基于组播时间扩展图的集合通信编排算法

刘飞[1]，王鹏[2]，麻涵[1]，李烨[1]，李红艳[1]
（1.西安电子科技大学空天地一体化综合业务网全国重点实验室，陕西 西安 710071；2.新加坡科技设计大学，新加坡 487372）

摘　要：集合通信时延已成为大模型分布式训练的瓶颈。针对现有算法拓扑感知不足、调度复杂度高等问题，提出一种基于组播时间扩展图的集合通信编排算法。通过构建组播时变图模型，刻画节点间的拓扑关系及时隙冲突约束，将集合通信编排问题转化为时变图中组播路径搜索问题，实现路由与时隙联合规划。仿真结果表明，所提算法完成时间接近最优，同时显著降低计算复杂度，适用于大规模分布式训练场景。

关键词：集合通信；拓扑感知；时变图；组播调度；分布式训练
中图分类号：TP393.0
文献标志码：A
DOI：10.11959/j.issn.1000–436x.2026062

Multicast time-expanded graph-based collective communication scheduling algorithm

Liu Fei[1], Wang Peng[2], Ma Han[1], Li Ye[1], Li Hongyan[1]
1. State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an 710071, China
2. Singapore University of Technology and Design, Singapore 487372, Singapore

Abstract: Collective communication latency has become a major performance bottleneck in large-scale distributed training. To address the limitations of existing algorithms in terms of insufficient topology awareness and high computational complexity, a topology-aware collective communication scheduling algorithm based on multicast time-expanded graph was proposed. By constructing a multicast time-varying graph model, both network topology constraints and time-slot conflict constraints were explicitly captured. The collective communication scheduling problem was then transformed into a multicast path search problem on the time-varying graph, enabling joint optimization of routing and time slot planning. Simulation results demonstrate that the proposed algorithm achieves near-optimal communication completion time while significantly reducing computational complexity, making it suitable for large-scale distributed training scenarios.

Keywords: collective communication, topology aware, time-varying graph, multicast scheduling, distributed training

0　引言

在大模型分布式训练中，分布在计算中心的计算节点需进行数据交互、规约等操作，节点间的集合通信[1]已成为大模型训练效率的瓶颈因素，据 Domino 报道，集合通信时延占 GPT-3-13B 大模型训练总时间的 17%~43%[2]。近年来，学术界分别从拓扑无感的集合通信算法、拓扑感知的集合通信调度算法等方面开展研究，旨在降低集合通信时延占比。

拓扑无感的集合通信算法假定拓扑为理想拓扑，以集合通信时延最小为优化目标，进行计算与通信的协同编排。常用环形算法进行全规约计算[3-6]，N 个计算节点构成逻辑环，每个节点接收来自上游节点的数据并进行规约计算，并将计算

收稿日期：2025–12–27；修回日期：2026–03–01
通信作者：李红艳，hyli@xidian.edu.cn
基金项目：国家自然科学基金资助项目（No.62371374）
Foundation Item：The National Natural Science Foundation of China (No.62371374)

结果发送至下游节点，经过2(N−1)轮迭代，完成全规约计算。若网络拓扑为理想拓扑，即每轮迭代过程中各节点的传输时延、计算时延均相等，环形算法的集合通信时延可逼近理论下限。若节点间传播时延无法忽略，随着大模型分布式训练节点数量的增加，环形算法的集合通信时延将显著增加。针对这一问题，腾讯提出了分层全规约算法[4-5]，构建分层环，通过缩短单环的长度，降低集合通信时延；索尼提出基于二维环的全规约算法[6]，通过在水平和垂直两个维度同时构建逻辑环，降低集合通信时延；高性能计算接口标准引入了基于二项树的集合通信算法[7]，将数据通过递归的方式，从持有节点转发至未持有节点，使通信步数由环形算法的线性复杂度$O(N)$降低至对数级复杂度$O(\mathrm{lb}N)$，但该算法每轮迭代都需要发送全部数据，单次通信链路负载高，传输时延显著增加；在此基础上，消息传递接口（MPI）标准进一步提出了基于双二叉树的集合通信算法[8]，通过构造两棵逻辑互补的二叉树，使两棵树并行工作，分别承载一半数据量的计算与传输，降低集合通信时延；英伟达在其集合通信数据库2.4（NCCL 2.4）中设计了环树结合的集合通信架构[9]，兼顾了计算时延与通信效率。

拓扑感知的集合通信调度算法依据分布式训练参与节点间的拓扑，设计适配网络拓扑的计算与通信协同编排方案。部分高性能计算中心构建专用拓扑，适配拓扑无感的集合通信算法。意大利欧泰股份有限公司[10]与国际商业机器公司（IBM）[11]均搭建了3维环形的网络架构，让物理网络拓扑适配优化的集合通信环形算法，支撑数据中心实现高性能计算。桑迪亚国家实验室提出了二进制立方体的高性能计算网络拓扑架构[12]，该架构可适配集合通信的多种算法。但专用拓扑的构建较为困难，且网络设备成本高。近年来，多篇文献提出了拓扑感知的集合通信调度算法，通过路由与计算编排，设计与网络拓扑相适配的集合通信方案，动态适配大模型训练需求、算力节点分布及网络拓扑。

微软设计了综合集合通信库（MSCCL）[13]，将物理网络带宽/时延、集合通信需求、大模型训练约束进行联合表征，求解适配网络与大模型训练的集合通信编排方案，降低了集合通信时延；微软和得克萨斯大学联合提出了拓扑感知的集合通信架构（TACCL）[14]，依据拓扑约束和集合通信需求将集合通信问题建模为混合整数线性规划（mixed integer linear programming，MILP）问题，并针对该问题设计了路由、时隙分别调度的快速算法，该算法复杂度低，适配集合通信算法的重构需求。宾夕法尼亚大学与openAI将集合通信编排问题归纳为多商品流问题[15]，模型考虑了链路容量与时延约束，并考虑了集合通信过程中数据传输时序约束，求解该问题可获得适配网络拓扑的最优集合通信方案，即最小化大模型训练时延。综上所述，拓扑感知的集合通信算法已成为降低集合通信时延的重要手段之一，目前面临的挑战是拓扑感知的集合通信编排算法复杂度高，难以适应算力资源时变环境。

为解决上述问题，本文提出了基于组播时间扩展图（multicast time-expanded graph，MTEG）的集合通信编排算法（MTEGCCA）。首先构建了组播时间扩展图模型，表征了算力节点之间的拓扑约束，以及集合通信过程中的时隙冲突约束，为无冲突集合通信编排提供了保障。针对集合通信普遍存在的一对多的传输需求，设计了MTEGCCA，利用组播机制降低集合通信时延，解决了集合通信过程中“从哪传、何时传”的问题。本文的主要工作如下。

1) 提出了MTEG模型，将网络拓扑、链路带宽、链路时隙等多维资源进行统一建模，精准表征了集合通信参与节点之间的拓扑约束关系、差异化的链路带宽、多节点业务传输冲突约束，为构建无冲突的集合通信提供了模型基础。

2) 设计了MTEGCCA，将指数级的MILP集合通信编排问题转化为MTEG上组播路由与时隙联合规划问题，在MTEG中搜索组播路由即可完成路由与时隙的联合规划，依托组播树可有效降低集合通信时延，依托时隙化的MTEG，保障集合通信编排无冲突。

3) 在不同规模的典型拓扑环境（如DGX1、DGX2、NDv2、AMD）中[15]进行了对比实验。结果表明，所提算法的集合通信完成时间逼近最优解，集合通信编排算法的求解时间与微软[14]和openAI[15]提出的算法相比降低了一个数量级，算法有效带宽提升30%~80%，具有良好的扩展性。

1　数学建模

1.1　集合通信编排

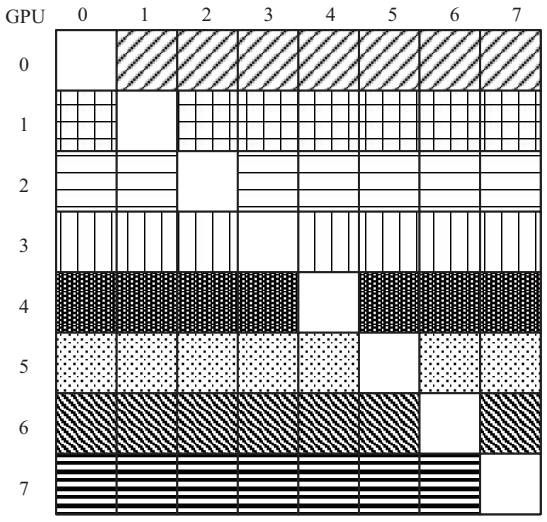
集合通信编排是指在给定网络拓扑、集合通信操作需求和传输数据规模的前提下，设计合理的集合通信编排方案，即确定数据在网络中的传输路径与传输时序，从而在满足链路带宽等资源约束的条件下，实现集合通信的完成时间最小化等优化目标。在分布式训练中，全收集（AllGather）是集合通信中最具代表性的操作之一，因此常被用作研究集合通信编排问题的典型场景[13-15]。AllGather 的目标是在一组参与训练的节点之间完成数据的全量交换，实现全数据同步。具体而言，在初始阶段，每个节点仅持有自身生成的数据块。AllGather 操作通过组织多轮数据传输，使每个节点将自身数据通过广播（Broadcast）发送至其他节点，最终保证通信组内的每个节点均获得完整的数据集合。

图 1 展示了在 DGX1 拓扑中执行集合通信中 AllGather 的操作过程。在上述 AllGather 操作中，各节点之间的数据交换需求可用需求矩阵进行刻画。如图 1(a)所示，矩阵中非空元素 $d_{ij}(i \neq j)$ 代表源图形处理单元（GPU）节点 i 需向目的 GPU 节点 j 传输大小为 d 的数据总量。图 1(b)描述了数据在网络中的传输路径与时序关系。由于物理链路在同一时刻只能承载有限的数据传输任务，因此仅确定数据的传输路径是不充分的，还必须进一步编排不同数据块在链路上的传输时序。以链路(0,1)为例，根据需求矩阵，GPU0 与 GPU2 均需向 GPU1 发送数据。假设数据都选择链路(0,1)传输，如果没有确定时序，链路(0,1)有可能会先传 GPU2 的数据再传 GPU0 的数据，GPU0 的数据需要等待 GPU2 的数据传完后才传，导致 GPU2 的数据在 t=1 时到达 GPU1，GPU0 的数据在 t=2 时到达 GPU1。实际上，在 t=0 时传输 GPU0 的数据给 GPU1，同时 GPU0 接收 GPU2 的数据，再在 t=1 时转发 GPU2 的数据至 GPU1，可更早地完成传输。

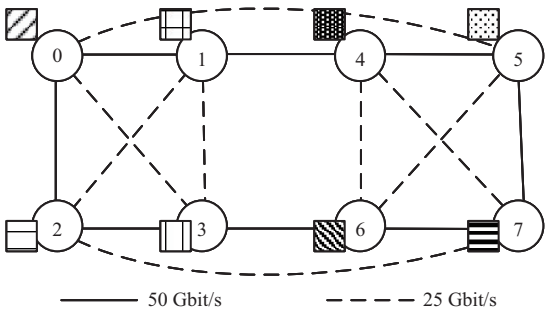
由此可见，集合通信编排不仅涉及“从哪传”的路径选择问题，还涉及“何时传”的时序安排问题。不同数据块在链路上的发送顺序将直接影响整体通信完成时间，若缺乏合理的时序规划，将导致链路冲突和带宽利用率下降。

1.2　数学建模

针对上述集合通信编排问题，本文将其建模为 MILP 问题。具体地，本文采用无向图 $\mathcal{G}=(\mathcal{V},\mathcal{E},\mathcal{B})$ 表示网络拓扑，其中 $\mathcal{V}=\{v_1,v_2,\cdots,v_n\}$ 为网络中的节点集合，$\mathcal{E}=\{(u,v)|u,v\in\mathcal{V},u\neq v\}$ 为网络中链路集合，$\mathcal{B}=[B_{(u,v)}]\in\mathbb{R}^{V\times V}$ 为链路带宽资源集合，其中 $B_{(u,v)}$ 表示链路 (u,v) 的可用带宽资源，单位为 Gbit/s。将节点 u 到节点 v 传输的总数据量记为 $d_{u,v}$，单位为 GB。为支持数据并行传输，本文将 $d_{u,v}$ 分为 C 个小数据块，单个数据块大小为 μ，即 $C=\left\lceil\frac{d_{u,v}}{\mu}\right\rceil$，并将数据块集合记为 $\mathcal{C}=\{1,2,\cdots,C\}$。在构建具体数学模型之前，先介绍一些概念，以明确本文所考虑的场景。



(a) 需求矩阵



(b) DGX1 拓扑

图1　集合通信示例

1) 时隙长度τ。时隙长度定义为单个数据块在网络中带宽最大的链路上传输所需的传输时延，即$\tau=\frac{\mu}{\max_{(u,v)\in\mathcal{E}}B_{u,v}}$。在该定义下，单个数据块在一个时隙内至多可传输一跳。假设一次集合通信操作的完成时间为D，选取足够长的时间段$T(T\geqslant D)$，并将其分为$H=\left\lceil\frac{T}{\tau}\right\rceil$个等长的时隙，时隙集合记为$\mathcal{H}=\{0,1,2,\cdots,H\}$。

2) 链路容量ω。链路容量定义为单位时间内链路能够传输的数据块数量，由链路带宽B和数据块大小μ共同决定，即$\omega=\frac{B}{\mu}$。相应地，在单个时隙内，每条链路所能承载的数据块数量至多为一个。

3) 链路时延。本文采用“$\alpha-\beta$”模型来刻画链路上的时延，具体地，α表示链路上的固定时延，通常为传播时延和处理时延；β表示链路上的传输时延。特别地，采用$\delta_{(u,v)}$表示链路(u,v)上固定时延所需的时隙数，即$\delta_{(u,v)}=\left\lceil\frac{\alpha_{(u,v)}}{\tau}\right\rceil$；采用$l_{(u,v)}$表示链路$(u,v)$上传输一个数据块所需的时隙数，即$l_{(u,v)}=\left\lceil\frac{1}{\omega_{(u,v)}\cdot\tau}\right\rceil$。因此，采用链路$(u,v)$传输一个数据块需要的总时隙数为$L_{(u,v)}=\delta_{(u,v)}+l_{(u,v)}$，所有链路时延组成时延矩阵$\boldsymbol{L}$。

为了精准地刻画网络资源的占用情况和数据传输情况，定义如下变量。

1) $x_{s,(u,v),c,t}\in\{0,1\}$，表示源自节点$s$的第$c$个数据块是否在时隙$t$通过链路$(u,v)$进行传输，若选择链路$(u,v)$传输，则值为1，否则为0。

2) $y_{s,v,c,t}\in\{0,1\}$，表示源自节点s的第c个数据块是否在时隙t开始之前已经缓存在节点v，若已经缓存，则值为1，否则为0。

3) $D_{s,v,c}\in\mathbb{Z}$，表示源自节点s的数据块c到达目的节点v的端到端时延对应的时隙数。

表1为本文涉及的主要数学符号及其含义。至此，本文已经具备将集合通信路由问题建模为MILP问题的所需的全部要素。具体而言，本文的优化目标为最小化集合通信中执行一次AllGather操作所需要的时间，定义为从任一节点s发送出第一个数据块开始，到所有节点均成功接收到其所需数据块为止的时间区间。

表1 本文涉及的主要数学符号及其含义

数学符号	含义
$\mathcal{V},\mathcal{E},\mathcal{B}$	节点集合、链路集合、链路带宽集合
$d_{u,v}$	节点u至节点v待传输的数据总量，单位为GB
μ	单个数据块大小
$\mathcal{C},C$	数据块集合、数据块数量
τ,T	时隙长度、选定足够长的时间段
$\mathcal{H},H$	时隙集合、时隙集合中的时隙个数
ω	链路单位时间可承载的数据块数量
$\alpha,\delta_{(u,v)}$	链路传播与处理时延之和、α对应的时隙个数
$\beta,l_{(u,v)}$	链路传输时延、传输时延对应的时隙个数
$L_{(u,v)},\boldsymbol{L}$	链路(u,v)总时延所对应的时隙数、链路总时延矩阵
$x_{s,(u,v),c,t}$	源自节点s的第c个数据块是否在时隙t通过链路(u,v)进行传输，若选择链路(u,v)传输，则值为1，否则为0
$y_{s,v,c,t}$	源自节点s的第c个数据块是否在时隙t开始之前已缓存在节点v，若已缓存，则值为1，否则为0
$D_{s,v,c}$	源自节点s的数据块c到达目的节点v的端到端时延对应的时隙数
$z_{s,v,c,t}$	节点v是否在时隙t首次接收来自节点s的数据块c，如果是首次接收，则为1，否则为0

该MILP问题可建模为

$$\min_{s\in\mathcal{V}}\max_{v\in\mathcal{V},c\in\mathcal{C}}D_{s,v,c}\tag{1}$$

且满足如下约束条件。

1) 初始状态约束。式(2)表示在初始时刻，每个节点上都缓存了自己的数据块。式(3)表示在初始时刻，每个节点还未缓存其他节点的数据块。

$$y_{s,s,c,0}=1,\quad\forall c\in\mathcal{C},\quad\forall s\in\mathcal{V}\tag{2}$$

$$y_{s,v,c,0}=0,\quad\forall c\in\mathcal{C},\quad\forall v\in\mathcal{V}\backslash\{s\}\tag{3}$$

2) 流守恒约束。式(4)表示在选取的整个时间区间内，每个节点s需要将其上的每个数据块至少发出一次，否则其他节点将不可能收到相应的数据，从而导致集合通信失败。每个节点都需要将其全部数据块发送给其他节点，而其他节点既可以从该节点接收相应的数据块，也可以通过向已经收到数据块的中间节点转发来获取相应的数据块。所以节点上每个数据块的发送次数不需要大于其余节点

数，只需确保数据块发出即可。

$$\sum_{t=0}^{H}\sum_{v:(s,v)\in\mathcal{E}} x_{s,(s,v),c,t} \geqslant 1, \quad \forall c\in\mathcal{C},\forall s\in\mathcal{V} \tag{4}$$

式(5)用于保证数据在传输过程中满足因果关系，即节点仅在已经持有数据块的情况下，才允许在下一时隙发起对该数据块的发送操作。

$$y_{s,u,c,t} + \sum_{\substack{u:(u,v)\in\mathcal{E}\\ t-L_{(u,v)}\geqslant 0}} x_{s,(u,v),c,t-L_{(u,v)}} \geqslant x_{s,(u,v),c,t+1},$$

$$\forall (u,v)\in\mathcal{E},\forall t\in\mathcal{H},\forall c\in\mathcal{C},\forall s,u\in\mathcal{V} \tag{5}$$

3) 缓存约束。式(6)用于保证数据块在节点上缓存的因果关系。具体而言，一个节点v在时隙t开始之前是否已经缓存了源自节点s的数据块c取决于以下两种情况：①节点v在前一时隙$t-1$就已经缓存了该数据块，即数据块在节点处保持；②节点v在时隙$t-1$通过其入边(u,v)接收到该数据块，这要求节点u在时隙$t-L_{(u,v)}-1$发起数据块c的传输，$t-L_{(u,v)}-1\geqslant 0$用于保证传输发起时刻不早于初始时隙。

$$y_{s,v,c,t} = y_{s,v,c,t-1} + \sum_{\substack{u:(u,v)\in\mathcal{E},\\ t-L_{(u,v)}-1\geqslant 0}} x_{s,(u,v),c,t-L_{(u,v)}-1},\forall v\in\mathcal{V},$$

$$\forall t\geqslant 1,\forall c\in\mathcal{C} \tag{6}$$

4) 容量约束。式(7)保证在任意时隙内，链路上承载的数据块都不会超过链路容量，其中$\omega_{(u,v),t}$表示链路(u,v)在时隙t的容量。

$$\sum_{\forall s\in\mathcal{V}}\sum_{c=1}^{C} x_{s,(u,v),c,t} \leqslant \omega_{(u,v),t}, \quad \forall (u,v)\in\mathcal{E},\forall t\in\mathcal{H} \tag{7}$$

5) 接收约束。式(8)保证所有节点最终均缓存其所需的全部数据块，从而确保集合通信的成功。

$$\sum_{t=0}^{H} y_{s,v,c,t} \geqslant 1, \quad \forall c\in\mathcal{C},\forall s,v\in\mathcal{V} \tag{8}$$

6) 接收时间追踪约束。为了精确刻画每个节点首次接收到各数据块的时隙，从而准确地获取集合通信完成时间，本文引入一个辅助二进制变量$z_{s,v,c,t}$，用于表示节点v是否在时隙t首次接收来自源节点s的数据块c，如果是首次接收，则为1，否则为0。

式(9)保证首次接收时间的唯一性，每个接收节点对每个数据块只能在一个时隙被标记为首次接收。

$$\sum_{t=1}^{H} z_{s,v,c,t} = 1, \quad v\neq s,\forall v,s\in\mathcal{V},\forall c\in\mathcal{C} \tag{9}$$

式(10)~式(13)用于标记节点v是否在时隙t首次接收源自节点s的数据块c。对于时隙$t\geqslant 1$，当且仅当节点v在时隙t收到了数据块，但在前一时隙还没有收到该数据块时，则标记节点v在时隙t首次接收源自节点s的数据块c。在初始时隙$t=0$，节点除了自身数据块外并没有缓存其他节点的数据块，因此首次接收时间为0。

$$z_{s,v,c,t} \leqslant y_{s,v,c,t},\forall s,v\in\mathcal{V},\forall c\in\mathcal{C},\forall t\in\mathcal{H}\backslash\{0\}\} \tag{10}$$

$$z_{s,v,c,t} \leqslant 1-y_{s,v,c,t-1},\forall s,v\in\mathcal{V},\forall c\in\mathcal{C},\forall t\in\mathcal{H}\backslash\{0\} \tag{11}$$

$$z_{s,v,c,t} \geqslant y_{s,v,c,t}-y_{s,v,c,t-1},\forall s,v\in\mathcal{V},\forall c\in\mathcal{C},\forall t\in\mathcal{H}\backslash\{0\} \tag{12}$$

$$z_{s,v,c,0} = y_{s,v,c,0},v\neq s,\forall v,s\in\mathcal{V},\forall c\in\mathcal{C} \tag{13}$$

若节点v在时隙t首次接收到数据块c，则其对应的端到端时延对应的时隙数$D_{s,v,c}$=t。这一关系通过式(14)进行刻画。

$$D_{s,v,c} = t\cdot z_{s,v,c,t}, \quad \forall s,v\in\mathcal{V},\forall c\in\mathcal{C},\forall t\in\mathcal{H}\backslash\{0\} \tag{14}$$

式(2)~式(14)仅涉及二进制变量$x_{s,(u,v),c,t}$，$y_{s,v,c,t}$,$z_{s,v,c,t}$和整数变量$D_{s,v,c}$，且所有的约束式都是线性的，因此本文构建的数学模型是一个MILP问题。假设$|\mathcal{V}|$、$|\mathcal{E}|$、C和H分别表示网络中的节点数、链路数、最小数据块数和时隙数，则该MILP问题总共含有$O(CH|\mathcal{E}|)$个决策变量，由于模型中引入了大量二进制决策变量，其解空间规模呈指数级增长，若采用穷举搜索，求解复杂度将高达$2^{O(CH|\mathcal{E}|)}$，在实际求解中将占用大量时间，甚至超过规划结果所需的通信时间，难以规模化使用。为了克服MILP计算复杂度高的问题，本文提出了基于图的启发式算法，该算法在显著降低求解开销的同时，能够高效地逼近最优性能。

2　MTEGCCA

2.1　组播时间扩展图的构建算法

集合通信中的AllGather机制同时受到多种时间相关约束（如流守恒、节点缓存及时隙化收发约束）的共同制约，其调度过程不仅涉及数据在网络中的传输路径选择，还依赖于不同数据块在时间维度上的发送与接收顺序安排。因此，AllGather的调度问题本质上是一个同时耦合网络拓扑、链路带宽及时隙资源的时空联合优化问题。

为了统一刻画上述多维资源之间的内在关联关系，并将复杂的时序依赖显式地纳入模型，本文引入时间扩展图的建模思想，将AllGather的最小完成时间目标等价转化为时间扩展图上的最小组播树构建问题，从而将复杂的集合通信调度统一映射为图上的路径规划问题。基于此，本文提出了MTEG模型。

MTEG是一种将原始网络沿时间轴离散化展开的时空图模型。其基本思想是在给定时间区间T内，将时间划分为H个等长的时隙，时隙长度为τ，在每个时隙内，MTEG均映射一份原始网络，其节点由“节点–时隙”二元组v_t表示，用以描述数据所在的网络节点位置及其所处的时隙。本文采用$\mathcal{G}_{\mathrm{M}}=(\mathcal{V}_{\mathrm{M}},\mathcal{E}_{\mathrm{M}})$来表示MTEG模型，$\mathcal{V}_{\mathrm{M}}$为MTEG的节点集合，$\mathcal{E}_{\mathrm{M}}$为MTEG的边集合，其具体构建方式如下所述。

1) 节点集合$\mathcal{V}_{\mathrm{M}}$：对于原始网络中每个节点$u\in\mathcal{V}$和每个离散的时隙$t\in\mathcal{H}$，在MTEG中都会创建唯一一个相应的节点–时隙对u_t，同时为了区分收到的数据块，引入一组虚拟节点$\mathcal{U}_{s,c}$，其中$s\in\mathcal{V}\backslash\{u\}$,$c\in\mathcal{C}$，标识从节点$s$发出的数据块$c$到达节点$u$，所有的$u_t$和$\mathcal{U}_{s,c}$构成了MTEG的节点集合$\mathcal{V}_{\mathrm{M}}$。

2) 边集合$\mathcal{E}_{\mathrm{M}}$：MTEG中边有如下3类。

①存储边(u_t,u_{t+1})。为了刻画节点处对数据块的持续保持，对于任意时隙节点u_t，当$t\leqslant H-1$时，在MTEG中添加一条由当前时隙节点u_t指向下一时隙节点u_{t+1}的有向边。该存储边表示数据可在节点v处被缓存并可延迟至下一时隙继续传输。不失一般性，本文将存储边的代价设为1，表示沿该边进行缓存操作需要消耗一个时隙。

②跨时隙传输边$(u_t,v_{t+L_{(u,v)}})$。对于任意物理链路$(u,v)\in\mathcal{E}$，若在时隙t发起一次数据传输，并且该传输能够在$t+L_{(u,v)}\leqslant T$之前完成，则在MTEG中添加一条从节点u_t指向节点$v_{t+L_{(u,v)}}$的有向传输边，其代价为链路时延$L_{(u,v)}$。

③虚拟边$(u_t,\mathcal{U}_{s,c})$。对于每一个接收节点u_t，在MTEG中添加一条从u_t指向$\mathcal{U}_{s,c}$的有向虚拟边，其代价为0，用以表示来自节点s的数据块c已成功传输至节点u，从而完成该数据块的接收过程。

引入虚拟节点和虚拟边的目的在于区分收到的数据块，从而确认每个节点都收到需要的全部数据块。一个数据块可能在任意时隙到达终端节点，因此本文将所有的终端–时隙节点通过代价为0的虚拟边连接至对应的虚拟节点。通过这一结构，一旦数据块到达虚拟终端，其实际到达时刻即可由其入边连接的前序节点的时隙索引唯一确定。算法1给出了MTEG的具体构建流程。

算法1 MTEG构建算法

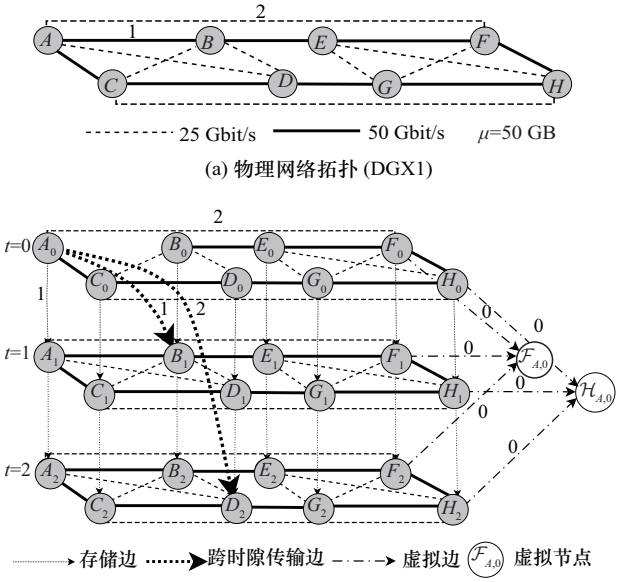
输入 物理网络$\mathcal{G}=(\mathcal{V},\mathcal{E},\mathcal{B})$，时延矩阵$\boldsymbol{L}$，时隙数$H$，数据块集合$\mathcal{C}$

输出 $\mathcal{G}_{\mathrm{M}}=(\mathcal{V}_{\mathrm{M}},\mathcal{E}_{\mathrm{M}})$

初始化$\mathcal{G}_{\mathrm{M}}\leftarrow\varnothing$
for $t=0$ to H
  for $v\in\mathcal{V}$
    创建节点v_t，并添加到集合$\mathcal{V}_{\mathrm{M}}$；
    if $t\leqslant H-1$ then
      创建存储边(v_t,v_{t+1})，链路权重为1，并添加到集合$\mathcal{E}_{\mathrm{M}}$；
    end if
  end for
  for $(u,v)\in\mathcal{E}$
    if $t+L_{(u,v)}\leqslant H$ then
      创建传输边$(u_t,v_{t+L_{(u,v)}})$，链路权重为$L_{(u,v)}$，并添加到集合$\mathcal{E}_{\mathrm{M}}$；
    end if
  end for
end for
for $u\in\mathcal{V}$
  for $s\in\mathcal{V}\backslash\{u\}$
    for $c\in\mathcal{C}$
      创建相应的虚拟节点$\mathcal{U}_{s,c}$，并添加到集合$\mathcal{V}_{\mathrm{M}}$；
      for $t=0$ to H
        创建虚拟边$(u_t,\mathcal{U}_{s,c})$，链路权重为0，并添加到集合$\mathcal{E}_{\mathrm{M}}$；
      end for
    end for
  end for
end for
返回$\mathcal{G}_{\mathrm{M}}=(\mathcal{V}_{\mathrm{M}},\mathcal{E}_{\mathrm{M}})$。

为便于直观理解MTEG的构造方式与结构特征，图2给出了一个示例说明。图2(a)展示了物理网络拓扑，假设在节点A处有一个数据块需要传输到节点F和H，数据块的大小为50 GB，因此在网

络中最快链路（50 Gbit/s）上传输需要的时间为1 s，固定时延α为微秒级，这里可忽略不计。将时隙长度设为1 s，则在最快链路上传输需要一个时隙，例如链路(A,B)权重标注为1，在最慢链路上传输需要两个时隙，例如链路(A,F)权重标注为2。



(a) 物理网络拓扑 (DGX1)

(b) DGX1的组播时间扩展图模型

图2　MTEG模型示意

图2(b)给出了对应的MTEG模型。该模型共考虑了3个时隙，每一行对应一个时隙层，每一列对应同一物理节点在不同时隙下的状态。例如，节点A_0、A_1、A_2分别表示物理节点A在第0、1、2个时隙的状态。在MTEG中，存储边用于连接同一物理节点在相邻时隙的状态节点，例如(A_0,A_1)表示数据在节点A可缓存后再传。跨时隙传输边用于刻画数据沿物理链路传输所需的时延，其构造以物理链路(A,D)为例进行说明。由于在原始网络中链路(A,D)为带宽最小的链路，对应的传输时延为两个时隙，因此在MTEG中会引入从A_t指向D_{t+2}的跨时隙传输边，如图2(b)中的链路(A_0,D_2)所示，表示在时隙0发起的数据将在时隙2的开始时刻到达D。考虑到原始网络是双向的，MTEG中同样会构造反向链路对应的跨时隙传输边，例如(D_0,A_2)。为便于示意，图2(b)中仅绘制了部分跨时隙传输边（如(A_0,D_2)和(A_0,B_1)），在实际构造中，所有满足时延约束的物理链路均会在不同时隙之间生成对应的跨时隙传输边。此外，MTEG中还引入了虚拟边及对应的虚拟节点，用于刻画具体的数据块在目的节点完成接收。如图2(b)右侧所示，对于每一个目的节点及其对应的数据块，都会构造一个虚拟节点（如$\mathcal{F}_{A,0}$），并将该终端在所有时隙下的状态节点均通过代价为0的虚拟边连接至对应的虚拟节点，例如$(F_0,\mathcal{F}_{A,0})$、$(F_1,\mathcal{F}_{A,0})$、$(F_2,\mathcal{F}_{A,0})$。只要数据在任一时隙到达目的节点，就立即汇入对应的虚拟节点，从而标记该数据块在该节点的接收完成。

2.2　基于MTEG的AllGather调度算法

广播操作，乃至更一般的组播操作，其本质均可抽象为从单一源节点向多个接收节点进行数据分发的问题。基于此，本文在构建的MTEG上利用斯坦纳树构造算法，为组播操作规划一棵满足时序与链路约束的组播树。该组播树在MTEG中同时确定了数据块从源节点到各接收节点的传输路径及其对应的传输时隙，从而实现对“从哪传”和“何时传”的联合规划。

对于AllGather操作，可将其分解为多次Broadcast操作的组合。具体而言，每个节点均需向其他节点广播其本地数据块。考虑到不同节点所需广播的数据量可能存在差异，本文首先根据广播数据量大小对各Broadcast操作进行排序，并优先调度数据量较大的广播任务。随后，对于每一次Broadcast操作，基于当前的MTEG构建其对应的最优组播树。

为避免不同Broadcast操作占用同一物理链路的同一时隙资源而产生冲突，在完成一次Broadcast的组播树规划后，本文将该组播树占用的边从MTEG中移除或标记为不可用，并在更新后的MTEG上继续为后续Broadcast操作规划组播树。上述过程持续迭代，直至所有Broadcast操作均完成调度，从而实现对AllGather操作的无冲突调度。基于此，本文提出了基于MTEG的集合通信编排算法，具体步骤如下。

步骤1　集合通信原语分解。若任务为Broadcast/Multicast，则将其视为单个组播任务；若任务为AllGather，则将其分解为一组Broadcast任务，其中每个节点依次作为源节点向其余节点广播本地数据块。

步骤2　Broadcast任务排序。对于所有待调度的Broadcast任务，根据其源节点传输的数据量大

小进行降序排序，记录排序后的任务序列为$\{k_1,k_2,\cdots,k_M\}$。

步骤 3 对于排序后的每一个 Broadcast 任务k_m，依次执行以下步骤。

1) 在当前的$\mathcal{G}_{\mathrm{M}}$上，以源节点s_{k_m}在初始时隙对应的节点为根节点，以所有目的节点对应的虚拟节点为终点集合，如$\mathcal{F}_{s_{k_m},c}$, $c \in \mathcal{C}$。

2) 采用斯坦纳树构造算法，在$\mathcal{G}_{\mathrm{M}}$中搜索一棵满足可达性与时隙约束的最小代价组播树$\mathbb{R}_{k_m}$；由于斯坦纳树问题本身是一个NP-难问题，这里采用高桥–松山（Takahashi-Matsuyama，TM）算法来寻找组播树$\mathbb{R}_{k_m}$，关于 TM 算法的执行过程可参考文献[16]。其基本思想是以源节点为初始生成树，通过迭代方式逐步将尚未覆盖的目的节点并入当前生成树：在每一次迭代中，选择与已有生成树距离最短的目的节点，并将对应的最短路径加入组播树中，直至所有目的节点均被覆盖。文献[16]已证明 TM 算法复杂度为$O(kn^2)$，其中k为目的节点数，n为网络节点数，且在最坏情况下，它与最优解的差异为$2(1-\frac{1}{k})$。

3) 根据组播树$\mathbb{R}_{k_m}$，确定该Broadcast任务中所有的数据块从源节点到各接收节点的传输路径及其对应的传输时隙。

步骤 4 资源更新。将组播树$\mathbb{R}_{k_m}$中使用的所有边在$\mathcal{G}_{\mathrm{M}}$中标记为已占用或直接移除；更新$\mathcal{G}_{\mathrm{M}}$，以避免后续Broadcast任务在物理链路与时间维度上产生资源冲突。

步骤 5 重复步骤 3 和步骤 4，直至所有的Broadcast任务均完成调度，输出最终的集合通信路由与时隙调度结果。

2.3 MTEGCCA与MILP建模的对应关系分析

在MTEG中，原网络中节点v在每个时隙t内被展开为一个时空节点v_t，其可达性状态在语义上等价于MILP中的变量$y_{s,v,c,t}$。对于原网络中的链路(u,v)，MTEG通过引入跨时隙传输边$(u_t,v_{t+L_{(u,v)}})$表示一次沿链路的数据传输过程，该边对应于MILP中的链路选择变量$x_{s,(u,v),c,t}$，并在MTEGCCA的资源更新步骤中体现MILP中的链路容量约束（式(7)）。MTEG中的存储边(u_t,u_{t+1})表示数据块在节点处的保持行为和时间推进，对应于MILP中的缓存约束（式(6)）。此外，引入虚拟节点和虚拟边用于标记数据块在某节点的接收状态及接收时隙，对应于MILP中的接收变量$D_{s,v,c}$及接收时间追踪约束（式(9)~式(14)）。在构造组播树时，以初始时隙对应的时空节点为根节点，对应于MILP中的初始状态约束（式(2)、式(3)）。通过TM算法在MTEG上搜索可行组播树，则对应于满足MILP中流守恒约束和接收约束（式(4)、式(5)、式(8)）的可行解；进一步地，寻找代价最小的组播树则等效于MILP中最小化集合通信时延的优化目标。因此，原MILP问题中复杂的可行解搜索过程，可转化为MTEG上寻找满足容量约束与时序约束的组播树问题。尽管该问题在一般情况下仍为NP-难问题，但其图结构特性使得可以采用成熟的斯坦纳树近似算法，在可接受的计算复杂度内获得高质量的可行解。

2.4 算法复杂度分析

假设$|\mathcal{V}|$、$|\mathcal{E}|$、C和H分别表示网络中的节点数、链路数、最小数据块数和时隙数，在最坏情况下，每个节点都要给网络中其余节点广播数据，即AllGather操作被分解为$|\mathcal{V}|$次Broadcast操作。在MTEG中，节点数量为$|\mathcal{V}_{\mathrm{M}}|=|\mathcal{V}|\cdot H+C\cdot(|\mathcal{V}|-1)$，其中$C\cdot(|\mathcal{V}|-1)$为虚拟节点数量，$|\mathcal{V}|\cdot H$为时隙节点数量；链路数量为$|\mathcal{E}_{\mathrm{M}}|=|\mathcal{V}|\cdot(H-1)+|\mathcal{E}|\cdot H+(|\mathcal{V}|-1)\cdot CH$，其中$|\mathcal{V}|\cdot(H-1)$为存储边数量，$|\mathcal{E}|\cdot H$为跨时隙传输边数量，$(|\mathcal{V}|-1)\cdot CH$为虚拟边数量。对于每一个Broadcast操作，采用TM算法在MTEG中寻找代价最小组播树的算法复杂度为$O(kn^2)$，其中k为终端节点数，这里可以视为虚拟节点的数量$C\cdot(|\mathcal{V}|-1)$，n为节点数量，即$|\mathcal{V}_{\mathrm{M}}|=|\mathcal{V}|\cdot H+C\cdot(|\mathcal{V}|-1)$，因此单次Broadcast的最坏复杂度为$O(C|\mathcal{V}|^3(H+C)^2)$，整个算法的时间复杂度为$O(C|\mathcal{V}|^4(H+C)^2)$。在大规模分布式训练场景中，数据块数量$C$和时隙数$H$通常远小于节点数$|\mathcal{V}|$，上述复杂度可进一步简化为$O(|\mathcal{V}|^4)$。

3 仿真分析

3.1 参数配置

为验证本文提出的MTEGCCA的性能，本节采用Python 3.12.3搭建仿真平台开展实验评估。首先，采用Gurobi求解器[17]对小规模示例（如在

DGX1 网络中进行 Broadcast 操作）进行了求解，以验证所提出的数学建模在刻画问题约束与优化目标方面的正确性。已有研究基于流量工程的集合通信库（TECCL）[15]充分利用了 Gurobi 的建模与求解特性，在求解效率方面表现更优。因此，在后续的仿真对比中，本文选取 TECCL 作为基线，而不再直接采用本文构建的 MILP 问题进行对比。

事实上，TECCL 提供了从精确求解到近似求解的一系列实现形态，本文选择了最具代表性的 TECCL_fast（对比方案 1）、TECCL_early stop（对比方案 2）、TECCL_slow（对比方案 3）3 种形态进行对比。其中，TECCL_fast 采用最快链路上的传输时延作为时隙长度，通过求解完整的 MILP 问题来获得最优调度方案，但求解开销较高。为此，文献[15]引入基于最优性间隙的提前终止策略，在获得高质量可行解后即终止求解（当可行解与最优解的差距小于 30% 时停止求解），即 TECCL_early stop 方法。此外，文献[15]还给出了 TECCL_slow 实现，其采用最慢链路上的传输时延作为时隙长度，通过调整时隙长度来评估不同时间建模精度下方法的性能。拓扑感知的集合通信库（TACCL）[13]作为学术界广泛采用的代表性集合通信调度方法，其设计目标是在显式建模物理拓扑与通信约束的前提下，生成具有高通信效率的调度方案。文献[13]已在多种典型物理拓扑上（如 DGX2、NDv2）及不同数据规模条件下对 TACCL 与英伟达集合通信库（NCCL）进行了系统对比，验证了 TACCL 在通信性能方面能够达到与 NCCL 相当的水平。因此，除 TECCL 的 3 种方法外，本文还选取 TACCL（对比方案 4）作为对比方法，间接反映所提算法相对于 NCCL 的性能优势。

本文采用的网络拓扑参数如表 2 所示，与文献[15]保持一致，其中 NDv2 和 AMD 拓扑允许通过增加机柜（chassis）数量扩展网络规模。本文采用的仿真参数如表 3 所示。

表 2　　网络拓扑参数

拓扑	GPU/chassis	边数/chassis
DGX1	8	32
DGX2	8	32
NDv2	8	32
AMD	16	56

表 3　　仿真参数

仿真参数	参数值
最小数据块大小 μ/KB	$2^1 \sim 2^{20}$[15]
时隙数 H/个	40 ~ 200
机柜数量	2 ~ 10
固定时延 α/μs	{0.7, 1.3, 0.35, 2.6}[15]

本文选取集合通信时延、求解时间和算法带宽 3 项指标进行对比分析。具体地，集合通信时延指在给定网络拓扑与集合通信需求的条件下，按照所规划的路由完成一次集合通信操作所需的时间；求解时间是指算法计算路由方案消耗的时间；算法带宽通常定义为集合通信操作传输的总数据量与集合通信时间之比，用于刻画集合通信过程中的有效数据传输速率。

3.2　仿真结果对比

图 3 为不同集合通信算法在 NDv2（2 chassis）拓扑下执行 AllGather 操作的集合通信时延对比。随着数据量逐渐增大，各算法的集合通信时延均呈现出单调上升趋势，这是因为链路的传输时延随着数据量的增大而增大。在小数据规模（1~64 KB）下，TACCL 的集合通信时延明显高于其他算法，其完成时间为 10~13 μs，TECCL_fast、TECCL_early stop、TECCL_slow 以及 MTEGCCA 均集中在 4~7 μs。该现象表明，在固定时延占主导的场景下，TACCL 由于其算法合成与调度机制带来的额外调度开销，难以充分发挥网络性能，其余算法在该区间内几乎重合，均能达到接近最优的通信完成时间。随着数据规模增大（≥256 KB），集合通信时延逐渐由带宽利用效率主导。此时，MTEGCCA 在整个数据规模范围内始终可以逼近 TECCL_fast 和 TECCL_early stop。在 1 MB 数据规模时，MTEGCCA 的集合通信时延略高于 TECCL_fast，但显著低于其他算法。TECCL_fast 允许在有效时间内（文献[15]设定为 2 h）尽可能遍历整个解空间，因此可以取得更优的性能。上述结果表明，MTEGCCA 能够在小规模与大规模通信场景下逼近最优的集合通信时延。

图 4 为不同集合通信算法在 NDv2（2 chassis）拓扑下执行 AllGather 操作的求解时间对比。在固定 NDv2 拓扑规模下，MTEGCCA 在不同数据规模条件下均保持稳定且极低的求解时间，求解时间始终控制在 1 s 以内，且几乎不随数据规模的增长而

显著变化，这是由于在拓扑稳定的情况下，MTEGCCA并不需要调整已构成的组播树。相比之下，TECCL_fast的求解时间随数据规模急剧上升，当数据规模达到16 MB及以上时，其求解时间已达到10^3~10^4 s量级；在1 GB数据规模下，TECCL_fast的求解时间约为7.2×10^3 s，MTEGCCA仍维持在约1 s内，求解时间提升7 000多倍。即使采用提前终止策略的TECCL_early stop，在中大规模数据场景下的求解时间仍处于2~3 s量级，为MTEGCCA的2~3倍。TECCL_slow由于采用更粗粒度的时隙划分，其运行时间与MTEGCCA接近，但以牺牲集合通信时延为代价；TACCL的运行时间整体保持在5~10 s，为MTEGCCA的5~10倍。因此，MTEGCCA在保证集合通信时延接近最优的同时，求解时间相较TECCL_fast降低了4个数量级以上，相较TACCL仍可获得一个数量级的运行时间优势，能够更好地适用于大规模训练场景。

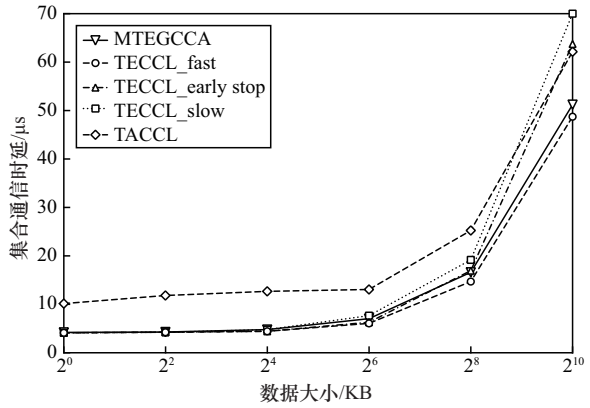


图3　不同集合通信算法在NDv2（2 chassis）拓扑下执行AllGather操作的集合通信时延对比

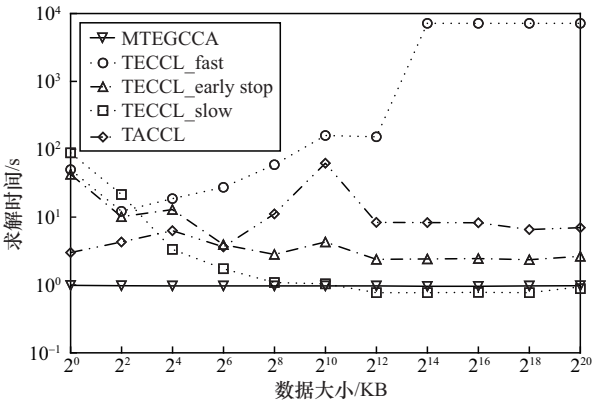


图4　不同集合通信算法在NDv2（2 chassis）拓扑下执行AllGather操作的求解时间对比

图5为不同集合通信算法在NDv2（2chassis）拓扑下执行AllGather操作的算法带宽对比。从整体趋势来看，随着数据规模增大，各算法的带宽逐步提升，并在中大规模数据场景下趋于稳定。在小数据规模（≤16 KB）时，所有算法的带宽均较低，主要受固定通信时延主导，不同算法之间差异不明显。随着数据规模增大，链路带宽利用率逐步成为主导因素，算法间性能差距开始显现。相比TACCL和TECCL_slow，MTEGCCA能够更快地逼近网络可用带宽上限，在256 KB~4 MB，其带宽相较TACCL提升30%~80%，相较TECCL_slow提升10%~30%。在大数据规模（≥16 MB）下，MTEGCCA的有效带宽稳定在22~23 Gbit/s，与TECCL_fast基本一致，表明MTEGCCA在不需要求解MILP的情况下，仍能获得接近最优的通信性能。同时，相比TECCL_early stop，MTEGCCA在整个数据区间内均表现出更稳定的带宽收益，避免了因提前终止导致的性能波动。

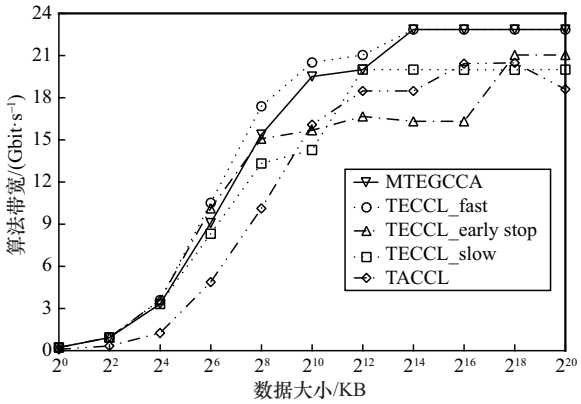


图5　不同集合通信算法在NDv2（2 chassis）拓扑下执行AllGather操作的算法带宽对比

图6为不同集合通信算法在多种网络拓扑下执行AllGather操作的集合通信时延对比。由于各拓扑在链路结构与带宽配置等方面存在差异，集合通信时延在不同拓扑中并不呈现随网络规模线性变化的规律，因此图6主要用于比较不同算法在各典型拓扑下的相对性能。具体地，在所有给定拓扑中，MTEGCCA在集合通信时延方面始终能够与最优的TECCL_fast保持逼近关系，并优于TECCL_early stop与TECCL_slow。随着机箱数量的增加，调度问题的搜索空间显著扩大。在AMD（2 chassis）场景中，TECCL_fast与TECCL_early stop在运行超

过 4 h 后仍未获得可行解，因此在图 6 中标注为 N/A。相比之下，MTEGCCA 能够在该场景下成功完成调度，其集合通信时延约为 0.02 s，表现出良好的可行性与鲁棒性。尽管 TECCL_fast 在可求解场景中能够获得最优或近最优的集合通信时延，但其在多机箱、大规模拓扑下的可扩展性受到显著限制；MTEGCCA 在不同网络规模下均能够稳定获得高质量调度方案，在保持集合通信性能逼近最优的同时，显著提升了算法的实用性。

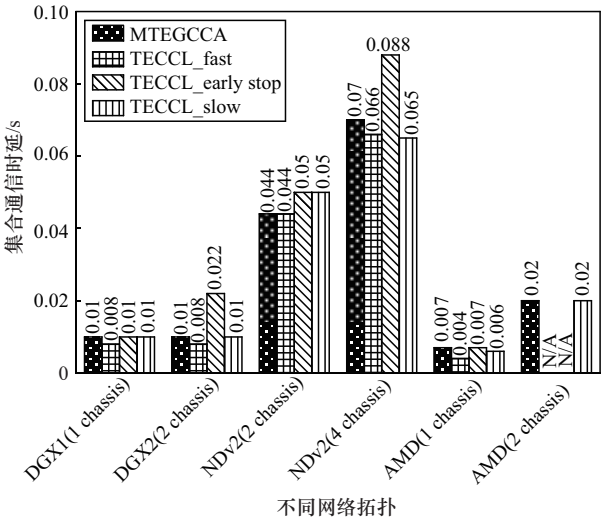


图 6　不同集合通信算法在多种网络拓扑下执行 AllGather 操作的集合通信时延对比

图 7 为不同集合通信算法在多种网络拓扑下执行 AllGather 操作的求解时间对比。不同的算法在求解时间方面存在显著差异，且该差异在多机柜拓扑下尤其明显。在 1 chassis 场景下，如 DGX1、AMD（1 chassis）中，各算法均能够在较短时间内完成求解，求解时间整体处于毫秒至秒级范围内，其中，MTEGCCA 的求解时间始终保持最短。随着拓扑复杂度的增加，TECCL 系列算法的求解时间迅速增长。在 DGX2 和 NDv2（2 chassis）场景中，TECCL_fast 与 TECCL_early stop 的求解时间已经达到了数十秒至数千秒，MTEGCCA 仍能够在秒级时间内完成求解。当拓扑扩展至 NDv2（4 chassis）时，TECCL_fast 与 TECCL_early stop 的求解时间进一步上升至数小时量级，MTEGCCA 的求解时间依然可以保持在数秒内，求解时间降低了 2~4 个数量级。在 AMD（2 chassis）场景下，TECCL_early stop 在运行了超过 2 h 后仍未获得可行解，TECCL_fast 的求解时间则更长，因此在图中标注为 N/A。TECCL_slow 在除了 AMD（2 chassis）的其他拓扑中，求解时间基本在可承受范围内，从秒级到几分钟级，而在 AMD（2 chassis）中却达到了小时级，在实际求解过程中难以接受。MTEGCCA 仍可以在 18.7 s 内获取可行解，因此所提算法能够在保证调度质量的同时，显著降低求解时间。

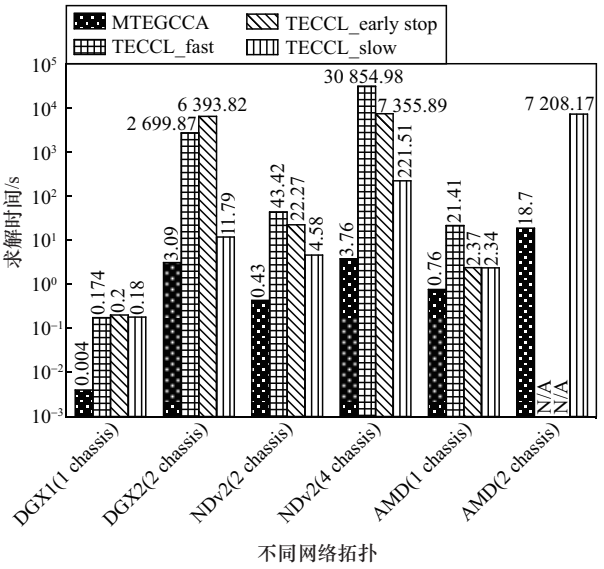


图 7　不同集合通信算法在多种网络拓扑下执行 AllGather 操作的求解时间对比

从采用 TECCL 进行求解的过程中可以看出，其对时隙参数的设置较为敏感。当集合通信时延对应的时隙数较多而预先设定的时隙上界偏小时，TECCL 在求解过程中容易陷入不可行解空间，即经过长时间搜索仍无法获得可行解，从而导致大量计算资源被消耗在一次无解的求解过程中。该现象在多机箱、大规模拓扑场景下尤其明显。相比之下，MTEGCCA 能够在较低的计算开销下快速生成一组可行的调度方案。基于此，先利用 MTEGCCA 得到的调度结果对所需时隙数进行上界评估，并将该上界作为 TECCL 的时隙数配置参数，可有效避免因时隙数设置不足而导致的不可行搜索问题，也可有效避免因时隙设置过多而导致的解空间过大，从而导致求解时间过长。这一观察结果表明，除直接用于集合通信调度外，MTEGCCA 还可作为一种实用的上界估计算法，为后续精确或启发式算法提供有效的参数参考。

图 8 为不同集合通信算法在多种网络拓扑下执行 AllGather 操作的算法带宽对比。在所有可求解场景中，MTEGCCA 整体优于 TECCL_early stop 和

TECCL_slow，且逼近TECCL_fast。在1 chassis场景中，由于拓扑规模小、解空间受限，TECCL_fast能够通过充分搜索获得最优或近最优解，因此在算法性能方面表现最优。随着机柜数量的增加和网络拓扑复杂度的提升，解空间迅速扩大，TECCL_fast和TECCL_early stop求解代价显著上升。例如在AMD（2 chassis）场景中，TECCL_fast和TECCL_early stop未能在给定时间内获得可行解，因此未给出对应的算法性能结果。相比之下，MTEGCCA与TECCL_slow均能够完成调度求解，在相同算法带宽下，MTEGCCA能够以显著更低的求解时间获得可行解，表明MTEGCCA在复杂拓扑条件下具有更稳定的链路资源利用能力与更好的实际可用性。

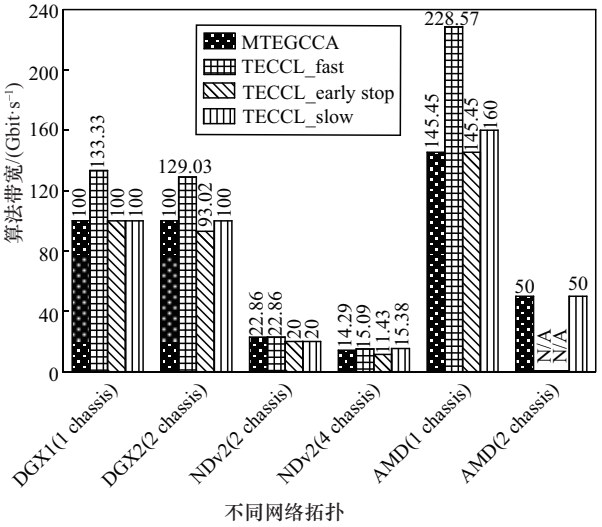


图8　不同集合通信算法在多种网络拓扑下执行AllGather操作的算法带宽对比

图9为MTEGCCA在不同网络规模下执行AllGather操作所需的算法运行时间。随着网络规模由2 chassis逐步扩展至10 chassis，算法运行时间呈近似线性增长趋势，表明MTEGCCA具有良好的可扩展性。为对比求解效率，本文在4 chassis的NDv2拓扑中测试了TECCL的3种实现形态。其中，TECCL_fast的求解时间为30 854.98 s，TECCL_early stop为7 355.89 s，TECCL_slow为221.5 s，MTEGCCA仅需3.79 s即可完成路由规划，求解效率分别提升57.4~8 139.8倍。同时，各算法所生成路由方案的通信完成时间基本集中在65~88 ms，表明在通信性能接近最优的前提下，MTEGCCA显著降低了调度开销。随着网络规模进一步扩大，TECCL系列算法在给定时间（2 h）内难以获得可行解，因此图9中仅展示了MTEGCCA的运行时间结果。在10 chassis规模下，网络中每个节点需向除自身外的所有节点发送数据，总计形成6 320对通信节点对，MTEGCCA仅需170.82 s即可完成路由方案计算，验证了其在大规模集合通信场景下的高效性与适用性。

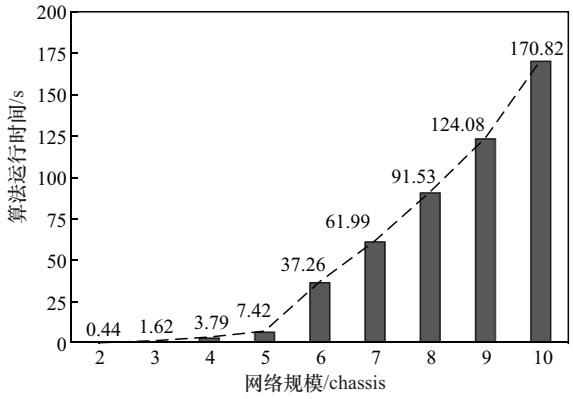


图9　MTEGCCA在不同网络规模下执行AllGather操作所需的算法运行时间

4　结束语

本文针对大规模分布式训练中集合通信编排算法拓扑感知不足，求解复杂度高的问题，提出了一种基于MTEG的集合通信编排算法。通过将网络拓扑与时间维度统一建模，并构建高效的集合通信编排方法，实现了对数据传输路径与时序的联合优化。仿真结果表明，与现有方案相比，所提算法的集合通信时延可逼近最优解，同时显著降低计算复杂度，在大规模网络场景下仍具有良好的可扩展性。未来工作将考虑采用图模型解决规约等多到一的通信操作，解决集合通信过程中计算、通信与时隙联合规划问题，构建一套完整的基于图模型的集合通信方法库。

参考文献：

[1] 高翔, 董斌, 肖晴, 等. 智算场景下集合通信库的挑战与发展趋势[J]. 电信科学, 2025, 41(4): 81-94.
Gao X, Dong B, Xiao Q, et al. Challenges and development trends of collective communication libraries in intelligent computing scenarios[J]. Telecommunications Science, 2025, 41(4): 81-94.

[2] Liu T R, Hei C Y, Li F L, et al. ResCCL: resource-efficient scheduling for collective communication[C]//Proceedings of the ACM SIGCOMM 2025 Conference. New York: ACM Press, 2025: 55-70.

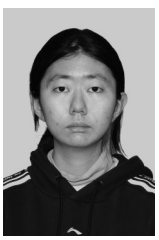
[3] Hui L H, Yang W, Wu F, et al. DirectReduce: a scalable ring AllReduce offloading architecture for torus topologies[J]. IEEE Internet of Things Journal, 2025, 12(16): 32951-32964.
[4] Zhang Z X, Wen Y B, Lyu H Q, et al. AI computing systems for large language models training[J]. Journal of Computer Science and Technology, 2025, 40(1): 6-41.
[5] Geng J K, Li D, Cheng Y, et al. HiPS: hierarchical parameter synchronization in large-scale distributed machine learning[C]//Proceedings of the 2018 Workshop on Network Meets AI & ML. New York: ACM Press, 2018: 1-7.
[6] Jiang Y H, Gu H X, Lu Y F, et al. 2D-HRA: two-dimensional hierarchical ring-based all-reduce algorithm in large-scale distributed machine learning[J]. IEEE Access, 2020, 8: 183488-183494.
[7] Pjesivac G J. Towards automatic and adaptive optimizations of MPI collective operations[D]. Knoxville: The University of Tennessee, Knoxville, 2007.
[8] Sanders P, Speck J, Träff J L. Two-tree algorithms for full bandwidth broadcast, reduction and scan[J]. Parallel Computing, 2009, 35(12): 581-594.
[9] Weingram A, Li Y K, Qi H, et al. xCCL: a survey of industry-led collective communication libraries for deep learning[J]. Journal of Computer Science and Technology, 2023, 38(1): 166-195.
[10] Cavazzoni C. EURORA: a European architecture toward exascale[C]//Proceedings of the Future HPC Systems: the Challenges of Power-Constrained Performance. New York: ACM Press, 2012: 1-4.
[11] Cho M, Finkler U, Serrano M, et al. BlueConnect: decomposing all-reduce for deep learning on heterogeneous network hierarchy[J]. IBM Journal of Research and Development, 2019, 63(6): 1-11.
[12] Kalb J L, Lee D S. Network topology analysis[R]. 2008.
[13] Cai Z X, Liu Z Y, Maleki S, et al. Synthesizing optimal collective algorithms[C]//Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: ACM Press, 2021: 62-75.
[14] Shah A, Chidambaram V, Cowan M, et al. TACCL: Guiding collective algorithm synthesis using communication sketches[C]//Proceedings of the 20th NSDI-2023 USENIX Symposium on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2023: 593-612.
[15] Liu X T, Arzani B, Kakarla S K R, et al. Rethinking machine learning collective communication as a multi-commodity flow problem[C]//Proceedings of the ACM SIGCOMM 2024 Conference. New York: ACM Press, 2024: 16-37.
[16] Takahashi H, Matsuyama A. An approximate solution for steiner problem in graphs[J]. Math. Japonica, 1980, 24(6): 573-577.
[17] GUROBI optimization, LLC. Gurobi optimizer reference manual[R]. 2024.

[作者简介]

刘飞（1996– ），男，山西吕梁人，西安电子科技大学博士生，主要研究方向为算力网络、集合通信和时变图理论。

王鹏（1995– ），男，河南济源人，博士，新加坡科技设计大学博士后研究员，主要研究方向为算力网络、时间确定性网络和6G网络。

麻涵（1997– ），男，宁夏银川人，西安电子科技大学博士生，主要研究方向为时变图理论、自组织网络资源调度。



李烨（2004– ），女，吉林珲春人，西安电子科技大学博士生，主要研究方向为时变图理论、集合通信。

李红艳（1966– ），女，陕西西安人，博士，西安电子科技大学教授、博士生导师，主要研究方向为时间确定性网络、天地一体化网络和新一代无线局域网等。