基于神经模拟退火优化高性能计算集群上的拓扑感知作业分配

Zekang Lan[1], Yan Xu[2], Yingkun Huang[1],Dian Huang[1],Shengzhong Feng[1]

arXiv:2302.03517v1 [cs.DC] 6 Feb 2023

摘要——**高性能计算（HPC）集群上的作业可能因作业间网络干扰而导致性能显著下降。拓扑感知作业分配问题（TJAP）旨在决定如何将节点分配给特定应用程序，以减轻作业间网络干扰。本文研究胖树网络上的基于窗口的TJAP，目标是最小化通信跳数成本，这是一种定义的作业间干扰度量。基于窗口的调度方法会周期性地重复执行：获取队列中的作业，并求解将作业映射到可用节点的分配问题。本文考虑了两种特殊的分配策略，即静态连续分配策略（SCAS）和动态连续分配策略（DCAS）。对于SCAS，本文构建了一个0-1整数规划模型。对于DCAS，本文提出了一种称为神经模拟算法（NSA）的方法。该方法是模拟算法（SA）的扩展，能够学习修复算子，并在引导式启发搜索中使用该算子。通过与SA和SCIP进行计算研究，验证了NSA的有效性。数值实验结果表明，本文提出的模型和算法均有效。**

索引词—**高性能计算、作业分配、神经模拟退火、拓扑感知**

一、引言

高性能计算（HPC）集群是一种并行处理系统，它利用数千台互连的独立计算机协同工作，作为一个集成的计算资源。互连拓扑——包括计算机节点（下文简称节点）、交换机和信道的布局——在决定网络性能方面发挥着关键作用，而网络性能是HPC集群的关键因素。近年来，作业间网络干扰已被确定为导致高性能计算系统中作业性能波动的原因之一。对于采用不同互连拓扑的集群，例如基于环面网络、蜻蜓网络和胖树的集群，作业间网络干扰会使生产应用程序的运行速度降低66%–150% [1] [2] [3] [4]。

拓扑感知作业分配问题（TJAP）旨在确定如何为特定应用程序分配节点，以减轻作业间的网络干扰。由于并行应用程序的运行时间会受到这些应用程序线程之间频繁通信的影响，这些线程通过消息传递接口等通信基础设施交换数据和消息，因此，通信密集型应用程序的性能高度取决于彼此通信的各个计算节点的位置。如今，资源与作业管理系统可以配置为支持 TJAP，例如，广为人知且应用广泛的 SLURM 支持三维环面互连、胖树和蜻蜓网络，并采用略有不同的算法。然而，现有调度系统以逐作业方式做出决策，例如先来先服务（FCFS），即每个作业在派发到系统资源时都不考虑后续作业。虽然单独做出作业决策可能带来良好的短期优化，但从长远来看很可能导致性能不佳。基于窗口的方法不同于作业到达后立即将其分配给节点的逐作业模式，它会先将作业收集到队列中，直到满足指定条件 [5] [6]。基于窗口的分配步骤会定期重复。每个周期都必须解决一个将作业映射到可用资源的分配问题。通常会采用整数规划技术。与逐作业方式相比，基于窗口的方法会全面考虑等待队列中的作业，从而得出更优的作业分配方案。然而，由于基于窗口的系统需要探索更大的解空间，因此需要更高效的算法。

通常，TJAP问题与二次分配问题（QAP）十分相似，而后者已被证明是NP完全问题。因此，人们提出了一些启发式算法，例如遗传算法 [7]和进化算法 [8]。为了满足实时决策所要求的严格时限，一种常用的解决方案是采用连续分配策略，即调度器为每个作业分配一组紧凑且连续的计算节点。这种策略通过保持已分配节点的局部性，减少多个作业同时运行并共享网络带宽所造成的网络竞争，从而提升应用性能。根据连续性规则，我们不必将所有节点的分配情况都作为决策变量。相反，只需将一个请求节点的分配情况作为决策变量，其他节点的分配则按照连续性规则确定。因此，TJAP的建模并非复杂的QAP，而是一个线性规划问题。

[1]深圳国家超级计算中心（深圳云计算中心），中国 {lanzk,huangyk, huangdian,fengsz}@nsccsz.cn[2]北京工业大学，中国 yxu@bjut.edu.cn

本文研究胖树网络上的基于窗口的TJAP。作业间网络干扰通过定义的通信跳数（CH）成本来衡量。本文提出了一种特殊的连续分配策略：根据一维空闲节点序列，连续分配作业所请求的节点。根据分配一个作业后是否更新该序列，连续分配策略进一步分为两种：静态连续分配策略（SCAS）和动态连续分配策略（DCAS）。本文的贡献有两点。首先，我们将基于SCAS的TJAP建模为一个整数规划问题。其次，我们针对基于DCSA的TJAP开发了一种新算法，即神经模拟退火（NSA）。

本文其余部分安排如下：第二节介绍与高性能计算作业调度相关的工作，尤其是 TJAP。第三节详细介绍 TJAP。第四节给出一种基于 SCAS 的模型。第五节提出一种名为 NSA 的新算法，基于 DCAS 求解 TJAP。第六节给出所得结果。最后，第七节总结全文。

本文相关源代码可在以下网址获取：https://github.com/lzk23/hpcschedule。

第二节 相关工作

严格来说，调度器系统中的作业调度包含两部分，即作业优先级排序和作业分配。作业优先级排序用于决定作业获准运行的顺序。先到先服务和 EASY回填（回填）是非常简单的优先级调度算法示例。回填是一种在有足够空闲节点时，将作业插队执行（按队列顺序之外运行作业）的方法，且插队作业不会延迟队列首部的作业。约 90% ~95% 的作业调度器采用回填 [9]。作业调度主要考虑若干因素，包括公平性 [9],能耗[10],利用率 [3], 和性能。本文聚焦于 TJAP，该方法旨在减轻作业间干扰并提升性能。

大多数关于 TJAP 的文献采用逐作业方式。Kaplan等人 [11] 提出了一种作业分配方法，可在降低冷却能耗的同时，联合最小化高性能计算应用程序的通信成本。Georgiou等人[12] [13]考虑了应用通信矩阵。Yan等人 [14] 提出了一种基于标签传播的进程映射方法，即 LPMS，该方法成本低廉，且非常适用于共享高性能计算系统（多个作业可以共享一个节点）。Ryu等人 [15] 提出了一种具备拓扑感知能力的深度强化学习调度器，可同时选择作业并为其分配资源。代表集群拓扑的机器带宽和当前图形处理器到作业的分配情况被作为输入提供给深度神经网络。Smith等人 [3] 针对三级胖树开发了一种新型作业隔离调度方法，在保持较高系统利用率的同时消除作业间网络干扰。为了分析网络拓扑及其不同配置在构建前的表现，网络设计者会借助解析建模和仿真，例如[16]和 [17]。

针对 TJAP 的研究中，采用基于窗口的方法并不多。Soner 等人 [5]提出了一种名为 AUCSCHED 的调度算法，尝试在一维节点数组上分配连续块。该方法使用 CPLEX 求解 0-1整数模型。Yang 等人 [6] 维护了一个时隙列表，以保留资源分配的节点连续性信息。他们分别构建了一个 0-1 多重背包模型，并使用分支定界法和贪心算法求解。

III. 问题陈述

本节首先简要介绍胖树网络拓扑，并详细说明作业间干扰度量（即通信跳数开销）。随后，我们介绍两种不同的连续分配策略，即 SCAS 和 DCAS。最后，我们正式阐述 TJAP。

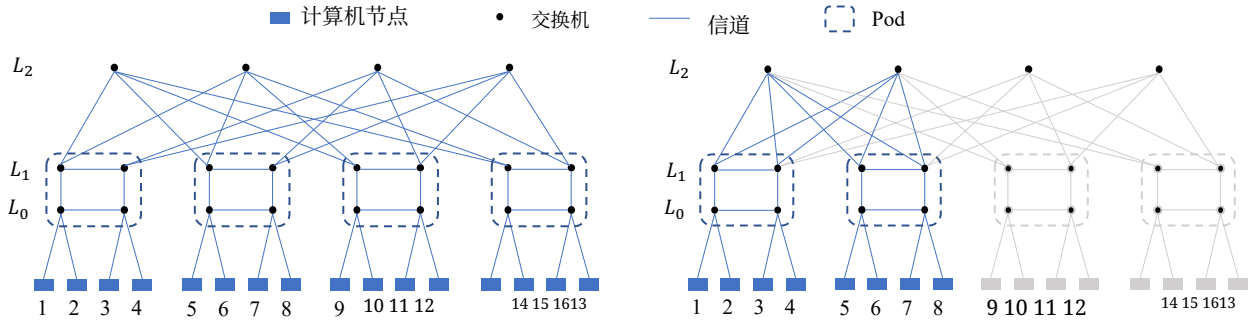
A. 胖树网络和作业间干扰度量

胖树网络是一种可实现可证明高效通信的通用网络。大多数高性能网络中使用的胖树类型是 k叉胖树。具体而言，对于给定的基数 k （即一个交换机上的端口数），胖树拓扑中有 k 个 Pod。每个 Pod 中有 k 个交换机，均匀分布在两层中，且每个交换机有 k 个端口。第 L_0 层的每个交换机连接到 $k/2$ 个节点和第 L_1 层的 $k/2$ 个交换机（各占用 1 个端口）。类似地，第 L_1 层的每个交换机连接到第 L_0 层的 $k/2$ 个交换机，以及第 L_2 层的 $k/2$ 个交换机。因此，胖树网络具有全二分带宽。由此，每个 Pod 连接 $(k/2)\cdot(k/2)$ 个节点，系统中有 $k\cdot(k/2)\cdot(k/2) = k^3/4$ 个节点互连。图 1(a) 展示了一个 4叉胖树，其中每个 Pod 有 4 个节点，总计 16 个节点。

对于具有一定数量节点的实际系统，需要对全二分胖树拓扑进行调整。因此，生产系统通常采用剪枝胖树配置。基于 k 元胖树网络模型，会剪除部分 Pod 以匹配实际系统。例如，图 1(b) 展示了一个包含两个 Pod 的剪枝胖树模型，对应于图 1(a) 中的全二分胖树。

如前所述，胖树网络拓扑可能会遭受严重的作业间干扰，从而影响性能。在共享的高性能计算系统中，作业间干扰发生在节点内。本文假设所有作业都申请多个节点。由于恰好申请一个节点的作业不会产生作业间干扰，因此不予考虑。由此，节点是独占的：一旦某个节点被一个尚未完成的作业占用，就不能分配给其他作业。因此，节点内部不会发生作业间干扰。

直观而言，为了减少作业间干扰，应尽可能将作业放置在少数 Pod 中。本文引入一种定制的 CH 成本，用于衡量作业间干扰。CH 是连接两个设备（即节点或交换机）的中间连接链路。对于两个指定的节点



(a) k 叉胖树架构　　(b) 剪枝胖树配置

图 1. 胖树网络拓扑。

在胖树网络中，如果它们具有相同的父节点的交换机（在第 L_0 层直接连接这两个节点），则 CH 的数量为 2；否则，如果它们位于同一个 Pod 中，则 CH 的数量为 4；否则，如果它们位于不同的 Pod 中，则 CH 的数量为 6。给定分配给作业 j 的一组节点 $N_j(N_j \geq 2)$，计算 CH 成本的公式如下：

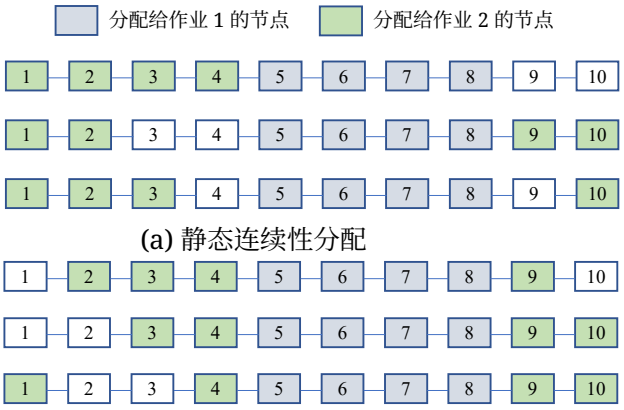
$$C(N_j) = \frac{1}{n_j} \sum_{i \in N} \sum_{k \in N: i \neq k} ch_{i,k} \quad (1)$$

其中，n_j 表示作业 j请求的节点数；c 表示每个 CH 的成本；$h_{i,\ k}$ 表示节点 i和 k之间 CH 的数量。

B. 连续分配策略

为避免 TJAP 落入难以求解的 QAP，我们采用一种特殊的连续分配策略。具体而言，胖树网络中的节点按照从左到右的线性顺序赋予唯一标识。在每个调度周期，将所有空闲节点按其标识以非降序排列，组成序列 Q。根据序列 Q，我们需要为每个作业确定首个分配节点。确定首个分配节点后，按照序列 Q中给出的节点顺序，连续分配其余节点。特别是，如果从节点 i开始没有足够的连续节点可供分配，则从序列 Q的开头选取剩余节点。遵循此规则，同一 Pod 中的节点很可能会分配给同一个作业，从而有望降低 CH 成本。

根据分配一个作业时是否允许更新序列 Q，连续分配策略分为两类：SCAS 和 DCAS。在 SCAS 中，分配一个作业时不会更新序列 Q；而在 DCAS 中，作业分配完成后，会移除已分配的节点，动态更新序列 Q。每个作业都基于新的空闲节点序列进行连续分配。例如，对于图 1 中的 4叉胖树集群，假设只有节点 1-10 处于空闲状态，即 Q= 1-10。现在需要分配两个作业，分别称为作业 1 和作业 2，且两者都需要 4 个节点。作业 1 被分配到节点 5-8。对于 SCAS，作业 2 有三个可行解，如图 2(a) 所示。



(a) 静态连续性分配

(b) 动态连续性分配

图 2. 静态和动态连续性分配策略。

对于 DCAS，在作业 1 被分配到节点 5-8 后，序列 Q 将更新为 Q= {1-4,9,10}，作业 2 有六个可行解，其中三个见图 2(a)，另外三个见图 2(b)。也就是说，SCAS 的解对于 DCAS 同样是可行解。

C. TJAP 描述

我们的 TJAP 基于胖树网络。TJAP 的目标是根据 SCAS 或 DCSA 为每个作业分配一组节点，以优化式（1）所示的 CH 成本。具体而言，我们假设作业动态到达，即输入队列中等待的作业数量不断变化，并且随着时间推移，会有更多作业到达。我们的 TJAP 采用基于窗口的方式，也就是说，TJAP 会定期从队列中取出作业，并求解将作业映射到可用节点的分配问题。每个时间周期的长度为 τ（例如，30 秒）。也就是说，调度方法必须在每个时间周期内以在线方式做出决策。每个作业 j 都有一个到达时间、请求节点数和处理时间（用户提交的是估计处理时间；作业完成时才能得知实际处理时间）。作业 j 只有在到达时间之后、由调度器派发时，调度器才能获知。作业不允许被抢占或迁移，也就是说，

作业一旦开始执行，就必须在同一组节点上运行直至完成。

需要注意的是，在一个时间周期内，可用的空闲节点可能不足以满足等待队列中所有作业的需求。因此，除了作业分配之外，我们还必须决定作业优先级排序。具体而言，我们为每个作业设置一个名为“等待周期数”（NWP）的属性，其数值初始化为 0。对于当前周期未分配的作业，其 NWP 数值将增加 1。在每个决策周期中，NWP 数值较大的作业优先级较高。对于 NWP 数值相同的作业，请求节点数较少的作业优先级较高。根据所定义的优先级规则，尽可能多地选择作业进行分配。其余未分配作业将在下一个时间周期等待分配。

IV. 静态连续性分配

本节首先基于第四节 A 部分中的 SCAS 对 TJAP 进行建模，然后在第四节 B 部分证明该模型的 NP难性。

A. 基于 *SCAS* 的 *TJAP* 建模

为了基于 SCAS 建立模型，我们首先作如下定义：J，如第三节-C 末尾所述，从等待队列中选出的作业集合；N，空闲节点集合。因此，可以根据 N 中标识递增的节点构建节点序列 Q；n_j，作业 j 请求的节点数；$D(i,\ n_j)$，序列 Q 中从节点 i 开始的 n 个连续节点；$l_{j,\ i}$，$D(i,\ n_j)$ 中节点的 CH 成本；$y_{j,\ i}$，一个二元变量，表示作业 j 是否选择节点 i 作为其首个分配节点。下面给出基于 SCAS 的模型，记为 $\mathrm{M_{SCAS}}$:

$$(\mathrm{M_{SCAS}})\min \sum_{j\in J}\sum_{i\in N} l_{j,i}y_{j,i} \tag{2}$$

$$\sum_{i\in N} y_{j,i} = 1,\ \forall j \in J \tag{3}$$

$$y_{j,i} \le x_{j,k},\ \forall j \in J, i \in N, k \in D(i,n_j) \tag{4}$$

$$x_{j,i}, y_j \in \{0,1\},\ \forall j \in J, i \in N \tag{5}$$

其中，目标函数 (2) 表示最小化总 CH 成本。约束 (3) 限制每个作业恰好选择一个首个分配节点。约束 (4) 确保如果节点 i 被选为作业 j 的首个分配节点，那么 $D(i,\ n_j)$ 中的所有节点都会分配给作业 j。约束 (5) 设定变量的取值域。

B. 基于 *SCAS* 的模型的 *NP*难性

上述模型实际上可以转换为约束最短路径问题（CSP），而该问题已被证明是 NP完全问题 [18]。TJAP 与 CSP 的等价性将在后文证明。

给定一个空闲节点序列 $Q = \{1, 2, \ldots, i, \ldots, n\}$ 和$J = \{0, 1, .., j, \ldots, |J|\}$。作业所请求节点数的集合为 R。请求 r 个节点的作业数为 ε_r，

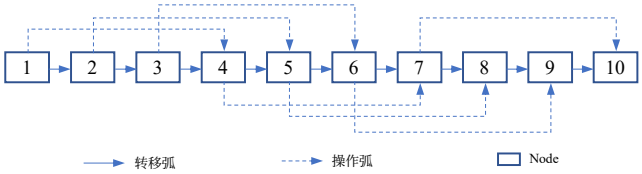


图3. 将 TJAP 转换为约束最短路径问题的示意图。

因此 $|J| = \cup_{r\in R}\varepsilon_r$。构造有向图 $G = (V,\ E)$，其中 V 是对应于 Q中节点的节点集，E 是弧集。弧集 E 包含两种类型的弧：(1) 第一种是转移弧，即根据序列 Q连接任意节点与下一个首节点的有向弧；(2) 第二种是操作弧，即对于每个节点 $i \in V$ 和每个 $r \in R$，从节点 i 连接到其下一个第 $(r-1)$ 个节点（如果存在）的弧。对于一条连接节点 i 与其下一个第 $(r-1)$ 个节点的操作弧，其跨越节点集为 $\{i, i+1, \ldots,\ i+r-1\}$。为简洁起见，不考虑沿序列 Q 返回“反向”连接到首节点的操作弧。跨越 r 个节点的操作弧集合为 E_r。任意相邻弧的权重均为 0，而操作弧的权重则为该弧所跨越节点的 CH成本。随后，TJAP 可转化为约束最短路径问题，即在 G中寻找从节点 1 到节点 n 的最短路径，并满足该路径经过 E_r中的 ε_r 条操作弧这一要求。确定最短路径后，作业 j 的分配节点为同时位于最短路径和 E_{n_j}中的任意操作弧所跨越的节点。

例如，假设 $Q = \{1-10\}$，$J = \{0, 1\}$。任何作业请求的节点数均为 4，即 $R = \{4\}$和 $n_4 = 2$。对应的图 G 如图3所示。由于所有作业都请求 4 个节点，因此通过将每个节点与其后第三个节点（如果存在）相连来构造操作弧。例如，存在一条从节点 1 连接到节点 4 的操作弧，其跨越节点集合为 {1,2,3,4}。随后，TJAP 被转换为约束最短路径问题，该问题旨在寻找一条从节点 1 到节点 10 的最短路径，且该路径恰好经过两次操作弧。

V. 动态连续性分配

考虑到基于 DCAS 的模型表示为 $\mathrm{M_{DCAS}}$。直观而言，$\mathrm{M_{SCAS}}$ 的可行解空间包含于 $\mathrm{M_{DCAS}}$ 的可行解空间中。因此，$\mathrm{M_{DCAS}}$ 也是 NP难。此外，$\mathrm{M_{SCAS}}$ 可以由求解器（例如 CPLEX、GUROBI）直接求解，而 $\mathrm{M_{DCAS}}$ 是一个多阶段优化问题，无法由求解器直接求解。

为了求解 $\mathrm{M_{DCAS}}$，我们提出一种名为神经模拟退火（NSA）的新算法。它是对模拟退火（SA） [19] 的扩展，能够通过强化学习（RL）自动完成生成邻域解这一复杂任务。模拟退火算法已得到广泛发展，并广泛应用于许多优化问题中。它能够避免陷入局部最优解，并获得

更优的解，其方法是以一定概率接受较差的解。我们的方法受到 [20], 的启发；该研究提出了一种神经大邻域搜索算法，用于学习车辆路径问题的修复算子。

A. NSA 总体算法

NSA 总体算法如算法 1 所示。除了步骤 4(b) 中使用的强化学习算法外，NSA 的流程与模拟退火相同。因此，NSA 的流程也由三个参数 $T_{\max}$、$T_{\min}$ 和 $t^{\max}$ 控制：$T^{\max}$ 是最大温度；$T^{\min}$ 是最小温度；$t^{\max}$ 是最大迭代步数。只要迭代次数尚未达到 $t^{\max}$，循环就会继续。在第 t 次迭代时，温度 T 降至 $T^{\max}e^{\frac{rt}{t^{\max}}}$，其中 r 是冷却因子，其计算方式为 $-\log T_{\max}/T_{\min}$。随着 T 降低，接受较差解的接受率也会降低，如算法 1 的步骤 8 所示。

算法 1 用于 TJAP 的 NSA

输入： 成本函数 COST0, 最大（最小）温度 ($T^{\min}$), 最大迭代步数 $t^{\max}$ （$T_{最大值}$）

输出： 找到的最佳可行解 x_{best}

1: 设置 $r = -log(T^{\max}/T_{最小值})$
2: 通过启发式算法生成初始解 x 并记忆 x 作为 x_{best}
3: **当** $t \le t^{\max}$ **时执行**
4: 从 x 中随机生成 x'：(a) 随机移除一定数量的作业从 x 的分配解中移除；(b) 使用强化学习算法
5: **若** x' 优于 x **则**
6: $x \leftarrow x'$
7: **else**
8: $x \leftarrow x'$ 以 $e^{[\text{COST}(x)-\text{COST}(x')]/T}$
9: **结束 if**
10: 若 COST(x')<COST(x)，则将 x记为 x_{best}
11: $T \leftarrow T^{\max}e^{\frac{rt}{t^{\max}}}$
12: **结束 while**
13: **返回** x_{best}

更具体地说，设 x 为通过启发式算法生成的初始解，例如使用 SCAS 按顺序逐个调度作业。SA 通过应用移动算子生成 x 的邻域解 x'。移动算子可视为先执行移除算子、再执行修复算子的组合。移除算子会删除解中的部分内容。具体而言，给定作业的最大移除数量 n^D（$n^D \ge 1$），将在 $[1, n^D]$ 之间随机生成一个数，作为需要移除的作业数量。随后，修复算子根据 DCAS 将未分配作业补充分配至可用节点，从而修复破坏解并生成可行解 x'。具体来说，修复算子首先按照所需节点数量，对破坏算子移除的所有作业进行排序，然后依据 DCAS 按顺序重新插入这些作业。采用 Metropolis准则这一接受准则来决定是否接受新解，即是否应以 x' 替换 x。更新x（或不更新）后，搜索会持续进行，直至达到终止准则。

SA 的性能很大程度上取决于领域专家开发的破坏算子和修复算子的质量。尽管即便是仅以随机方式破坏解中部分内容的简单移除算子，也能得到优质解，但修复算子通常需要实现复杂的启发式算法。NSA 与 SA 唯一的区别在于修复算子。NSA 中的修复算子对应于我们提出的神经网络（NN）模型的学习参数化 θ，该模型通过顺序修复过程修复解。在修复算子的训练过程中，模型会反复接收由特定破坏算子破坏的不完整解。训练的目标是调整模型参数 θ，使模型能够构造高质量解。对于实际应用中经常需要求解具有相似特征的实例的问题而言，这项能力尤为重要。这也意味着，如果遇到的实例特征发生变化，可以重新训练 NSA，从而避免设计新算子所需的大量人力投入。

B. 学习修复解

修复一个破坏解的问题可以表述为一个强化学习问题：智能体（一个学习得到的模型）在多个离散时间步内与环境（一个不完整的解）交互。在每个时间步，模型从 J 中选取一个未分配作业，并输出该作业的首个分配节点。随后，根据序列 Q，按照连续分配策略分配该作业。分配完成后，将根据 DCAS 更新序列 Q。重复这一过程，直至 J 中不再有未分配作业。J 中作业的调度序列可以随机生成，也可以采用其他方法。本文按请求节点数降序排列 J 中的作业，然后依次进行选择。下文首先介绍强化学习的一般数学基础，随后各小节聚焦于 TJAP，并详细定义强化学习中的各个组成部分。

在强化学习中，环境通常用马尔可夫决策过程（MDP）来建模。马尔可夫决策过程是一种用于建模决策的离散时间随机控制过程。形式上，MDP 可表示为一个四元组 $M = (S, A, P_a, R_a)$，其中 S是有限状态集；A 是有限动作集；$P_a(s, s')$ 是采取动作 a 时，状态 s 转移到 s' 的概率（由于在我们的情形中环境是确定性的，因此该概率设为 1）；$R_a(s, s')$ 是从状态 s 转换到 s'后获得的奖励。

强化学习训练算法的目标是学习一个策略π，该策略将每个状态与一个特定动作关联起来，以最大化期望累积奖励。对于状态 s_0，我们可以根据以下公式计算策略 π 的数值 V^π：

$$V^\pi(s_0) = E^\pi\left[\sum_{t=0}^{+\infty}\gamma^t R(s_t, a_t)\right]$$

其中，$V^\pi(s_0)$ 表示对通过采样策略中的动作 V^π 而获得的可容许轨迹分布（s_0，a_0，r_0，s_1，a_1，r_1，.....）求期望

；γ 表示折扣率，这是一种超参数，用于调节智能体向未来展望的范围。如果没有其他策略能产生更高的回报数值，则该策略为最优策略。

1) 状态表示：回想一下，修复算子实际上是逐个调度作业（见第V-B节开头）。考虑到马尔可夫决策过程中的当前时间步为 t，且空闲节点序列遵循 DCAS 更新为 Q。设 n^Q 为 Q中的节点数。粗略而言，状态表示可以是一个 $\left(n^Q \times \beta\right)$ 矩阵，其中 β 是 Q中每个节点的属性数。不过，序列Q 会根据 DCAS 动态更新。当强化学习模型架构包含全连接层时（见后文），状态规模应保持固定。为此，我们的状态表示为一个（$q \times \beta$）矩阵，其中 q 是给定的常数参数。如果 $n^Q \geq q$，则将 Q 中的前 q 个节点纳入状态表示；如果 $n^Q < q$，则选取 Q 中的所有节点，并将其余元素填充为 0。

Q 中的每个节点都有 4 个属性（即 β=4）：(1) 父节点的交换机。(2) 该节点所属的 Pod。(3) 当前调度作业请求的节点数信息。当前调度作业是指强化学习需要在当前时间步 t 决定其首个分配节点的作业。(4) 其他未分配作业请求的节点总数信息。具体而言，对于前两个属性，状态矩阵中相应的值可分别表示为父节点的交换机和 Pod 的归一化标识。对于后两个属性，如果该节点处于空闲状态，状态矩阵中相应的值可分别表示为 $n_j/n^{\max}$ 和 $\sum_{j' \in J' \setminus \{j\}} n_{j'} / \left[(|J'|-1)\, n^{\max}\right]$，其中 $n^{\max}$ 表示作业请求的最大节点数；j 指当前调度作业；J' 表示 J 中未分配作业的集合；否则，相应的值设为 0。

2) 动作选择：对于每个调度作业，动作是选择其首个分配节点。一旦确定首个分配节点，该作业的分配节点就会根据序列Q，按照连续分配策略确定。强化学习的动作空间由 $A=\{0,1,\ldots,q-1\}$定义。当$n^Q < q$时，$\{n^Q,\ldots,q-1\}$ 中的动作属于非法动作。可以采用不同技术来跳过非法动作。一种方法是，当智能体执行一个非法动作时给予负奖励，并希望智能体能够学会识别这些动作。由于这种方法要求智能体同时学习区分合法/非法动作以及优劣动作，从而使问题更加复杂，因此效果较差。

我们的方法是在神经网络的输出上使用掩码，将非法动作值转换为一个较小的负数，即可表示的最小数值。当使用softmax函数时，非法动作的概率趋近于零。该技术此前已有研究，并且似乎能够取得最佳效果 [21]。

3）奖励函数：请注意，移除算子移除的作业数量是在$[1, n^D]$之间随机生成的。如果随机数为 n，则需要由修复算子调度 n 个作业。对于前（$n-1$）个作业，奖励为 0；对于第 n 个作业，奖励按 n 个作业的 CH成本的倒数计算。由于 CH成本越大，奖励越小，因此取其相反值。

4）模型架构：全连接网络。全连接网络（FCN） [22]由多个全连接层组成，每一层中的每个神经元都与另一层中的每个神经元相连。FCN 的主要优势在于其“不依赖结构”，这意味着无需对输入作出任何特殊假设。尽管 FCN 不依赖结构这一特点使其适用范围非常广泛，但与针对特定问题进行优化的专用网络相比，这类网络的表现往往较差。

卷积神经网络。FCN 不利用像素之间的位置信息。对于图像识别问题（高性能计算集群可视为图像，节点可视为图像中的像素），每个像素都与其周围的像素密切相关，而与距离较远的像素关联较弱。如果某个神经元与上一层的所有神经元相连，那么在学习每个连接的权重后，最终可能会发现大量权重的数值非常小，也就是说，这些连接实际上并不重要，无疑会导致训练效率低下。卷积神经网络（CNN）架构[18] 明确假设输入为图像，因此可以将像素的位置属性编码到模型架构中。CNN 采用局部连接，每个神经元不再与上一层的所有神经元相连，而只与少数神经元相连。同时，CNN 采用权重共享。一组连接可以共享相同的权重，而不是每个连接都使用不同的权重，从而减小参数规模。

5）近端策略优化：我们使用近端策略优化（PPO） [23] 作为强化学习算法来训练智能体。PPO采用演员-评论家架构，处于基于策略的强化学习与基于价值的强化学习的交汇处，因此策略（演员）会沿着价值函数（评论家）所指示的方向进行优化。针对任何特定策略 π_θ 计算出的更新步骤，只能保证在 θ 附近的邻域内具有预测性，这是基于策略的算法 [24]的一项基本属性。在参数空间中，基于策略的算法会使策略更新保持接近。然而，即使参数空间中看似微小的差异，也可能产生重大影响并降低策略性能。PPO力求在不导致性能崩溃的前提下，实现尽可能大的改进。为此，它会针对裁剪目标函数执行一系列随机梯度上升步骤。在这种情况下，损失函数 L定义为：

$$L = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} D^{\pi_{\theta_k}}(s,a),\ \ g(\epsilon, D^{\pi_{\theta_k}}(s,a))\right)$$

$$g(\epsilon, D) = \begin{cases} (1+\epsilon)D & D \geq 0 \\ (1-\epsilon)D & D < 0 \end{cases}$$

其中， θ 和 θ_k 分别是新策略和旧策略的参数； ε 是一个（较小的）超参数，大致表示新策略与旧策略之间允许偏离的幅度； D^{π_θ} 是优势函数的估计值。

VI. 结果

本节介绍用于验证所提出模型并评估 NSA 性能的实验。第六节 A 介绍所有实例所依据的高性能计算集群网络及其参数设置。第六节 B 至第六节 E 分别比较强化学习、模拟退火和 NSA 的不同神经网络架构，SCIP 与 NSA，以及不同的调度方法。所有算法和仿真均使用 Python 编写，并在一台配备 2 个英特尔至强 6226 处理器和一块 Nvidia Tesla V100-PCIE 图形处理器的服务器上运行。最后，本项目开发的所有软件和测试实例（可用于复现本节所呈现的所有结果）均以开源形式提供，网址为 https://github.com/lzk23/hpcschedule。

A. 高性能计算集群网络和参数设置

高性能计算集群网络采用具有基数 $k = 20$ 的 1000 节点修剪胖树结构。工作负载由 10^5 个作业组成，这些作业随机生成，请求节点数为 $[1, 40]$，处理时间为 $[10, 1800]$。除非特别说明，其他参数均按表 I 所示设置。

B. 强化学习的神经网络架构比较

本节比较基于两种不同神经网络架构（即全连接网络和卷积神经网络）的强化学习算法，具体内容见第V-B节。此外，如表 II 所示，分别基于全连接网络和卷积神经网络设计了三种具有不同网络规模（以参数数量表示）的神经网络架构。基于全连接网络的神经网络由全连接（线性层） [22], 组成，而基于卷积神经网络的神经网络由一维卷积层 [25] 和全连接层组成。特别是在基于卷积神经网络的神经网络中，卷积层和全连接层之间有一个展平层（展平），用于将多维数据转换为一维数据。Tabh [26] 用作激活函数。请记住，PPO采用演员-评论家架构。表 II 仅提供与行动者模型相关的信息。行动者模型和评论家模型的神经网络架构通常只在最后一层有所不同。例如，对于第一个基于全连接网络的神经网络，将“Linear(64,100)”替换为“Linear(64,1)”，即可得到相应的评论家模型神经网络架构，从而仅输出一个评论家数值。

图 4 比较了 RL 与六种神经网络的学习曲线，其中曲线采用移动平均进行了平滑处理，以提高可读性。为便于记述，三种基于 FCN 的神经网络分别表示为 FCN-1、FCN-2 和 FCN-3。同样，三种基于 CNN 的神经网络分别表示为 CNN-1、CNN-2 和 CNN-3。可以看出，CNN-1、CNN-2 和 CNN-3 之间的性能存在明显差异，而 FCN-1、FCN-2 和 FCN-3 之间则没有明显差异。此外，基于 CNN 的模型优于基于 FCN 的模型。即使是性能最差的

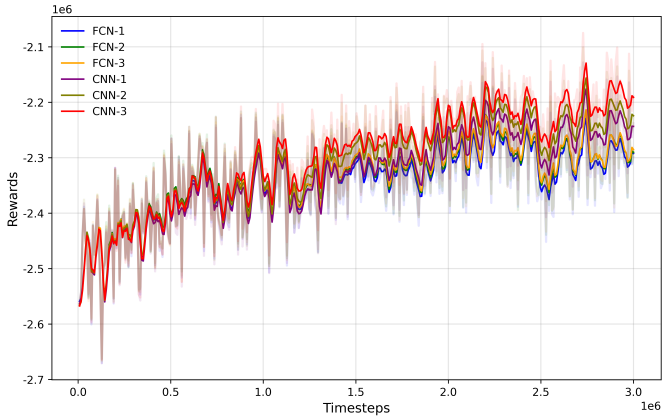


图 4. RL采用不同模型架构时学习曲线的比较。

基于 CNN 的模型（即 CNN-1），其性能也优于性能最佳的基于 FCN 的模型（即 FCN-3）。尽管 CNN-1 的参数数量少于 FCN-3。

C. 模拟退火与 *NSA* 的比较

本节将比较 NSA 与模拟退火，以评估 NSA 相较于模拟退火的改进。NSA 修复算子中使用的强化学习神经网络架构为 CNN-3。比较包含 10 个时段，每个时段包含 100 个动态生成的实例。具体而言，每个时段开始前，高性能计算集群中的所有节点均初始化为空闲状态。随着时间推移，作业会持续到达。在每个决策周期，分别使用 NSA 和模拟退火求解 TJAP问题。同时，根据 NSA 得出的结果更新等待队列和集群的状态，从而确保 NSA 和模拟退火求解的是同一个实例。为方便表示，迭代次数为 $t^{\max}$ 的 NSA（模拟退火）记作 NSA-$t^{\max}$（SA-$t^{\max}$）。例如，“SA-500”指迭代 500 次的模拟退火算法。表 III 展示了比较结果，其中“成本”和“时间”两列分别表示每个时段 100 个实例的平均 CH成本和平均求解时间；“序列”表示根据 SCAS 逐个调度作业而得出的初始解；“改进”表示 NSA-500 相较于 SA-1000 在解的质量方面的提升百分比。由于 NSA 中嵌入了神经网络计算，在迭代步数相同的情况下，NSA 的计算时间长于模拟退火。为评估在计算时间相同的情况下 NSA 相较于模拟退火的性能提升，表 III 给出了 SA-1000 的结果及“改进”数据。

表 III 显示，NSA-500 和 SA-1000 的求解时间大致相当。因此，我们选择将 NSA-500 与 SA-1000 进行比较。从“改进”列可以看出，NSA-500 在所有时段的性能都优于 SA-1000。十个时段的平均改进幅度为 1.34%。为了体现基于不同神经网络的带修复算子的神经模拟退火算法的改进效果

表 I 参数设置。

参数	定义	数值
$T^{\max}\left(T^{\text{最小值}}\right)$	模拟退火和 NSA 的最大（最小）温度	2500(2.5)
$t^{\max}$	模拟退火和NSA的最大迭代步数	500
c	一个CH成本	1000
n^D	移除算子中作业的最大移除数量	2
q	状态表示中使用的常数参数	100
τ	每个时间周期的长度	60s

表 II 强化学习的神经网络架构。

神经网络架构	1	2	3
FCN	线性层(400, 64) 双曲正切函数() 线性层(64,64) 双曲正切函数() 线性层(64, 100)	线性层(400, 64) 双曲正切函数() 线性层(64,64) 双曲正切函数() 线性层(64,64) 双曲正切函数() 线性层(64, 100)	线性层(400, 96) 双曲正切函数() 线性层(96,64) 双曲正切函数() 线性层(64, 100)
参数数量	42724	46884	60804
CNN	一维卷积层(4, 4, 10, 1,0) 双曲正切函数() 一维卷积层(4,4,5,2,0) 展平层() 线性层(176,256) 线性层(256,100)	一维卷积层(4, 8, 10, 1,0) 双曲正切函数() 一维卷积层(8,4,5,2,0) 展平层() 线性层(176,256) 线性层(256,100)	一维卷积层(4, 16, 10, 1,0) 双曲正切函数() 一维卷积层(16,8,5,2,0) 展平层() 线性层(352,256) 线性层(256,100)
参数数量	37248	42097	54641

表 III 模拟退火与 NSA 结果对比。

轮次	Seq.	模拟退火-500		模拟退火-1000		NSA-500		改进
	Cost	Cost	时间/秒	Cost	时间/秒	Cost	时间/秒	
1	99152	87322	0.8	86958	1.5	85596	1.5	1.57%
2	105087	91662	0.7	90810	1.5	89026	1.5	1.96%
3	110728	97723	0.8	97006	1.6	96215	1.6	0.82%
4	102931	90909	0.9	89797	1.9	89127	1.9	0.75%
5	105780	94016	0.7	92483	1.5	91471	1.5	1.09%
6	106006	91388	0.7	90084	1.5	88919	1.5	1.29%
7	105448	93022	0.7	91804	1.5	91060	1.5	0.81%
8	104471	84483	0.8	83266	1.6	82668	1.4	0.72%
9	107885	97214	0.8	96337	1.5	94642	1.5	1.76%
10	102747	90393	0.7	89552	1.4	87163	1.5	2.67%
Avg.	105023.5	91813.2	0.76	90809.7	1.55	89588.7	1.54	1.34%

与 SA 相比（即表 II 所示的六种神经网络），我们分别使用基于不同神经网络的神经模拟退火算法和 SA，对均匀分布在十个时段中的一千个实例进行求解。此外，NSA 的迭代次数分别设为 500、600、700、800、900 和 1000。为保持求解时间一致，SA 的迭代次数分别设为 1000、1200、1400、1600、1800 和 2000。图 5 比较了六种神经网络在六种不同迭代次数下的平均 CH 成本。尽管采用 FCN-1 的 NSA-500 性能不如 SA-1000（对应的改进幅度为 -0.27%），但在其他所有实例中，神经模拟退火算法的 CH 成本都更低（改进幅度均为正）。尤其是，使用 CNN-3 网络能获得更大的改进比使用其他神经网络。

D. SCIP 与 *NSA* 的比较

为体现NSA算法相较于求解器SCIP的优势，将用于比较SA与NSA的实例（见表 III）也作为输入传入模型M SCAS，并由SCIP求解。结果如表 IV所示，其中“SCIP（无时间限制）”表示SCIP的求解时间不受限制；“SCIP（时间限制=30s）”表示将SCIP的求解时间限制为30秒；“取消”指每个轮次中未分配作业占所选作业的百分比。有时，尽管资源分配充足，但由于SCIP无法在有限时间内获得可行解，部分作业仍未被分配。“改进”表示

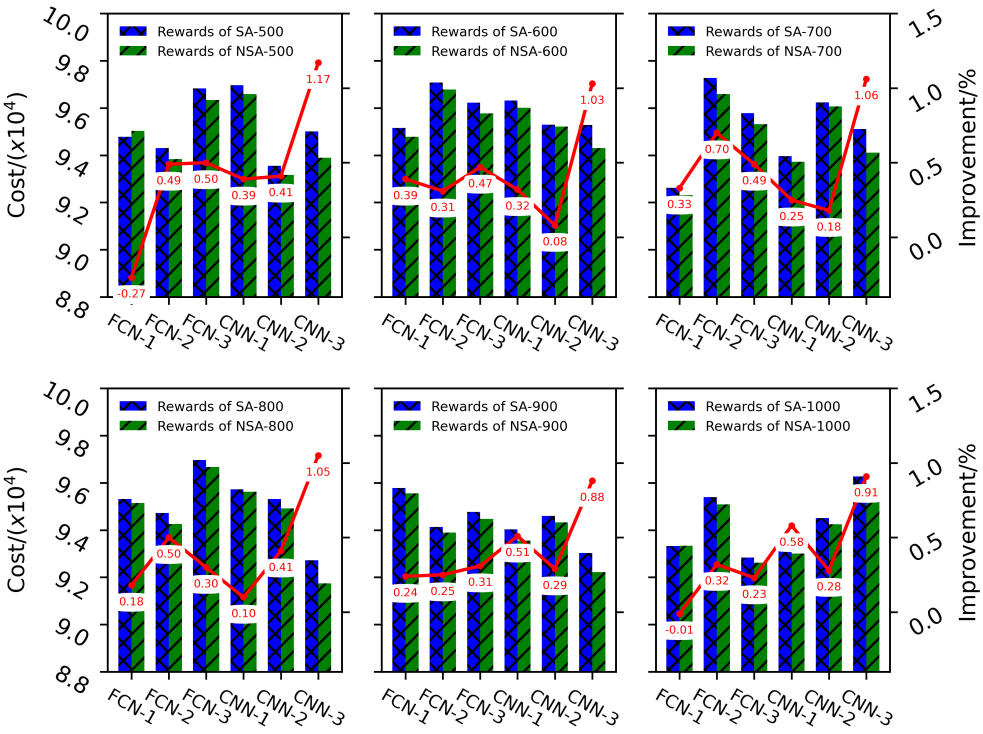


图 5. 不同神经网络的 NSA 与模拟退火的比较。

NSA相对于“SCIP（无时间限制）”所实现的解质量改进百分比。

表 IV 显示，SCIP 的效率无法满足实际需求。当求解时间限制为 30 秒时，部分实例无法获得可行解，也就是说，仍有一些作业尚未分配。当 SCIP 的求解时间不受限制时，平均求解时间高达 18.96 秒；对于实时应用而言，这一响应时间过长。根据我们的观察，SCIP 的求解时间与空闲节点数量及已分配作业数量密切相关。相比之下，在所有时段中，NSA 的平均求解时间都不到 2 秒。关于解的质量（见“改进”列），NSA 得到的解劣于“SCIP（无时间限制）”的解。不过，在所有时段中，解的质量差异均不超过 10%。

E. 不同调度方法的比较

本节将基于窗口的调度方法与逐作业方式（即先到先服务和回填）进行比较。此外，将基于窗口的调度方法中每个时间周期的长度分别设置为10秒、20秒、30秒、40秒、50秒和60秒，以分析其对CH 成本和等待时间（分配时间减去到达时间）的影响。每种方法均测试10个时段。在每个时段中，分别使用不同方法为连续到达的300个作业进行分配。图6展示了结果。需要指出的是，CH成本会影响作业的处理时间，进而影响平均等待时间。然而，CH

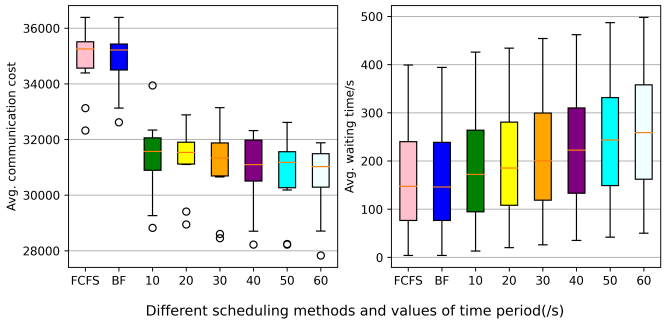


图6. 不同调度方法及时间周期长度的结果比较。

成本与运行时间之间的具体关系不在本研究范围内，因此仍假设作业的处理时间不会随CH成本的变化而改变。

图6表明，与先到先服务和回填方法相比，基于窗口的调度方法能够以增加平均等待时间为代价，降低平均CH成本。此外，随着每个时间周期长度的增加，平均CH成本呈下降趋势，而平均等待时间则会增加。

VII. 结论

本文研究了基于胖树的高性能计算系统中的基于窗口的TJAP，旨在降低CH成本。为避免处理QAP这一难题，提出了特殊的SCAS和DCAS。对于SCAS，我们构建了一个

表 IV NSA 与 SCIP 的结果比较。

	SCIP（无时间限制）		SCIP（时间限制=30秒）			NSA		
轮次	Cost	时间/秒	Cost	取消	时间/秒	Cost	时间/秒	改进
1	79597	26.2	79484	0.01	8.5	85596	1.6	-7.54%
2	83863	14.7	81421	0.04	8.5	89026	1.5	-6.16%
3	89422	21.2	86687	0.04	9.2	96215	1.6	-7.60%
4	82598	21.5	82410	0.02	8.7	89127	1.9	-7.90%
5	84438	22.0	80421	0.03	8.1	91471	1.5	-8.33%
6	80905	16.6	80380	0.01	7.9	88919	1.5	-9.91%
7	84358	19.5	85253	0.01	8.1	91060	1.5	-7.94%
8	76331	12.2	75734	0.01	6.3	82668	1.4	-8.30%
9	88091	22.6	87345	0.02	8.5	94642	1.6	-7.44%
10	81320	13.1	82013	0.01	7.9	87163	1.5	-7.19%
Avg.	83092.3	18.96	82114.8	0.02	8.17	89588.7	1.56	-7.82%

0-1整数规划模型，可由求解器直接求解；而对于DCAS，则提出了一种新算法，即NSA。实验表明，与SA相比，在求解时间相同的情况下，NSA能够获得更好的结果。在使用SCIP求解模型$\mathrm{M_{SCAS}}$时，我们发现SCIP的求解时间对求解规模更加敏感，对于TJAP而言难以接受。此外，在限定时间为30秒时，SCIP的解的质量明显劣于NSA。与逐作业调度方法（即先到先服务和回填）相比，平均CH成本可以降低，但需要以增加等待时间为代价。

致谢

本研究部分由中国博士后科学基金资助，合同编号为2022M720931。

参考文献

[1] N. Jain、A. Bhatele 和 L. H. Howell，“预测不同胖树配置的性能影响”，载于 *SC17*：高性能计算、网络、存储与分析国际会议，2017 年。[2] S. A. Smith、C. E. Cromey、D. K. Lowenthal、J. Domke、N. Jain、J. J. Thiagarajan 和 A. Bhatele，“利用自适应流感知路由缓解作业间干扰”，载于 *SC18*：高性能计算、网络、存储与分析国际会议，2018 年。[3] S. A. Smith 和 D. K. Lowenthal，“Jigsaw：面向胖树集群的高利用率、无干扰作业调度器”，载于 *HPDC '21*：第 *30* 届高性能并行与分布式计算国际研讨会，2020 年。[4] A. Bhatele、J. J. Thiagarajan、T. Groves、R. Anirudh 和 D. K. Lowenthal，“基于蜻蜓网络系统的性能变化问题”，载于 *2020 IEEE* 国际并行与分布式处理研讨会（*IPDPS*），2020 年。[5] S. Soner 和 C. Ozturan，“基于整数规划的异构 CPU-GPU 集群调度器：面向 SLURM 资源管理器”，载于《计算机与系统科学杂志》，2012 年。[6] Y. Xu、Z. Zhou、T. Wei、X. Zheng 和 Z. Lan，“通过环面互连系统上的局部性感知调度平衡作业性能与系统性能”，载于*IEEE* 集群计算国际会议，2014 年。[7] J. M. Orduna、F. Silla 和 J. Duato，“一种面向通信感知调度策略的新任务映射技术”，载于 国际并行处理会议论文集，2002 年。[8] T. Chockalingam 和 S. Arunkumar，“映射问题的随机启发式方法：遗传方法”，《并行计算》，第 18 卷，第 10 期，第 1157–1165 页，1992 年。

[9] D. Tsafrir、Y. Etsion 和 D. G. Feitelson，“使用系统生成的预测而非用户运行时间估计进行回填调度”，*IEEE* 并行与分布式系统汇刊，第 18 卷，第 803 页，2007 年 6 月，第 789 页。[10] Z. Zhou、Z. Lan、W. Tang 和 N. Desai，“通过功耗感知作业调度降低 IBM Blue Gene/P 的能耗成本”，载于 并行处理作业调度策略研讨会，2013 年。[11] F. Kaplan、M. Jie 和 A. K. Coskun，“通过智能作业分配优化高性能计算数据中心的通信与冷却成本”，载于 绿色计算会议（*IGCC*），2013 年。[12] Y. Georgiou、E. Jeannot、G. Mercier 和 A. Villiermet，“面向高性能计算应用的拓扑感知资源管理”，载于第 *18* 届分布式计算与网络国际会议论文集，2017 年 1 月，第 1–10 页。[13] Y. Georgiou、G. Mercier 和 A. Villiermet，拓扑感知资源管理的大规模实验。《Euro-Par 2017：并行处理研讨会》，2018 年。[14] B. Yan、Z. Yang、L. Xiao、Y. Zhou、B. Wei 和 Y. Song，“LPMS：一种面向共享高性能计算系统上大规模并行应用的低成本拓扑感知进程映射方法”，载于第 *48* 届并行处理国际会议：研讨会（*ICPP*），第 5–8 页，2019 年。[15] B. Ryu、A. An、Z. Rashidi、J. Liu 和 Y. Hu，“利用深度强化学习实现面向拓扑感知的抢占式作业调度”，载于第 *30* 届计算机科学与软件工程年度国际会议论文集（*CASCON'20*），2020 年。[16] M. Mubarak、C. D. Carothers、R. B. Ross 和 P. Carns，“实现大规模高性能计算网络系统的并行仿真”，*IEEE* 并行与分布式系统汇刊，第 87–100 页，2016 年。[17] P. Qiao、X. Wang、X. Yang、Y. Fan 和 Z. Lan，“应用通信模式、作业放置与网络路由对胖树系统的联合影响”，2018 年，第 1–10 页。[18] M. Ziegelmann，“受约束最短路径及相关问题”，萨尔大学博士论文，2001 年。[19] S. Kirkpatrick、C. D. Gelatt 和 A. Vecchi，“通过模拟退火进行优化”，《科学》，1983 年。[20] A. Hottung 和 K. Tierney，“用于带容量约束车辆路径问题的神经大邻域搜索”，2019 年。[21] S. Huang 和 S. Onta´on，“深入探究策略梯度算法中的无效动作屏蔽”，载于 *arXiv* 预印本 *arXiv:2006.14171*，2020 年。[22] S. Basha、S. R. Dubey、V. Pulabaigari 和 S. Mukherjee，“全连接层对用于图像分类的卷积神经网络性能的影响”，《神经计算》，2020 年。[23] J. Schulman、F. Wolski、P. Dhariwal、A. Radford 和 O. Klimov，“近端策略优化算法”，*arXiv* 预印本 *arXiv:1707 06347*，2017 年。。[24] D. Silver、G. Lever、N. Heess、T. Degris、D. Wierstra 和 M. Riedmiller，“确定性策略梯度算法”，机器学习国际会议，第 387–395 页，2002 年。[25] M. Zeiler 和 R. Fergus，“卷积网络的可视化与理解”，载于 *ECCV 2014*，2014 年。[26] M. Mirhassani 和 B. Zamanlooy，“带双曲正切激活函数的神经网络高效超大规模集成电路实现”，*IEEE*《超大规模集成电路系统》汇刊，2014 年。