

在网计算中资源受限的流汇聚算法*

盛家华,洪佩琳[†],王航

(中国科学技术大学电子工程与信息科学系信息网络实验室,合肥 230027)

(2022 年 12 月 27 日收稿;2023 年 4 月 27 日收修改稿)

Sheng J H, Hong P L, Wang H. Flow aggregation with constrained resource for in-network computation[J]. Journal of University of Chinese Academy of Sciences, 2025, 42(2): 248-259. DOI:10. 7523/j. ucas. 2023. 044.

摘要 考虑在总资源量和节点容量的限制下,如何为任务找到一棵最小代价汇聚树。该问题是一个 NP 难问题,分别提出问题的线性整数规划模型和启发式算法 GCAT。仿真结果显示,相比于其他的启发式算法,GCAT 算法生成的汇聚树代价更小,同时有更高的资源利用率,小规模网络下性能接近于最优解。

关键词 在网计算;汇聚树;资源分配;线性整数规划;GCAT

中图分类号:TN929.5 文献标志码:A DOI:10. 7523/j. ucas. 2023. 044

Flow aggregation with constrained resource for in-network computation

SHENG Jiahua, HONG Peilin, WANG Hang

(Laboratory of Information Network, Department of Electronic Engineering and Information Science,
University of Science and Technology of China, Hefei 230027, China)

Abstract In-network computation can greatly reduce the traffic generated during many-to-one transmission by establishing aggregation trees to merge data streams at the aggregation nodes. In this paper, we consider finding the minimum-cost aggregation tree under the constraints of a given amount of resources and the capacity of switch nodes. Since this problem is an NP-hard problem, a linear integer programming model and a heuristic algorithm called greedy cost aggregation tree (GCAT) are given to solve it. Simulation results show that the GCAT algorithm can generate a tree with less cost and utilize the resource more efficiently than other heuristics, and the performance is close to the optimal solution for small-scale networks.

Keywords in-network computation; aggregation tree; resource allocation; linear integer programming (ILP); greedy cost aggregation tree (GCAT)

* 国家自然科学基金(61671420)资助

[†] 通信作者, E-mail: plhong@ustc.edu.cn

随着大数据时代到来,数据中心承载着比以往更多的分布式计算任务,如分布式机器学习^[1]、MapReduce^[2]大数据处理任务等。这些应用在计算过程中需要传输大量数据同步计算结果,不仅导致应用自身存在末端拥塞问题^[3],也持续占用大量网络带宽,影响其他任务的执行效率^[4-6]。

针对此问题,研究者提出在网计算的概念^[7-8],即把一部分计算卸载到网络设备单元(如智能网卡,可编程交换机等)上完成,我们以图 1 为例说明在网计算在分布式应用执行过程中起到的作用。

图 1(a) 是一个简化的数据中心网络 FatTree^[9],架内的矩形框代表服务器,单词右侧数字代表其出现的次数。此时网络中正运行着一个统计单词“hello”数量的 MapReduce 任务,每台相关服务器需要将自己统计的结果发送给接收端机架 D 上的服务器做求和处理。

图 1(b) 表示任务在网络中产生的汇聚树,阴影节点表示交换节点被选作计算节点(下文也称汇聚节点),可对进入的数据包进行合并,虚线框是合并的结果。“/”前后数字分别代表不使用和使用在网计算技术边上经过的数据包数量。我们定义汇聚树的代价为其在网络中产生的数据量总和,则通过图中汇聚点 2、3、5、8 的聚合作用,总代价从原来的 30 减为 13。

图 1(b) 中的完全汇聚树是理想情况下的结果,即任务可以使用任意交换节点作为汇聚点使得每条边上经过的数据量等于 1。此时求最小代

价的汇聚树问题等价于斯坦纳树问题^[10],无线传感器领域的信息收集^[11-12],以及组播领域的组播树构造^[13-14]对此问题都有深入研究。本文的不同之处在于考虑了资源对汇聚树代价的影响,这包含两个方面。

一方面,交换节点的容量有限,以内存资源为例,由于到达交换节点的各条流存在时间差,节点需要为汇聚树预留内存保存中间结果^[15-16],如果交换节点没有空闲内存使用,将只能作为转发节点将收到的所有数据包原封不动发往下一跳,此时汇聚树中只有部分交换节点可以起到汇聚的作用;另一方面,考虑在网计算作为服务提供给租户付费使用^[17],租户可能只购买了少量资源,此时如何在交换节点上分配这些资源也是影响汇聚树代价的重要因素,如图 2(a) 和 2(b),假如用户购买的资源只够使用 2 个汇聚点,则图 2(b) 的分配方式能够产生的汇聚树代价更小。容易证明,在上述 2 个资源限制下为任务找出最小代价汇聚树是一个 NP 难问题。

定理 1 资源限制下的最小汇聚树问题 (minimum aggregation tree with constrained resource, MATCR) 是 NP 难问题。

证明 当任务可使用资源量和节点可使用资源量无限大时,任意交换节点都可以作为汇聚点,于是所有边上的数据量都为 1,最小代价汇聚树问题就退化为在图中寻找斯坦纳树,这个问题已经被证明为 NP-complete 问题^[18]。由于此问题只是本文问题的特例,因此 MATCR 问题是 NP 难问题。 □

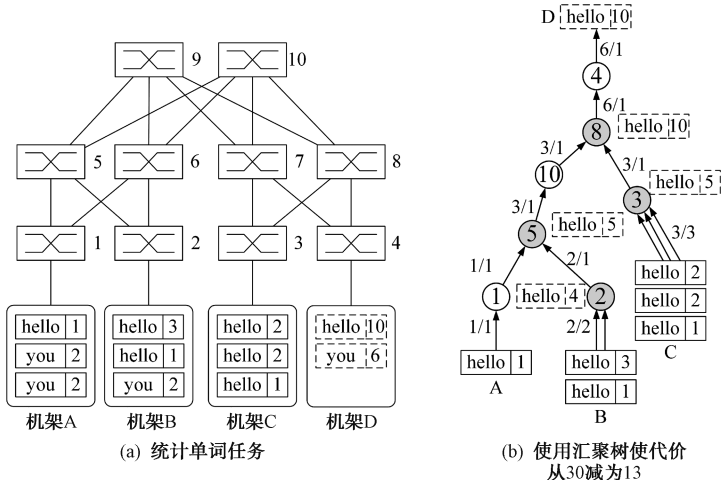


图 1 在网计算示意图

Fig. 1 Sketch of in-network computation

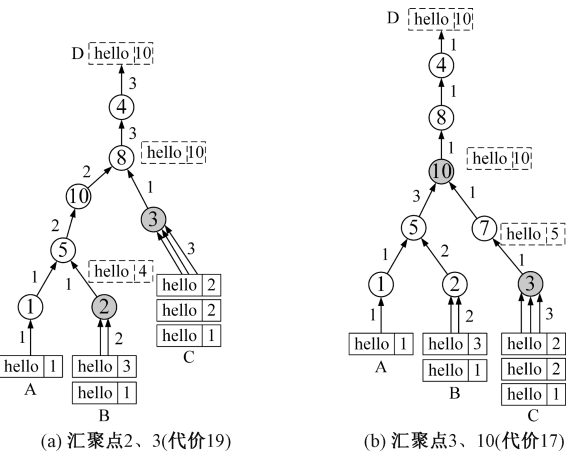


图 2 部分汇聚树

Fig. 2 Partial aggregation tree

文献[19]针对 FatTree 网络提出多播树的构造方法;文献[20-21]分别针对网络拓扑 CamCube^[22]、BCube^[23]提出汇聚树的构造方法,但是都没有考虑资源对汇聚结果(树的代价)的影响。

文献[17]研究当已知汇聚树时汇聚点的选择问题,提出一个基于双层动态规划的算法 SOAR,但是并没有给出汇聚树的构造方法,并且该方法只适用于单棵汇聚树的情况。

综上,本文的贡献在于提出一种在任意网络中构造(多棵)汇聚树的算法,同时还考虑了资源对于汇聚树代价的影响。

1 问题描述

表 1 列出本文中出现的重点符号和变量, G 表示原始物理拓扑(有向图),总的节点集合 N 包含交换节点 V 和终端节点 H 。终端节点指的是可以接收或发送数据包的节点,发送端 S 和接收端 r 都属于此类节点。 H 中的节点只和 V 中的节点相连, H 内部节点之间不相连。

T 代表一次传输 $S \rightarrow r$ 形成的汇聚树(有向图),发送端集合 S 是汇聚树的叶子节点,接收端 r 是根节点。一个任务往往包含多棵汇聚树,用上标 $k \in \mathcal{K}$ 区分不同的汇聚树,即 T_k 代表传输 $S_k \rightarrow r_k$ 形成的汇聚树。

本文假设在汇聚树中每个汇聚点上消耗的资源都是相同且已知的(单位 1),这类资源包括保存中间结果的内存资源^[15]、指示汇聚数据包下一跳的路由表条目^[24]等。对于使用多种资源的情况,只要这些资源的消耗不互相影响,模型里只需

表 1 符号和变量

Table 1 Symbols and variables

物理网络	
$G(N, E)$	物理网络拓扑(有向图), $N = V \cup H, E$ 为节点之间的有向边集合
V	交换节点集合
H	终端集合
m_i	交换节点 $i \in V$ 的容量(剩余资源量)
汇聚任务	
T	汇聚树(有向图),上标 $k \in \mathcal{K} = \{1, 2, \dots, K\}$ 区分不同的汇聚树
P	汇聚树的流路径集合,上标 k 表示汇聚树 T^k 的流路径集合 P^k
$S \in H$	发送端集合
$r \in H$	接收端
M	任务可使用的资源量
变量	
$a(i)$	0/1 变量, $a(i) = 1$ 则交换节点 i 是汇聚点; $a(i) = 0$ 则交换节点 i 是转发节点
$x(e)$	边 e 上经过的实际数据量
$y(e)$	辅助变量,不考虑汇聚点的聚合作用,边 e 上经过的数据量
$z(i)$	辅助变量,表示节点 i 消耗的数据量
$I(i)$	节点 i 的入向边集合
$O(i)$	节点 i 的出向边集合

加入相应的约束即可。为简化分析,本文只考虑一种资源约束。由于一棵汇聚树在汇聚点上消耗的资源为单位 1,所以节点上的剩余资源量 m_i 也可以理解为节点容量,即节点能同时容纳的汇聚树数量。任务消耗的总资源量不能超过可使用的总资源量 M 。

本文使用交换节点、转发节点、汇聚节点 3 种称呼区分汇聚树上的节点:一棵汇聚树除了发送端和接收端以外都是交换节点,交换节点不受资源限制,可以在网络中任取。只有当 $a(i) = 1$ 时,交换节点 i 才可以作为汇聚节点合并数据包,否则只能作为转发节点(即输入的数据量等于输出的数据量)。

汇聚树 T_k 在边 e 上经过的数据量为 $x^k(e)$,一棵树的代价是其所有边上的数据量之和,本文的目标是最小化任务的所有汇聚树代价之和,即
$$\min \sum_{k \in \mathcal{K}} \sum_{e \in E} x^k(e)。$$

2 单汇聚树的线性整数规划建模

这一节讨论任务只包含一棵汇聚树的情况,因此不涉及上标 k 的使用。单汇聚树的线性整数规划(linear integer programming, ILP)建模有 2 个关键部分,一是确定汇聚树经过的边,二是确定资

源量的分配,即路径约束和资源约束。

2.1 流的路径约束

$$\sum_{e \in O(i)} y(e) = 1, \forall i \in S, \quad (1)$$

$$\sum_{e \in O(i)} y(e) - \sum_{e \in I(i)} y(e) = 0, \forall i \in V, \quad (2)$$

$$\sum_{e \in I(r)} y(e) = |S|, \quad (3)$$

$$x(e) \leq y(e), \forall e \in E, \quad (4)$$

$$|S| \cdot x(e) \geq y(e), \forall e \in E, \quad (5)$$

$$\sum_{e \in O(i)} x(e) - \sum_{e \in I(i)} x(e) = 1, \forall i \in S. \quad (6)$$

定理1说明斯坦纳树问题只是本文的特例,所以并不能直接使用斯坦纳树问题的ILP模型^[10],原因在于交换节点的输入、输出量之间存在多种情况。以图3(a)为例,节点1和节点5作为汇聚节点,输入的数据量分别是2和4,而输出量只有1,流量不守恒;节点2、3、4作为转发节点,输入量等于输出量,流量守恒。然而交换节点的类型是资源分配结果 $a(i)$ 决定的,于是路径约束和资源分配之间存在耦合,难以列出相关约束。

为此引入辅助汇聚树(图3(b)),辅助汇聚树和实际的汇聚树(图3(a))拓扑相同,但是没有汇聚节点对流起汇聚作用,设实际和辅助汇聚树边 e 上的流量分别为 $x(e), y(e)$ 。辅助汇聚树可以看成从发送端集合 S 到接收端 r 的 $|S|$ 条流的路径形成的汇聚树,所以容易写出路径约束(1)~(3)。这样 $y > 0$ 的边就构成了最终汇聚树的边,然后利用约束(4)(5)限制实际数据量 x 只经过这些边,就能保证辅助汇聚树和实际汇聚树经过相同的边。加入约束(6)是因为本文不考虑终端节点作为转发节点。

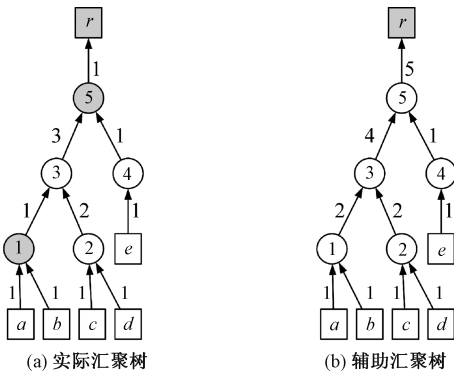


图3 辅助汇聚树的引入

Fig. 3 Introduction of auxiliary aggregation tree

2.2 资源约束

$$\sum_{e \in O(i)} x(e) - \sum_{e \in I(i)} x(e) + z(i) = 0, \forall i \in V, \quad (7)$$

$$z(i) \leq |S| \cdot a(i), \forall i \in V, \quad (8)$$

$$z(i) \geq a(i), \forall i \in V, \quad (9)$$

$$a(i) \leq m_i, \forall i \in V, \quad (10)$$

$$\sum_{i \in V} a(i) \leq M. \quad (11)$$

容易看出,节点的输入输出量之间只存在2种情况: $a(i) = 0$ 时,节点 i 为转发节点,流量守恒; $a(i) = 1$ 时,节点 i 为汇聚节点,流量不守恒,但是输出量之和为1。因此,引入辅助变量 $z(i)$ 和约束(7)~(9),在不破坏模型的线性前提下区分这2种情况。

$z(i)$ 的作用原理解释如下:当 $a(i) = 1$ 时,约束(8)、(9)允许 $z(i)$ 大于0,因为目标是最小化总代价, $x(e)$ 应尽可能小,所以在约束(7)中 $z(i)$ 会尽可能地大,但是约束(5)又保证了节点 i 起码有一条出边 e 使得 $x(e) = 1$,所以 $z(i)$ 最大为 $\sum_{e \in I(i)} x(e) - 1$,因此 $z(i)$ 可以理解为汇聚节点 i 消耗掉的流量。约束(9)还有一个作用是防止最后解中出现无效汇聚点,因为其要求 $a(i) = 1$ 时,节点一定消耗了流量($z(i) > 0$),所以不会出现某个节点只有一条流进入但是却被指定为汇聚点,这种情况会出现在比如总资源量 M 很大时。当 $a(i) = 0$ 时, $z(i)$ 被约束(8)限制为0,此时约束(7)就是普通的流量守恒约束。

$$\begin{aligned} \min_{a(i), y(e), x(e)} \quad & \sum_{e \in E} x(e), \\ \text{s. t.} \quad & (1) - (11). \end{aligned} \quad (P1)$$

约束(10)和(11)分别对应节点的容量限制以及总资源量约束,最终完整的最小代价汇聚树模型如(P1)所示。

2.3 提取流路径算法 EXTRACTPATHS

在网计算的部署需要在交换节点上配置相应的路由项才能引导流到达汇聚节点进行合并,为此需要得到每条流的路径。

因为树状图中每个节点的父节点(即汇聚树中每个节点的下一跳)只有一个,所以大多数情况下,沿着 $x(e) > 0$ 的出边寻找就能确定每一条流的路径,但是当流的路径发生重叠,一个节点就会有2条出边:如图4(b)所示,以发送端 c, d 为起点的流先到汇聚节点1,和 a, b 汇聚后再一起发送到接收端 r ,产生的代价会比直接到接收端 r

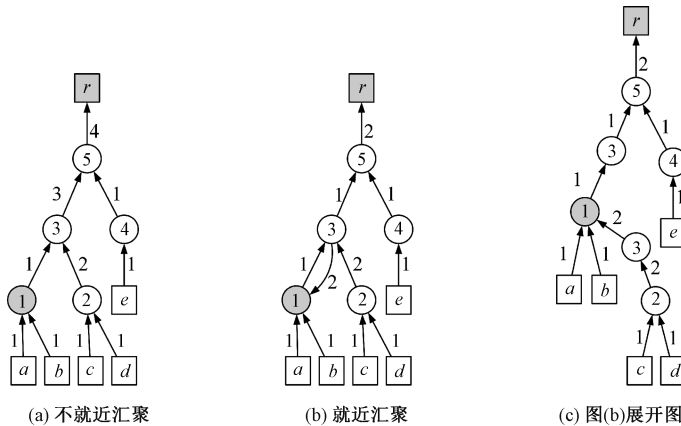


图 4 特殊情况
Fig. 4 Special cases

(图 4(a))产生的代价更小。虽然图 4(b)看上去有环路,但是经过节点 3 的实际是 2 条不同的流(见展开图 4(c))。

造成这个问题的根本原因在于模型(P1)是一个基于边的模型,其中变量 $x(e)$ 只代表边上的流量,而无法区分流量具体属于哪条流,为此需要一个额外的算法 EXTRACTPATHS(表 2)来准确提取每条流的路径,算法的正确性证明见下文。

在说明算法之前先引入 2 个记号 PATH 和 DIST:PATH(v_1, v_2, \dots, v_n)代表 $v_1 \rightarrow v_n$ 的最短路径,同时这条路径必须按照顺序依次经过点 v_1, \dots, v_n 。其中 v_i 可以是单独的点也可以是点集合,当表示点集合时,PATH(v_1, v_2, \dots, v_n)代表所

有可能路径中最短的那一条。DIST 表示相应 PATH 的长度。

算法分成 2 步:首先确定流起始点属于 A 的路径,然后确定流起始点属于 S 的路径:

第 1 步,行 4~13,对于起点是汇聚点 $a \in A$ 的流,用 Prim 算法从 r 开始生成关于 $A \cup r$ 的最小树,即只考虑汇聚点(转发节点只会影响汇聚点之间的距离)。

第 2 步,行 14~16,对于起点是发送端 $s \in S$ 的流,其终点是离其最近的汇聚点或终点。

定理 2 已知汇聚点集合 A,如果按照 EXTRACTPATHS 找出的汇聚树是合理的,则此汇聚树是关于 A 的最小代价汇聚树。

证明 第 1 步,只考虑汇聚点和接收端集合 $A \cup r$,以汇聚点为起点的汇聚流一定有且只有 1 条,即节点的出度为 1,并且终止于另外一个汇聚点或接收端 r 。符合所有节点出度 ≤ 1 ,且保证 $A \cup r$ 连通的最小图显然是一棵最小有向树。

因为本文的优化目标只和边上的数据量有关,和边的方向无关,所以使用无向图的最小生成树算法 Prim 算法即可(只要每次加入新边时,保证一个节点只有 1 条出向边即可)。如果优化的目标和边的方向有关,即边的权重不是对称的(如路径的时延),则需要使用专门的有向树生成算法^[25]。

第 2 步,对于 $A \cup r$ 这个整体,以发送端为起点的流只要取到这个集合内最近的节点的最短路径即可。因为如果没有汇聚点,某个发送端 s 对于汇聚树贡献的代价最小是 DIST(s, r),而如果有汇聚点,流 s 到汇聚点可能会比直接到达接收

表 2 提取路径算法

Table 2 Extract paths algorithm

算法 1 EXTRACTPATHS
输入: S, A, r
输出: 流的路径集合 P
1: $P \leftarrow \emptyset; Q \leftarrow \{r\}; T \leftarrow A \cup r$
2: $\text{dist}[i] \leftarrow \infty, \forall i \in A$
3: $\text{dist}[r] \leftarrow 0$
4: while $A \neq \emptyset$ do
5: $v \leftarrow \operatorname{argmin}_{i \in T} \text{dist}[i]$
6: $P[v] \leftarrow \text{PATH}(v, Q)$
7: $Q \leftarrow Q \cup v; A \leftarrow A \setminus v$
8: for $i \in A$ do
9: if DIST(i, A) < $\text{dist}[i]$ then
10: $\text{dist}[i] \leftarrow \text{DIST}(i, A)$
11: end if
12: end for
13: end
14: for $s \in S$ do
15: $P[s] \leftarrow \text{PATH}(s, T)$
16: end for
17: return P

端增加的代价更小,因此取这两类距离中最小的即可。□

要注意的是,定理 2 中强调了最后的汇聚树必须是合理的,如果给定的汇聚点集合 A 不合理,按照 EXTRACTPATHS 生成的汇聚树也可能是不合理的,如某个汇聚点处于单条流的路径上,并没有起到聚合作用。

2.4 汇聚树问题的复杂度分析

利用上一步的 EXTRACTPATHS 算法,容易得到这样一个暴力搜索算法:给定任务可使用的资源量 M (目前还只考虑单棵汇聚树,因此 M 即汇聚节点数量),依次尝试在网络中选择 $n = 1, 2, \dots, M$ 个汇聚点,每个 n 会带来 $C_{|V|}^n$ 种组合,然后根据 EXTRACTPATHS 求出可能的汇聚树,删去不合理的汇聚树,从剩余汇聚树中选出代价最小的。

而 EXTRACTPATHS 算法复杂度分析如下:选用没有任何优化的 Dijistra 算法作为寻找节点之间最短路径算法,算法复杂度为 $O(|V|^2)$ 。一开始确定 S, A, r 之间的所有最短路径对的复杂度为 $O((|S| + |A|) |V|^2)$ 。第 1 步,确定 $A \cup r$ 构成的最小树的算法复杂度为 $O(|A|^2)$ 。第 2 步,确定以发送端 S 为起点的流路径,每个发送端需要比较 $|A| + 1$ 条路径,这一步和上一步可以独立进行,而且这种比较在刚开始寻找最短路径时就可以顺带完成,因此这一步没有产生额外的复杂度。综上,整个算法复杂度为 $O((|S| + |A|) |V|^2 + |A|^2)$,最差情况为 A 取 V 时,复杂度为 $O(|V|^3)$ 。

因此,假如采用暴力搜索的方法求解汇聚树,考虑资源 M 足够大时 ($M \geq |V|$),需要遍历 V 的所有子集和,此时复杂度为 $O(2^{|V|} \cdot |V|^3)$ 。

3 GCAT 算法

考虑到大规模求解 ILP 的复杂度过高,本文提出一个启发式算法 (greedy cost aggregation tree, GCAT) (表 3),其输入为发送端集合 S ,接收端 r ,最大可用资源量 M ,以及节点容量 C 向量。输出为流的路径集合 P (汇聚树可由流路径构造出来)。

算法原理类似梯度下降法:令 $f(S, A, r)$ 表示由发送端集合 S ,集合 A (只有 A 中节点可作为汇聚节点)以及接收端 r 构成的最小汇聚树的代价。容易看出,随着集合 A 内节点的增加, $S \rightarrow r$ 这棵

表 3 贪心最小代价汇聚树

Table 3 Greedy cost aggregation tree

算法 2 GCAT
输入: S, r, M, C
输出:流的路径集合 P
1: $A \leftarrow \emptyset; W \leftarrow \{r\}$
2: for $s \in S$ do
3: $P[s] = r$
4: $W \leftarrow W \cup \{i \mid i \in \text{PATH}(s, r) \text{ and } C[i] > 0\}$
5: end for
6: while $M > 0$ do
7: $c \leftarrow \text{COST}(P); v \leftarrow \text{None}$
8: for $i \in W$ do
9: $P_i \leftarrow \text{EXTRACTPATHS}(S, A \cup i, r)$
10: if $\text{COST}(P_i) < c$ then
11: $c \leftarrow \text{COST}(P_i); v \leftarrow i; P \leftarrow P_i;$
12: end if
13: end for
14: if $v = \text{None}$ then
15: break
16: end if
17: $A \leftarrow A \cup v; M \leftarrow M - 1;$
18: $W \leftarrow W \setminus v \cup \{i \mid i \in P \text{ and } i \notin A\};$
19: end while
20: return P

汇聚树的代价 f 总是单调不减,因此 GCAT 算法每次扩张 A 时总是选择使得 f 下降最多的节点 (行 8~13) 进行扩张。

由 2.3 节提出的 EXTRACTPATHS 算法可得到所有流的路径集合 P ,而 f 实际上就是 P 中所有流的路径长度之和。以图 4(c) 为例,所有流的路径长度之和为 $1 + 1 + 3 + 3 + 3 + 3 = 14$ (表 3 中 $\text{COST}(P)$ 的计算方式),这和按照 ILP 模型 (P1) 基于边的计算方式 $1 \times 8 + 2 \times 3 = 14$ 是相等的。

如果遍历 V 的所有子集计算 f ,2.4 节已指出其具有指数复杂度,因此问题关键在于如何确定最小的集合 A 使得汇聚树代价最小。

3.1 逐点加入的贪心算法

为减少排列组合复杂度,算法 GCAT 做了 2 处改进。一是采取贪心的策略避免穷举:每一轮只从待选节点集合 W 中选出 1 个价值最大 (使树代价减少最多) 的节点加入 A ,后一轮的价值计算基于上一轮的选择结果,而不是从头开始 (行 17);二是待选节点集合 W 动态地进行扩展,一开始 W 为 $S \rightarrow r$ 的最短路径树上的交换节点 (行 4),在汇聚树构造过程中,不断向 W 内添加新探索到的路径上的节点,同时删除已经加入 A 的节点 (行 18)。这样就避免从一开始就考虑整个节

点集合 V 。

以图 5(a)中的拓扑为例,假设任务可使用 4 份资源,每个交换节点的容量为 1,图 5(b)是根据 GCAT 算法最终得到的汇聚树,图中的虚线是没有数据流经过的物理链路。图 6(a)~6(d)展示了算法的每一轮结果,有向边上标出的是经过的数据量(只有大于 1 才标出),图中的阴影圆节点代表汇聚点。一开始, $W = \{1, 2, 3, 4, 6, 7, 8\}$, $A = \emptyset$, $COST(P) = 24$,即集合 S 到 r 的所有最短路径长度之和。

第 1 轮,从集合 W 逐点代入 EXTRACTPATHS 得到路径并计算代价,最终发现节点 4 能够使代价减少最多,为 4,因此节点 4 加入集合 A ,从集合 W 中删除节点 4。第 1 轮结束,结果如图 6(a)所示。

第 2 轮,在 $W = \{1, 2, 3, 6, 7, 8\}$, $A = \{4\}$ 的前提下,继续从集合 W 中代入节点计算,选出能够

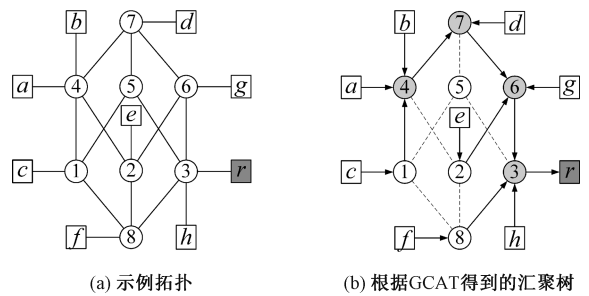


图 5 GCAT 算法示例
Fig. 5 Example of algorithm GCAT

使得代价减少最多的节点也就是节点 3 加入集合 A ,从集合 W 中删除节点 3。第 2 轮结束,结果如图 6(b)所示。

第 3、4 轮同理。整个过程中值得注意的有 2 处:1)从节点 d 出发的流在前 3 轮并没有直接走到 r 的最短路径($d \rightarrow 7 \rightarrow 6 \rightarrow 3 \rightarrow r$),而是选择先到交换节点 4 和其他流合并($d \rightarrow 7 \rightarrow 4$)后再从节点 4 走最短路径,因为这样从总的代价上看会减少 2。2)原始拓扑中的节点 5 始终没有加入待选节点集合 W ,这样就减少了需要考虑的节点数量。

3.2 复杂度分析

考虑当 M 充分大时,因为最后形成的汇聚树上必然只有流的交汇点作为汇聚点,而 $|S|$ 个叶子节点的树,最多只会有 $|S| - 1$ 个交汇点(考虑二叉树的情况),所以最多只需要循环 $|S| - 1$ 轮。

在确定第 i 个汇聚点时,已经确定的汇聚点有 $i - 1$ 个,构造这 $i - 1$ 个汇聚点和 r 的生成树复杂度为 $O(i^2)$,每个发送端出发的流判断是否要切换终点,总判断次数为 $|S|$,需执行 $|W|$ 次,总的复杂度为 $\sum_{i=1 \dots |S|-1} (O(i^2) + O(|S|)) \cdot O(|W|) = O(|S|^3 |V| + |V|^3)$ 。虽然 W 的大小是动态变化的,但是总是 V 的子集,因此总的复杂度为 $O(|S|^3 |V| + |V|^3)$ 。

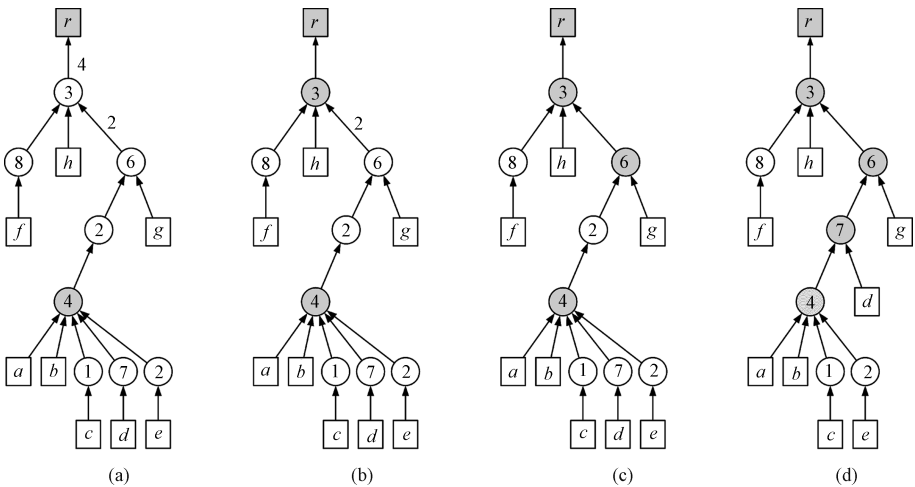


图 6 每轮得到的汇聚树(代价依次为 20,17,16,15)
Fig. 6 Each round's aggregation tree(the costs are 20,17,16, and 15 respectively)

4 多汇聚树模型

上文针对单棵汇聚树给出了 ILP 模型 (P1) 和启发式算法 GCAT, 这部分讨论如何将这两者推广到处理多棵汇聚树的场景。

$$\sum_{k \in \mathcal{K}} a^k(i) \leq m_i, \forall i \in V, \quad (12)$$

$$\sum_{k \in \mathcal{K}} \sum_{i \in V} a^k(i) \leq M. \quad (13)$$

模型 (P1) 的约束 (1) ~ (9) 对于各汇聚树而言都是独立的, 我们用 $(i)^k$ 表示汇聚树 T^k 的有关约束。由于每棵汇聚树共享了节点上资源以及任务可使用资源量, 约束 (10) (11) 需要重写为约束 (12) (13)。

$$\min \sum_{k \in \mathcal{K}} \sum_{i \in V} x^k(e), \quad (P2)$$

$$\text{s. t. } (1)^k - (9)^k, (12) (13), \forall k \in \mathcal{K}.$$

优化目标是最小化所有汇聚树代价之和, 最后关于多棵汇聚树的 ILP 模型整理为 (P2)。

算法 EXTRACTPATHS 只和汇聚树的发送端 s^k , 汇聚点集合 A^k 和接收端 r^k 有关, 因此为每棵汇聚树单独执行一次即可得到每棵汇聚树上的流路径。

算法 GCAT 运用在多棵汇聚树上时, 只要为每棵树分别记录 A^k, W^k, P^k, v^k , 然后在 GCAT 的 while 循环内每次从 K 棵树中选出能够使 $c^k - \text{COST}(P^k)$ 最小的汇聚点 v , 更新相应的汇聚树的路径。此处无法直接比较 $\text{COST}(P^k)$, 因为生成的汇聚树自身的代价小并不能说明加入的汇聚点减少的代价多, 可能汇聚树原本代价就很小。

需要注意的是, 由于所有的汇聚树共享了节点资源, 当某些节点资源被一棵树使用后为 0 时, 也要将此节点从其他 W^k 中删除。

除了第 1 轮所有树都要计算一遍以外, 之后每一轮只有上一轮被加入了汇聚点的树才需要重新评估节点, 所以 K 棵树的 GCAT 复杂度为 $O(K|S|^3|V| + |V|^3)$ 。

5 仿真结果与分析

5.1 生成树算法和网络拓扑

已有算法^[19-21]都只针对特定网络拓扑提出生成树的构造方法, 本实验分别使用各自文献中的网络拓扑和其对比效果, 同时还额外加入 Erdos-Rényi 随机网络^[26]和 2 个通用算法: 最短路径树 (Shortest) 和 Takashami 算法^[18] (一种斯坦纳

树算法) 用作比较。最后为比较 GCAT 算法和最优解 (即 2.4 节提出的暴力搜索算法) 之间的差距, 对于每种网络拓扑都做了小规模仿真验证。下面对涉及到的算法以及网络拓扑规模作简要说明。

Avalanche 算法^[19]在 FatTree 网络中构造多播树, 目标是使得最后多播树总的边数尽可能少。本实验中 FatTree 网络 pod 数量为 16, 即总共有 320 台交换机, 终端只连接在边缘交换机上。Camdoop 算法^[20]在 Torus 网络 (一种立方体结构的网络, 每个交换机和上下左右前后连接) 中构造汇聚树, 这种网络中每个节点都有 1 个三维坐标, 该算法根据发送端的三维坐标在每个维度上依次汇聚。本实验中的 Torus 网络规模为 $8 \times 8 \times 8 = 512$ 的交换机阵列, 每个交换机都连接 1 个终端。IRS 算法^[21]在 BCube(n, k) 网络 (一种递归结构网络, n 为交换机端口数量, k 为递归的次数, 终端和交换机都参与数据转发, 但本实验中只允许交换机转发) 中构造汇聚树, 由于每个节点也有相应坐标, 因此方法跟 Camdoop 算法类似, 也是根据发送端的坐标维度依次汇聚。本实验中选择 BCube(4, 3) 作为实验网络, 该网络共由 $4^{3+1} = 256$ 个终端和 $(3+1) \times 4^3 = 256$ 个交换机组成。

随机网络中交换节点以一定概率 p 和其他交换机相连。交换节点数量 $|V|$ 设置为 300~400 之间的随机值, 服从均匀分布, 每个交换节点上都连有一个终端。交换节点之间的连接概率设置为 $p = 2\ln(|V|)/|V|$ (这样设置能够保证生成的随机图大概率是连通的), 则每个节点连接的边数服从关于节点数量 $|V|$ 和 p 的二项分布, 1 个节点和大约 $|V|p = 2\ln(|V|) \approx 12$ 个节点相连, 方差为 $|V|p(1-p) \approx 12$ 。

由于这些算法没有考虑资源的分配问题, 还需加入一个资源分配策略: 当可使用资源量 M 小于生成树上的交汇点数目时, 优先选择能使得流合并后减少代价最多的交汇点作为汇聚点, 同时最后输出路径集合 P 时检查以发送端为起点的流终点是否为汇聚点, 如果不是, 要将其恢复成到 r 的最短路径, 因为在执行生成树算法时这些流的路径可能被延长了。这些算法扩展为处理 K 棵树的方法和第 4 节类似。

5.2 仿真参数

仿真参数以及性能指标已经列在表 4 中, 这里做简要说明。一棵有 $|S|$ 个发送端的汇聚树

表 4 参数和性能指标

Table 4 Simulation parameters and performance indicators

仿真	数值/定义
交换节点数量 $ V $	FatTree 为 320; Torus 为 512; BCube 为 256; ER 随机图为 [300, 400]
汇聚树数量 K	[20, 50]
发送端数量 $ S $	[50, 100]
总资源充裕度 $\alpha = M/KS$	[0.1, 0.2, \cdots, 1]
节点资源充裕度 $\beta = m/K$	[0.1, 0.2, \cdots, 1]
代价减少率 θ	$1 - \text{COST}(P)/\text{COST}(P_0)$
资源利用率 η	θ/M_{used}

最多需要 $|S| - 1$ 个汇聚点,因此 K 棵汇聚树需要的资源量不超过 KS 。任务可使用的资源量为 M ,定义任务的资源充裕度为 $\alpha = M/KS$ 。 $\alpha = 0$ 时,由于任务没有资源可使用,此时相当于 $|S|$ 条最短路径构成的汇聚树; $\alpha \geq 1$ 时,任务可使用资源量大于其所需的,多余资源量不起作用。因此我们只需关注 $0 \leq \alpha \leq 1$ 的部分。所有交换节点的容量都设置为 m ,定义交换节点资源充裕度为 $\beta = m/K$ 。当 $\beta = 0$ 时,由于节点上没有资源,此时相当于 $|S|$ 条最短路径构成的汇聚树。当 $\beta \geq 1$ 时,每个交换节点都能容纳 $\geq K$ 棵汇聚树,多余的容量不起作用。因此只需关注 $0 \leq \beta \leq 1$ 的部分。

任务的汇聚树数量 K 为 20~50 之间的随机值,每棵汇聚树包含的发送端 $|S|$ 数量为 50~

100 之间的随机值,两者均服从均匀分布,接收端 r 任取不同于 S 的终端。

将 $\alpha = 0$ 或 $\beta = 0$ 时的汇聚树代价,即 $|S|$ 条最短路径 P_0 长度之和 $\text{COST}(P_0)$ 作为基准,将加入汇聚点后的代价的减少量 $\text{COST}(P) - \text{COST}(P_0)$ 除此基准称作代价减少率 θ ,其含义为汇聚树每多使用一份资源带来的传输数据量的减少,这个值越大越好。

由于 K 棵汇聚树并不一定会使用完 $K|S|$ 大小的资源量,即任务实际使用的资源量 $M_{\text{used}} \leq K|S|$,定义 θ/M_{used} (每单位资源带来的代价减少率)为资源利用率,其含义是每消耗一份资源能够减少的数据量(百分比),这个值越大越好。

每次实验确定 $|V|, K, |S|$,调节 α 和 β 值,可分别得到 $|\alpha| \times |\beta| = 100$ 个数据点,将其分成 4 组场景:1. $\alpha \in (0, 0.5], \beta \in (0, 0.5]$,节点容量和资源量都偏少;2. $\alpha \in (0, 0.5], \beta \in (0.5, 1]$,节点容量偏少,资源量充足;3. $\alpha \in (0.5, 1], \beta \in (0, 0.5]$,节点容量充足,资源量偏少;4. $\alpha \in (0.5, 1], \beta \in (0.5, 1]$,节点容量和资源量都充足,此种情况接近于理想情况,最优解应是一棵斯坦纳树。最后统计各自区间平均值,各网络拓扑下各算法的 θ 和 η 结果如图 7、图 8 所示。

对于和最优解对比的仿真实验,拓扑规模和前文所述一致,为减少穷举时间,网络中只有 40

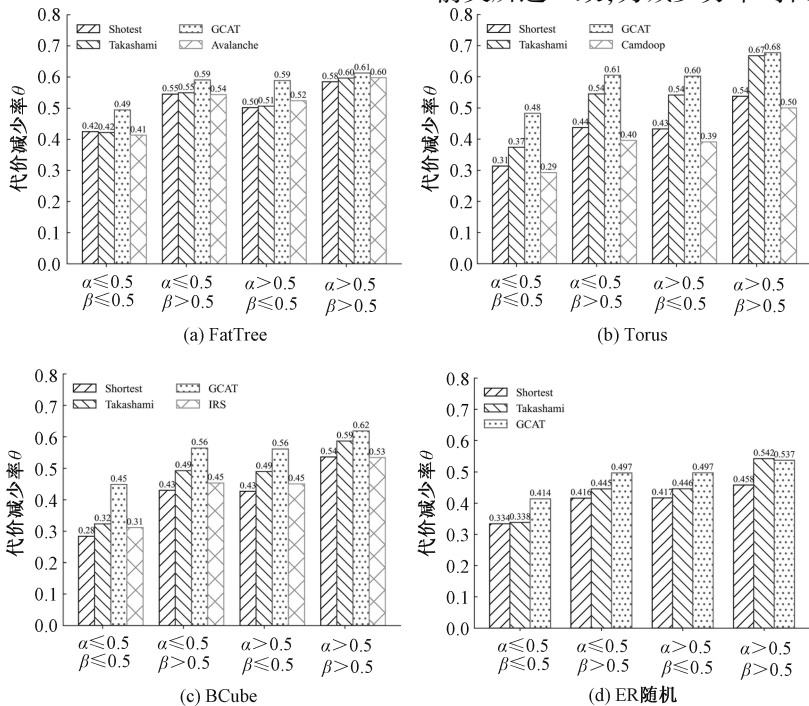


图 7 各算法在不同网络拓扑下的 θ

Fig. 7 Each algorithm's θ under different network topologies

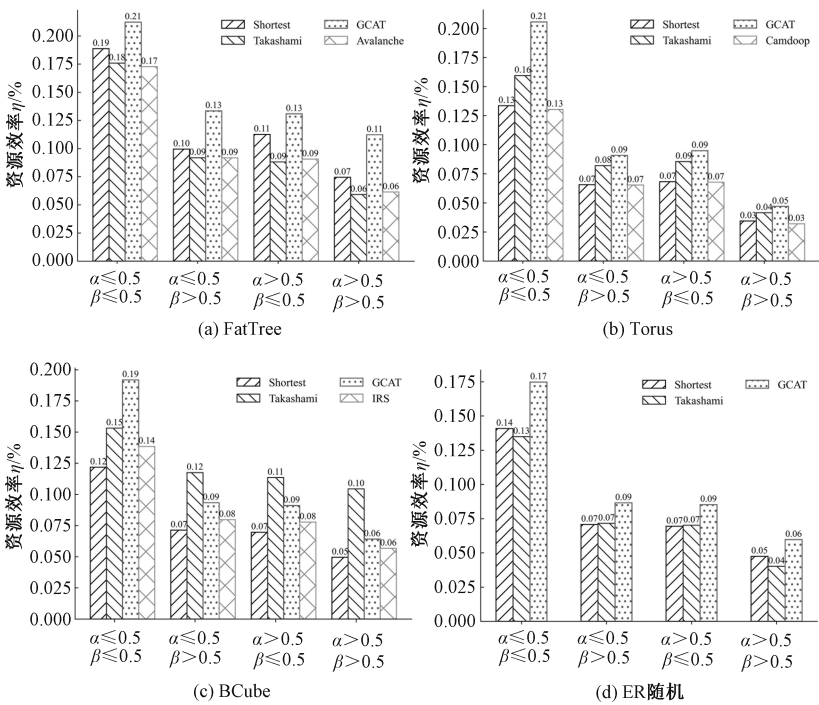


图 8 各算法在不同网络拓扑下的 η

Fig. 8 Each algorithm's η under different network topologies

个节点可作为汇聚点,任务只包含一棵发送端数量为 10 的汇聚树。实验结果展现的是场景 1~4 下启发式算法和最优解比值即 $\theta/\theta_{\text{OPT}}$ 的平均值,结果如图 9 所示。

5.3 实验结果分析

可以看出,无论是节点容量还是总资源量成为瓶颈(场景 1~3),在 4 种网络下 GCAT 对于汇聚树代价的减少都是最多的(图 7(a)~7(c))。当资源几乎不会成为瓶颈(场景 4),即此问题等价于斯坦纳树问题时,GCAT 在 ER 随机网络中的表现要略逊于启发式斯坦纳算法 Takashami

(图 7(d))。

从资源利用率(图 8)的角度看,GCAT 在各个场景下均显著高于其他算法。原因在于,最初的每份资源带来的代价减少较大,随着资源的增加,每份资源带来的代价减少会逐渐变小,当代价的减少需要 ≥ 2 个汇聚点时,由于 GCAT 一次只考虑 1 个汇聚点,会提前停止对汇聚点的使用,这种特性使其对资源有着较高的利用率,但同时也减少了其进一步降低代价的可能性。因此 GCAT 算法尤其适用于资源受限的场景,这一点在图 7、图 8 的场景 1 体现得尤为明显。在小规模实验中(图 9),GCAT 算法相比其他启发式算法明显更接近最优解,带来显著差异的主要原因是其他算法缺少像 GCAT 一样的探索机制(代码行 18),因此如果一开始确定的生成树上缺少可作为汇聚点的交汇点,这些算法不能有效寻找到新的可用节点。

6 总结

本文提出并研究在资源限制下的最小代价汇聚树问题。首先说明此问题和斯坦纳树问题之间的关系,证明其是 NP 难问题,然后分别给出问题的 ILP 模型和启发式算法 GCAT。仿真结果表明,相较于其他启发式算法,算法 GCAT 能更高效地减少汇聚树在网络中产生的数据量,尤其是在

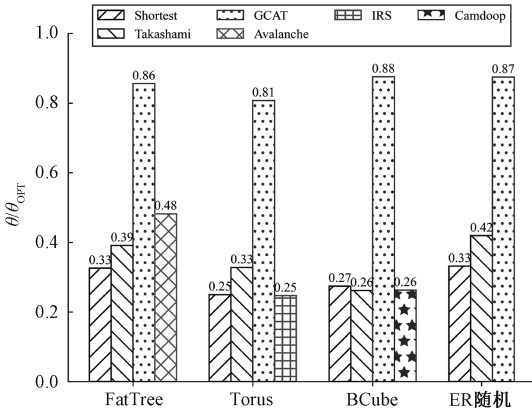


图 9 各算法的 $\theta/\theta_{\text{OPT}}$

Fig. 9 Each algorithm's $\theta/\theta_{\text{OPT}}$

资源受限明显的场景下性能更为突出。

参考文献

- [1] Li M, Andersen D G, Smola A, et al. Communication efficient distributed machine learning with the parameter server[C]//Proceedings of the 27th International Conference on Neural Information Processing Systems-Volume 1. December 8-13, 2014, Montreal, Canada. New York: ACM, 2014; 19-27. DOI: 10.5555/2968826.2968829.
- [2] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51 (1): 107-113. DOI: 10.1145/1327452.1327492.
- [3] Liu K X, Tian C, Wang Q Y, et al. Floodgate: taming incast in datacenter networks [C] // Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies. December 7 - 10, 2021, Virtual Event, Germany. New York: ACM, 2021; 30-44. DOI: 10.1145/3485983.3494854.
- [4] Li Y L, Miao R, Liu H H, et al. HPCC: high precision congestion control [C] // Proceedings of the ACM Special Interest Group on Data Communication. August 19-23, 2019, Beijing, China. New York: ACM, 2019; 44-58. DOI: 10.1145/3341302.3342085.
- [5] Viswanathan R, Balasubramanian A, Akella A. Network-accelerated distributed machine learning for multi-tenant settings[C]//Proceedings of the 11th ACM Symposium on Cloud Computing. October 19-21, 2020, Virtual Event, USA. New York: ACM, 2020; 447-461. DOI: 10.1145/3419111.3421296.
- [6] McCauley J, Panda A, Krishnamurthy A, et al. Thoughts on load distribution and the role of programmable switches[J]. ACM SIGCOMM Computer Communication Review, 2019, 49 (1): 18-23. DOI: 10.1145/3314212.3314216.
- [7] Ports D R K, Nelson J. When should the network be the computer? [C] // Proceedings of the Workshop on Hot Topics in Operating Systems. May 13-15, 2019, Bertinoro, Italy. New York: ACM, 2019; 209-215. DOI: 10.1145/3317550.3321439.
- [8] Sapio A, Abdelaziz I, Aldilajan A, et al. In-network computation is a dumb idea whose time has come [C] // Proceedings of the 16th ACM Workshop on Hot Topics in Networks. 30 November, 2017, Palo Alto, CA, USA. New York: ACM, 2017; 150-156. DOI: 10.1145/3152434.3152461.
- [9] Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture[C]//Proceedings of the ACM SIGCOMM 2008 conference on Data communication. August 17-22, 2008, Seattle, WA, USA. New York: ACM, 2008; 63-74. DOI: 10.1145/1402958.1402967.
- [10] Maculan N. The steiner problem in graphs* [J]. North-Holland Mathematics Studies, 1987, 132: 185-211. DOI: 10.1016/S0304-0208(08)73236-5.
- [11] Intanagonwiwat C, Govindan R, Estrin D. Directed diffusion: a scalable and robust communication paradigm for sensor networks[C] // Proceedings of the 6th annual international conference on Mobile computing and networking. August 6-11, 2000, Boston, Massachusetts, USA. New York: ACM, 2000; 56-67. DOI: 10.1145/345910.345920.
- [12] Villas L A, Boukerche A, Ramos H S, et al. DRINA: a lightweight and reliable routing approach for in-network aggregation in wireless sensor networks [J]. IEEE Transactions on Computers, 2013, 62 (4): 676-689. DOI: 10.1109/TC.2012.31.
- [13] Sahasrabudhe L H, Mukherjee B. Multicast routing algorithms and protocols: a tutorial [J]. IEEE Network, 2000, 14 (1): 90-102. DOI: 10.1109/65.819175.
- [14] Ramanathan S. Multicast tree generation in networks with asymmetric links [J]. IEEE/ACM Transactions on Networking, 1996, 4 (4): 558-568. DOI: 10.1109/90.532865.
- [15] Yang F, Wang Z, Ma X X, et al. Understanding the performance of In-network computing: a case study [C] // 2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom). December 16-18, 2019, Xiamen, China. IEEE, 2020; 26-35. DOI: 10.1109/ISPA-BDCloud-SustainCom-SocialCom48970.2019.00016.
- [17] Segal R, Avin C, Scalosub G. SOAR: minimizing network utilization with bounded in-network computing [C] // Proceedings of the 17th International Conference on emerging Networking EXperiments and Technologies. December 7-10, 2021, Virtual Event, Germany. New York: ACM, 2021; 16-29. DOI: 10.1145/3485983.3494853.
- [18] Takashami H, Matsuyama A. An approximate solution for the Steiner problem in graphs [J]. Math Japonica, 1980, 24 (6): 573-577.
- [19] Iyer A, Kumar P, Mann V. Avalanche: data center Multicast using software defined networking [C] // 2014 Sixth International Conference on Communication Systems and Networks (COMSNETS). January 6-10, 2014, Bangalore, India. IEEE, 2014; 1-8. DOI: 10.1109/COMSNETS.2014.6734903.
- [20] Costa P, Donnelly A, Rowstron A, et al. Camdoop: exploiting in-network aggregation for big data applications [C] // 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12). April 25-27, 2012, San Jose, CA. USENIX Association, 2012; 29-42. DOI: 10.5555/2228298.2228302.

[21] Guo D K, Xie J J, Zhou X L, et al. Exploiting efficient and scalable shuffle transfers in future data center networks[J]. IEEE Transactions on Parallel and Distributed Systems, 2015, 26 (4): 997-1009. DOI: 10.1109/TPDS.2014.2316829.

[22] Abu-Libdeh H, Costa P, Rowstron A, et al. Symbiotic routing in future data centers[C]//Proceedings of the ACM SIGCOMM 2010 conference. 30 August, 2010, New Delhi, India. New York: ACM, 2010: 51-62. DOI: 10.1145/1851182.1851191.

[23] Guo C X, Lu G H, Li D, et al. BCube: a high performance, server-centric network architecture for modular data centers [C]//Proceedings of the ACM SIGCOMM 2009 conference on Data communication. August 16-21, 2009, Barcelona, Spain. New York: ACM, 2009: 63-74. DOI: 10.1145/1592568.1592577.

[24] Graham R L, Bureddy D, Lui P, et al. Scalable hierarchical aggregation protocol (SHArP): a hardware architecture for efficient data reduction [C] // 2016 First International Workshop on Communication Optimizations in HPC (COMHPC). November 18-18, 2016, Salt Lake City, UT, USA. IEEE, 2017: 1-10. DOI: 10.1109/COMHPC.2016.006.

[25] Edmonds J. Optimum branchings[J]. Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics, 1967, 71B (4): 233-240. DOI: 10.6028/jres.071b.032.

[26] Ganesh A, Massoulié L, Towsley D. The effect of network topology on the spread of epidemics[C]//Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies. March 13-17, 2005. Miami, FL, USA. IEEE, 2005: 1455-1466. DOI: 10.1109/INFCOM.2005.1498374.