Inferring Network Topology for Distributed Machine Learning Model Training

Renjun An, Kui Wu, and Zewen Huang
Department of Computer Science, University of Victoria, Canada
renjunan@uvic.ca, wkui@uvic.ca, zwhuang@uvic.ca

***Abstract*—With the adoption of distributed machine learning across industries, there is an increasing demand for model training on cloud computing resources. However, many cloud computing service providers refuse to disclose information about the underlying network topology to end-users for commercial and security reasons. Due to this opacity, it is challenging to distribute the computation modules across different Virtual Machines (VMs) to achieve optimal resource utilization. To address this problem, we propose an algorithm, Flow Tracking (FT), that uses external measurements to infer the internal structure of a general graph. Compared with state-of-the-art topology inference algorithms, FT achieves the most accurate topology, as measured by four metrics. Notably, FT achieves 100% reconstruction of the underlying topology under the shortest-path routing strategy. We also developed two resource allocation methods, called Cluster Embedding and Average-Based Matching, that leverage topology information for distributed model training. In experiments, resource allocation based on the inferred topology significantly improves model training efficiency compared with random allocation.**

I. Introduction

Cloud computing has become the mainstream practice for training large machine learning models [1]. Its appeal lies in exceptional flexibility and cost-effectiveness, allowing users to scale resources on demand without significant upfront investment. Despite these advantages, a critical challenge for cloud computing clients is the opacity of the cloud's internal network topology. The network topology, which describes how network elements are interconnected, significantly influences the performance of cloud services [2] and the strategies for distributed model training [3]. However, cloud clients often have limited visibility into the physical or virtual network configurations due to the inherent design of cloud services, which prioritize security and abstraction [4]. As a result, clients are aware of the capacity of their allocated computing and storage resources, but the specifics of how these resources are interconnected remain hidden and accessible only to cloud service providers.

Data centers constitute the foundational infrastructure of cloud computing, in which services are typically delivered via a collaborative network of one or more data centers. Fig. 1 illustrates a typical data center network topology, structured into core, aggregation, and access layers [5], [6]. Cloud computing users access files or computational resources hosted on servers via the internet, sequentially connecting through the core, aggregation, and access layers. When training large machine learning models using multiple virtual machines (VMs)[1], cloud users establish an overlay network at the application layer. In this overlay network, a link between two VMs signifies the need to transfer data between them. It is crucial to note that there is strict information isolation: the overlay network topology and its traffic patterns are private to cloud users, while the data center network topology is visible only to cloud service providers.

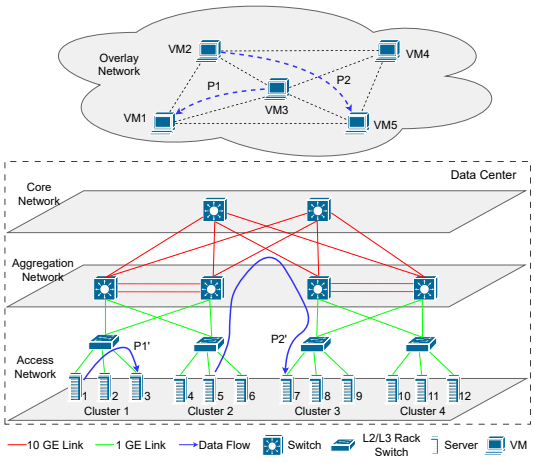


Fig. 1: Typical data center network topology.

From the perspective of cloud users, effectively utilizing allocated cloud resources requires tackling two critical challenges:

- How to infer the structure of the underlying data center network that connects VMs?
- Given the inferred data center network topology and a specific machine learning training framework, how to allocate machine learning modules to different VMs?

We use Fig. 1 to further illustrate the importance of the above problems. Due to the resource provisioning strategies employed by cloud service providers, server resources allocated to individual users may be co-located on a single server cluster or distributed across multiple clusters within the same or different data centers. For instance, VM 1 resides on Server 3, VM 3 on Server 1, VM 2 on Server 5, and VM 5 on Server 7. When considering traffic patterns between VMs, such as from VM 3 to VM 1 (referred to as path $P1$ in the

[1]Here, we use the term VM as a general abstraction for computing or storage units without specifying detailed resource types or implementations.

ISBN 978-3-903176-82-9 © 2026 IFIP

overlay and $P1'$ in the underlay), optimal performance benefits from higher bandwidth and lower latency in communication between Server 1 and Server 3. Conversely, traffic from VM 2 to VM 5 (path $P2$ in the overlay and $P2'$ in the underlay) faces reduced bandwidth and increased latency. These performance discrepancies stem from inter-cluster communication, which requires data packets to traverse both the aggregation and core layers, thereby increasing the risk of network congestion and queuing delays [6]. If cloud users could infer the underlay network structure connecting VMs, strategically placing two machine learning modules that frequently communicate in VM 3 and VM 1 would be more advantageous than placing them in VM 2 and VM 5. However, cloud users lack the necessary information to make such informed decisions.

We are motivated to answer the above questions. The first question addresses a longstanding challenge in network tomography [7]–[11]. Nevertheless, most existing research assumes a multicast tree structure for routing from a source to multiple destinations, a topology inference approach termed tree-based. This approach, however, proves inadequate for data center networks where such a tree structure cannot always be assumed. Recently, graph-based topology inference solutions [11]–[13] have emerged, but the state-of-the-art methods were primarily developed for different contexts such as content distribution networks (CDNs) [13] or network function virtualization (NFV) [11]. Our investigation reveals that topologies inferred using existing graph-based methods may not align well with the practical needs of distributed machine learning model training. This seemingly counterintuitive assertion is because of two reasons: (1) achieving 100% accuracy in inferring ground-truth topology is often impossible, and (2) there lacks a universally accepted metric to judge one inferred topology's superiority over another. Thus, *evaluating the quality of inferred topologies must be contextualized by the specific applications that run on these topologies.*

The second question concerns research on virtual graph embedding [14]. This is, in general, an NP-hard problem [15], and our problem is no exception. Nevertheless, our problem has unique features that allow us to develop an approximate greedy solution: a distributed machine learning framework can be (roughly) modelled as a weighted graph, and data center networks usually have special topological features, particularly VM clusters. In addition, we can formulate a small-scale problem as a mixed integer nonlinear programming problem and use existing optimization solvers to find the solution.

The main contributions of this work are:

1) To our knowledge, we are the first to perform topology inference studies on general graphs in a cloud computing scenario.
2) We propose a topology inference algorithm, Flow Tracking (FT), suitable for general graphs and aligned with evaluation criteria for topology inference in data centers. We also develop two topology-aware task allocation schemes, Cluster Embedding and Average-Based Matching, for distributed training of machine learning models.
3) We comprehensively evaluate the performance of FT with four different metrics, demonstrating its superiority over existing state-of-the-art topology inference algorithms. Notably, FT achieves 100% topology reconstruction under Dijkstra's shortest-path routing strategy. Our topology-aware task allocation schemes significantly reduce communication costs for distributed model training.

II. System Model and Background

A. Network Model

In modelling the data center network topology, we use a weighted and directed graph $G = (V, E)$, where V represents the nodes in the network, including switches, routers and servers. The edges, denoted by E, represent the physical connections between nodes, and each edge $e \in E$ carries a weight. This weight, denoted by u_e, evaluates the performance of the edge, which could be related to loss, delay, or bandwidth. We use link and edge interchangeably in this work.

A sequence of nodes and edges forms a network path p from one server to another. This can be regarded as communication from one compute instance or virtual machine to another one. We only consider additive metrics, such as delay and loss[2]. A metric is called additive if, measured in this metric, the end-to-end performance is the sum of the performance of individual links along this end-to-end path. Thus, the performance of p can be assessed by the sum of the weights of the edges it passes through, calculated as $\sum_{e \in p} u_e$.

B. Measurement Model

The advantages of using an additive metric in network topology inference [16] include: 1) it is non-negative and satisfies the additive property; 2) it can be measured end-to-end without requiring cooperation from the internal network; 3) it follows specific rules when measuring multiple paths simultaneously. Multicast probes are highly suitable for measuring multiple network paths simultaneously. However, due to their limited application in IP networks and SDN, [17] utilized a back-to-back approach, also known as sandwich probes, to simulate multicast probes. Initially, this simulation was restricted to two paths. Lin et al. [12] extended this method to k paths, referring to it as k-cast. We establish our measurement model within this framework.

We define a set $P = \{p_1, p_2 ... p_n\}$ that contains all measurement paths, with each subset $C \subseteq P$ including one or multiple paths. For an edge e in the network, we denote α_e as the probability of a successful probe transmission over e, where $\alpha_e \in [0, 1]$, thus this edge's additive weight will be defined as $u_e = -\log \alpha_e$. If a probe traverses all edges, denoted by E_C, of paths in the subset C successfully, it will give us a weight, called cast weight [12] ϕ_C, which can be formulated as

$$\phi_C = -\log\left(\prod_{e \in E_C} \alpha_e\right) = \sum_{e \in E_C} u_e \tag{1}$$

[2]The loss metric becomes additive if we take a logarithm operation on it.

where u_e embodies the edge weight. Thus, ϕ_C quantifies the collective performance of edges within E_C, serving as a comprehensive performance index of network paths.

Remark 1. *While we use loss as an example additive metric, the method developed in this paper is applicable to other additive metrics, such as utilization and delay. Methods for measuring these metrics in practice are beyond our scope, but an introduction can be found in [16].*

C. Principles in Topology Inference

Inferring topology through end-to-end measurements has been proven challenging. Different network topologies can exhibit identical measurement results, which is an inherent limitation of the topology inference problem [12]. Therefore, it is essential to consider practical application scenarios and propose constraints suitable for specific contexts.

Returning to our primary concern, we aim to optimize communication efficiency between virtual machines through topology awareness. This involves choosing servers that are physically closer and selecting paths with minimal blocking probabilities in the underlying topology to achieve high-capacity, low-latency communication. In cloud computing network topologies, the distance of two servers can be represented by the number of hops their communication traverses, and the degree of a node can represent the extent to which it is shared by multiple paths. Generally, traversing more hops and more high-degree nodes implies higher latency and higher blocking probability. In other words, we should prioritize the accuracy w.r.t. hop counts and node degree during the topology inference. *To be specific, we prioritize the topology that meets the following conditions*:

1) The topology should closely replicate the hop count information of the ground truth paths. If an exact replication is not possible, it should at least correlate positively with the hop counts of the ground truth paths.
2) The topology should identify and replicate high-degree nodes within the ground truth topology.

D. Why Is Network Tomography Necessary?

Astute readers might question the necessity of network tomography over simpler network diagnostic tools like traceroute for inferring network topology. Although it might seem that traceroute can easily provide hop count information, this assumption is overly optimistic. In our context, the network tomography approach is indispensable for two compelling reasons.

First, data centers and cloud providers often restrict the use of traceroute by tenants due to security, privacy, and network performance concerns. Previous investigation has obtained "empirical evidences suggesting the presence of mechanisms nullifying traditional topology-discovery strategies such as ICMP filtering. These mechanisms prevented us to gain knowledge about the relative position of the VMs through the adoption of tools like traceroute and ping [18]." Traceroute can expose details about the network infrastructure and topology, which could potentially lead to security vulnerabilities if misused by malicious actors. It also generates additional traffic that could affect network performance for other users. Due to these reasons, network diagnostic tools like traceroute are generally not allowed to generate inside-cloud ICMP probing traffic, and their use will be restricted or monitored closely.

Second, even if a cloud tenant can find out the hop count between its VMs, accurately identifying the network topology—especially the shared links among different paths—requires network tomography techniques.

TABLE I: Notations

Notation	Explanation
α_e	The probability of a probe transmission over edge e successfully.
u_e	$u_e = -\log \alpha_e$
ϕ_C	Measurement result of a subset C paths, defined in (1) and (4)
Γ_A	A set of edges with the same category A, defined in (2)
w_A	The sum of the weights of all edges belonging to the same category A
δ_A	Amendment to category weight w_A

III. A New Topology Inference Algorithm

A. An Amendment to Category Weight

There are very few topology inference algorithms based on general graphs. Lin et al. [12] proposed a new perspective called category weight. In this approach, edges traversed by the measured paths are divided into different categories. The specific method involves considering a non-empty subset A of n measured paths as a category Γ_A. All edges traversed by this subset of paths, and *only by this subset*, are assigned to this category. Formally, category Γ_A is defined as

$$\Gamma_A = \{e \in E : e \in p_i \text{ iff } i \in A\}, \tag{2}$$

where $A \subseteq [n]$ and $A \neq \emptyset$. Note that $[n] := \{1, ..., n\}$. The sum of the edge weights within the same category A is referred to as the *category weight* w_A. We call a category non-zero if its category weight is non-zero. In addition, $|A|$ is also called the size of the category Γ_A, e.g., the size of $\Gamma_{1,2}$ is 2.

To help better understand the practical meaning of category and category weight, let's assume we have two measurement paths p_1 and p_2. We only consider edges that are traversed by the measurement paths because we have no information to infer the existence of an edge if no measurement covers it. Then, each edge covered by the measurements must fall into one of three categories: Γ_1 (those only covered by p_1), Γ_2 (those only covered by p_2), and $\Gamma_{1,2}$ (those covered by both p_1 and p_2). Finding non-zero categories is critical for topology inference because it indicates whether or not a subset of measurement paths share edges. For instance, if $\Gamma_{1,2}$ is non-zero, then we know paths p_1 and p_2 share edges. Also, it is important to note that category weights have been shown to be the finest granularity information that we can obtain using only end-to-end measurements [12], [13].

By relating the concept of category weight to the measurement model introduced in Section II-B, we can derive the following equations:

$$w_A = \sum_{e \in \Gamma_A} u_e \tag{3}$$

$$\phi_C = \sum_{A \cap C \neq \emptyset} w_A \tag{4}$$

Then, we solve the category weight inference problem, i.e., inferring the category weights w_A from the measured cast weights ϕ_C. This problem has been well addressed in existing work [11]–[13]. As such, we re-use existing work for category weight inference but propose a new topology inference method based on category weights.

Category weight is considered one of the most innovative concepts in general graph-based topology inference. Here, we further enrich this concept with a minor yet necessary amendment. The amendment fixes an issue when the measurement path is non-simple, i.e., the path can traverse the same node or edge multiple times. The previously defined category weights and corresponding equations may not hold in this case. We illustrate this issue using a simple topology with two paths shown in Fig. 2, where p_1 traverses e_2 twice. According to the definition (2), e_1 and e_2 belong to $\Gamma_{1,2}$, e_3 and e_4 belong to Γ_1, and e_5 belongs to Γ_2. According to cast weight (1), $\phi_1 = u_{e_1} + 2 \times u_{e_2} + u_{e_3} + u_{e_4}$, but based on (4), $\phi_1 = w_{1,2} + w_1 = u_{e_1} + u_{e_2} + u_{e_3} + u_{e_4}$. There is an inconsistency here.

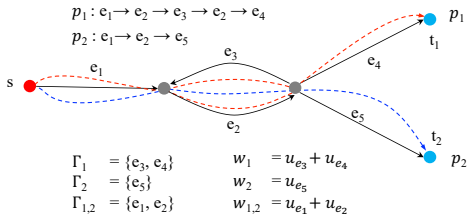


Fig. 2: Network topology with non-simple paths.

To address this issue and ensure that category weights apply to both non-simple and simple paths, we need to revise the category weight calculation in (3). When $|A| = 1$ (i.e., there is only one measurement path in the category A), we add an amendment δ_A to the definition of category weights. δ_A is the sum of the weights of all edges that this path traverses multiple times, as defined in (5), where k_e denotes the number of times the path traverses edge e. The reason for adding this amendment δ_A is that a path only contributes to the category classification of the edges it traverses the first time; repeated traversals do not change the category classification but do contribute to the category weights. Also, note that we do not need to adjust for cases when $|A| > 1$ because doing so will lead to redundant summation. For example, suppose an edge e belongs to $\Gamma_{1,2}$ and is traversed twice simultaneously by both paths p_1 and p_2. According to (3), $w_{1,2}$ only accounts for the weight u_e from the first traversal. After applying the amendment in (5), the second traversal is already included in w_1 and w_2, respectively. Therefore, no further amendment is needed for $w_{1,2}$ to avoid double counting. *In short, to ensure the correctness of the equation, all repeated traversals starting from the second one are accumulated in the corresponding category weights where* $|A| = 1$.

$$\delta_A = \sum_{e \in p_A} (k_e - 1) \cdot u_e \quad \text{where} \quad |A| = 1 \tag{5}$$

$$\hat{w}_A = \begin{cases} \delta_A + \sum_{e \in \Gamma_A} u_e & |A| = 1 \\ \sum_{e \in \Gamma_A} u_e & \text{Otherwise} \end{cases} \tag{6}$$

$$\phi_C = \sum_{A \cap C \neq \emptyset} \hat{w}_A \tag{7}$$

B. Topology Inference for Cloud Computing

Studies on algorithms that use category weights for topology inference are relatively limited. Lin et al. [11] have introduced a technique known as Clique Embedding (CE), which aims to identify the smallest or simplest topological structure that reflects the measurement results. CE involves assigning a category weight by randomly selecting an edge within the smallest clique until all non-zero categories have been allocated. Additionally, CE incorporates a dummy node r, and uses edges with 0 weights to connect r with other nodes to form valid paths. This method does not align with the criteria for inferring data center topology, discussed in Section II-C.

Let us re-consider what information category weights can provide to us. From the set of category weights, we can also derive the following *extra* information:

1) Number of non-zero categories (c_1). This value indicates that there are at least c_1 edges in the ground truth topology. Since the network must be connected, the number of nodes must be at least $c_1 + 1$.
2) Number of non-zero categories that a path hits (c_2). We call that a path hits a non-zero category if this category includes at least one edge of this path. For instance, path p_1 hits $\Gamma_{1,2}$ if this category is non-zero. The value c_2 suggests that the path traverses at least c_2 hops.
3) Number of shared categories across paths (c_3). It reveals that multiple paths share at least c_3 common edges. For instance, if $\Gamma_{1,2}$ and $\Gamma_{1,2,3}$ are both non-zero, we know that paths p_1 and p_2 share at least two edges.

Our idea: Our core idea is to *treat every non-zero category* Γ_A *as a node (with weight* w_A*) instead of an edge.* This conversion, while simple, drastically changes the way we utilize the category information. This step is critical for us to form valid paths without adding many 0-weight edges like the Clique Embedding (CE) algorithm mentioned before. With this conversion, the shared information among multiple paths will be directly converted to the node's degree information. As a result, the inferred topology better meets our criteria in Section II-C, since we aim to restore as much as possible the information of the hops in the path and the information of high-degree nodes in the ground truth topology. After this step,

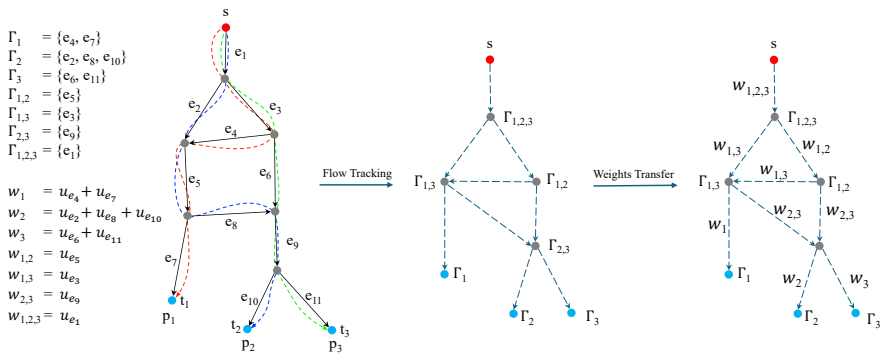


Fig. 3: An example of topology inference using Flow Tracking (FT).

we build an initial topology which has $c_1 + 1$ nodes (i.e., c_1 non-zero category nodes plus the source node) and no edges.

Then, we gradually add new edges to the initial topology by tracking the non-zero categories that each path hits. Our goal is to reconstruct each path by adding edges to existing nodes. For a path, its length in the recovered topology is recorded by c_2, and we order all non-zero categories that this path hits by their sizes. Since each non-zero category corresponds to one node in the topology, the above ordering suggests a path from the source node to the last-hop node (i.e., Γ_i for path p_i) in the current topology. We add edges as needed to complete the path in the current topology along this route. We refer to this process as *Flow Tracking*, analogous to how an imaginary water flow would move from the source to the last-hop node. The detailed steps are outlined in Algorithm 1.

Algorithm 1 Flow Tracking (FT)

Input: Set of non-zero categories Γ_A, category weights w_A, total number of paths n
Output: Directed graph G, path route L

1: $G \leftarrow$ new directed graph with a source node s
2: **for** each Γ_A **do**
3: Add node Γ_A with weight w_A to G
4: **for** $i = 1$ to n **do**
5: $L_i \leftarrow$ sort categories containing i by size, descending
6: Prepend s to L_i
7: **for** $j = 0$ to length$(L_i) - 1$ **do**
8: **if** edge $(L_i[j], L_i[j+1])$ does not exist in G **then**
9: Add directed edge $(L_i[j], L_i[j+1])$ to G
// The following weights transfer step is optional.
10: **for** each node Γ_A in G **do**
11: **for** each edge e incoming to Γ_A **do**
12: $w_e \leftarrow w_A$
13: **return** G, L

After reconstructing the topology, we may convert the topology into a weighted directed graph. This step is optional and is needed only if we want to use existing path measurements for performance estimation [13]. We employ a procedure called *Weights Transfer* (Line 10 to Line 12). For every node in the graph, we transfer the node's weight to the edges that flow into this node. This method transforms a node-weight graph into an edge-weight graph to restore all path measurement results. Fig. 3 shows an example of topology inference using FT, including the optional Weights Transfer step.

Complexity Analysis: The total time complexity of Algorithm 1 is $O(n(c_1 \log c_1))$, and the total space complexity is $O(n \cdot c_1)$, where c_1 is the total number of non-zero categories and n is the total number of paths.

IV. Task Allocation for Model Training

A. Machine-learning Framework

This section answers the second research question: Given the inferred underlay data center network topology and a specific machine learning training framework, how to allocate machine learning modules/tasks to different VMs? To formally address this question, we need an abstract model that captures the data transfer of a machine-learning framework.

Definition 1. ***Machine-learning framework*** *We model a machine-learning framework as a weighted overlay network $G_o = (V_o, E_o)$, where V_o denotes the set of nodes, E_o denotes the set of links in the network. Each node in V_o represents a machine learning module/task. A link in E_o connects two nodes, and each link is associated with a weight w_o, which (roughly) captures the total amount of data that needs to be transferred between the two nodes. We use W_o to denote the set of link weights.*

The above graph model is general to model the data transfer in various machine learning frames, such as parameter servers [19] and All-Reduce [20]. We then formally define the task allocation problem:

Task Allocation Problem (TAP): *Assume that we have an overlay machine-learning framework $G_o = (V_o, E_o)$ and an underlay data center network $G = (V, E)$. Assume that a link $(v_o, u_o) \in E_o$ has a weight w_o and $V_m \subseteq V$ includes only nodes (i.e., VMs) to which we can assign machine learning modules/tasks. Assume that $v_o \in V_o \rightarrow v \in V_m$ and $u_o \in V_o \rightarrow u \in V_m$, where $\rightarrow$ means assigning an overlay node to an underlay node. The cost of w_o in the underlay is calculated as $c_w = w_o * l$, where l is the shortest hop distance between u and v in the underlay network. Note that v and u could be the same node in G, and l is set to 0 if $u = v$. The optimal*

task allocation problem is finding a node assignment scheme: $V_o \to V_m$ such that the total cost $\sum_{w_o \in W_o} c_w$ is minimized.

Remark 2. *We intentionally use hop count between VMs to estimate the data transfer overhead. This is because delay and loss can fluctuate with network conditions, meaning that the measurement results used for topology inference may not reflect the true network performance in a long time horizon since training a large machine model normally takes a long time. In contrast, the underlying network topology does not change frequently. Therefore, the hop distance between two VMs is a more stable and reliable metric for task allocation. Also, refer to Section II-D to understand why the hop distance and topology information can not be obtained with simple tools like traceroute.*

Assume that we can only assign no more than K overlay nodes to an underlay node. Clearly, if $K \geq |V_o|$, **TAP** is trivial because we can assign all overlay nodes to one underlay node to obtain the total cost of zero. In addition, if $|V_o| > K|V_m|$, **TAP** has no solution because there is insufficient capacity to hold all the modules/tasks.

Task scheduling problems of the same kind have been proven to be NP-hard [15], particularly when they involve distributing multiple tasks across multiple processing units to minimize overall costs. We tackle this problem from two angles: (a) formulate **TAP** as a Mixed Integer Nonlinear Programming (MINLP) problem (8), enabling us to use existing tools, such as Gurobi, to find solutions for small-scale **TAP**, and (b) propose a greedy algorithm for large-scale **TAP**.

$$\min \quad \sum_{i,i'=1}^{m} \sum_{j,j'=1}^{n} X_{ij} \cdot X_{i'j'} \cdot M_{ii'} \cdot D_{jj'} \tag{8}$$

$$\text{s.t.} \quad X_{ij} \in \{0,1\} \quad \forall 1 \leq i \leq m, 1 \leq j \leq n, \tag{9}$$

$$m \leq n \cdot K, \tag{10}$$

$$\sum_{i=1}^{m} X_{ij} \leq K \quad \forall 1 \leq j \leq n. \tag{11}$$

$$\sum_{j=1}^{n} X_{ij} = 1 \quad \forall 1 \leq i \leq m, \tag{12}$$

The objective of the MINLP problem is to minimize the total communication cost. Note that m represents the number of machine-learning modules and n denotes the number of VMs. The matrix M represents the data amounts between modules, where $M_{ii'}$ denotes the data amount between module i and module i'. The matrix D represents the shortest hop counts between VMs, where $D_{jj'}$ denotes the shortest hop count between VM j and VM j'. The binary decision variable X_{ij} indicates whether or not the i-th module is placed on the j-th VM, taking a value of 1 if yes and 0 otherwise. Given that each VM can be allocated at most K modules, we have constraints (10) and (11). Constraint (12) means that a module can be assigned to only one VM.

B. Cluster Embedding

Due to the NP-hardness of **TAP**, we propose a greedy algorithm, called *Cluster Embedding*, which uses the cluster information of the inferred topology. Here, *cluster* refers to the specific topological feature in the inferred underlay network. We treat VMs connecting to the same intermediate router as a cluster. The number of modules that VMs in a cluster can accommodate is called the capacity of the cluster. We allocate modules with the highest communication costs to the same cluster as much as possible. To reduce computational overhead, we ignore the communication costs among the VMs within the same cluster and use the capacity of the cluster to approximate the total communication cost in the cluster. Such an approximation leads to a good solution if we order the clusters by their capacity and the edges in the overlay network by their weight.

Complexity Analysis: The time complexity of Algorithm Cluster Embedding is $O(n_o \log n_o + n_u \log n_u + n_o n_u)$, where n_o is the total number of edges in the overlay, n_u is the total number of clusters in the underlay.

C. Average-Based Matching (ABM)

Cluster embedding ignores the communication costs within the same cluster in the underlay network. While this approach reduces computational overhead, it may not perform well when the majority of VMs are concentrated in a few clusters. Thus, considering the communication costs within the same cluster would lead to a better solution, and we propose another greedy algorithm called *Average-Based Matching*. We compute the weighted average communication cost for each machine learning module, defined as the sum of the weights of its incident edges divided by $m-1$, where m is the total number of modules. Similarly, we calculate the weighted average for each VM in the underlying topology based on its hop count to other VMs, defined as the sum of its shortest hop counts to all other VMs, also divided by $n-1$, where n is the total number of VMs. Finally, the algorithm prioritizes assigning modules with higher average communication costs to virtual machines with lower average hop counts.

Complexity Analysis: The time complexity of Algorithm ABM is $O(m \log m + n \log n)$, where m is the total number of machine learning modules, n is the total number of VMs.

V. Performance Evaluation

A. Experimental Setup

We use two common data center network topologies, the 4-ary Fat-tree and the Dcell 1-4 network topology, as our ground truth topology. Fig. 4 (left) depicts the 4-ary Fat-tree network topology, which consists of three layers of switches. In this structure, each switch has four ports. Edge layer switches connect to two servers each, with the remaining two ports linking to the aggregation layer. Aggregation layer switches deliver traffic to the core layer. Fig. 4 (right) shows the Dcell 1-4 topology, built on the Dcell0 unit. Each Dcell0 includes a switch directly connected to four servers. To construct Dcell1, five such Dcell0 units are used. Servers within each unit

connect directly to servers in other Dcell0 units through a specific connectivity pattern, forming a larger network unit.

We employ four types of end-to-end measurement strategies to form paths between servers: 1) Random walk paths, which can create both non-simple and simple paths, meaning that paths may pass through the same node or edge multiple times; 2) Random selection from multiple simple paths; 3) Random selection from multiple shortest paths; 4) Shortest paths using Dijkstra's algorithm. By considering different ways of constructing end-to-end paths, we ensure a comprehensive evaluation of our topology inference method. Unless otherwise specified, all results are averages from 20 Monte Carlo runs.

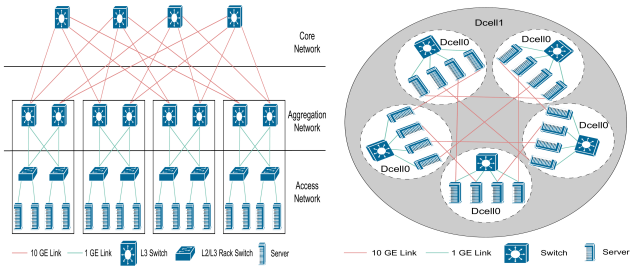


Fig. 4: Ground truth topologies.

B. Performance in Topology Inference

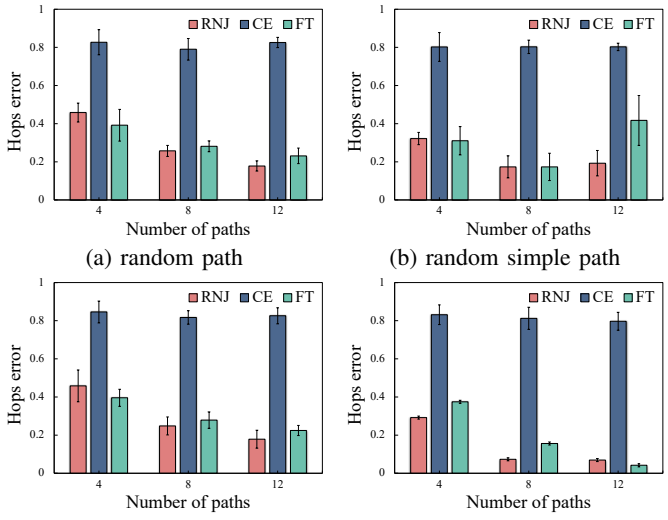
Baselines: We selected two algorithms as the baselines. One is Clique Embedding (CE) [11], which is also the state-of-the-art topology inference algorithm for general graphs. The other is Rooted Neighbor-Joining (RNJ) [16], which achieves highly accurate results if the ground truth topology is a tree.

Performance Metrics:

- *Hops error:* This metric specifically calculates the mean ratio of the hop counts on the shortest paths in the inferred topology to those in the ground truth topology.
- *Degree error:* We use two types of degree errors: the first is global degree error (GDE), which is the discrepancy between the average node degree in the inferred topology and that in ground truth topology. The second is path degree error (PDE), which measures the difference between the average node degree along the measured paths in the inferred topology and that in the corresponding paths in the ground truth topology.
- *Normalized graph edit distance (NGED):* Graph edit distance (GED) quantifies the minimum number of edit operations (i.e., insert, delete, and substitute) required to transform one graph into another. To account for the impact of network size on GED results, we normalize the GED by dividing it by the total number of edges and nodes in the ground-truth network.
- H_2 *index:* The H_2 index refers to the proportion of servers within 2 hops in the ground truth topology that also appear within 2 hops in the reconstructed topology. This metric is used because we care about VMs connecting to the same intermediate router.

Hops error results: Figs. 5 and 6 illustrate the hops error across different numbers of measurement paths, using Fat-tree or Dcell as the ground truth topology. Both figures demonstrate that our proposed FT algorithm consistently attains the lowest hops error across various path measurement strategies and the number of measurement paths. Notably, Dijkstra's shortest path routing yields superior inference accuracy, which is further enhanced as the quantity of measurement paths increases.

Degree error results: Figs. 7 and 8 show that FT achieves lower errors in both GDE and PDE. This is because FT transforms edges into nodes and incorporates path-sharing information more effectively into topology reconstruction. Similar to hops error, the degree error decreases as the complexity of the routing method is reduced (from random paths to shortest paths) and as the number of measurement paths increases. It is worth noting that the Clique Embedding (CE) algorithm infers the ground truth topology using the fewest nodes possible, which results in substantial degree errors ranging from 1 to 4. We thus omit its performance from Figs. 7 and 8 to make the figures visually clear.



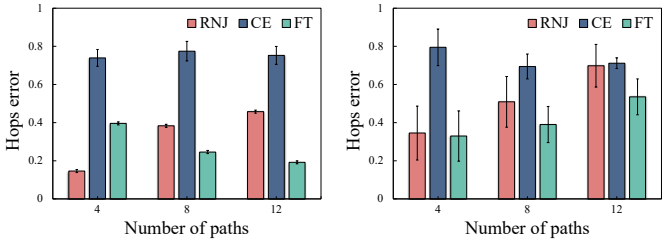
(a) random path (b) random simple path

(c) random shortest path (d) Dijkstra's shortest path

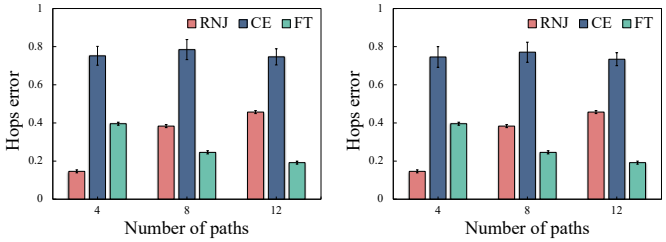
Fig. 5: Hops error in Fat-tree topology as ground truth.

NGED results: Our algorithm achieved very small errors using Dijkstra's shortest path, especially as the number of measurement paths increased. This prompts the question: Have we completely reconstructed the ground truth topology? To answer this, we show the NGED results in Fig. 9. As the number of measurement paths increased, NGED gradually decreased and ultimately approached zero, indicating that FT can successfully reconstruct the ground-truth topology with 100% accuracy under Dijkstra's shortest-path routing policy. This is because under this routing policy, every edge in the ground truth belongs to a different category if all the end-to-end paths were measured.

H_2 *index results:* Fig. 10 shows that RNJ and FT demonstrate similar trends in H_2 index performance, with FT surpassing RNJ. There is a gradual increase in the H_2 index for both Fat-tree and Dcell ground truth topologies, correlating with a progression from random path (Routing Method 1) to Dijkstra's shortest path (Routing Method 4), ultimately achiev-

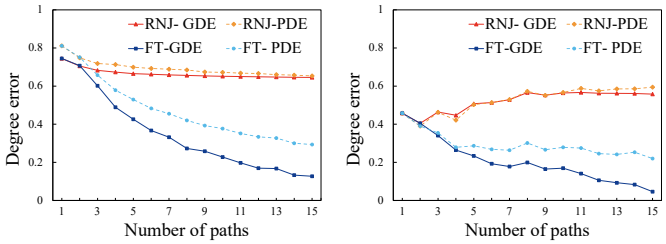


(a) random path (b) random simple path

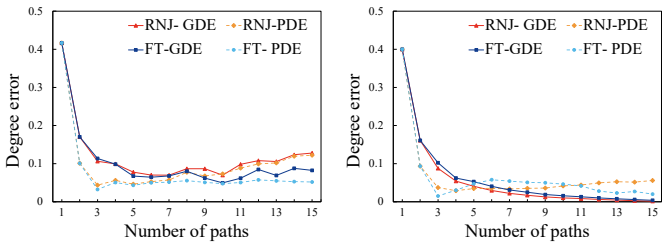


(c) random shortest path (d) Dijkstra's shortest path

Fig. 6: Hops error in Dcell topology as ground truth.



(a) random path (b) random simple path



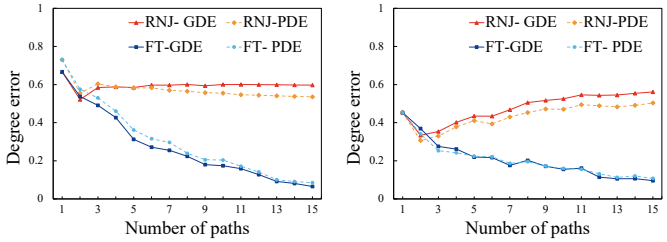
(c) random shortest path (d) Dijkstra's shortest path

Fig. 7: Degree error in Fat-tree topology as ground truth.

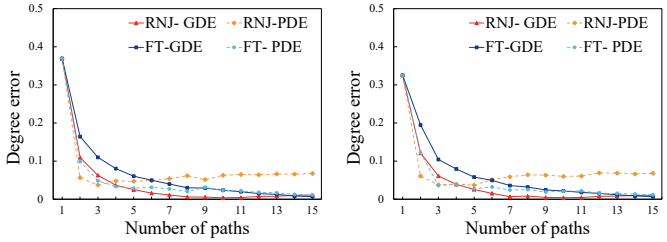
ing 100% accuracy. This trend suggests that transitioning from complex to simpler measurement paths enhances clustering accuracy. In contrast, CE exhibits varying performance and has the lowest clustering accuracy in several cases.

C. FT Helps Distributed Model Training

We evaluate how topology inference results reduce communication cost, using two commonly used overlay topologies in distributed deep learning: the Parameter Server (PS) and All-Reduce (AR). In PS, all workers download the global model from a central Parameter Server. They perform one iteration of training, compute gradients, and send them back to the Parameter Server. The server updates the global model with the new received gradients and then distributes the updated model back to all workers for the next iteration. In AR, all workers perform one iteration of training, compute gradients, and exchange them with every other worker. We assume that

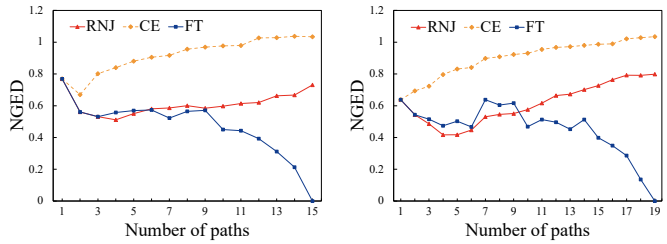


(a) random path (b) random simple path



(c) random shortest path (d) Dijkstra's shortest path

Fig. 8: Degree error in Dcell topology as ground truth.



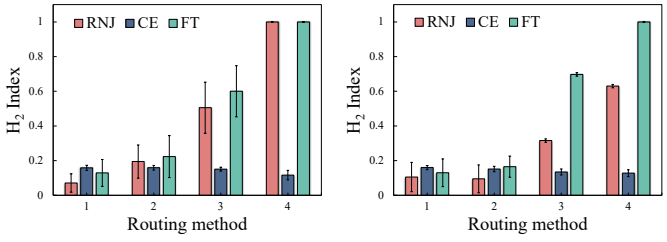
(a) Fat-tree (b) Dcell

Fig. 9: NGED in two different ground truth topologies.

the raw model parameters are transmitted without any compression. Hence, the data amount in one-hop communication is constant, and the communication cost can be solely determined by aggregated hop counts in the underlay topology.

We simulated a machine learning model consisting of 8 computational modules. We set $K = 1$ in Equations (10) and (11), allowing each VM to host only one machine learning module. Note that using different K values will not affect our conclusion. We utilize the MINLP, *Cluster Embedding* (CLE) and *Average-Based Matching* (ABM) methods, along with a random approach, to distribute these 8 modules across a varying number of VMs within the Fat-tree and Dcell underlay topologies.

Fig. 11 reveals that, compared to random allocation, MINLP



(a) Fat-tree (b) Dcell

Fig. 10: H_2 index in two different ground truth topologies.

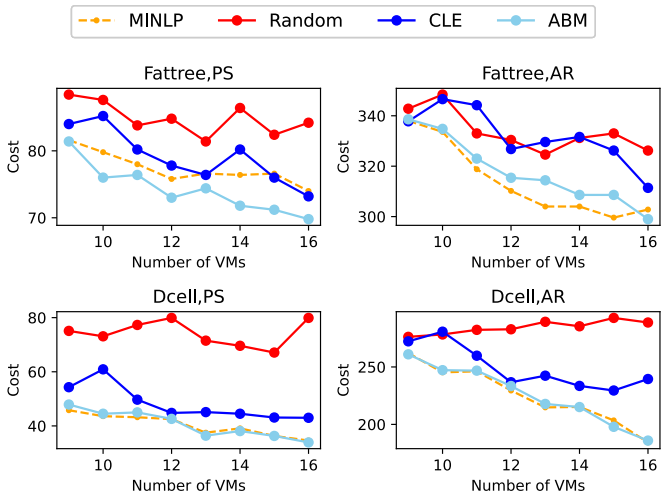


Fig. 11: Communication cost in different overlay-underlay combinations.

allocation results in lower communication costs. This is due to Gurobi's ability to derive precise solutions in small-scale models. However, MINLP does not always give the smallest cost, because MINLP gives the best solution based on the inferred topology, which is not necessarily the ground truth topology. ABM outperforms CLE, because CLE ignores the communication costs within clusters, whereas ABM accounts for them. In addition, running on a commodity computer (macOS 14.5, Intel Core i5, 16 GB RAM), MINLP requires around 10^4 times the computation time of CLE and ABM, with this ratio increasing as the task scale grows. This shows that the proposed two algorithms are more feasible for large-scale machine learning task allocation. Overall, the benefit of topology awareness for task allocation is apparent in all underlay-overlay combinations. The only exception is that CLE does not show clear advantages in the Fattree-AR combination. As we discussed above, CLE ignores the communication costs within clusters and may not perform well if most VMs are within a few clusters.

VI. Conclusion

Targeting network topology-aware distributed machine learning, we have developed a **general graph-based** topology inference method using category information. By modifying the original category weights model, we addressed its limitations concerning non-simple paths. Our proposed topology inference algorithm, Flow Tracking (FT), has been extensively validated through simulation, demonstrating superior performance over existing state-of-the-art algorithms across a comprehensive list of metrics. Notably, FT achieves 100% topology reconstruction accuracy when routing uses Dijkstra's shortest path algorithm. When applied to distributed machine learning training models, the inferred topology enables the design of effective task allocation schemes.

References

[1] Y. Wang, Q. Bao, J. Wang, G. Su, and X. Xu, "Cloud computing for large-scale resource computation and storage in machine learning," *Journal of Theory and Practice of Engineering Science*, vol. 4, no. 03, pp. 163–171, 2024.

[2] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, 2010, pp. 267–280.

[3] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th USENIX Symposium on operating systems design and implementation (OSDI 14)*, 2014, pp. 583–598.

[4] R. Jain and S. Paul, "Network virtualization and software defined networking for cloud computing: a survey," *IEEE Communications Magazine*, vol. 51, no. 11, pp. 24–31, 2013.

[5] S. K. Uzaman, J. Shuja, T. Maqsood, F. Rehman, S. Mustafa *et al.*, "A systems overview of commercial data centers: initial energy and cost analysis," *International Journal of Information Technology and Web Engineering (IJITWE)*, vol. 14, no. 1, pp. 42–65, 2019.

[6] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM computer communication review*, vol. 38, no. 4, pp. 63–74, 2008.

[7] R. Caceres, N. Duffield, J. Horowitz, F. L. Presti, and D. Towsley, "Loss-based inference of multicast network topology," in *Proceedings of the 38th IEEE Conference on Decision and Control (Cat. No. 99CH36304)*, vol. 3. IEEE, 1999, pp. 3065–3070.

[8] S. Ratnasamy and S. McCanne, "Inference of multicast routing trees and bottleneck bandwidths using end-to-end measurements," in *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, vol. 1. IEEE, 1999, pp. 353–360.

[9] N. G. Duffield, J. Horowitz, and F. L. Prestis, "Adaptive multicast topology inference," in *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, vol. 3. IEEE, 2001, pp. 1636–1645.

[10] N. G. Duffield, J. Horowitz, F. L. Presti, and D. Towsley, "Multicast topology inference from measured end-to-end loss," *IEEE Transactions on Information Theory*, vol. 48, no. 1, pp. 26–45, 2002.

[11] Y. Lin, T. He, S. Wang, K. Chan, and S. Pasteris, "Looking glass of nfv: Inferring the structure and state of nfv network from external observations," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1477–1490, 2020.

[12] ——, "Multicast-based weight inference in general network topologies," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[13] Y. Huang and T. He, "Overlay routing over an uncooperative underlay," in *Proceedings of the Twenty-fourth International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing*, 2023, pp. 151–160.

[14] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[15] J. D. Ullman, "Np-complete scheduling problems," *Journal of Computer and System sciences*, vol. 10, no. 3, pp. 384–393, 1975.

[16] J. Ni, H. Xie, S. Tatikonda, and Y. R. Yang, "Efficient and dynamic routing topology inference from end-to-end measurements," *IEEE/ACM transactions on networking*, vol. 18, no. 1, pp. 123–135, 2009.

[17] M. Coates, R. Castro, R. Nowak, M. Gadhiok, R. King, and Y. Tsang, "Maximum likelihood network topology identification from edge-based unicast measurements," *ACM SIGMETRICS Performance Evaluation Review*, vol. 30, no. 1, pp. 11–20, 2002.

[18] V. Persico, P. Marchetta, A. Botta, and A. Pescape, "On network throughput variability in microsoft azure cloud," in *2015 IEEE Global Communications Conference (GLOBECOM)*, 2015, pp. 1–6.

[19] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[20] Y. Bao, Y. Peng, Y. Chen, and C. Wu, "Preemptive all-reduce scheduling for expediting distributed dnn training," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*. IEEE, 2020, pp. 626–635.