HierRL: Hierarchical Reinforcement Learning for Task Scheduling in Distributed Systems

研究背景

分布式系统在大规模任务处理中的重要性不言而喻。在这个领域，Ray作为一种先进的分布式系统，受到了广泛的关注。然而，Ray目前采用的两层任务调度机制（本地调度器和全局调度器）虽然具有一定的灵活性和可扩展性，但其性能在某些应用场景中表现较为有限。比如：全局调度器尝试将任务分配给预计等待时间最短的节点，但任务运行时间和到达时间具有随机性，难以精确估计。现有基于遗传算法、蚁群算法的启发式方法可用于特定任务，但在分布式系统中设计和实现代价较高，适用性有限。

研究贡献

- **目标**：针对Ray的两层调度问题，提出一种基于**分层强化学习**的解决方案（HierRL）。
- 方法设计：
  - 构建两层调度问题的优化模型，以最小化任务完成时间为目标。
  - 引入分层强化学习：
    - **高层代理**负责跨节点的任务分配。
    - **低层代理**在节点内选择任务执行顺序。
    - 分别为高层和低层设计独立的状态空间、动作空间和奖励函数。
  - 提出分层策略学习过程，联合训练高低层代理。

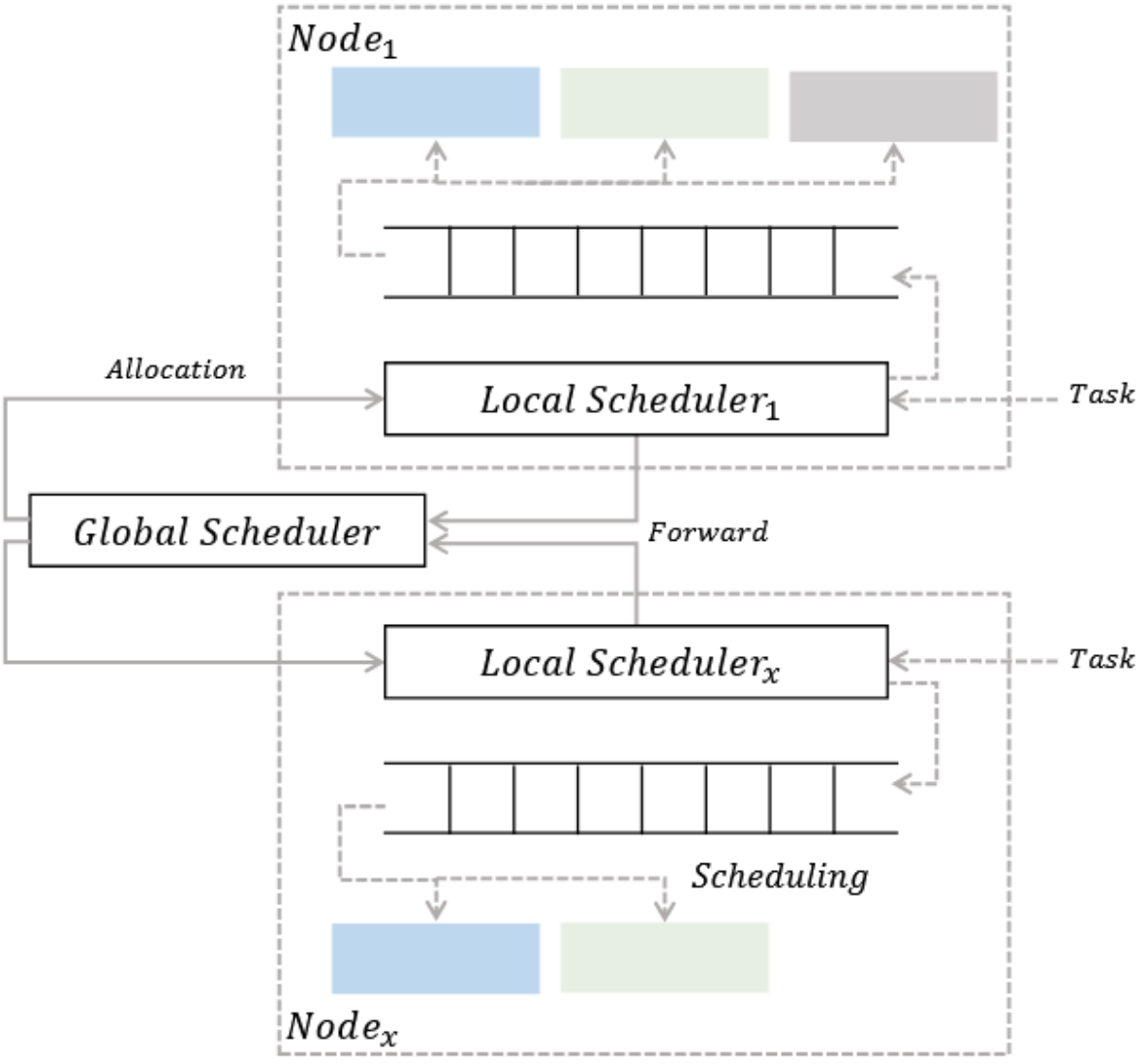
背景知识

Ray的调度框架

- **框架结构**：

- Ray 的调度框架由一个**全局调度器**和若干**局部调度器**组成。
- 每个局部调度器与一个节点对应，并关联一个任务队列，用于存储待本地调度的任务。

- 调度流程：
  - 任务最初被提交到局部调度器。
  - 局部调度器判断任务是否可以本地调度：
    - 如果节点超载（如资源不足或队列超出阈值），任务会被转移到全局调度器处理。
    - 否则，任务被放入局部队列，等待调度。
  - 全局调度器在接收到任务后，根据一定原则将任务分配给适当的节点。



问题建模

1. 系统描述
    - 系统包含 I个节点，每个节点具有一定的资源（如 CPU、内存、磁盘资源）。
    - 每个节点有一个队列，可以容纳一定数量的任务。
2. 任务与资源表示：
    - 节点集合用 $I = \{1, 2, ..., I\}$表示；
    - 任务集合用 $J = \{1, 2, ..., J\}$表示；
    - 每个任务 j 在时间 λ_j从节点i提交，对应资源需求为 $r_{jk}, k = 1, 2, 3$;
    - $C_{ik}, k = 1, 2, 3$表示第i个节点的资源量；
3. 变量定义
    - x_{ij}^t：是否在时间t时，任务j被全局调度器分配到节点i（1 表示是，0 表示否）。
    - y_{ij}^t：是否在时间t时，任务j被节点i执行（1 表示是，0 表示否）。
    - z_{ij}^t：是否在时间t时，任务j被节点i的队列中缓存（1 表示是，0 表示否）。
    - u_{ik}^t：在时间t时，第i个节点的可用资源量；
    - w：最后一个任务完成的时间；
4. 优化目标：**最小化所有任务的完工时间**（makespan），即最小化最后一个任务完成的时间 w
5. 问题表述：
    - 在调度过程中，任务可以：
        1. 被直接调度到节点并执行。
        2. 放入节点的本地队列缓存。
        3. 超过队列阈值或资源不足时，转发至全局调度器处理。

优化模型

目标：最小化所有任务的完成时间

$$Minw$$

约束条件

- 任务完成时间约束：w必须大于任何任务完成的时间t；

$$w \geq t \times t_{ij}^{t}$$

- 任务分配约束：每个任务 j在整个时间范围内只能被分配一次，无论本地缓存还是全局调度；
- 任务执行约束：每个任务必须被执行一次；
- 资源限制约束：确保任务j的资源消耗 r_{jk}不超过节点i的总资源；
- 任务缓冲和调度的时间顺序约束：任务只有在被缓冲或由全局调度分配给节点i后，才能在节点i上执行；

问题性质：NP-hard问题；

研究方法

高层调度

1. 简介：
    a. **全局调度器**（对应高层代理H-Agent）负责从局部调度器接收任务 j，并将任务分配到 I 个局部调度器中的一个。
    b. 模型：构建了一个多层感知机模型，以参数 θ^h表示；
    c. 动作 a_t^h：从集合 $\{1, 2, ..., I\}$ 中选择一个局部调度器；
    d. 输入状态 s_t^h：通过MLP模块转换为嵌入向量，再通过另一个MLP模块计算 $Q^h(s_t^h, a_t^h; \theta^h)$
2. 状态表示 s_t^h： $s_t^h = \{c_t^0, l_t^0, n_t^0, \ldots, c_t^I, l_t^I, n_t^I\}$
    a. 剩余资源：节点i在时间t的剩余资源量，包括CPU、内存、磁盘等；
    b. 任务队列长度：节点i在时间t的任务队列长度 l_t^i；
    c. 全局分配任务数：节点i到时间t位置从全局调度器接收的任务总数 n_t^i；
3. 奖励设计：$r_t^h = \alpha \frac{c_t^{a_t^h}}{\sum_{i \in \mathcal{I}} c_t^i} - \beta \frac{l_t^{a_t^h}}{\sum_{i \in \mathcal{I}} l_t^i} - \gamma \frac{n_t^{a_t^h}}{\sum_{i \in \mathcal{I}} n_t^i}$
    - 剩余资源分布（第一项）：鼓励均衡使用各节点资源。
    - 任务队列长度（第二项）：鼓励减少任务队列长度，平衡负载。

- 分配任务总数（第三项）：鼓励任务在所有节点间均匀分布。
- 最终奖励 r_t^f：在整个调度过程结束时，若所有任务的完成时间小于基线，则奖励为正，否则为负。

4. 网络训练：使用DDQN；

低层调度

1. 简介：
    a. 负责从任务队列中选择一个任务并执行。每次从队列头部提取一定数量的任务 m，然后决定选择哪个任务执行
    b. 所有低层调度网络共享网络参数和重放缓冲区；
2. 状态表示
    a. 动作空间：低层调度代理的动作为 a_t^l，取值范围为 $\{1, 2, \ldots, m\}$，表示从 m 个候选任务中选择的任务编号。
    b. 状态表示： $s_t^l = \{\langle \sigma_0, \lambda_0 \rangle, \ldots, \langle \sigma_m, \lambda_m \rangle\}$，包含候选任务的资源需求和提交时间；
3. 奖励设计： $r_t^l = \eta_1 f_1(\sigma) + \eta_2 f_2(\lambda)$，基于评分的机制进行设计；
    - 第一项：与资源需求相关的评分函数，评估任务资源利用率；
    - 第二项：与提交时间相关的评分函数，鼓励更早处理任务；
4. 损失函数与更新：基于Q-learning更新；

分层策略学习算法

输入：节点集、任务流；

输出：优化后的高层和低层策略 (θ^h, θ^l)

步骤：

1. 初始化：初始化高层和低层经验回放存储器 (D^h, D^l)和神经网络参数 (θ^h, θ^l)；
2. 训练过程（重复N个回合）
    a. 初始化分布式系统；
    b. 当任务流没有结束时：
        i. 任务j从节点i生成：
            1. 如果节点i超载，将任务提交到全局调度器；

2. 反之：将任务存储在节点i的任务队列中；

ii. 如果全局调度器中存在任务：

1. 获取高层状态；
2. 使用探索策略选择节点g；

iii. 否则，设置g为None；

iv. 对每个节点：

1. 获取节点i的底层状态；
2. 使用探索策略选择任务 a_i；
3. 执行选中的任务，获得新的低层状态和奖励；

v. 经验存储

1. 获取新的高层状态和高层奖励；
2. 对每个节点：存储低层经验到低层经验回放区；
3. 存储高层经验到高层经验回放区；

vi. 策略更新

1. 如果满足更新条件；
    a. 从高层与低层经验回放区中分别采样一个批次；
    b. 分别执行梯度下降更新高层和低层网络参数；

3. 返回优化后的策略参数；

实验设置

- 模拟环境：使用CloudSim作为仿真工具；
    - CloudSim 是一种模拟和仿真分布式系统的工具，支持数据中心、调度和分配策略等建模。
- 数据集与验证
    - 使用阿里巴巴真实生产数据集 cluster-trace-v2017；
    - 包含在线服务和批处理任务的分配信息，约1300台机器在12小时内的任务记录。为简化模型，任务数据进行了缩放。

    - 构建基于 CloudSim 的自底向上的调度框架，其中全局调度器可获取所有局部调度器的状态和任务信息。
- 实验平台：
    - 硬件：使用配备 i7-11700F CPU、RTX3060 GPU 和 16GB RAM 的计算机。
    - 软件：TensorFlow；
- 性能指标：表示 Ray 算法与 HierRL 算法在任务完成时间上的相对改善程度。

$$rate_{\Delta m} = \frac{m(R) - m(H)}{m(R)}$$

- $m(R)$：Ray 算法的 makespan（任务完成时间）。
- $m(H)$：HierRL 算法的 makespan。