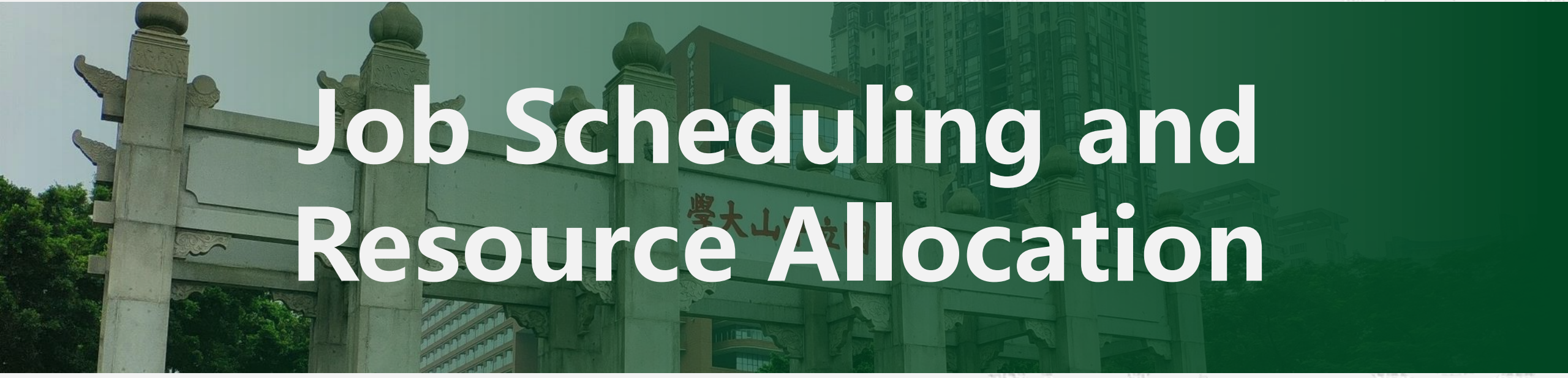
中山大學
SUN YAT-SEN UNIVERSITY



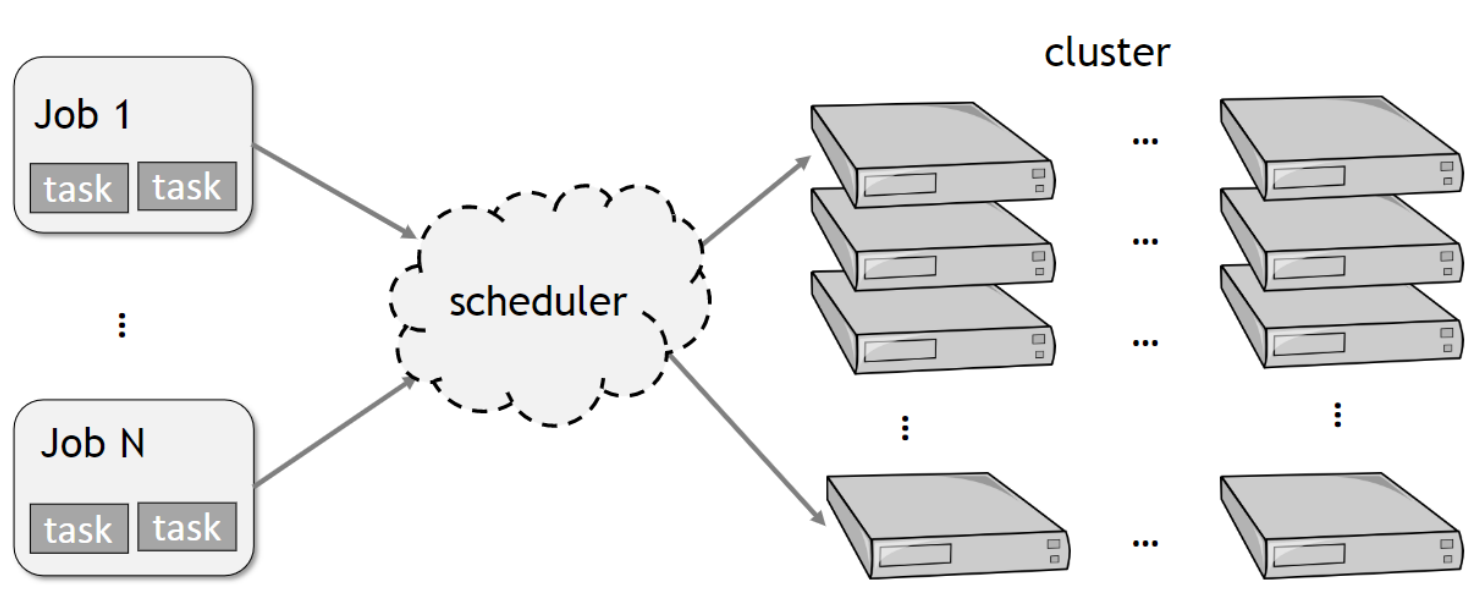
Job Scheduling and Resource Allocation

汇报人：肖霖畅

目录

CONTENTS

任务调度和资源分配

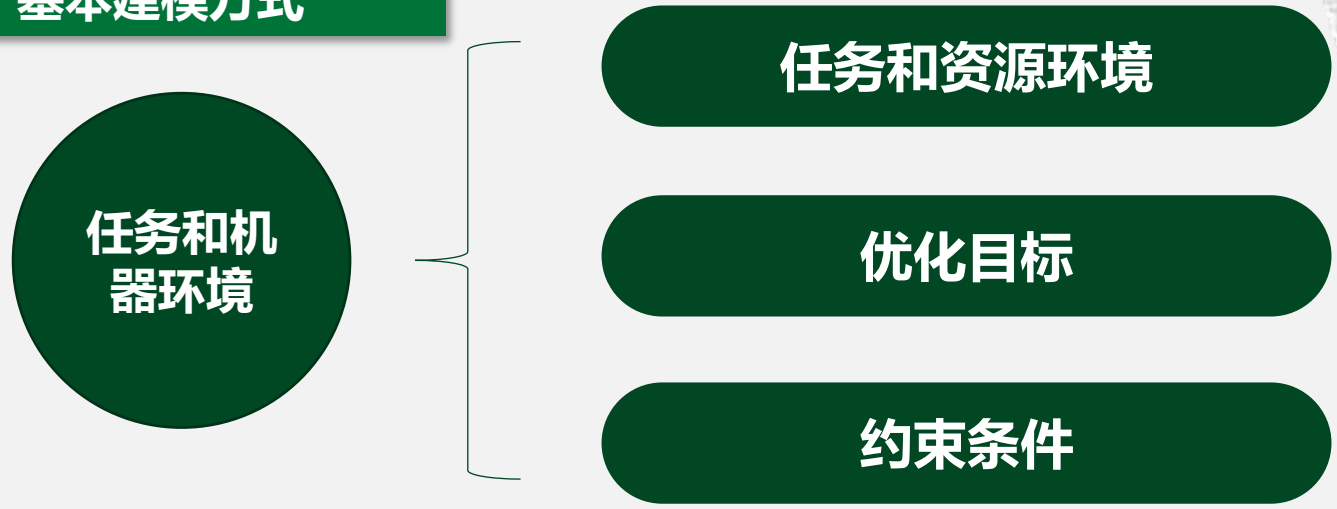


- 在目前的实际环境中，任务调度和资源分配很多时候耦合在一起。
- 一般来说，一个任务调度和资源分配问题可以表示为：一系列任务需要分配到一些机器上，满足某些约束，并且优化一个特定的目标函数。

广泛的问题建模方式

- 一个CPU需要处理不断到达的程序。如何安排程序处理的顺序，最小化程序的平均处理时间（任务到达至完成的时间）。
- （一系列建模方式）……
- 考虑一个生产不同类型产品的工厂。不同产品需要不同机器上的不同处理时间，需要首先在机器1处理，然后是机器2，最后机器3，不同产品在不同机器上处理时间不同。工厂会收到订单，每个订单有额定的完成时间，必须在此之前完成。如何安排机器的生产顺序，使得工厂完成尽可能多的订单。

基本建模方式



基本建模方式

- **任务和机器环境**
  - 1：单机问题
  - P：等价并行环境（每个资源完全相等）
  - Q：均匀相关环境（每台机器速度不同，不同任务在同个机器执行速度**一致**）
  - R：不相关环境（每台机器速度不同，不同任务在同个机器执行速度**不一致**）
  - O：开放工厂（任务可细分为多个**执行时间相同的操作**，操作可以**按任意顺序完成**）
  - J：任务工厂（任务可细分为多个**执行时间相同的操作**，操作必须**在前驱操作完成后执行**）
  - F：流水线工厂（任务可细分为多个**执行时间不同的操作**，操作必须**在前驱操作完成后执行**）

- **优化目标**
  - 平均完成时间 （JCT）
  - 所有任务完成跨度（Makespan）
  - （执行性能、服务价格等等……）

- **约束条件**
  - 支持抢占？任务/操作在线到达？任务存在DDL？任务中操作的组织形式（DAG/Tree/Chain）？
  - 任务存在资源需求？ 机器存在资源供给限额？带宽限额?
  - 机器是否分级（云/边缘/移动端）？任务是否有特殊的限制（AI/响应式）？

基本建模方式

Machine environment α

type : 1 | P | Q | R | O | F | J

Constraints β

number of jobs : Ø | $n=2$ | $n=3$ | $n=k$

precedence relation : Ø | prec | chains | tree | intree | outtree | opposing forest | sp-graph | bounded height | level order | interval order | quasi-interval order | over-interval order | Am-order | DC-graph | 2-dim partial order | k-dim partial order

time lags : Ø | $l=1$ | l | $l \leq 0$ | $l > 0$ | l_{ij} | $l_{ij} \leq 0$ | $l_{ij} > 0$

release time : Ø | r_j | online-r$_j$

preemption : Ø | pmtn

due date : Ø | $d_j = d$

processing times : Ø | $p_j = 1$ | $p_j = p$

batching : Ø | s-batch | batch(∞) | batch(b)

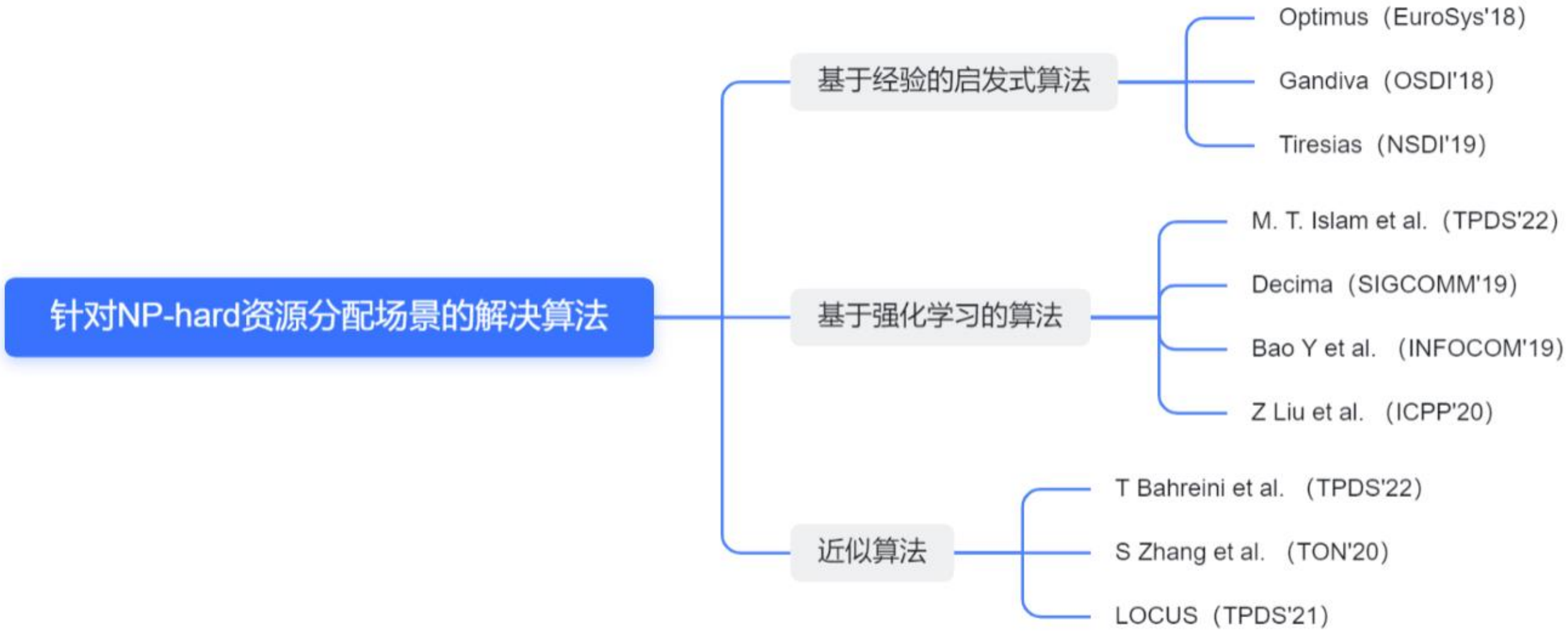
Objective function γ

Objective function : $C_{\max}$ | $C_{\min}$ | $\sum C_j$ | $\sum w_j C_j$ | $L_{\max}$ | $\sum U_j$ | $\sum w_j U_j$ | $\sum T_j$ | $\sum w_j T_j$

http://schedulingzoo.lip6.fr/

基本建模方式

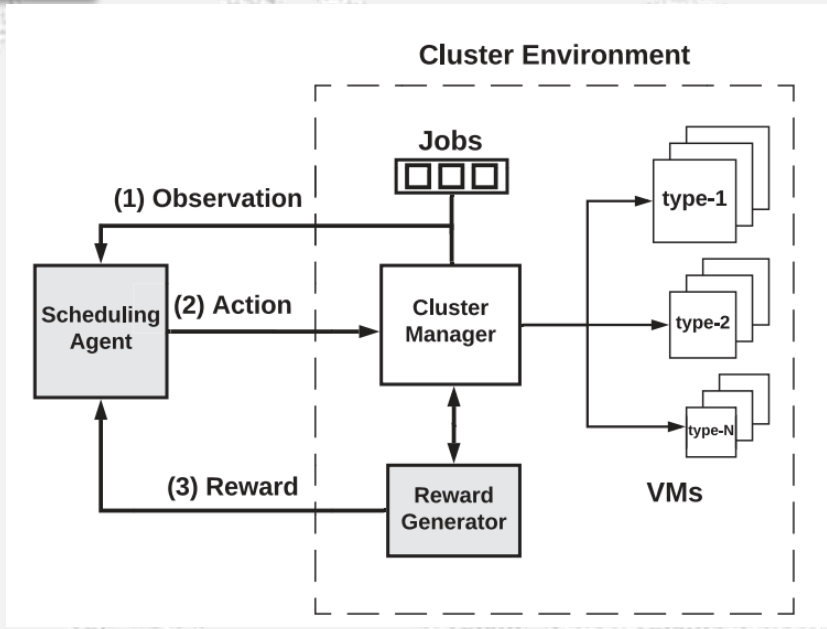
- **共有的挑战**
  - 随着场景定义的复杂，最优化问题变成NP-hard问题（在有限的时间内无法在可行解集中找到最优解）
- **任务调度问题中NP-hard问题举例**
  - 单机 + 任务存在发布时间 + 平均作业完成时间 $1 \mid\mid r_j \mid\mid \sum C_j$
  - 单机 + （任务存在发布时间 + 任务允许抢占) + 平均加权作业完成时间 $1 \mid\mid r_j, pmtm \mid\mid \sum w_j C_j$
  - 多机等价并行 + 最终任务完成时间 $P_m \mid\mid C_{max}$
- **解决方法**
  - 非NP-hard问题：确定方法求解【e.g. 线性规划求解】
  - **NP-hard问题：**近似方法求解【e.g. 基于经验的启发式算法、近似算法、强化学习等】


针对NP-hard资源分配场景的解决算法
基于经验的启发式算法
Optimus（EuroSys'18）
Gandiva（OSDI'18）
Tiresias（NSDI'19）
基于强化学习的算法
M. T. Islam et al.（TPDS'22）
Decima（SIGCOMM'19）
Bao Y et al.（INFOCOM'19）
Z Liu et al.（ICPP'20）
近似算法
T Bahreini et al.（TPDS'22）
S Zhang et al.（TON'20）
LOCUS（TPDS'21）

Performance and Cost-Efficient Spark Job Scheduling Based on Deep Reinforcement Learning in Cloud Computing Environments[1]

场景定义

- 任务和机器环境
  - 不相关环境（机器资源异构）：每台VM速度不同，不同任务在同个VM执行速度不一致
  - 每个Job包含多个异质的Executors
- 优化目标
  - **VM资源的价格**和**平均Job完成时间**加权求和
- 约束条件
  - Job在线随时到达，不允许抢占
  - Executor的资源需求与VM资源供给存在约束，Executor不能切割
- 决策场景
  - Job到来时存放在等待池中
  - 每个时间片依次从等待池中决策任务的Executor放置的VM位置



$$\sum_{k\in\omega}(e^k_{cpu}\times x_{ki}) \le vm^i_{cpu} \qquad \forall i \in \delta$$

$$\sum_{k\in\omega}(e^k_{mem}\times x_{ki}) \le vm^i_{mem} \qquad \forall i \in \delta,$$

[1]Islam M T, Karunasekera S, Buyya R. Performance and Cost-Efficient Spark Job Scheduling Based on Deep Reinforcement Learning in Cloud Computing Environments[J]. IEEE Transactions on Parallel and Distributed Systems, 2021, 33(7): 1695-1710.

Performance and Cost-Efficient Spark Job Scheduling Based on Deep Reinforcement Learning in Cloud Computing Environments[1]

问题定义

$$\min_{x_{ki}} = (\beta \times Cost_{total} + (1-\beta) \times Avg_T)$$

$$s.t. \quad Cost_{total} = \sum_{i \in \delta} (vm^i_{price} \times vm^i_T)$$

$$Avg_T = (\sum_{j \in \psi} job^j_T)/M$$

$$x_{ki} \in \{0, 1\}$$

$$\sum_{i \in \delta} x_{ki} = 1$$

$$\sum_{k \in \omega} (e^k_{cpu} \times x_{ki}) \leq vm^i_{cpu}$$

$$\sum_{k \in \omega} (e^k_{mem} \times x_{ki}) \leq vm^i_{mem}$$

挑战

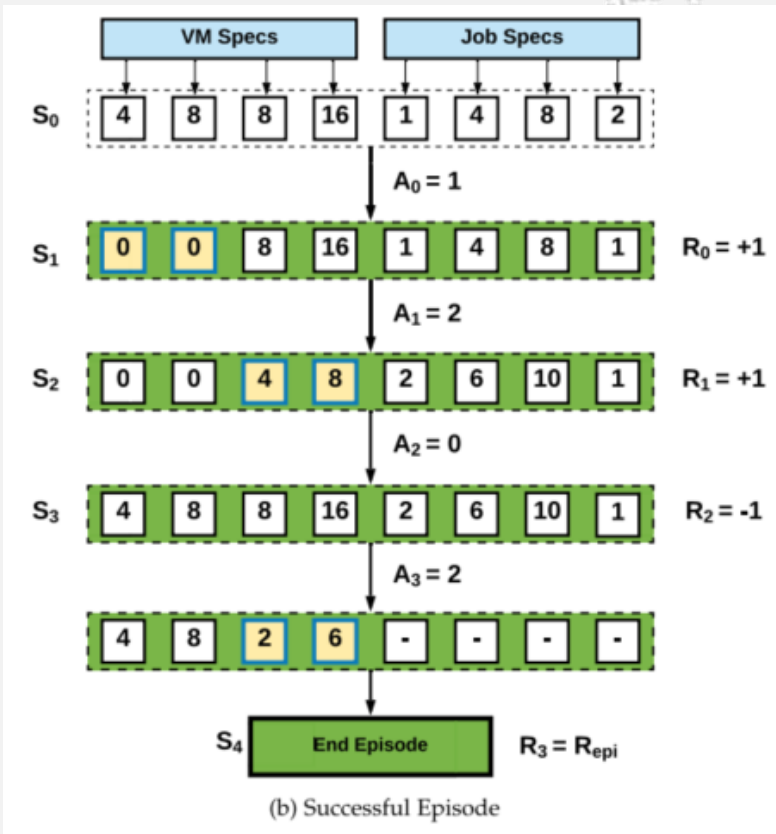
- 混合整数线性规划问题（MILP）
  - NP-hard Problem
- 场景在线动态变换
  - 工作负载动态到来的，任务的到达时间、**执行时间**、计算需求都是无法提前获知且变化巨大的
  - 传统简单的方法缺少对动态场景的考虑

基于强化学习的方法为解决此类复杂问题提供途径

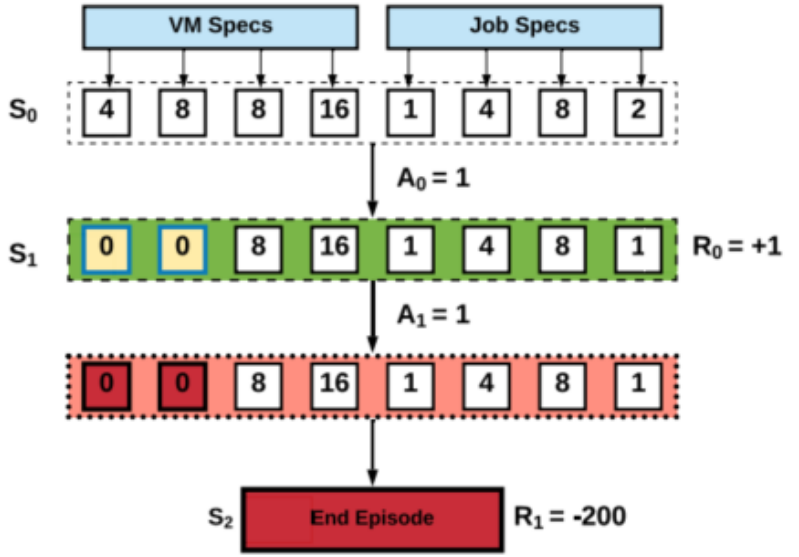
Performance and Cost-Efficient Spark Job Scheduling Based on Deep Reinforcement Learning in Cloud Computing Environments

MDP设计

- State
  - VM状态信息 + Job状态信息
- Action
  - 调度VM序号 + 不调度信号
- Reward
  - Episode Reward（基于此Episode的优化目标值与记录中最差的优化目标值）
  - Immediate Reward



(b) Successful Episode



(a) Failed Episode

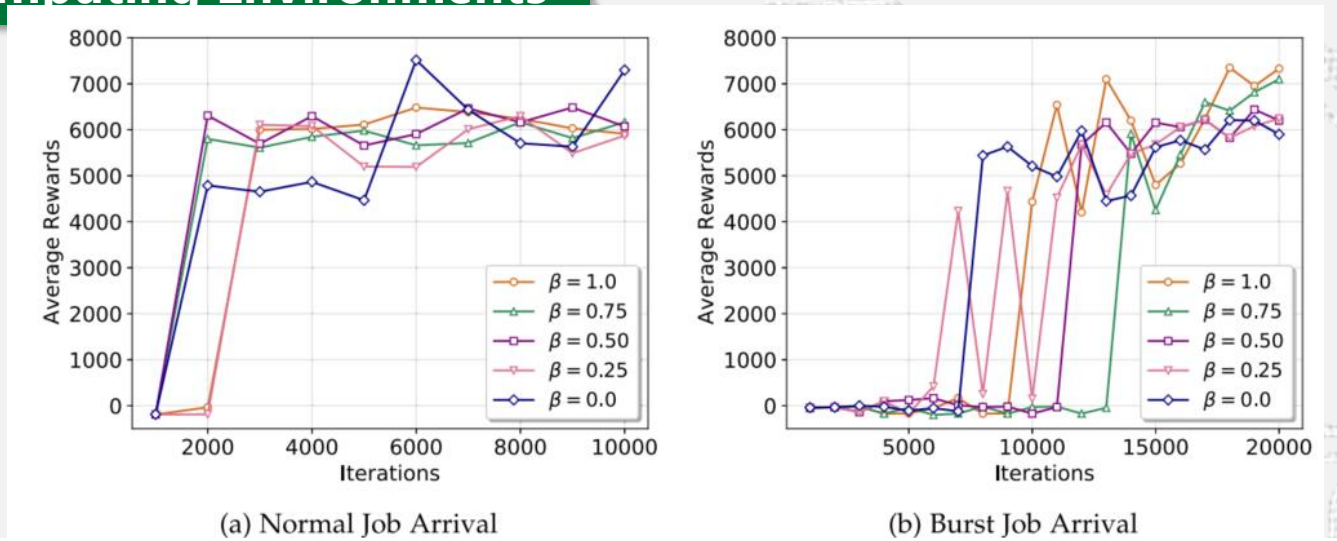
Performance and Cost-Efficient Spark Job Scheduling Based on Deep Reinforcement Learning in Cloud Computing Environments

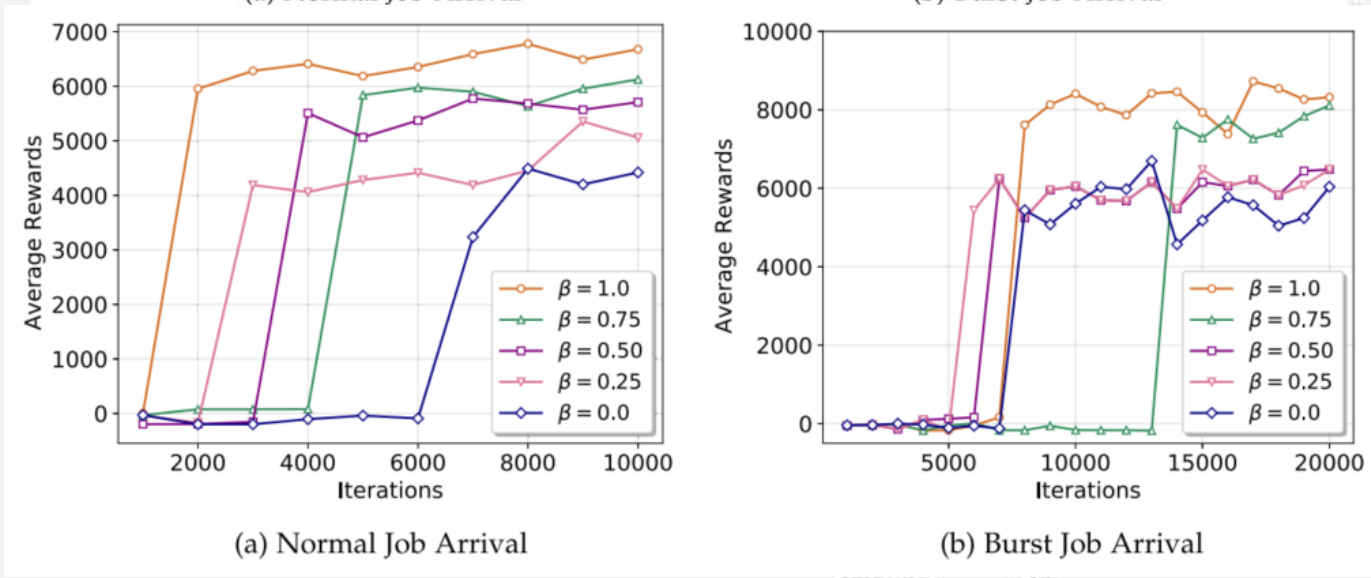
智能体实现与实验

- 实验数据来源
  - **Workload:** Facebook Hadoop Workload Trace[1]
  - **Resources Price:** AWS EC2
- 强化学习智能体算法实现
  - DQN
  - REINFORCE

- β -> 0: 优化平均完成时间，β -> 1: 优化VM租用成本
- Burst: Workload集中在一起到来

[1]https://github.com/SWIMProjectUCB/SWIM/wiki/Workloads-repository



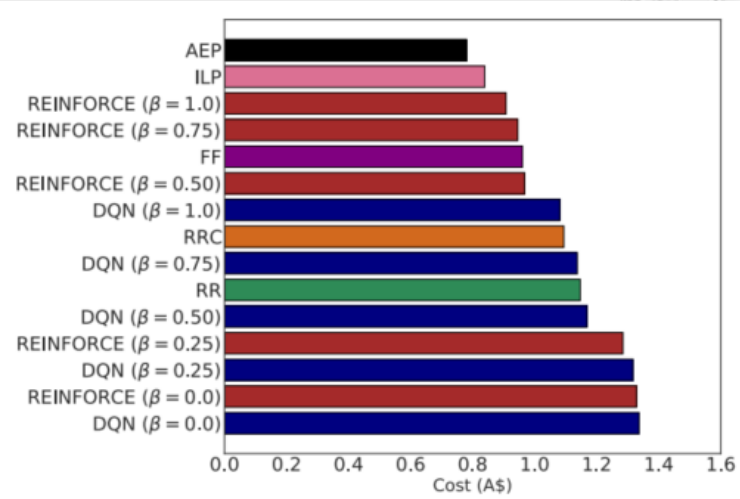


Performance and Cost-Efficient Spark Job Scheduling Based on Deep Reinforcement Learning in Cloud Computing Environments

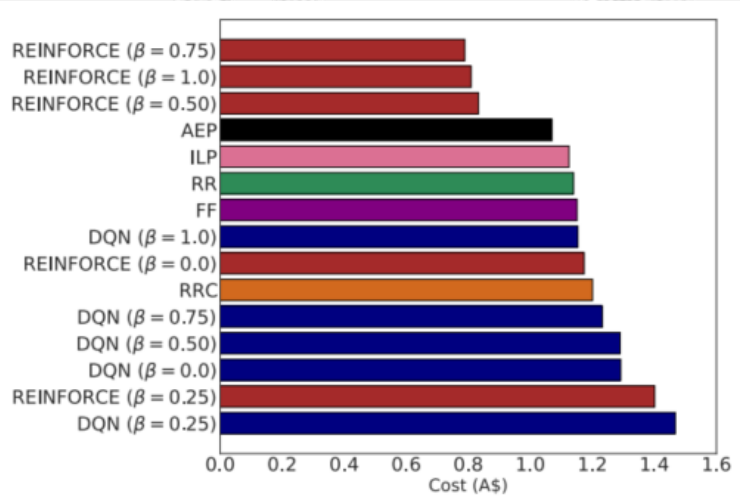
智能体实现与实验

- 收敛性
  - 上面是DQN
  - 下面是REINFORCE
- VM总Cost测试结果
- 任务平均完成时间测试结果

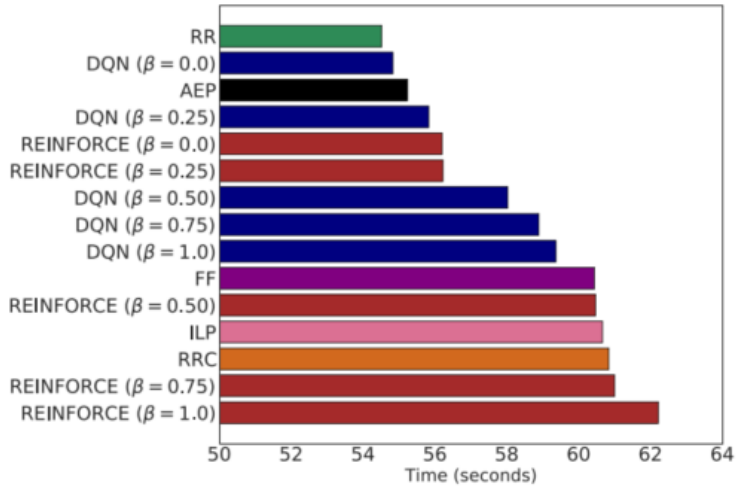
- 对比算法
  - RR：Round Robin
  - RRC：Round Robin Consolidate
  - FF：First Fit
  - ILP：Integer Linear Programming
  - AEP：Adaptive Executor Placement



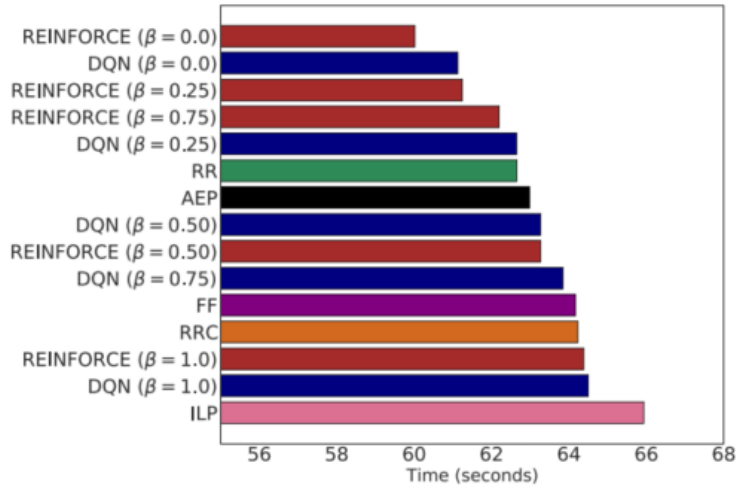
(a) Normal Job Arrival



(b) Burst Job Arrival



(a) Normal Job Arrival



(b) Burst Job Arrival

Performance and Cost-Efficient Spark Job Scheduling Based on Deep Reinforcement Learning in Cloud Computing Environments

优势

- 经过多轮训练效果得到明显提升
- 对动态场景适配性好
  - Brust Job Arrival
- 不需要很多先验知识（e.g. 任务执行时间），状态和动作设计比较简单

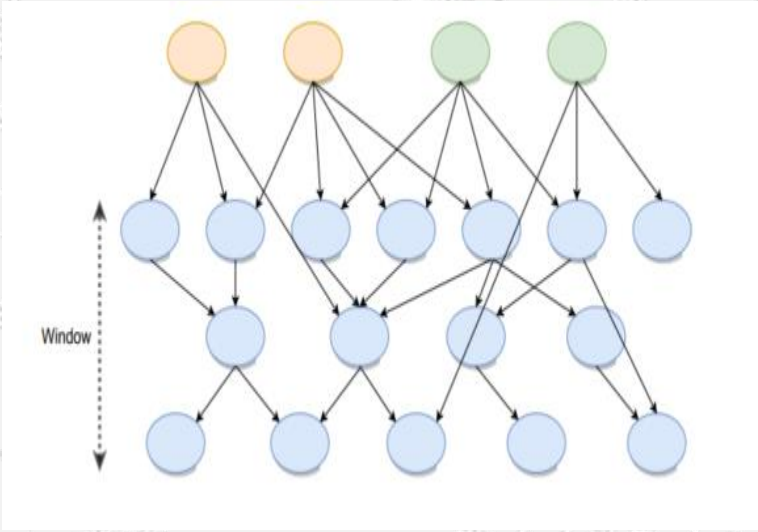
不足

- 训练轮数和训练时间太长，不适合大规模场景
- 状态空间和动作空间的设计不适合大规模的集群（决策速度会变慢）

READYS: A Reinforcement Learning Based Strategy for Heterogeneous Dynamic Scheduling

场景定义

- 任务和机器环境
  - 不相关环境（机器资源异构）：包括两种计算资源（CPU+GPU），每种计算资源速度不同，同个任务在不同计算资源执行速度不一致
- 优化目标
  - **每个Job所有Task的最终完成时间**
- 约束条件
  - 每个Job是由多个子Task构成，Task间依赖可用**DAG图**表示
  - 每个Job中子Task在线随时到达，且非同时到达
- 决策场景
  - 子Task到来时进入等待状态
  - 当计算资源存在空闲时，决策某个等待状态的子Task于其中执行



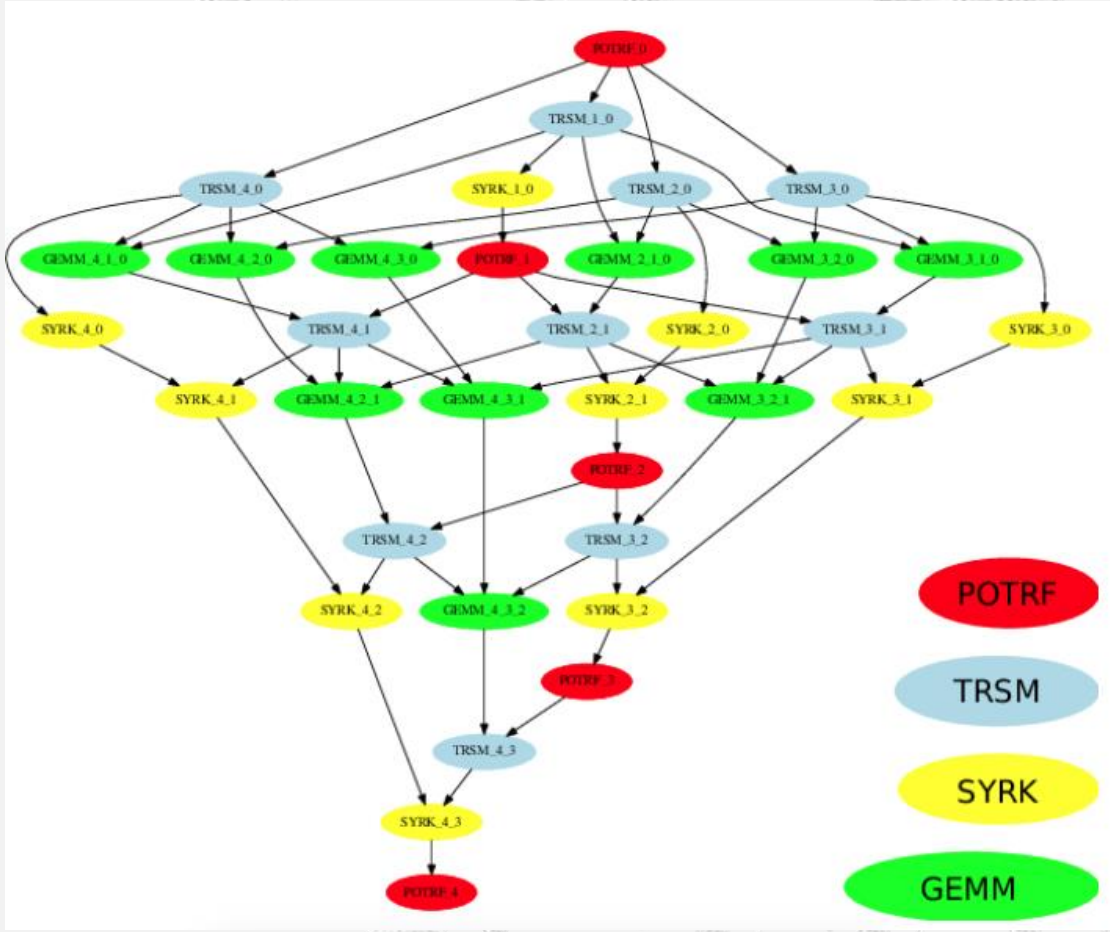
N. Grinsztajn, O. Beaumont, E. Jeannot and P. Preux, "READYS: A Reinforcement Learning Based Strategy for Heterogeneous Dynamic Scheduling," 2021 IEEE International Conference on Cluster Computing (CLUSTER), 2021, pp. 70-81

READYS: A Reinforcement Learning Based Strategy for Heterogeneous Dynamic Scheduling

挑战

- Task是动态到来的，任务的到达时间、**执行时间**、计算需求都是无法提前获知且变化巨大的
- DAG图层面
  - 非完全信息：每个时刻只能获得DAG图的一部分Task信息
  - 一个Job的DAG完整图比较复杂，包含大量任务、复杂依赖
  - 决策时需要处理大量的DAG图原始信息，可能影响决策速度

利用图卷积网络学习DAG特征
利用强化学习优化在线决策



READYS: A Reinforcement Learning Based Strategy for Heterogeneous Dynamic Scheduling

MDP设计

- State
  - 图卷积网络（GCN）编码

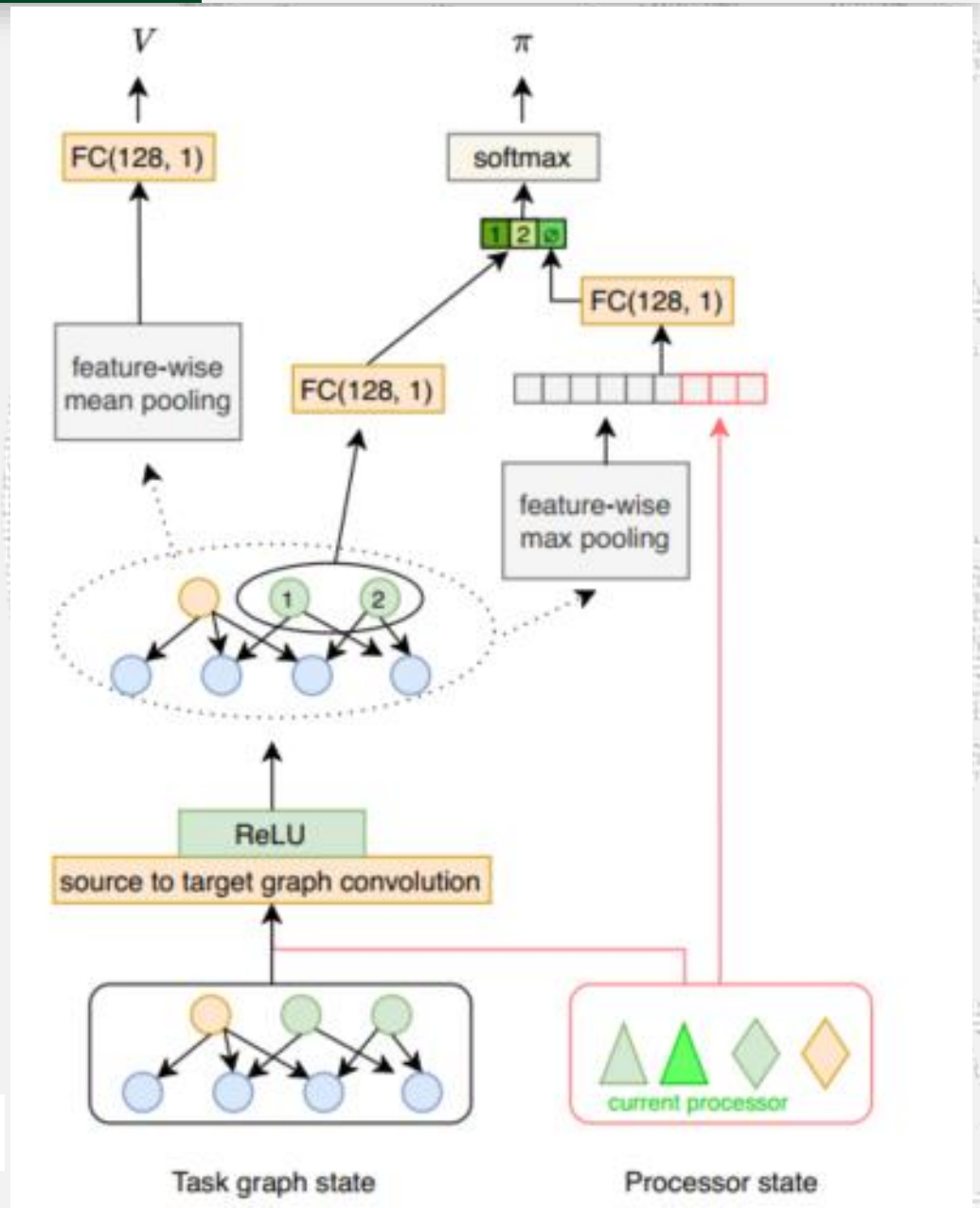
$$\hat{X}_i = [|\mathcal{S}(i)|, |\mathcal{P}(i)|, type(i), ready(i), F(i)],$$

$$\bar{F}(i) = \left(type(i) + \sum_{c \in S(i)} \frac{\bar{F}(c)}{|P(c)|} \right) \text{ and } F(i) = \frac{\bar{F}(i)}{\bar{F}(0)}$$

$$H^{(l+1)} = \varphi\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right). \quad H^{(0)} = [\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_n].$$

- Action
  - 可调度Task序号 + 不调度
- Reward
  - 完成整个DAG后才给出的Episode Reward
  - 启发式算法HEFT作为基准

$$R_a(s, s') = \begin{cases} 0 \text{ if } s' \text{ is non terminal,} \\ R(\text{makespan}) \text{ otherwise.} \end{cases} \quad R(\text{makespan}) = \frac{\text{makespan(HEFT)} - \text{makespan}}{\text{makespan(HEFT)}}.$$



READYS: A Reinforcement Learning Based Strategy for Heterogeneous Dynamic Scheduling

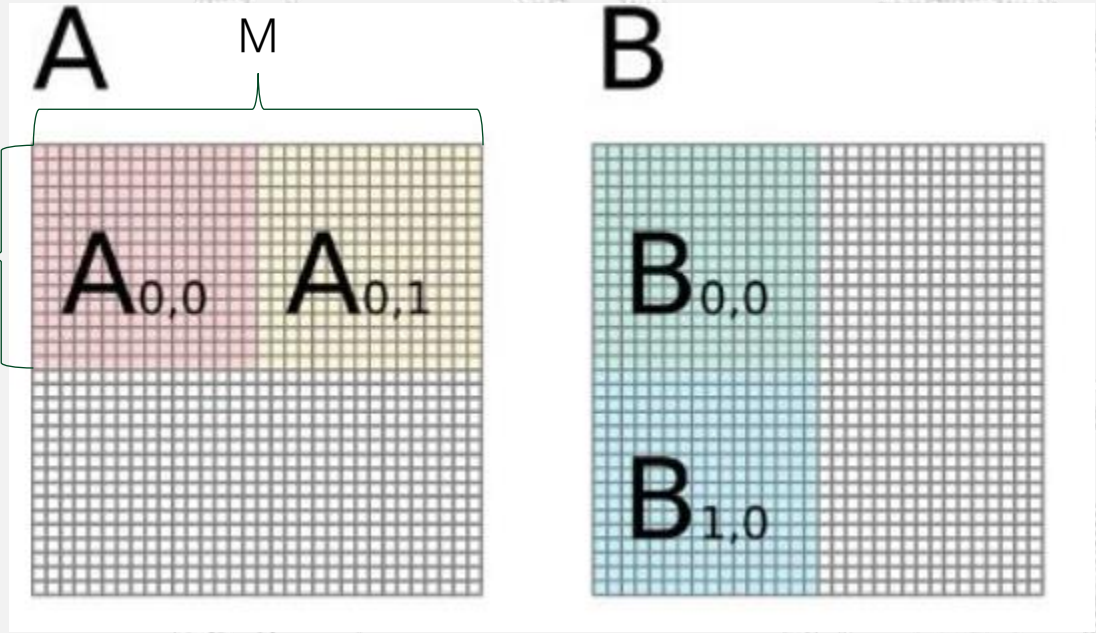
智能体实现与实验

- 强化学习智能体实现
  - Actor-Critic（A2C）
- 实验数据来源
  - 三种矩阵分解Job：Cholesky/QR/LU
  - 随机生成每个Task i在处理器p上的实际执行时间

$$d(i,p) = \max\left[0, \mathcal{N}\left(E(i,p), \sigma E(i,p)\right)\right],$$

- 对比算法
  - **（静态调度）**HEFT：关键路径节点Task优先调度，需要提前知道全图信息。
  - **（动态调度）**MCT（Minimum Completion Time）：节点Task执行时间短优先，需要提前知道每个Task的执行时间。
- 测试指标
  - **不同Tile数量**和**实际执行时间条件不同随机程度**下Makespan的提升率
  - Transfer Learning机制: 在小规模DAG训练，迁移到大规模DAG后测试Makespan的提升率

Tile数量为T^2 = (M / N)^2
此矩阵分解Job的DAG图包含Task节点数n = O(T^3)



READYS: A Reinforcement Learning Based Strategy for Heterogeneous Dynamic Scheduling

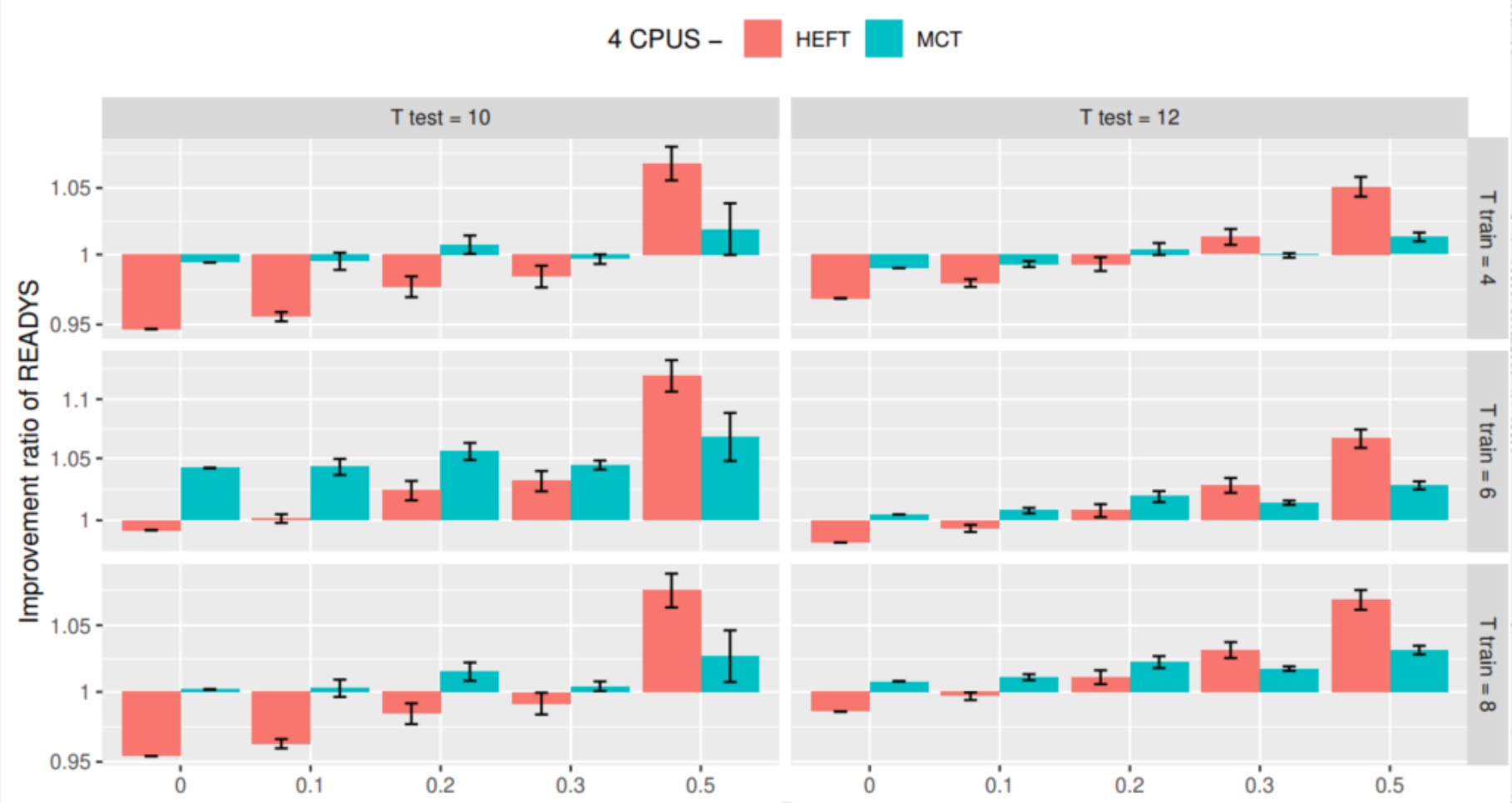
智能体实现与实验

- 测试集上 Makespan性能提升结果

READYS: A Reinforcement Learning Based Strategy for Heterogeneous Dynamic Scheduling

智能体实现与实验

- Transfer learning机制 Makespan性能提升结果



READYS: A Reinforcement Learning Based Strategy for Heterogeneous Dynamic Scheduling

优势

- 对动态环境适配性好
- 相比传统算法不需要很多先验知识（e.g. DAG全图信息、Task执行时间）
- 通过在小规模DAG图上训练并到大规模DAG图上测试的结果表明模型的鲁棒性强且可以减少训练消耗
- （对我们的研究而言）实验模拟场景类似CUDA对矩阵处理的优化机制，具有探索研究价值

不足

- 训练轮数和训练时间较长，不适合大规模场景
- Reward设计上存在缺陷：
  - 大量四元组的瞬时回报都是0，四元组训练利用率低
  - 缺少对错误决策场景的负面Reward和提前结束机制，部分episode训练时间会变得超级长

$$R_a(s,s') = \begin{cases} 0 \text{ if } s' \text{ is non terminal,} \\ R(\text{makespan}) \text{ otherwise.} \end{cases}$$

其他基于强化学习的调度算法

- （TPDS’21）DL2: A Deep Learning-Driven Scheduler for Deep Learning
  - 场景：针对PS-worker架构下的分布式深度学习任务集群进行**弹性资源伸缩**
  - 方法：基于已有工作轨迹进行离线监督学习，然后将神经网络插入强化学习进行微调，用于在线决定任务资源分配。
- （CoNEXT ‘20）Job Scheduling for Large-Scale Machine Learning Clusters
  - 场景：针对PS-worker架构下的以DAG图为构建方式的分布式机器学习任务进行调度
  - 方法：先设计了一个启发式调度算法，系统初始阶段使用该启发式调度算法，并基于这些工作轨迹进行**离线有监督学习**，模型达到一定优化程度后，使用RL模型进行**在线决策**。
  - 特点：少见地实现了很多知名Baseline进行对比实验

强化学习共同特点：训练成本高

近似算法

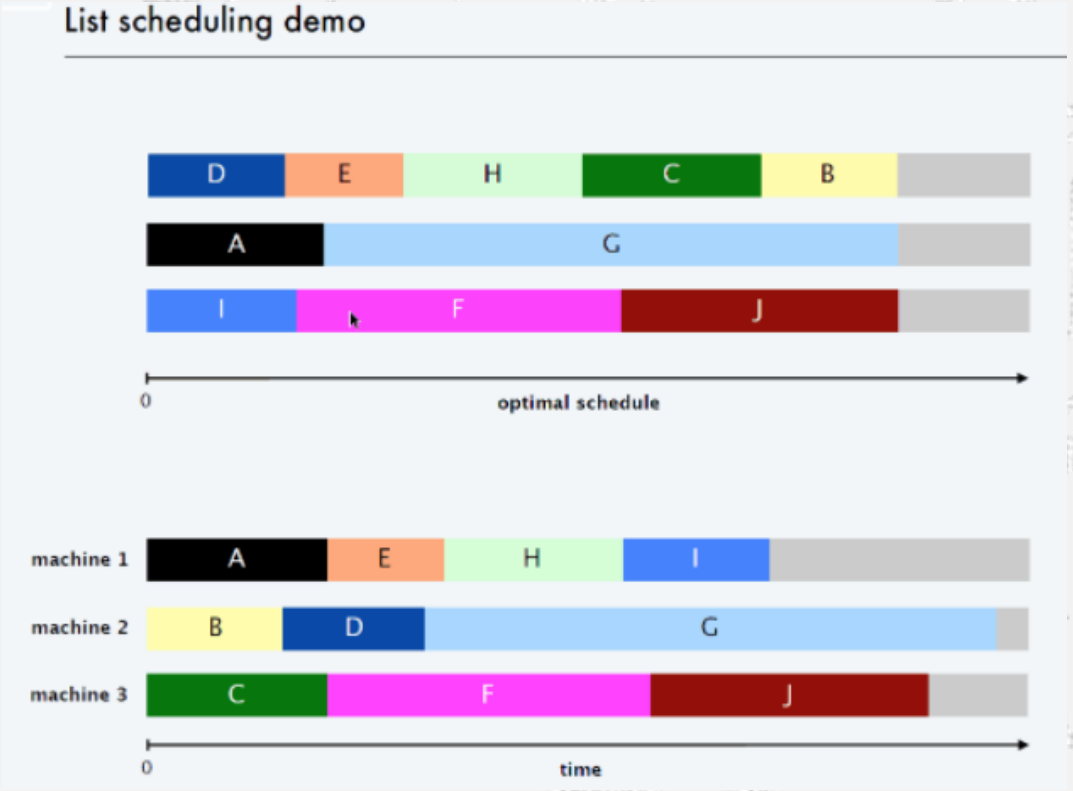
- **NP-hard问题**
  - 无法找到多项式时间普适算法
  - 牺牲算法的精确度，以在可计算时间内找到一个近似解。

- **近似算法满足条件**
  - 在多项式时间内完成
  - 具有普适性（能解决这类问题的任何实例）
  - 有一个确切的近似程度

简单场景下使用近似算法例子

- **场景定义（NP-hard问题）**
  - 任务与机器环境：M台机器
  - 优化目标：所有作业的最终完成时间
  - 约束条件：
- 算法
  - List Scheduling：贪心策略，它的核心思想是将各个工作依次安排到累计工作时长最短的机器中。

$$P||C_{max}$$



简单场景下使用近似算法例子

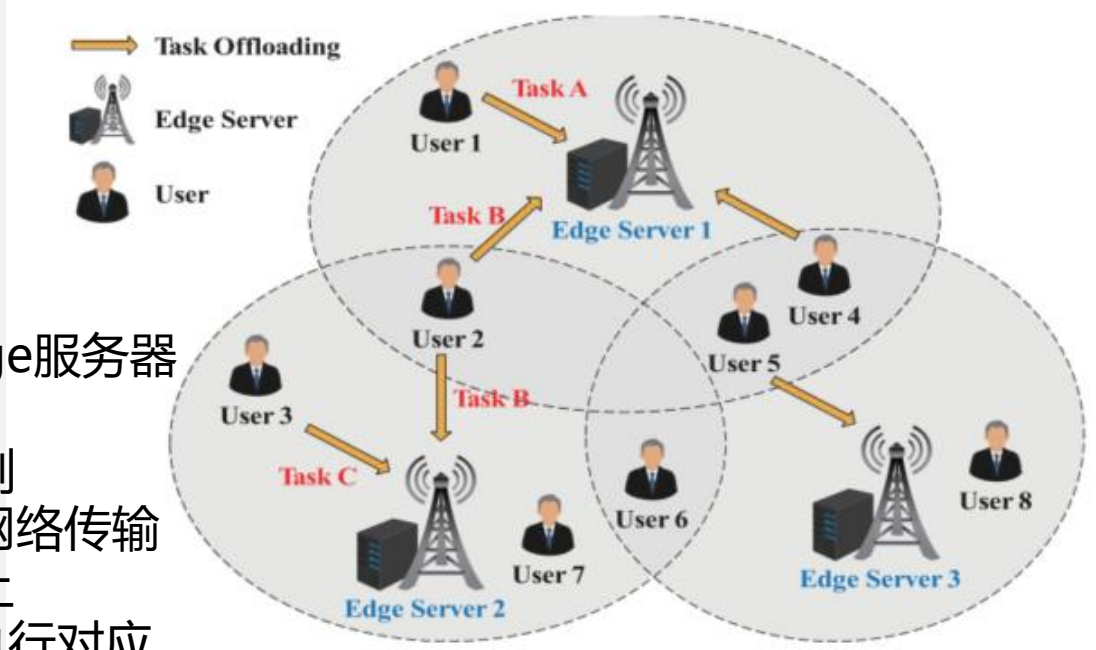
- 算法
  - List Scheduling：贪心策略，它的核心思想是将各个工作依次安排到累计工作时长最短的机器中。
- 近似比证明
  - 分析两个 C_{max} 的下界
    - $C^*_{max} \geq \sum_{j=1}^{n} p_j/m$ ：最优调度策略得到的完成时间至少是每个机器的平均负载
    - $C^*_{max} \geq \max_j p_j$ ：最终完成时间至少是任何一个任务的完成时间
  - 证明2-近似：假设j是最后一个完成的任务，则j的完成时间就是最终任务完成时间。假设j的开始时间为 s_j ，则 $C_{max} = s_j + p_j$ 。在 s_j 之前所有机器应该是繁忙的，否则j会更快开始。所有机器都繁忙的时间最多是 $\sum_{j=1}^{n} \frac{p_j}{m}$ ，因此： $C^{LS}_{max} \leq s_j + p_j \leq \sum_{j=1}^{n} \frac{p_j}{m} + p_j \leq 2C^*_{max}$

LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment

论文的场景定义

- 任务
  - **空间层面**：只有在服务范围的用户任务才可以被卸载到Edge服务器执行
  - **任务层面**：所有的任务都要被卸载到边缘端，任务可以分割
  - **资源异构**：边缘服务器的速度不同、用户提交任务和数据网络传输带宽不同，只有满足资源需求才能让任务掉调度到服务器上
  - **服务限制**：Edge服务器必须包含用户请求的Survice才能执行对应用户卸载的Task

- 决策场景
  - **User Allocation:** 决策用户的Task n分配到哪些服务器m执行
  - **Service Placement:** 决策Survice s是否放置到服务器m上



比较复杂

$$\mathcal{X} \triangleq \{x_{n,m} : n \in \mathcal{N}, m \in \mathcal{M}\}$$
$$\mathcal{Y} \triangleq \{y_{m,s} : m \in \mathcal{M}, s \in \mathcal{S}\}$$

Y. Chen, et al.,"LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment" in IEEE Transactions on Parallel & Distributed Systems, vol. 33, no. 07, pp. 1581-1592, 2022.

LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment

场景定义

- 任务和机器环境
  - 均匀相关环境Q：每个服务器具有不同的计算速度，不同任务在同一个服务器上执行速度一致
- 约束条件
  - 决策变量约束：
    - 任务分配变量x in [0,1]说明任务可以被切分到不同服务器上。（12b）
    - 服务放置变量y in {0,1} （12c）
  - 卸载场景约束：
    - 整个任务必须完整地被卸载到临近的边缘服务器上（12d）
    - 任务必须放置在具有survice的边缘服务器上（12e）
  - 服务器容量约束
    - 任务分配到服务器的计算容量不能超过限制（12f）
  - 用户感知时延约束
    - 任务分配到服务器的计算时延+传输时延必须小于用户的最低容忍延迟（12g）

$$D^c_{n,m} = x_{n,m} l_n / F_m, \qquad D^t_{n,m} = \frac{x_{n,m} b_n}{w \log_2(1 + p_n h_{n,m}/\sigma^2)}$$

$$\text{s.t.} \quad x_{n,m} \in [0,1], \qquad \forall n \in \mathcal{N}, \forall m \in \mathcal{M}, \tag{12b}$$

$$y_{m,s} \in \{0,1\}, \qquad \forall m \in \mathcal{M}, \forall s \in \mathcal{S}, \tag{12c}$$

$$\sum_{m \in \mathcal{M}_n} x_{n,m} = 1, \qquad \forall n \in \mathcal{N}, \tag{12d}$$

$$x_{n,m} \le y_{m,s}, \qquad \forall n \in \mathcal{N}_s, \tag{12e}$$

$$\sum_{n \in \mathcal{N}_m} x_{n,m} l_n \le L_m, \qquad \forall m \in \mathcal{M}, \tag{12f}$$

$$D^c_{n,m} + D^t_{n,m} \le d_n, \quad \forall n \in \mathcal{N}, \qquad \forall m \in \mathcal{M}. \tag{12g}$$

LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment

优化目标：任务卸载总成本（Total Cost of Task Offloading System）

- 服务放置成本（Service Placement Cost）
  - Service放置到服务器上需要消耗边缘服务器上的存储空间，这是货币化的表示。
- 边缘服务器使用成本（Edge Server Usage Cost）
  - 当处理负载转移到边缘服务器时，边缘服务器的计算资源被消耗带来成本这是货币化的表示。
- 能源消耗成本（Energy Consumption Cost）
  - 在用户和边缘端的数据传输过程中，能源消耗成本需要被考虑。

$$\mathbb{P}_1 : \min_{\mathcal{X},\mathcal{Y}} \quad C^p + C^u + C^e$$

$$C^p = \sum_{m\in\mathcal{M}} \sum_{s\in\mathcal{S}} y_{m,s} c^p_{m,s}$$

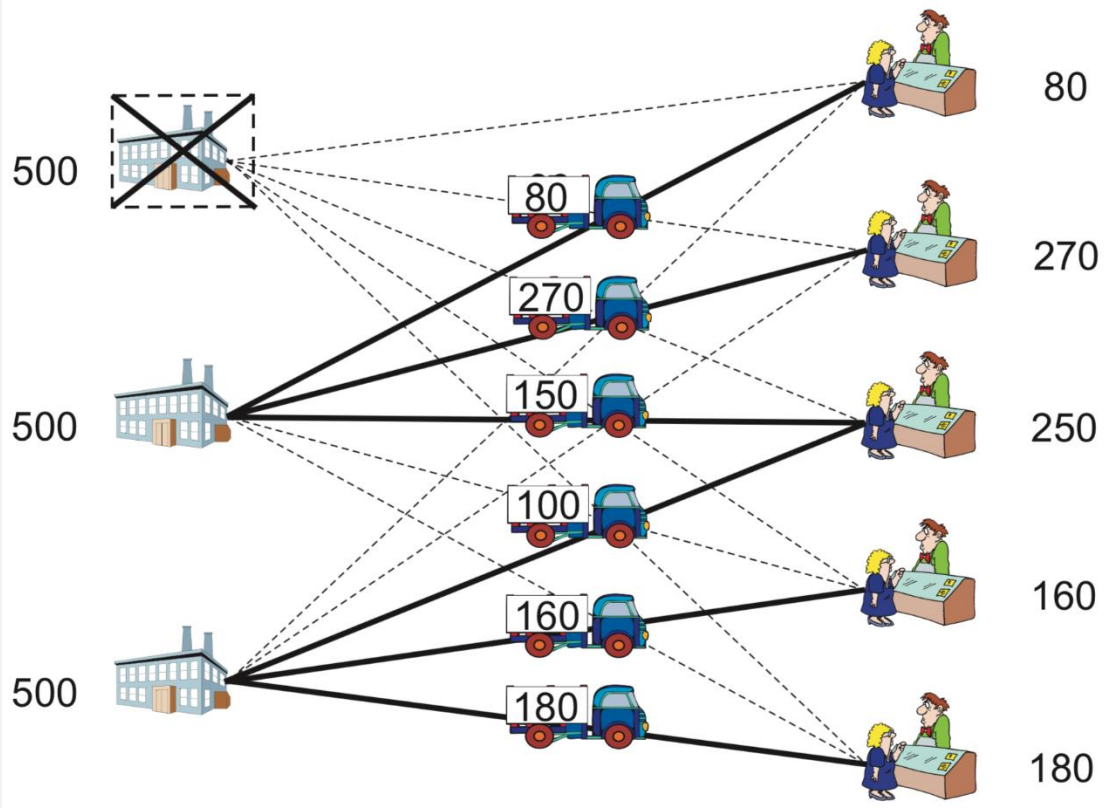
$$C^u = \sum_{m\in\mathcal{M}} \sum_{n\in\mathcal{N}_m} x_{n,m} l_n c^u_m,$$

$$C^e = \sum_{m\in\mathcal{M}} \sum_{n\in\mathcal{N}_m} x_{n,m} \frac{p_n b_n}{w \log_2(1 + p_n h_{n,m}/\sigma^2)} c^e$$

LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment

挑战

- **P1问题是一个NP-hard问题**
  - 证明方法：将NP-hard问题限量设备选址问题（Capacitated Facility Location Problem, CFLP）规约成P1问题



- **P1问题是一个多变量优化问题**

$$\mathcal{X} \triangleq \{x_{n,m} : n \in \mathcal{N}, m \in \mathcal{M}\}$$
$$\mathcal{Y} \triangleq \{y_{m,s} : m \in \mathcal{M}, s \in \mathcal{S}\}$$

LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment

算法设计

- **简化问题的LP求解**
  - 固定变量Y，转化为标准LP问题
  - 使用**内点法**可以在多项式时间内求解

$$\mathbb{P}_2(\mathcal{Y}): \min_{\mathcal{X}} \quad C^p + C^u + C^e$$

$$\begin{aligned}
\text{s.t.} \quad & x_{n,m} \in [0,1], && \forall n \in \mathcal{N}, \forall m \in \mathcal{M}, \\
& \textstyle\sum_{m \in \mathcal{M}_n} x_{n,m} = 1, && \forall n \in \mathcal{N}, \\
& x_{n,m} \le y_{m,s}, && \forall n \in \mathcal{N}_s, \\
& \textstyle\sum_{n \in \mathcal{N}_m} x_{n,m} l_n \le L_m, && \forall m \in \mathcal{M}, \\
& D^c_{n,m} + D^t_{n,m} \le d_n, && \forall n \in \mathcal{N}, \forall m \in \mathcal{M}.
\end{aligned}$$

Algorithm 1. LP Solver for $\mathbb{P}_2(\mathcal{Y})$

Input: Y
Output: $\mathcal{X}$
1 Solve problem $\mathbb{P}_2(\mathcal{Y})$ using interior point method in polynomial time;
2 Obtain the minimized total cost $w(Y)$;
3 Output optimal user-allocation decision $\mathcal{X}$;

LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment

算法设计

- 简化问题的LP求解
  - 固定变量Y，转化为标准LP问题
  - 使用**内点法**可以在多项式时间内求解

- **对含有变量Y问题求解**
  - 初始化服务放置Y的决策
  - 使用多轮迭代**局部搜索法**求解近似最优解

Algorithm 2. Local-Search Operation *New*

Input: Set Y of placed services
Output: $bstNeighbor$, $cstReduce$

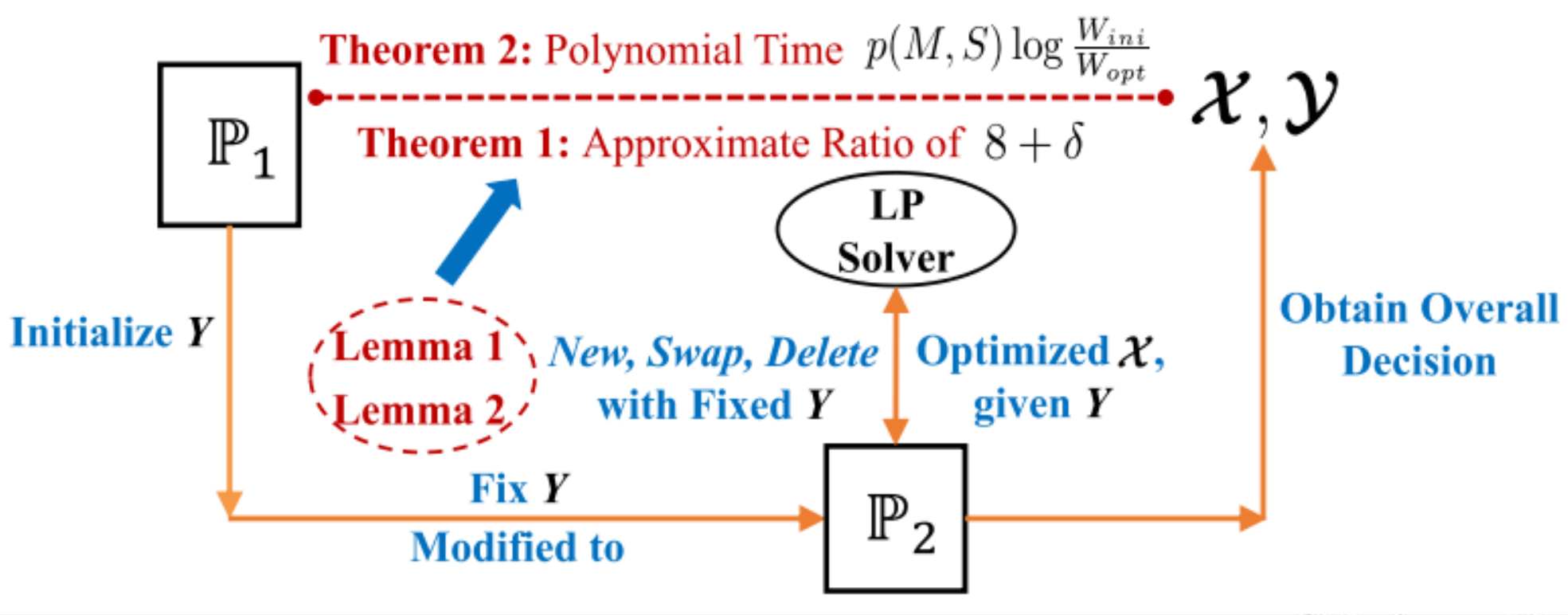
$bstNeighbor \leftarrow \varnothing$;
$cstReduce \leftarrow 0$;
for *each* $(m,s) \in \{(m,s) : m \in \mathcal{M}, s \in \mathcal{S}\} - Y$ **do**
  if $w(Y) - w(Y \cup \{(m,s)\}) > cstReduce$ **then**
    $bstNeighbor \leftarrow Y \cup \{(m,s)\}$;
    $cstReduce \leftarrow w(Y) - w(bstNeighbor)$;

Algorithm 5. LOCUS for User-Perceived Delay-Aware Service Placement and User Allocation

1 %Step 1:Initialize service placement decisions;
2 $Y = \varnothing$;
3 **for** *each* $m \in M$ **do**
4 **for** *each* $n \in \mathcal{N}_m$ **do**
5 $Y \leftarrow Y \cup \{(m, s_n^r)\}$;
6 %Step 2:Iterative local search for (near-)optimum;
7 **repeat**
8 $local\_success \leftarrow$ False;
9 $(bstNeighbor, cstReduce) \leftarrow New(Y)$;
10 **if** $cstReduce \geq w(Y)/p(M,S)$ **then**
11 $Y, local\_success \leftarrow bstNeighbor, True$;
12 Goto line 7;
13 $(bstNeighbor, cstReduce) \leftarrow Swap(Y)$;
14 **if** $cstReduce \geq w(Y)/p(M,S)$ **then**
15 $Y, local\_success \leftarrow bstNeighbor, True$;
16 Goto line 7;
17 $(bstNeighbor, cstReduce) \leftarrow Delete(Y)$;
18 **if** $cstReduce \geq w(Y)/p(M,S)$ **then**
19 $Y, local\_success \leftarrow bstNeighbor, True$;
20 **until** $local\_success = False$

LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment

算法设计



LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment

- 从**限量设备选址问题（Capacitated Facility Location Problem, CFLP）**的相关近似算法证明中找到和本文匹配的引理
- 对已有的引理进行形式化推导，得到算法优化目标值与OPT目标值比例的上下界

Lemma 1 ([44]). *If there is no (m, s) which is not in the current Y, such that*

$$w(\boldsymbol{Y}) - w(\boldsymbol{Y} \cup \{(m,s)\}) \geq w(\boldsymbol{Y})/p(M,S), \tag{15}$$

then

$$w_e(\boldsymbol{Y}) < w(\boldsymbol{Y}^*) + MSw(\boldsymbol{Y})/p(M,S), \tag{16}$$

Lemma 2 ([44]). *If there is no (m, s) in the current Y to perform the* Swap *or* Delete *operation so that the total cost is reduced by at least $w(Y)/p(M, S)$, then*

$$w_s(\boldsymbol{Y})(1 - \frac{(MS)^2}{p(M,S)}) < 5w(\boldsymbol{Y}^*) + 2w_e(\boldsymbol{Y}) + \frac{w(\boldsymbol{Y})}{MS}. \tag{17}$$

近似比证明过程

satisfied. Plugging (16) into (17), we get

$$w_s(\boldsymbol{Y})\left(1 - \frac{(MS)^2}{p(M,S)}\right) < 7w(\boldsymbol{Y}^*) + \frac{2MSw(\boldsymbol{Y})}{p(M,S)} + \frac{w(\boldsymbol{Y})}{MS}. \tag{18}$$

Based on inequation (16), we can also get

$$w_e(\boldsymbol{Y})(1 - \frac{(MS)^2}{p(M,S)}) < w(\boldsymbol{Y}^*) + \frac{MSw(\boldsymbol{Y})}{p(M,S)}. \tag{19}$$

Then we let inequation (18) plus (19), and we have

$$w(\boldsymbol{Y})\left(1 - \frac{(MS)^2}{p(M,S)}\right) \leq 8w(\boldsymbol{Y}^*) + \frac{3MSw(\boldsymbol{Y})}{p(M,S)} + \frac{w(\boldsymbol{Y})}{MS}. \tag{20}$$

$$\frac{w(\boldsymbol{Y})}{w(\boldsymbol{Y}^*)} < 8\frac{1}{(1 - \frac{(MS)^2}{p(M,S)} - \frac{3MS}{p(M,S)} - \frac{1}{MS})}. \tag{21}$$

$$\delta \geq 8\left(\frac{1}{(1 - \frac{(MS)^2}{p(M,S)} - \frac{3MS}{p(M,S)} - \frac{1}{MS})} - 1\right), \tag{22}$$

LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment

算法时间复杂度证明过程

- 对局部搜索算法证明迭代轮数的上界

$$W_k - W_{k+1} \geq \frac{W_k}{p(M,S)}, \quad \forall k \in \{0,1,2,\ldots,R-1\}. \tag{23}$$

$$\frac{W_k}{W_{k+1}} \geq \frac{1}{1-\frac{1}{p(M,S)}}, \quad \forall k \in \{0,1,2,\ldots,R-1\}. \tag{24}$$

$$\frac{W_{ini}}{W_{opt}} \geq \frac{W_0}{W_R} = \frac{W_0}{W_1}\cdot\frac{W_1}{W_2}\cdot\ldots\cdot\frac{W_{R-1}}{W_R} \geq \left(\frac{1}{1-\frac{1}{p(M,S)}}\right)^R. \tag{25}$$

$$\left(1-\frac{1}{p(M,S)}\right)^{-p(M,S)} \geq \mathrm{e}. \tag{26}$$

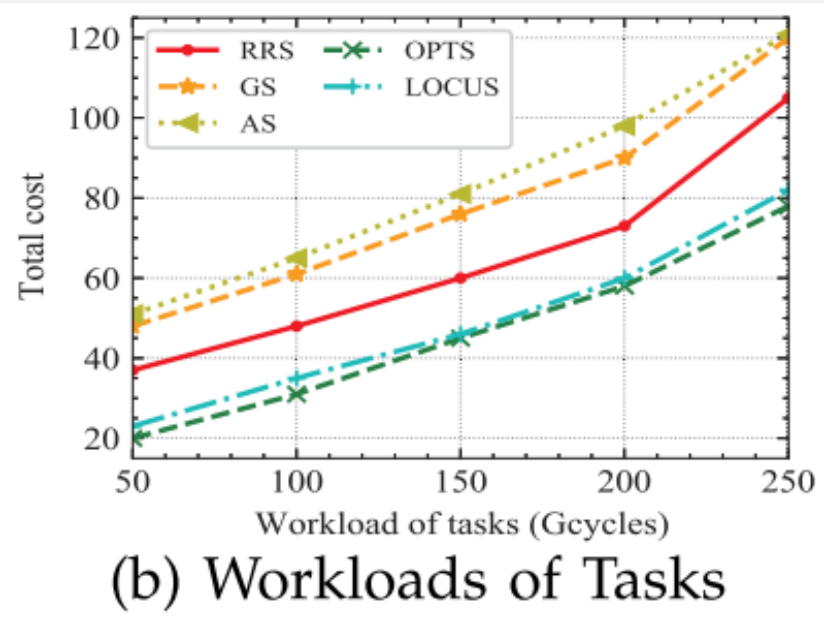
$$R \leq p(M,S)\log\frac{W_{ini}}{W_{opt}}. \tag{27}$$

LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment

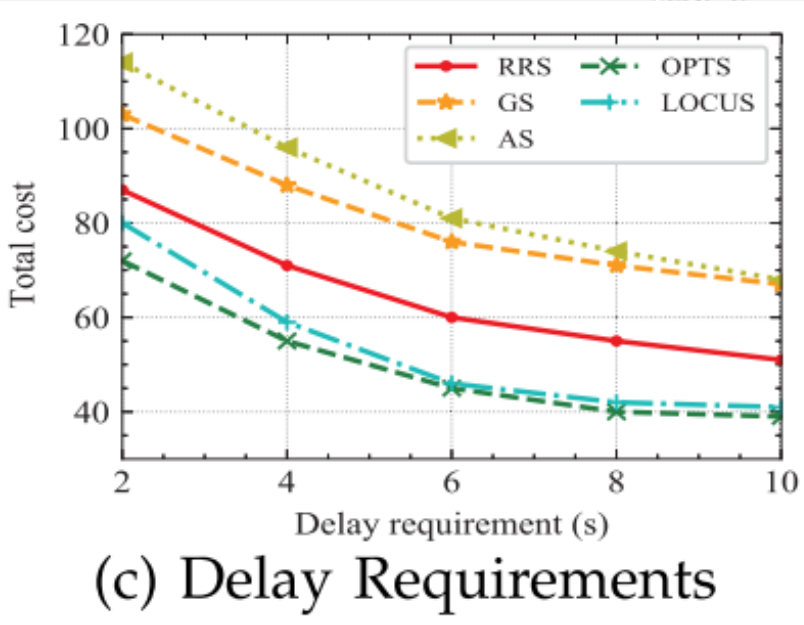
实验结果

- **对Total Cost的实验**
  - 接近OPT，优于baseline

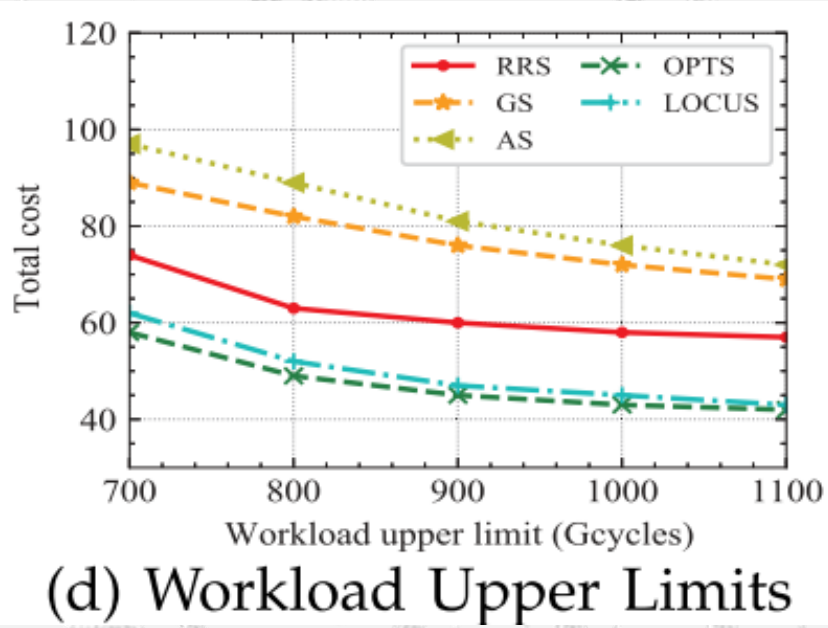
- RRS：Randomized Rounding Scheme
- GS：Greedy Scheme
- AS：All-service Scheme
- OPTS：MILP求解器得到的结果



(b) Workloads of Tasks



(c) Delay Requirements



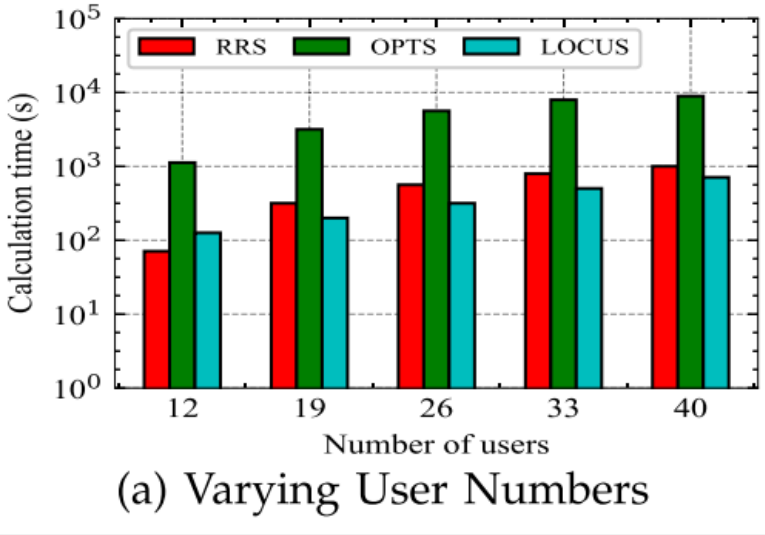
(d) Workload Upper Limits

LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment

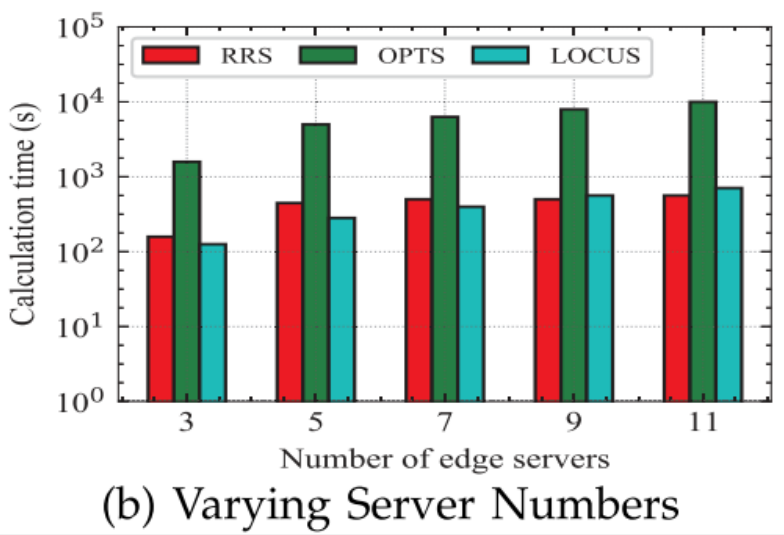
实验结果

- **对计算时间的实验**
  - 明显短于使用求解器得到的OPT解

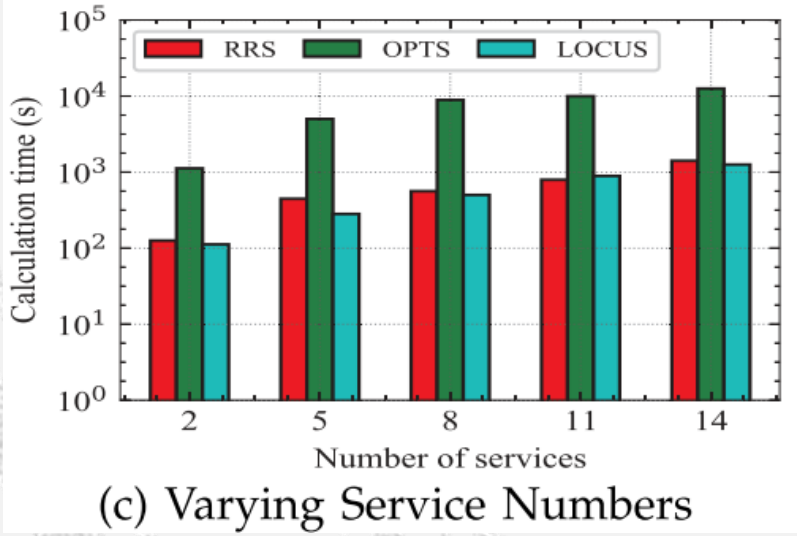
- RRS：Randomized Rounding Scheme
- OPTS：MILP求解器得到的结果



(a) Varying User Numbers



(b) Varying Server Numbers



(c) Varying Service Numbers

LOCUS: User-Perceived Delay-Aware Service Placement and User Allocation in MEC Environment

优势

- 基于复杂的离线分配场景进行建模，考虑了服务放置、任务分配、期望延迟等复杂因素
- 算法对提出的启发式算法具有较完备的理论证明，算法对性能的下界有保证

不足

- 本场景假设Task可以被无限拆分放置，但实际上任务通常具有最小的单位。
- 对比算法过于简单，没有和一些启发式算法或基于学习的算法进行比较。
- 本场景在调度开始前，所有任务都已到达且任务执行时间可以直接被获取。缺少对实际在线提交任务场景的考虑。

其他基于近似算法的调度算法

- (TOC’ 22) Online Scheduling of Distributed Machine Learning Jobs for Incentivizing Sharing in Multi-Tenant Systems
  - 针对PS-worker架构下的分布式机器学习任务在线调度场景建模
  - 将在线场景划分为很多个离线调度time-slot，每个time-slot应用近似算法
  - 提供激励机制鼓励用户将手头剩余的资源出租，同时在资源不足时租用其他用户的资源
  - 提供了理想资源共享特性（e.g. 帕累托效率）的证明，提供了近似比保证和复杂度分析

- （TPDS’ 22）Mechanisms for Resource Allocation and Pricing in Mobile Edge Computing Systems
  - 场景：用户为资源提供竞价，调度器根据用户的出价分配合适的资源，目标是整个系统的收益最大
  - 基于混合整数线性规划（MILP）建模问题，提出了满足Individually-rational和envy-free allocations机制的贪心算法
  - 提出了可证近似比的任务分配算法

- （TON ‘20）Provably Efficient Resource Allocation for Edge Service Entities Using Hermes
  - 场景：终端用户卸载任务到边缘端，边缘端服务器决策资源的伸缩，目标是租用资源成本最低
  - 解决方法：将问题转化为最小集合覆盖问题，从最小集合覆盖问题中找解决算法和已知近似比

研究方向特点和成本

系统派

- 重视实验和落地实现
- 现实场景中任务在资源上的执行通常是复杂多变的，任务执行时间、执行结果、到达时间、资源需求通常不可知，算法复杂且需要大量知识
- 通过大规模实验验证有效性，容易忽视理论证明
- **主要研究成本**：系统实现和大规模实验

强化学习派

- 兼顾系统和理论
- 通常需要利用大规模数据集进行多轮次的训练
- 需要长时间的调参
- 复杂的模型导致算法相比其他两派具有更长的决策时间
- **主要研究成本**：训练和调参

理论派

- 重视理论证明
- 通过近似比、时间复杂度、博弈论或调度论中的性质证明来说明有效性
- 理论证明难度限制了算法的复杂程度
- 实验简单，但大概率比不过其他两派的算法结果
- **主要研究成本**：理论证明

针对AI任务弹性调度研究 未来方向想法

系统派

- 可以借鉴READYS，对深度学习任务提取DAG图，借鉴深度学习框架做调度优化?
- 可以针对实际调度系统中的痛点做优化?
  - 对深度学习任务在不同计算资源上实际执行时间/实际执行资源的预测?
  - 对大型集群中的未来工作负载变化做预测?
  - 直接优化针对AI任务的集群调度器?

强化学习派

- 可以针对强化学习调度的痛点去做优化?
  - 智能体推断时间缩减
  - 智能体训练成本缩减
- 可以针对调度场景设计专用强化学习算法?

理论派

- 可以将问题建模成经典问题，将前人在经典问题的研究成果迁移到新型调度场景中?
- 将博弈论、激励机制等方法引入场景中，提出近似算法并证明近似比?

中山大學
SUN YAT-SEN UNIVERSITY

敬请批评指正

汇报人：肖霖畅