调度器，并对提出的调度器进行了端到端延迟分析. 目前，针对 ROS2 系统的分析，现有工作都集中在单线程执行器的分析上，而对于多线程执行器的调度行为和优先级的设置问题，还没有相关工作研究. 本文基于这个原因，开展了 ROS2 多线程执行器上的研究分析.

在多线程或者多核平台上提高并行处理能力的

主要方式是设计有效的任务调度算法. 近年来，研究者对于 DAG 系统进行了深入的分析，采用了不同的调度策略，包括全局调度[19-22]和联合调度[9,23-24]. 例如，Zhao 等人[25]探索了 DAG 结构中的并行性和依赖关系，并基于并发提供者和消费者模型提出了优先级分配方法和响应时间界限. Pathan 等人[26]提出了一种利用任务内优先级分配来提高资源利用率的方法，并利用准备时间和结点响应时间的思想来减少任务内干扰. He 等人[27-28]开发并设计启发式方法以确定并行任务的优先级策略，提出了一种计算具有任意优先级分配的 DAG 任务响应时限的分析方法. 还有很多关于在具有任务内优先级分配的多处理器平台上的调度技术，他们的目标是减少端到端响应时间. 文献 [15,29] 考虑了 DAG 静态调度算法的优先级分配. Kwok 等人[30]提出了一种基于任务图的关键路径，将任务图分配给全连接多处理器的静态调度算法.

通过在多个处理器上调度大量任务来最小化执行时长被认为是一个 NP 难问题. 因此，为了获得接近最优的解决方案，研究者也在尝试使用基于机器学习的启发式方法. 例如，文献 [31−32] 使用机器学习来学习作业结点的近似最优排序. 但这些研究没有考虑由 DAG 定义的作业之间的依赖性，也没有在分配执行器和其他资源时利用 DAG 拓扑. 强化学习(reinforcement learning, RL)作为一种重要的机器学习算法，越来越受到关注. 强化学习是一个与未知环境反复交互以获得最优解[33]的学习过程. 基于强化学习的任务调度，既考虑短期价值，又考虑长期价值. 因此，使用强化学习进行任务调度相比启发式等算法，陷入局部最优解的可能性降低. Lee 等人[34]提出了一种使用图卷积网络来处理复杂的相互依赖的 DAG 任务结构，并使用策略梯度方法开展了关于优先级的调度策略的研究，以最小化 DAG 任务的执行时间. Decima[35]使用强化学习来选择 DAG 作业的排序以及执行器的分配. 但强化学习方法的好坏受到奖励函数、特征提取、探索过程等多个因素的影响，例如强化学习的奖励函数无法兼顾策略安全性与稳定性，对于给连续任务分配优先级的问题，策略梯度估计的方差会随时间的推移被放大，并且策略的探索与输出也难以预测和约束. 本文受强化学习在调度任务的启发，提出了基于强化学习的蒙特卡洛树搜索方法，该方法不仅可以利用训练的策略网络指导搜索过程，提高树搜索的性能，还可以通过扩展树形结构模拟各种可能的未来轨迹进行评估，从而选择更有前景的方向来探索最佳策略，最终达到最小化任务执行时间的目的.

2 背景介绍

本节主要介绍了 ROS2 体系结构以及其执行流程和调度机制.

2.1 ROS2 体系结构

如图 1 展示了 ROS2 的体系结构，ROS2 应用程序通常由节点组成，每个节点都是负责一个单一的用途模块，多个节点相互组合协同，从而简化了代码结构和提高了代码重用性. 节点是发布和接收消息的主体，使用发布—订阅范例相互通信：节点在主题

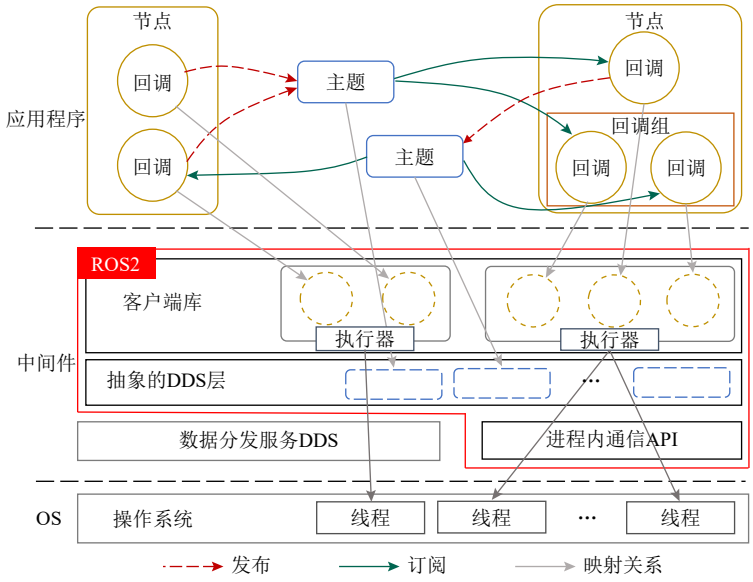


Fig. 1 Illustration of ROS2

图 1 ROS2 示意图

上发布消息，通过通信层数据分发服务 DDS 将消息广播给订阅主题的节点. 节点通过激活回调来处理每个消息，从而对传入消息做出反应. 为了部署 ROS 应用程序，需要将各个节点分布到主机，然后映射到操作系统进程. ROS2 中的执行器用于协调操作系统进程中节点回调的执行. ROS2 提供了 2 个内置的执行器：一个是在单个线程中执行回调的顺序执行器，另一个是在多个线程中处理回调的并行执行器.

一个执行器中同一节点内的回调可以属于某个回调组. 回调组有 2 种类型：互斥或可重入. 在运行时，不同类型回调组中的回调根据特定规则进行调度（将在 3.2 节介绍）. 回调也可能属于不同的节点，默认情况下，同一节点中未指定给任何回调组的回调被视为位于互斥回调组中.

2.2 多线程执行器的调度方法

本节介绍了 ROS2 Foxy Fitzroy[36] 中多线程执行器下的调度行为，该行为基于对源代码和相关文档的仔细检查[1-2,37]，并通过使用跟踪工具 ROS2_tracing[4] 进行的大量实验进行了验证.

回调是程序设计者可以处理的最小调度实体. 图 2 阐述了当回调被激活后，线程调度回调的工作流. 执行器中的所有线程都维护一个公共集 *wait_set*,

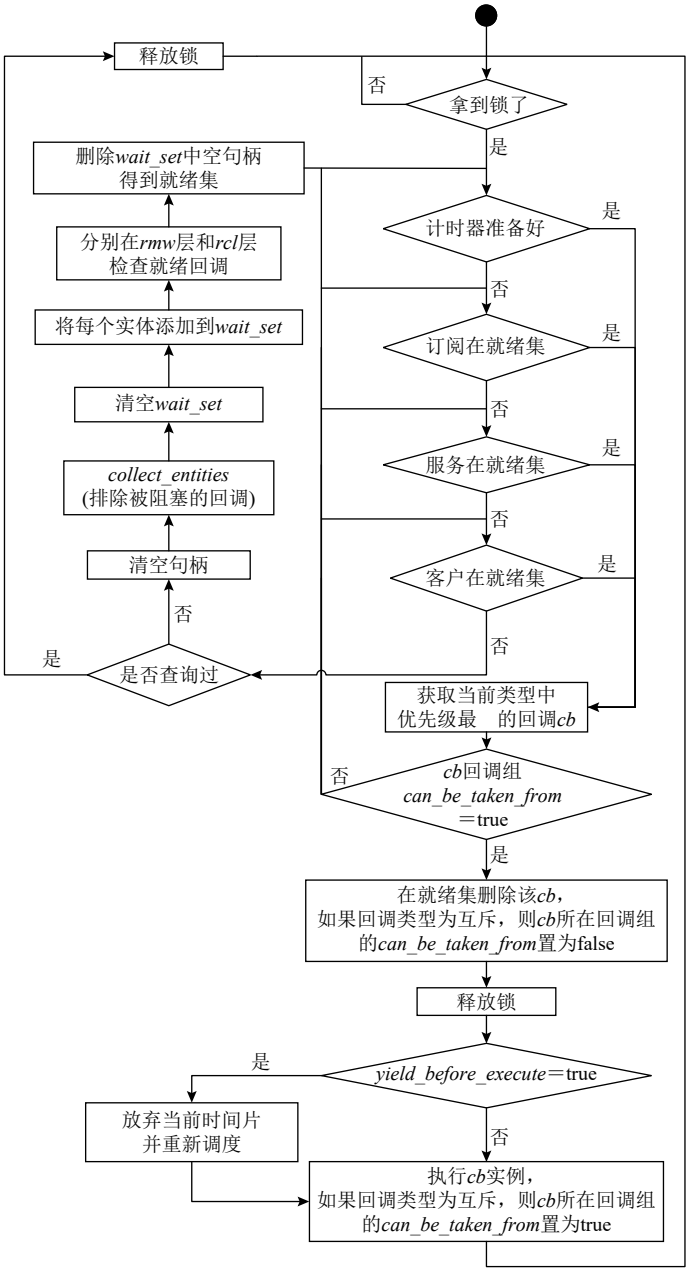


Fig. 2 The workflow of a thread in a ROS2 multi-threaded executor

图 2 ROS2 多线程执行器中线程的工作流

以记录有可用消息的回调. *wait_set* 由互斥锁 *mutex* 保护，线程在访问 *wait_set* 之前必须持有锁. 也就是说，*wait_set* 在任何时候最多只能由一个线程访问和修改；否则，线程将被阻塞，等待锁被释放. 当持有锁时，线程通过固定顺序搜索不同类别的回调来寻找要执行的回调.

正如在文献 [2−3,5] 中介绍的那样，回调的优先级(即执行时选择执行的顺序)由 2 个级别决定：1) 回调类型. 不同回调类型具有不同的优先级，回调分为计时器、订阅、服务和客户 4 种类型. 总的来说，4 种回调类型的优先级顺序，如图 2 所示，依次是计时器、订阅、服务和客户. 2)注册顺序. 同一类型回调的优先级取决于它们的注册顺序，较早注册的回调具有更高的优先级，这样每个回调就静态分配一个唯一的优先级. 在执行回调之前，还要进一步查验是否受到互斥影响. 对于可重入模式的互斥组，可以直接进入执行阶段；而对于互斥回调组需要查验标志位 *can_be_taken_from*，以确定是否可以执行. 只有当 *can_be_taken_from* = true 时，才能选择属于互斥回调组的回调；否则，线程将跳过此回调并继续查验下一个回调. 当选择此组中的回调执行时，标志设置为 false，然后在回调执行完成后设置为 true.

一旦成功选择回调，线程就会释放锁，并从就绪集中移除. 之后，ROS2 通过判断标志位 *yield_before_execute* 来决定所选回调是否可以立即执行. 如果设置 *yield_before_execute* = true，线程将把处理器交付给操作系统，直到操作系统再次调度它；否则，线程开始以非抢占方式执行回调. 在本文中，由于本文内容不关注 ROS2 和操作系统之间的交互，本文只考虑 *yield_before_execute* = false 的情况，这也是默认配置.

如果在查询过程中，没有发现可以执行的回调，那么线程将进入更新阶段(此时仍然持有锁). 在经过一些列初始化后，如在 *rcl* 层和 *rmw* 层的句柄重置和实体收集等，之后等待新的消息到来并更新 *wait_set*(如果有可更新的). 在等待函数 *wait_for* 设置中，还可以通过设置一定的时间限制 *timeout*(默认情况无等待时长约束)，即如果到达一定时间仍无法更新到新的回调，则线程将释放锁并变为空闲.

3 任务执行模型

本节将介绍抽象出来的 ROS2 多线程执行器的相关行为的模型.

3.1 负载模型

描述并行任务最通用的模型之一是有向无环图模型，该模型允许将任务表示为一系列子任务，这些子任务描述了潜在的并行计算，以及表示子任务的执行顺序的约束情况. 本文考虑在具有 M 个相同核的系统上执行应用程序. 程序以有向无环图 $G=(V,E)$ 的形式表示，其中 V 表示回调集合，$E\in V\times V$ 表示受到执行约束的回调对之间的边的集合. 每个回调 v_i 执行需要一定的时间，其最坏执行时间(worst case execution time, WCET)用 e_i 表示. 一条边 $(v_i,v_{i'})$ 表示回调 v_i 和 $v_{i'}$ 的优先关系，即 $v_{i'}$ 只能在 v_i 执行完毕后才开始执行. 没有传入(传出)边的顶点称为源点(汇点). 在不丧失一般性的情况下，本文假设 G 只有一个源点(表示为 v_{src})和一个汇点(表示为 v_{sink}). 在 G 有多个源点或汇点的情况下，可以添加一个具有零 WCET 的虚拟源点或汇点，以符合本文的假设. 另外，每个回调属于一个唯一的回调组，本文使用 $G(v_i)$ 表示 v_i 所属组的索引，如果 $G(v_i)=0$，表示 v_i 属于可重入组.

路径 $\lambda=[v_{\lambda_1},v_{\lambda_2},\cdots,v_{\lambda_k}]$ 是一个有限的回调序列，其中 $(v_{\lambda_i},v_{\lambda_{i+1}})\in E$，$i\in\{1,2,\cdots,k-1\}$. 如果路径同时包含源点和汇点，则称为完整路径. 路径长度是路径中结点的执行时间之和，即 $\sum_{v_i\in\lambda}e_i$. 完整路径中路径长度最长的路径称为关键路径，本文使用 $\mathcal{L}$ 表示关键路径的工作负载，即仅关键路径中结点的 WCET 之和. 在执行过程中，G 会释放无限序列的实例. 周期 T 是 G 的 2 个连续实例的释放时间之间的最小间隔. DAG 有一个相对的最后期限 D，在时刻 r 释放的 G 实例必须在时刻 $r+D$ 之前完成，也就是说最长路径的工作负载之和 $\mathcal{L}\leqslant r+D$，在本文中主要考虑了满足这种约束条件下的任务执行模式，称为可调度的 G. $\mathcal{L}$ 的负载之和为本文优化的 DAG 总执行时间，即最长完工时间 *makespan*.

3.2 调度模型

在运行时，G 中源点后的回调实例首先被释放，而 G 中的其他回调实例将在前任实例的输入消息可用时才被释放. 被释放的回调 v_i 的实例能否执行还取决于回调组的情况：

1) v_i 属于互斥回调组. v_i 所在的回调组中没有回调(包含 v_i)的实例正在执行时，表示处于准备就绪状态；否则，v_i 将被阻塞.

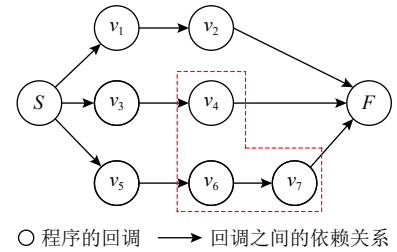
2) v_i 属于可重入回调组. v_i 在释放后始终处于就绪状态.

执行器维护一个就绪集合 Ω，它记录了可以执行

的就绪回调. 执行器可以选择 Ω 中的就绪回调进行执行. 从上面的描述中可以看到, 同一互斥回调组中回调的回调实例不能并行执行, 而可重入回调组中回调的多个回调实例可能并行执行. 另外, 回调实例准备就绪后不会立即添加到 Ω. 相反, 只有在 Ω 中没有符合条件的回调且线程空闲时, 才能将就绪回调实例添加到 Ω. 此外, 当更新 Ω 时, 首先设置 $\Omega=\varnothing$, 然后将所有当前就绪的回调实例添加到 Ω. 即在同一时刻(通常称为轮询点)Ω 中加入了一组回调实例.

在执行阶段, M 个线程逐个选择 Ω 中符合条件的回调实例, 并以非抢占方式执行. 而选择合格回调实例的顺序取决于它们的优先级. 每个回调都有一个固定且唯一的优先级, 其所有实例都继承该优先级. 当选择执行回调实例时, 回调实例将从 Ω 中删除.

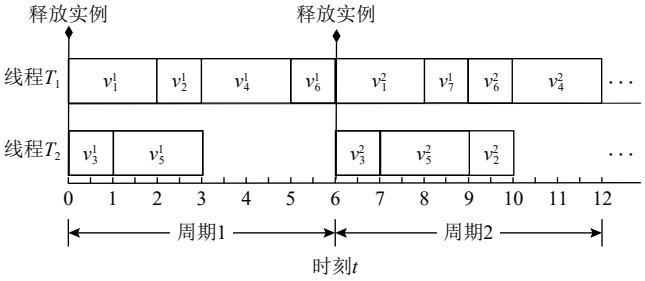
图 3 阐述了使用 2 个线程在 ROS2 系统上运行 DAG 程序执行的示例. 该程序由 7 个回调组成(源点和汇点不考虑在内), 圆形符号表示程序中的回调, 箭头表示回调之间的依赖关系, 其中回调 v_4, v_6, v_7 属于同一互斥回调组, 其他回调属于可重入回调组. 所有回调的优先级和最坏情况执行时间如图 3(b)所示. 向上菱形符号表示 DAG 实例释放时刻, 释放周期为 6, 其中 v_j^i 表示第 i 个回调的第 j 个实例. 我们通过设置 2 种不同的优先级来展示优先级对 DAG 任务执行的影响, 其中优先级 1 的结果如图 3(c), 优先级 2 的结果如图 3(e) . 对应的每个轮询点的 Ω 中的回调实例如图 3(d)(f)所示, 集合的内容由任务依赖关系和线程更新 Ω 时刻所决定. 例如, 在时刻2执行完 v_1^1 后线程 T_1 空闲, 且此时 Ω 为空, 则线程 T_1 将就绪的 v_2^1 和 v_4^1 更新到 Ω. 比较 2 种优先级配置的运行结果可发现, 尽管只是对 v_2 和 v_4 优先级进行了互换就导致了整个系统由可调度变成了不可调度的情况, 观察图 3(c)(e)可发现, DAG 中最长路径的一次实例的执行时长也由6变成了9. 原因就是在时刻2, 在优先级 2 的配置下, v_2 先于 v_4 执行, 又因互斥组的原因导致了 v_6 和 v_7 的延后执行. 这说明合理设置优先级以及 DAG 中回调的约束关系对程序运行和执行时长的重要性.



(a) 回调关系

组别	回调	执行时间	优先级1	优先级2
组1	v_1	2	1	1
	v_2	1	6	5
组2	v_3	1	2	2
	v_4	2	5	6
组3	v_5	2	3	3
	v_6	1	4	4
	v_7	1	7	7

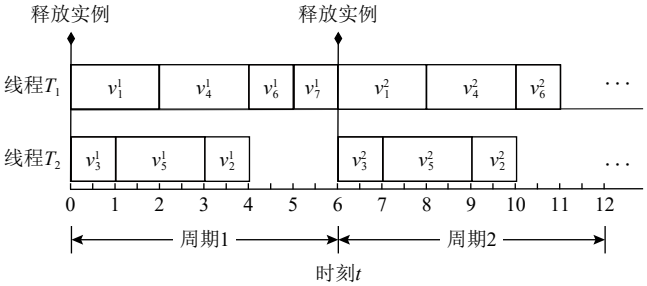
(b) 回调配置



(c) 基于优先级1的执行结果

t	就绪集合 Ω
0	$\{v_1^1, v_3^1, v_5^1\}$
2	$\{v_4^1, v_2^1\}$
4	$\{v_6^1\}$
5	$\{v_7^1\}$
6	$\{v_1^2, v_3^2, v_5^2\}$
8	$\{v_4^2, v_2^2\}$
9	$\{v_6^2\}$
10	$\{v_7^2\}$

(d) 优先级1的就绪集合状态



(e) 基于优先级2的执行结果

t	就绪集合 Ω
0	$\{v_1^1, v_3^1, v_5^1\}$
2	$\{v_2^1, v_4^1\}$
5	$\{v_6^1\}$
6	$\{v_1^2, v_3^2, v_5^2, v_7^1\}$
9	$\{v_6^2, v_2^2, v_4^2\}$

(f) 优先级2的就绪集合状态

Fig. 3 Example of task scheduling under ROS2 multithread executor

图 3 ROS2 多线程执行器下的任务调度示例

4 提出的方法

本节描述了优先级配置方法，包括马尔可夫决策过程(Markov decision process, MDP)模型和带有 RL 智能体的 MCTS 算法. 图 4 说明了整体架构及其主要组件. 该方法遵循通用的强化学习框架，关键组件为环境状态、动作、奖励和模拟环境. 智能体按照典型的 Actor-Critic 算法执行训练，并结合 MCTS 方法计算得到最优方案. 执行过程从初始化状态开始，根据跟踪的文件数据 1 获取下一时刻的环境状态，并以简明的形式提取状态特征 2，并反馈到 RL 智能体以生成可行的动作结果 3，最后，计算该动作的奖励 4 并返回给智能体进行训练. 由于大多数调度任务的执行性能指标只能在整个回调序列被调度后才能计算，因此在奖励计算期间，本方案将每个单独动作的中间奖励保持为 0，只计算完成回调序列中的最后一项回调后的最终奖励. 一旦安排了整个 DAG 回调执行序列，本方案就会建立一个包含优先级特征的执行轨迹.

4.1 马尔可夫决策过程

本节从马尔可夫决策过程的角度讨论了在 ROS2 系统下 DAG 的执行方式. 一般来说，MDP 具有一组状态、一组动作、奖励函数和状态转移概率.

1) 状态. 首先，本文方法封装了当前时刻t、核的工作状态以及回调间的互斥约束关系. 为了通过顺序选择来调度 DAG 任务，状态还需要关于随时间演变的子任务分区的信息，即处于该状态下满足当前依赖关系的就绪回调集合 Ω 和已经执行完的回调集合. 而这些状态信息是通过模拟器按 ROS2 的执行策略和 DAG 约束条件，随时间变化模拟得到的.

2) 动作. 动作行为表示时刻t在就绪集合 Ω 中选择一个回调进行执行. 本文将拥有 N 个回调程序的动作，选择操作a编码为一个离散空间$\{0, 1, \cdots, N\}$，当 $a = m$时表示选择第m个回调，如果满足约束条件，则将回调挂载到空闲核上，此动作并不占用实际的执行时间. 如果$a = 0$，则表示对挂载的回调进行处理，将占用核上资源并产生执行时间.

3) 奖励. 为了学习得到最合理的优先级配置，以最小化 DAG 工作流完成时间作为最优策略奖励参数. 我们将程序完成时的总累积奖励设置为负的工作流完成时间，即v_i的$-makespan$. 而在任务未完成时的动作选择奖励为0. 因此，最大化强化学习模型的回报相当于最小化工作流完成时间. 奖励函数的定义为

$$reward = \begin{cases} 0, & a = 1, 2, \cdots, N, \\ v_i\text{的} - makespan, & a = 0. \end{cases} \tag{1}$$

4) 转移概率. 在 MDP 中，过渡状态是确定性的，即对于一个状态和动作，下一个状态是没有随机性的. 这是因为调度操作不执行任务，它只影响调度策略并更改任务的排列(优先级顺序). 当采用强化学习处理大规模组合优化问题时，这种设置很常见[38].

4.2 MCTS 过程

MCTS 是一种最佳优先搜索算法，由 Coulom 在 2006 年引入[39]，用于在围棋中找到好的决策，但也可用于不同的 MDP，如集群资源调度[35]、生产调度[40]. 使用 MCTS 可以有效地搜索决策的空间，MCTS 由 4 个部分组成：选择、扩展、模拟和回溯. 在本文中，我们使用训练好的策略网络来指导 MCTS 过程，即在搜索过程中使用经过强化学习训练的智能体给出最有希望的动作，在缩小搜索空间的同时使 MCTS 可以更有效地找到任务的执行顺序. 具体步骤有 5 个：

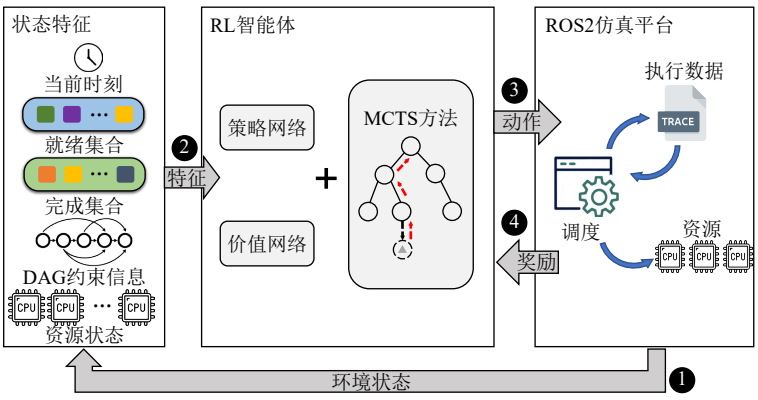


Fig. 4 Frame diagram of reinforcement learning-based priority assignment method

图 4 基于强化学习的优先级分配方法框架图

1）选择阶段. 在算法的初始阶段，蒙特卡洛树中只有一个初始结点. 这一步从根结点开始，逐树探索下一个最有希望的子结点，以保持探索和利用之间的平衡. 一般情况下，使用上置信界（upper confidence bound, UCB）作为每个结点优劣的评价标准，即

$$UCB\_score = \bar{V}_i + c\sqrt{\frac{\ln N_i}{n_i}}, \tag{2}$$

其中$\bar{V}_i$为该结点的平均价值大小，c 为常数探索因子，N_i为总探索次数，n_i为当前结点的探索次数. 为了适应本文最小化最长完工时间. $\bar{V}_i$设置为该结点所有访问中的最大负 *makespan* 值，新的 UCB 即为

$$NewUCB\_score = \max\nolimits_{-makespan} + c\sqrt{\frac{\ln N_i}{n_i}}. \tag{3}$$

2）扩展阶段. 如果选择最有希望的结点不是终止结点，则通过一个未探索的动作来扩展搜索树. 在这个阶段，可用的子结点是就绪任务队列中的一组动作或处理动作（$a = 0$）.

3）模拟阶段. 在经典的 MCTS 方法中，扩展后的搜索树中添加了新结点，选择一个新结点作为模拟的开始，其中移动一般由随机策略选择，并进行快速模拟到游戏结束及获得结果. 然而，大量的决策维度导致搜索算法开销很大，无法获得有希望的结果. 所以本文使用经过训练的强化学习智能体来替换随机策略.

4）回溯阶段. 在模拟结束时，根据模拟的 *makespan* 结果，从叶结点到根结点的访问路径上的所有结点，使用反向传播方式更新当前动作序列的值. 对于每个结点，该值被更新为当前负 *makespan* 和新 *makespan* 的最大值，表示为

$$value = \max\{-makespan, current\_value\}. \tag{4}$$

5）迭代预算. MCTS 算法重复上面 4 个步骤来增加搜索树的规模，直到计算资源耗尽. 最后，MCTS 将选择导致最高可能值的操作. 受文献 [41] 启发，迭代次数被设置为资源预算. 值得注意的是，本方案还为每个级别的预算设计了一个适当的衰减函数，在 DAG 调度问题中，当向下遍历搜索树时，搜索空间呈指数下降. 因此，可用的决策减少，然后随着树的深入，将迭代的上限设置为

$$budget = \max\left(\frac{b_{\text{initial}}}{d_i}, b_{\min}\right), \tag{5}$$

其中d_i是树的当前深度，b_{initial}和$b_{\min}$分别表示初始和最小迭代预算.

5 实验和评估

本节评估了提出的优先级分配方法，通过生成随机任务集和使用离散事件模拟器模拟多线程执行器的执行来评估本文设计的方法.

5.1 实验参数

该实验通过随机生成的 DAG 任务进行评估，任务集和测试集生成遵循在各种现有可调度性分析论文中使用的相同程序，例如文献 [42−43]，以便在可调度性测试和模拟结果之间进行比较. 生成各种 DAG 的具体输入参数有：1）回调数N；2）线程数M；3）DAG 的长度L；4）结点的最大出度max_{out}；5）结点的最大入度max_{in}；6）DAG 的形状参数α；7）DAG 的正则性参数β；8）任务执行时间e；9）DAG 中的互斥组个数上限g；10）互斥组中任务个数上限u. 在实验中，这些参数的值是从给定的集合中均匀采样得来的，其中$N \in \{5, 10, 20, 30, 40, 50\}$；$M \in \{2, 3, 4\}$；$max_{\text{out}} = 3$；$max_{\text{in}} = 3$；$\alpha \in \{0.5, 1.0, 1.5\}$；$\beta \in \{0.0, 0.5, 1.0, 2.0\}$；每个回调的最坏执行时间$e$在 [1, 100] 随机采样；互斥组个数在$\{0, 1, 2, \cdots, g\}$随机选择，$g = N/10$；互斥组内回调个数在$\{1, 2, \cdots, u\}$随机选择，其中$u = (N/10) + 1$. 通过上述设置，可以获得大量具有不同属性的任务流，从而防止对特定调度方法的偏见.

Actor 网络和 Critic 网络使用相同的配置，即宽度均为 128 网络的隐藏层神经网络. 使用了 tanh 激活功能. softmax 函数为 Actor 网络的输出层中的动作空间生成一系列概率. 受近端策略优化（proximal policy optimization, PPO）[44] 良好性能的影响，本文应用 PPO 来训练调度器. Actor 网络和 Critic 网络的学习率分别设置为3×10^{-4}和1×10^{-3}，折扣率$\gamma = 0.99$，裁剪范围系数为 0.2. 训练数据和测试数据来自由随机参数生成的 1 000 个 DAG. 在 MCTS 实现中，本文使用预先训练的 PPO 智能体作为指导，$b_{\text{initial}} = 100$ 和$b_{\min} = 10$.

所有实验都是基于 Python 3.6.13，torch 1.7.0 和 gym 0.19.0 环境实施，并在具有 128 GB 内存的 Intel® Xeon® Bronze 3104 CPU@1.70 GHz 和具有 CUDA 11.1 NVIDIA geforce rtx 3090 GPU 的 ubuntu 18.04.6 LTS 系统上进行训练和测试.

5.2 基准测试方法

对于本文的多线程执行器模型，DAG 任务在 M 个同构处理器上执行，则核数 M 和回调数 N 设置为数据规格，并包含在测试数据集中. 也就是说，在测试阶段，从数据集采样的每个 DAG 任务被配置在 M

个处理器的系统上运行. 本文的模型旨在最小化单个 DAG 任务的完成时间. 因此, 本文通过模拟器环境测量了各个调度方法的最长完工时间并将其用作评估指标. 比较的方法有:

1) 随机方法(Random). 在动作空间中随机采样动作, 在存在空闲核的情况下, 如果采样的动作无效, 如受到回调间约束关系和互斥关系而无法执行, 则重新选择动作. 如果发现此时没有空闲的核挂载该就绪回调, 则将当前动作变更为 0.

2) 短任务优先方法(shortest job first, SJF). 该方法首先在就绪集合 Ω 中根据最坏执行时间按降序对回调进行排序, 然后选择第 1 个回调作为下一个动作. 同随机方法, 如果发现此时没有空闲的核挂载在就绪回调, 则将当前动作变更为 0. 因为增加了核的状态判断, 所以本方法相比于随机选择动作的方法, 在对空闲核的利用上会表现得更有优势.

3) PPO. 使用 PPO 算法训练的网络, 根据环境状态选择下一个可执行的回调.

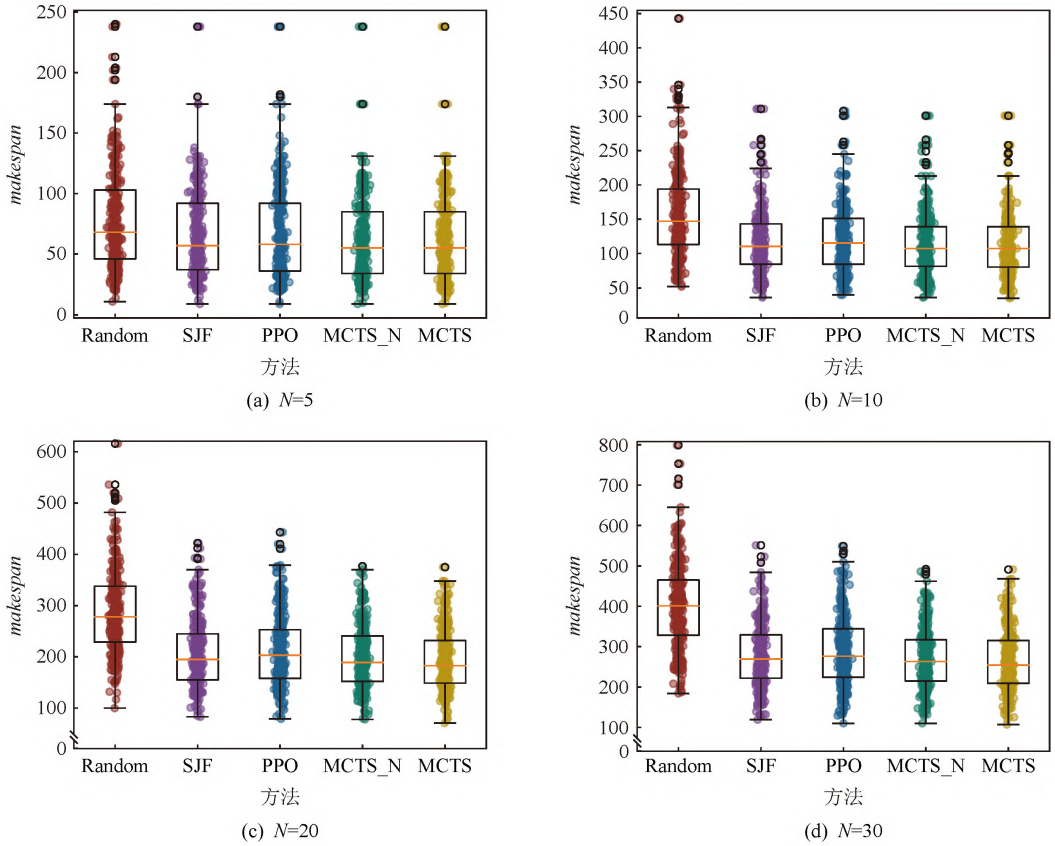
4) 没有 PPO 的 MCTS(MCTS_N). 为了验证强化学习对 MCTS 调度程序的关键影响, 分别对有和没有 PPO 智能体的 MCTS 进行了消融实验.

5.3 模拟环境

为了评估本文提出的方法, 本工作需要搭建与执行环境交互的平台. 虽然可以直接以 ROS2 真实系统作为交互环境, 但由于 ROS2 的性能依赖于系统的调度策略且受限于系统资源状态, 如果直接在 ROS2 系统中训练会存在训练时间长以及参数测量不准等潜在问题. 为了解决此问题, 本文实现了一个基于 ROS2 执行策略的 gym[45] 调度模拟器, 模拟器对于 DAG 调度方法至关重要, 为智能体和环境之间的交互提供平台. 该模拟器可以模拟 M 个同构处理器的 ROS2 系统执行行为, 并精确计算每个 DAG 任务按给定优先级顺序下的最大完工时间. 回调序列和执行器的设置参数被输入至模拟器, 以模拟 ROS2 处理平台如何执行回调. 在初始化步骤, 初始化回调序列、执行器状态和就绪回调集合等. 然后, 按优先级对就绪回调进行排序. 模拟器从就绪回调逐个选择回调实例, 并根据所选回调实例的约束情况决定执行步骤, 如挂载到相应的空闲核上并记录此时状态等. 最后, 模拟器根据核上回调的执行时长, 并按降序排列, 更新模拟时间以确定下一个调度时刻, 并更新核的工作状态. 模拟器重复上述过程, 直到所有作业都分配给执行器, 最后算出 DAG 任务的最大完工时间.

5.4 实验评估与分析

图 5 展示了优先级分配方法与基准测试在不同 DAG 尺寸下随机选取的 100 组测试集数据的最长完工时间情况. 表 1 展示了增加核数区分后的实际平均结果.



(a) N=5 (b) N=10

(c) N=20 (d) N=30

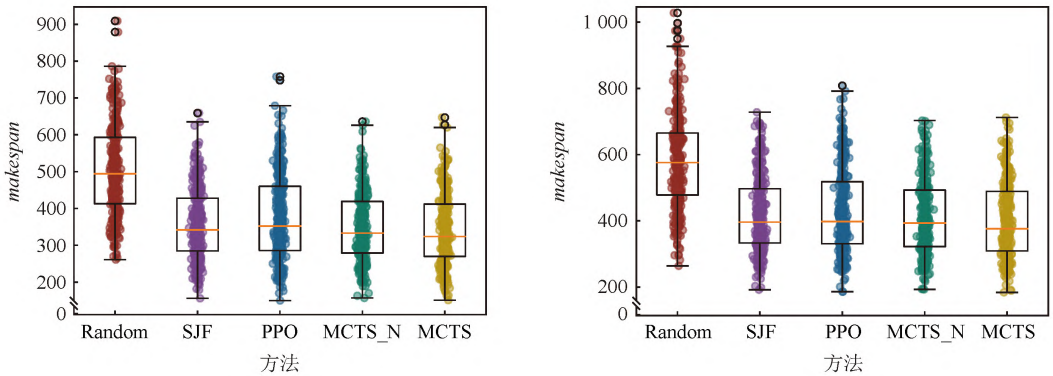


Fig. 5 Performance of *makespan* in different DAG sizes

图 5 不同 DAG 尺寸下的最大完工时间性能

从表 1 可以看出，本文所提出的优先级分配方法明显优于其他所有方法，且随着回调数 N 和核数 M 的增加效果更加明显. 其中由于随机方法选择动作的不确定性，对核的利用不佳，执行时长最长且波动范围最大. SJF 直接在就绪集合选择回调，再选择执行动作，所以整体来说核的利用率更高，运行时间相对较短. PPO 的执行时间略高于 SJF，这是因为虽然本文对 DAG 约束关系进行了表征，但仍然在某些情况不能完全涵盖，如虽然当前存在空闲核，但由于互斥原因，在 Ω 集合中选择的回调无法执行，这给 PPO 的训练过程带来了偏差，而且随着核数减少，复杂的约束关系将变得更加明显.

Table 1 Average *makespan* with Different Methods

表 1 不同方法的平均最长完工时间

核数 M	回调数 N	Random	SJF	PPO	MCTS_N	MCTS
2	5	83.35	73.04	75.26	65.21	65.00
	10	165.15	131.18	135.42	125.37	121.52
	20	321.05	246.96	257.30	237.75	231.42
	30	455.91	347.18	368.32	335.67	334.33
	40	606.56	454.41	504.51	442.52	440.59
	50	706.24	546.83	577.66	534.23	532.03
3	5	77.06	64.64	65.20	60.67	60.67
	10	152.58	113.33	116.30	109.64	108.65
	20	280.47	195.45	200.25	188.30	182.60
	30	396.23	261.62	265.66	256.59	246.37
	40	479.27	332.80	340.18	323.17	315.78
	50	554.64	390.28	387.93	377.91	374.54
4	5	74.19	61.76	61.69	59.88	59.88
	10	150.93	108.57	111.35	106.97	106.30
	20	263.40	177.03	183.62	173.89	170.59
	30	353.06	229.29	228.06	221.08	214.17
	40	431.72	282.73	282.55	277.86	265.42
	50	479.38	322.66	321.32	313.90	296.72

加入 MCTS 后，MCTS_N 和 MCTS 方法的最长完工时间 *makespan* 好于 Random, SJF, PPO 三种方法. 通过消融实验，也说明了 PPO 在帮助 MCTS 快速搜索相关结点的前提下，带来了更好的计算结果，并且随着结点数越多，核数越多，可选择的动作越多，可操作的空间越丰富时，其效果越明显. 除此之外，本文以核数为 4 的 ROS2 模拟系统为例，对提出方法的求解过程带来的时间开销进行了测量. 本文随机选取了测试集中 DAG 结点数为 30 的 50 个组进行测量，其中 PPO 测试的平均 *makespan* 为 238.59，平均时间开销为 843 ms. 对不同资源限制配置下的 MCTS 和 MCTS_N 计算时间也进行了测量，如表 2 所示.

Table 2 Execution Time Analysis for MCTS Method

表 2 MCTS 方法执行时间分析

b_{initial}	$b_{\min}$	MCTS_N		MCTS	
		makespan	执行时间/ms	*makespan*	执行时间/ms
10	5	220.79	70 572	215.20	141 526
10	10	219.29	134 610	213.31	176 093
50	5	220.57	99 651	213.22	146 856
50	10	222.10	160 808	212.14	213 164
100	5	220.47	158 221	212.12	233 844
100	10	220.37	186 324	211.84	282 441
500	10	219.90	512 627	210.90	1 023 218
1000	10	217.47	722 909	208.73	1 299 164

从表 2 可以看到 MCTS 在 PPO 的帮助下，虽然同样配置下的运行时间要高于 MCTS_N，但 *makespan* 的结果要优于 MCTS_N，而且随着b_{initial}的增加，*makespan* 效果趋于更好的结果. 横向比较这 2 种方法，MCTS

在$b_{initial}=10$和$b_{min}=5$时就能达到很好的效果，运行时间和运行结果都好于在更高迭代预算情况下MCTS_N的结果. 这说明引入的神经网络有效地指导了MCTS的探索过程，提高了MCTS的执行效率.

本文还对2个核10个回调的DAG进行了针对性的案例研究：在设置优先级时，由于ROS2采用的是在轮询点一次性放入多个就绪回调到Ω的方式，并且只有Ω为空时线程才能再次更新的机制. 优先级设置的分析可以简化为只考虑在2次更新阶段之间的时间段能够进入Ω的回调的优先级关系. 但需要注意的是，不同的优先级设置，会导致Ω中的回调在更新点时刻并不是一成不变的. 通过实验观察到的现象是：1)如果Ω中回调均为可重入模式，本文提出的方法会倾向于将执行时长最长的回调置于较高优先级，然后依次类推. 其目的是尽量防止执行时间较长的回调被阻塞. 2)如果Ω中存在属于互斥组的回调，方法会倾向于将属于回调组的且执行时间最长的回调置成较高的优先级. 尽量将可能带来后续影响的任务提前执行，以提高并行的概率，防止如图3(e)所示的互斥影响. 虽然在本文中没有进一步研究这种规律的深层原因，但这些发现可以通过形式化的方式进行进一步论证，这样不仅可以使基于强化学习的方法更具有可解释性，也可以从实践角度作为一个较好的优先级设计技巧去简化复杂的调度分析过程. 在未来工作中，我们将进一步分析和解决这个问题.

6 结 论

本文提出了一种基于强化学习和MCTS的在ROS2调度机制下的优先级配置方法，即设计回调的执行顺序. 该方法通过使用PPO训练策略价值网络指导MCTS方法寻找最佳的方案以最小化DAG形式表示的回调集合的最大完工时间. 基于这种方法，在深入研究了ROS2执行模式前提下，搭建了模拟平台，并通过实验验证了本文所提出的方法相比基准测试方法更具优势. 在未来的工作中，将专注于优化所提出的方法和进一步扩展，例如使用图卷积神经网络表征环境状态，支持单个DAG扩展到支持多个DAG的分析，优化DAG中回调组的设置.

作者贡献声明：纪东提出了算法思路和实验方案并撰写论文；魏阳杰和王义提出指导意见；李宇溪修改论文.

参 考 文 献

[1] Macenski S, Foote T, Gerkey B, et al. Robot operating system 2: Design, architecture, and uses in the wild[J]. Science Robotics 2022, 7(66): eabm6074

[2] Casini D, Blaß T, Lütkebohle I, et al. Response-time analysis of ROS 2 processing chains under reservation-based scheduling[C]// Proc of the 31st Euromicro Conf on Real-Time Systems (ECRTS). Dagstuhl, Germany: LIPIcs, 2019: 6: 1−6: 23

[3] Tang Yue, Feng Zhiwei, Guan Nan, et al. Response time analysis and priority assignment of processing chains on ROS 2 executors[C]// Proc of the 41st IEEE Real-Time Systems Symp (RTSS). Piscataway, NJ: IEEE, 2020: 231−243

[4] Bédard C, Lütkebohle I, Dagenais M. ROS2_tracing: Multipurpose low-overhead framework for real-time tracing of ROS 2[J]. IEEE Robotics and Automation Letters, 2022, 7(3): 6511−6518

[5] Blaß T, Casini D, Bozhko S, et al. A ROS 2 response-time analysis exploiting starvation freedom and execution-time variance[C]// Proc of the 42nd IEEE Real-Time Systems Symp (RTSS). Piscataway, NJ: IEEE, 2021: 41−53

[6] Kronauer T, Pohlmann J, Matthe M, et al. Latency analysis of ROS2 multi-node systems[J]. arXiv preprint, arXiv: 2101.02074, 2021

[7] Vazirani V V. Approximation Algorithms[M]. Berlin: Springer, 2001

[8] Martello S, Pisinger D, Vigo D. The three-dimensional bin packing problem[J]. Operations Research, 2000, 48(2): 256−267

[9] Li Jing, Chen Jianjia, Agrawal K, et al. Analysis of federated and global scheduling for parallel real-time tasks[C]// Proc of the 26th Euromicro Conf on Real-Time Systems(ECRTS). Piscataway, NJ: IEEE, 2014: 85−96

[10] Melani A, Bertogna M, Bonifaci V, et al. Response-time analysis of conditional dag tasks in multiprocessor systems[C]// Proc of the 27th Euromicro Conf on Real-Time Systems(ECRTS). Piscataway, NJ: IEEE, 2015: 211−221

[11] Jiang Xu, Guan Nan, Long Xiang, et al. Semi-federated scheduling of parallel real-time tasks on multiprocessors[C] //Proc of the 38th IEEE Real-Time Systems Symp (RTSS). Piscataway, NJ: IEEE, 2017: 80−91

[12] Baruah S, Bonifaci V, Marchetti-Spaccamela A. The global EDF scheduling of systems of conditional sporadic DAG tasks[C]// Proc of the 27th Euromicro Conf on Real-Time Systems(ECRTS). Piscataway, NJ: IEEE, 2015: 222−231

[13] Clausen J. Branch and bound algorithms-principles and examples [EB/OL]. Department of Computer Science, University of Copenhagen, 1999[2023-02-11].https://imada.sdu.dk/u/jbj/heuristikker/TSPtext.pdf

[14] Kohler W H, Steiglitz K. Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems[J]. Journal of the ACM, 1974, 21(1): 140−156

[15] Kwok Y K, Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors[J]. ACM Computing Surveys, 1999, 31(4): 406−471

[16] Wei Hongxing, Shao Zhenzhou, Huang Zhen , et al. RT-ROS: A real-time ROS architecture on multi-core processors[J]. Future Generation Computer Systems, 2016, 56: 171−178

[17] Saito Y, Azumi T, Kato S, et al. Priority and synchronization support for ROS[C]// Proc of the 4th IEEE Int Conf on Cyber-Physical Systems, Networks, and Applications (CPSNA). Piscataway, NJ: IEEE, 2016: 77−82

[18] Choi H, Xiang Yecheng, Kim H. PiCAS: New design of priority-driven chain-aware scheduling for ROS2[C]// Proc of the 27th IEEE Real-Time and Embedded Technology and Applications Symp (RTAS). Piscataway, NJ: IEEE, 2021: 251−263

[19] Li Jing, Agrawal K, Lu Chenyang, et al. Outstanding paper award: Analysis of global EDF for parallel tasks[C]// Proc of the 25th Euromicro Conf on Real-Time Systems(ECRTS). Piscataway, NJ: IEEE, 2013: 3−13

[20] Baruah S. Improved multiprocessor global schedulability analysis of sporadic DAG task systems[C]// Proc of the 26th Euromicro Conf on Real-Time Systems (ECRTS). Piscataway, NJ: IEEE, 2014: 97−105

[21] Fonseca J, Nelissen G, Nélis V. Improved response time analysis of sporadic DAG tasks for global FP scheduling[C]// Proc of the 25th Int Conf on Real-time Networks and Systems. New York: ACM, 2017: 28−37

[22] Fonseca J, Nelissen G, Nélis V. Schedulability analysis of dag tasks with arbitrary deadlines under global fixed-priority scheduling[J]. Real-Time Systems, 2019, 55(2): 387−432

[23] Baruah S. The federated scheduling of constrained-deadline sporadic DAG task systems[C]// Proc of the 18th Design, Automation & Test in Europe Conf & Exhibition (DATE). Piscataway, NJ: IEEE, 2015: 1323−1328

[24] Li Jing, Ferry D, Ahuja S, et al. Mixed-criticality federated scheduling for parallel real-time tasks[J]. Real-Time Systems, 2017, 53(5): 760−811

[25] Zhao Shuai, Dai Xiaotian, Bate I, et al. DAG scheduling and analysis on multiprocessor systems: Exploitation of parallelism and dependency[C]// Proc of the 41st IEEE Real-Time Systems Symp (RTSS). Piscataway, NJ: IEEE, 2020: 128−140

[26] Pathan R, Voudouris P, Stenström P. Scheduling parallel real-time recurrent tasks on multicore platforms[J]. IEEE Transactions on Parallel and Distributed Systems, 2017, 29(4): 915−928

[27] He Qingqiang, Guan Nan, Guo Zhishan. Intra-task priority assignment in real-time scheduling of dag tasks on multi-cores[J]. IEEE Transactions on Parallel and Distributed Systems, 2019, 30(10): 2283−2295

[28] He Qingqiang, Lv Mingsong, Guan Nan. Response time bounds for DAG tasks with arbitrary intra-task priority assignment[C]// Proc of the 33rd Euromicro Conf on Real-Time Systems (ECRTS). Dagstuhl, Germany: LIPIcs, 2021: 8: 1−8: 21

[29] Kasahara H, Narita S. Practical multiprocessor scheduling algorithms for efficient parallel processing[J]. IEEE Transactions on Computers, 1984, 33(11): 1023−1029

[30] Kwok Y K, Ahmad I. Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors[J]. IEEE Transactions on Parallel and Distributed Systems, 1996, 7(5): 506−521

[31] Mao Hongzi, Mohammad A, Ishai M, et al. Resource management with deep reinforcement learning[C]// Proc of the 15th ACM Workshop on Hot Topics in Networks. New York: ACM, 2016: 50−56

[32] Chen Xinyun, Tian Yuandong. Learning to perform local rewriting for combinatorial optimization[C]// Proc of the 33rd Int Conf on Neural Information Processing Systems. New York: ACM, 2019: 6281–6292

[33] Kober J, Bagnell J A, Peters J. Reinforcement learning in robotics: A survey[J]. The International Journal of Robotics Research, 2013, 32(11): 1238−1274

[34] Lee H, Cho S, Jang Y, et al. A global DAG task scheduler using deep reinforcement learning and graph convolution network[J]. IEEE Access, 2021, 9: 158548−158561

[35] Mao Hongzi, Schwarzkopf M, Venkatakrishnan S B, et al. Learning scheduling algorithms for data processing clusters[C]// Proc of the 33rd ACM Special Interest Group on Data Communication. New York: ACM, 2019: 270−288

[36] Macenski S, Foote T, Gerkey B, et al. ROS 2 Foxy Fitzroy [CP/OL]. 2020[2022-11-05].https://docs.ros.org/en/foxy/

[37] Foote T, Scott K. ROS community metrics report 2020[EB/OL]. 2021 [2022-11-05]. http://download.ros.org/downloads/metrics/metrics-report-2021-07.pdf

[38] Bello I, Pham H, Le Q V, et al. Neural combinatorial optimization with reinforcement learning[J]. arXiv preprint, arXiv: 1611.09940, 2016

[39] Coulom R. Efficient selectivity and backup operators in Monte-Carlo tree search[C] //Proc of the 5th Int Conf on Computers and Games. Berlin: Springer, 2006: 72−83

[40] Saqlain M, Ali S, Lee J Y. A Monte-Carlo tree search algorithm for the flexible job-shop scheduling in manufacturing systems[J/OL]. Flexible Services and Manufacturing Journal, 2022[2023-02-11].https://doi.org/10.1007/s10696−021-09437−4

[41] Hu Zhiming, Tu J, Li Baochun. Spear: Optimized dependency-aware task scheduling with deep reinforcement learning[C]// Proc of the 39th IEEE Int Conf on Distributed Computing Systems (ICDCS). Piscataway, NJ: IEEE, 2019: 2037−2046

[42] Arabnejad H, Barbosa J G. List scheduling algorithm for heterogeneous systems by an optimistic cost table[J]. IEEE Transactions on Parallel and Distributed Systems, 2014, 25(3): 682−694

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

[44] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint, arXiv: 1707.06347, 2017

[45] Brockman G, Cheung V, Pettersson L, et al. OpenAI gym[J]. arXiv preprint, arXiv: 1606.01540, 2016

Ji Dong, born in 1989. PhD. Student member of CCF. His main research interests include timing analysis in embedded systems, machine learning and its various applications in renewable energy.

纪　东, 1989 年生. 博士. CCF 学生会员. 主要研究方向为嵌入式系统时序分析和机器学习在可再生能源领域的应用.

Wei Yangjie, born in 1978. PhD, professor, PhD supervisor. Member of CCF. Her main research interests include speech and audio signal processing, image processing, and computer vision.

魏阳杰, 1978 年生. 博士, 教授, 博士生导师. CCF 会员. 主要研究方向为语音和音频信号处理、图像处理和计算机视觉.

Li Yuxi, born in 1990. PhD, lecturer. Member of CCF. Her main research interests include cyberspace security and Internet of things security.

李宇溪, 1990 年生. 博士, 讲师. CCF 会员. 主要研究方向为网络空间安全和物联网安全.

Wang Yi, born in 1961. PhD, professor, PhD supervisor. Member of CCF. His main research interests include modeling and validation of timed systems, timing analysis of embedded software on multicores and multiprocessor real-time scheduling.

王　义, 1961 年生. 博士, 教授, 博士生导师. CCF 会员. 主要研究方向为实时系统的建模和验证、多核嵌入式软件的时序分析和多处理器实时调度.