Inferring Network Topology for Distributed Machine Learning Model Training

by

Renjun An
B.Eng., Fudan University, 2012
M.Sc., Fudan University, 2015

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Inferring Network Topology for Distributed Machine Learning Model Training

by

Renjun An
B.Eng., Fudan University, 2012
M.Sc., Fudan University, 2015

Supervisory Committee

Dr. Kui Wu, Supervisor
(Department of Computer Science)

Dr. Jaya Prakash Champati, Departmental Member
(Department of Computer Science)

ABSTRACT

With the application of distributed machine learning in various industries, there is an increasing demand for model training using cloud computing resources. However, many cloud computing service providers refuse to provide end-users with information about the underlying network topology for commercial and security reasons. Due to this opaqueness, it is challenging to arrange the computation modules in different Virtual Machines (VMs) to achieve the best resource utilization efficiency. To address this problem, we propose an algorithm called Flow Tracking (FT), which uses external measurements to infer the internal structure of a general graph. Compared to the state-of-the-art topology inference algorithms, FT achieves the most accurate topology measured in four different metrics. Notably, FT achieves 100% reconstruction of the underlying topology under the shortest-path routing strategy of the underlying network. Experimentally, resource allocation using the inferred topology improves the model training efficiency significantly compared to random allocation.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

My wife, parents, and all my family members, for their unwavering support during my lowest moments. Their love and encouragement have been my greatest strength.

Supervisor Dr. Kui Wu, for mentoring, support, encouragement, and patience.

Drs. Jaya Prakash Champati and Lin Cai, for serving on the supervisory committee and helping me improve the thesis.

Thank you.
Renjun An

DEDICATION

Just hoping this is useful!

Chapter 1

Introduction

1.1 Background

1.1.1 Network Topology Inference

Network topology information plays a critical role in network management, fault diagnosis, traffic engineering, and performance optimization. Accurate knowledge of the network topology helps network administrators make efficient routing decisions, identify and eliminate bottlenecks, and improve the overall quality and reliability of network services. For instance, in traffic engineering, understanding the network's topology allows administrators to optimize traffic distribution, prevent congestion, and ensure the priority transmission of critical business traffic. Moreover, during network failures, topology information aids in quickly locating the fault, thereby reducing downtime and minimizing the impact on services [1].

However, as network security concerns have grown, many network operators and device manufacturers have increasingly restricted access to their network topology information. Traditionally, obtaining network topology data relied on routing protocols such as BGP (Border Gateway Protocol) and OSPF (Open Shortest Path First), which typically provide rich topology data [2]. Yet, to prevent potential attackers from exploiting this information for network attacks, many network devices have reduced the visibility of these protocols or even completely blocked external queries [3]. This has made it significantly more challenging to obtain network topology information, especially in the complex and dynamic environment of modern internet.

In this context, network topology inference has emerged as a crucial technique to address this challenge. Network topology inference is an important area of network

tomography, which aims to infer the internal topology of a network using external measurement data, without relying on data from internal routing protocols [4]. This approach typically involves sending probe packets and analyzing the returned data to infer the network's topology. For example, by using end-to-end delay measurements, the connectivity between nodes and link quality can be inferred, analyzing link packet loss rates can help identify potential network bottlenecks and faults [5].

In recent years, as the scale and complexity of the internet have grown, network topology inference has become a hot topic in both academia and industry. Researchers have proposed various topology inference methods based on different measurement techniques and algorithms, including maximum likelihood estimation [6], hierarchical clustering [7], and compressed sensing [8]. These methods have been widely applied in different types of networks, such as ISP (Internet Service Provider) networks, data center networks, and mobile networks.

1.1.2 Distributed Machine Learning

Distributed machine learning is a technique that divides machine learning tasks across multiple computing nodes to enable parallel processing. As the volume of data continues to grow exponentially and model complexity increases, traditional single-machine processing methods have become inadequate for practical applications. This has led to the rise of distributed machine learning, especially in fields such as big data, deep learning, and high-performance computing, where it provides robust support for training large-scale models [9].

In distributed machine learning, computational tasks are distributed across multiple nodes, which can be physically separated servers, virtual machines, or cloud resources. Each node is responsible for processing a portion of the data or model parameters and exchanges information with other nodes through communication protocols. Common methods of information exchange include synchronous and asynchronous communication. In synchronous communication, all nodes exchange and update parameters after each computation round, ensuring model consistency. In contrast, asynchronous communication allows nodes to independently update parameters, improving computational efficiency [10].

The network plays an important role in distributed machine learning. First, the bandwidth and latency of network connections directly impact the speed of data transfer and the efficiency of model parameter updates between nodes. In high-bandwidth,

low-latency network environments, the performance of distributed machine learning can be significantly enhanced. Additionally, the network topology influences how nodes communicate and the associated communication costs. For instance, in a fully connected topology, all nodes are directly linked, allowing for rapid information exchange, whereas in a ring or tree topology, the efficiency of information transfer may be somewhat limited [11, 12]. Moreover, the reliability and security of the network are critical to the stability of distributed learning and the security of data, especially in distributed environments spanning data centers or the cloud.

1.2 Why Topology Matters?

Cloud computing has become the mainstream practice for training large machine learning models [13]. Its appeal lies in exceptional flexibility and cost-effectiveness, allowing users to scale resources on-demand without significant upfront investment [14]. 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 [15] and the strategies for distributed model training [11]. 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 [16]. 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 form the foundational infrastructure of cloud computing, where services are typically delivered through a collaborative network of one or more data centers. Fig. 1.1 illustrates a typical data center network topology, structured into core, aggregation, and access layers [17,18]. Cloud computing users access files or computational resources hosted on servers by traversing through the internet interface, sequentially connecting via the core layer, aggregation layer, and access layer. 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

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

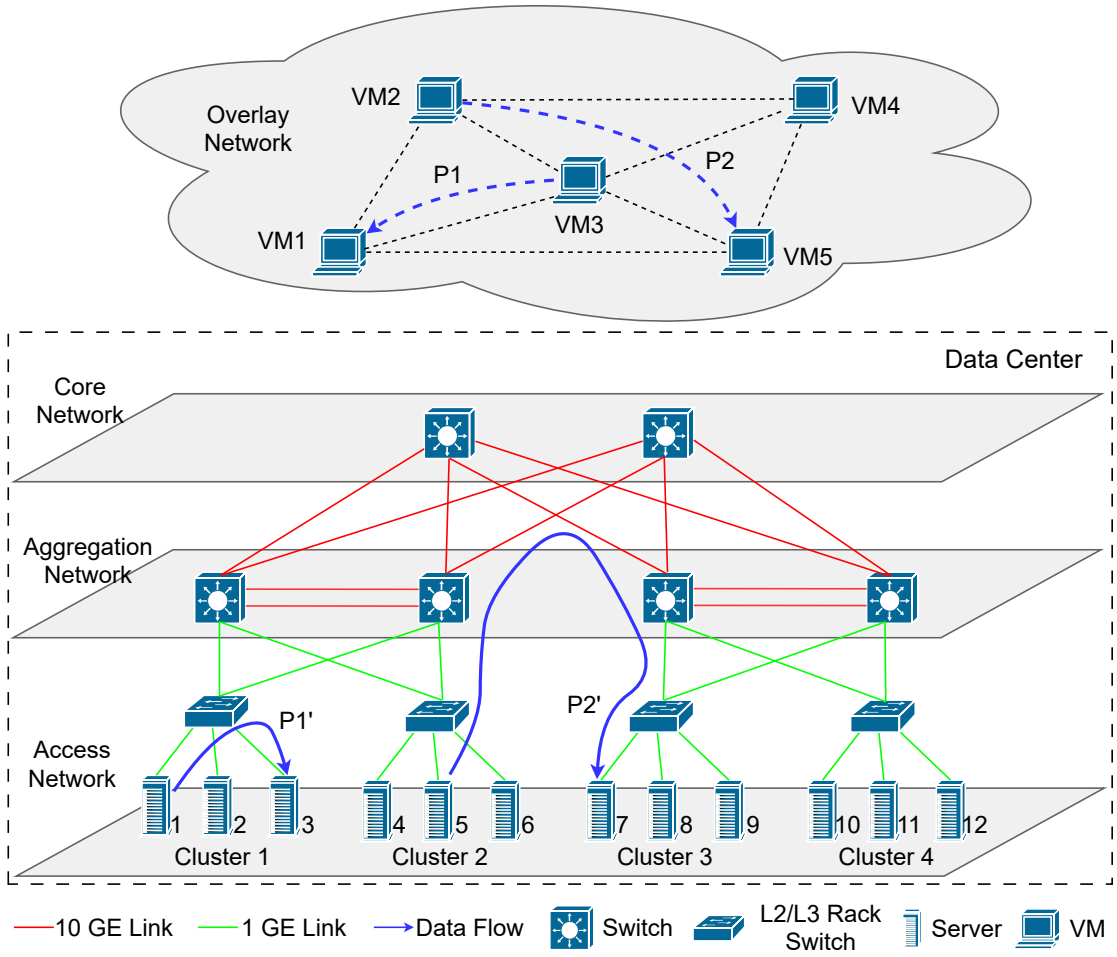


Figure 1.1: Typical data center network topology.

its traffic patterns are private to cloud users, while the data center network topology is visible only to cloud service providers.

From the perspective of cloud users, effectively utilizing allocated cloud resources needs to tackle 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. 1.1 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 within a single server cluster or distributed across various clusters within identical or disparate 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 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 pass through both aggregation and core layers, increasing the risk of network congestion and queuing delays [18]. 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. Beyond the significant impact that end-to-end data between servers has on our decisions, inferring the entire underlying network topology provides valuable insights into routing decisions, helps identify potential bottlenecks, and optimizes data flow, ultimately improving the overall performance of the network.

We are thus motivated to answer the above questions. The first question addresses a longstanding challenge in network tomography, spanning several studies [19–23]. Nevertheless, most of the existing research assumes a multicast tree structure for routing from a source to multiple destinations, termed tree-based topology inference. This approach, however, proves inadequate for data center networks where such a tree structure cannot always be assumed. Recently, graph-based topology inference solutions [23–25] have emerged, but the state-of-the-art methods were primarily developed for different contexts such as content distribution networks (CDNs) [25] or network function virtualization (NFV) [23]. 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 running on these topologies.*

The second question relates to the research on virtual graph embedding [26]. This

is, in general, an NP-hard problem [27], 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, e.g., Gurobi, to find the solution.

1.3 Related Works

1.3.1 Topology Inference with Network Tomography

Network topology inference has been one of the main themes in network tomography. Initially, correlation among packet losses observed in multicast networks at multicast receivers was used to infer the structure of the multicast tree [19, 20]. Following this, techniques based on measurements of delay and combinations of delay and loss were subsequently developed [21, 22]. However, due to the limited practical applicability of multicast configurations, researchers proposed unicast-based methods. One such method involves the use of back-to-back probing, also known as sandwich probing [6]. Utilizing this technique, several tree-based topology inference algorithms have been proposed. A Markov Chain Monte Carlo (MCMC) procedure was devised to infer the most likely network topology [6], and the Agglomerative Likelihood Tree (ALT) algorithm framed the inference challenge as a hierarchical clustering problem [28]. These topology inference algorithms can provide a tree-like topology with a certain level of accuracy. Nevertheless, the actual routing topology of data centers, shown in Fig. 1.1, is not purely tree-based and may include mesh-like configurations, e.g., when an intermediate route uses a load balancer to split a flow among multiple paths. Unfortunately, tree-based topology inference algorithms fail to capture the presence of these network structures.

Only recently have a few topology inference algorithms been based on general graphs. An algorithm called Clique Embedding was proposed to find the smallest or simplest topology that reflects measurement results, thereby inferring the structure and state of upper-layer network function virtualization (NFV) [23]. In the context of overlay networking, a polynomial-complexity algorithm was developed [25] to detect the existence of underlay links shared by each subset of paths between overlay

nodes from end-to-end measurements, where the underlay routing may form a general graph. As we will point out later, existing algorithms seek the simplest or smallest solution [23], and such criteria to judge the "quality" of inferred topology do not align with the cloud computing scenarios, where server clusters become a norm and thus clustering property should be prioritized over other criteria.

1.3.2 Topology-aware Cloud Computing

The opaque underlying network topology makes topology awareness plays a very important role in the utilization of cloud computing resources. Tashi is a location-aware cluster management system [29], which incorporates a resource telemetry service capable of reporting distances between virtual machines based on user-specified metrics. However, the paper does not detail the methods used by the telemetry service to obtain the location data of the virtual machines. Utilizing the sandwich probing technique previously mentioned, Battré et al. [30] conducted topology inference based on end-to-end measurements on open-source virtual machine managers such as KVM and XEN. They developed an improved version of the Agglomerative Likelihood Tree (ALT) algorithm and conducted comparisons with results derived from Round-Trip Time (RTT) measurements. Extending Battré's research, Saad et al. [31] proposed a new analytical cloud model that describes the physical placement of virtual machines within the communication hierarchy and validated this model on Amazon's cloud infrastructure. Nevertheless, all the topology inference algorithms applied in these studies rely on tree-based topologies, which do not accurately represent the more complex network topologies typical of compute clouds.

1.4 Contributions

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 called Flow Tracking (FT), which is suitable for general graphs and aligns with the evaluation criteria for topology inference in data centers. We also develop topology-aware task allocation schemes for distributed machine learning model training.

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.

1.5 Organization of Thesis

The rest of the thesis is organized as follows:

Chapter 2 formalizes the network model, the measurement model, and the topology inference problem.

Chapter 3 amends the definition of category weight to broaden its applicability to include non-simple paths, and proposes a topology inference algorithm based on category weight information, called Flow Tracking.

Chapter 4 develops topology-aware task allocation schemes for distributed machine learning model training, and proposes two algorithms for large-scale model training.

Chapter 5 evaluates our topology inference results using four key metrics and assesses the proposed two algorithms on the distributed machine learning model.

Chapter 6 summarizes the thesis and provides directions for future research.

Chapter 2

System Model and Background

2.1 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[1]. 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 $\sum_{e \in p} u_e$.

2.2 Measurement Model

A network measurement method based on additive metrics provide a theoretical foundation for network topology inference. This approach enables the inference of the internal topology of a network by analyzing data obtained from end-to-end network measurements. The core idea is to use the correlation between performance metrics

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

observed at the destination nodes, such as loss and delay, to infer the routing topology and link performance between the source and destination nodes [4, 5, 22].

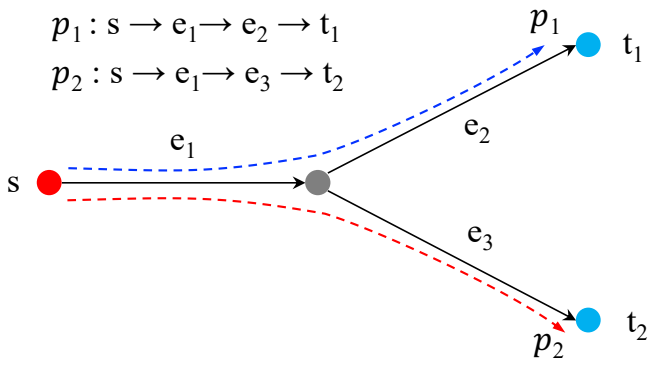


Figure 2.1: Example of Inferring Network Topology from Loss Metrics.

To illustrate this with an example, as shown in Fig. 2.1, consider performance metrics based on loss. Let α_i represent the success rate of link e_i (where $i = 1, 2, 3$). We send multicast (or unicast-multicast) probes from the source s to destinations t_1 and t_2, and let X_{p_i} (where $i = 1, 2$) denote the success rate metrics for the paths to t_1 and t_2, respectively. Assuming that the losses on all links are independent of one another, we can infer the success rates α_i using the following formula:

$$-\log \alpha_1 - \log \alpha_2 = -\log \Pr\{X_{p_1} = 1\} \tag{2.1}$$

$$-\log \alpha_1 - \log \alpha_3 = -\log \Pr\{X_{p_2} = 1\} \tag{2.2}$$

$$-\log \alpha_1 = -\log \left(\frac{\Pr\{X_{p_1} = 1\} \Pr\{X_{p_2} = 1\}}{\Pr\{X_{p_1} = X_{p_2} = 1\}} \right) \tag{2.3}$$

From the above formulas, it can be observed that if paths p_1 and p_2 share a common link, meaning they are correlated, the success rate of that link can be uniquely determined using equation (2.3). This characteristic makes it possible to explore the internal topology of the network through external measurements. However, it is important to note that each of the three links in Fig. 2.1 could represent a sequence of physical communication links. The approach described above infers only the logical topology, specifically a multicast tree with two receivers, rather than the actual physical topology.

The advantages of using an additive metric in network topology inference [32] 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, [6] 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. [24] 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 [24] ϕ_C, which can be formulated as

$$\phi_C = -\log\left(\prod_{e\in E_C} \alpha_e\right) = \sum_{e\in E_C} u_e \tag{2.4}$$

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 [32].*

2.3 Principles in Topology Inference

Inferring network topology through end-to-end measurements has long been recognized as a challenging task. One of the primary difficulties lies in the fact that different network topologies can produce identical measurement results, making it inherently difficult to distinguish between them based solely on these observations [24]. For instance, two distinct network configurations may yield the same packet loss rates or latencies when measured from the endpoints, leading to potential misinterpretations of the network's actual structure.

Given this challenge, it becomes crucial to take into account the practical scenarios in which topology inference is applied. By considering the specific context, researchers and network end users can introduce additional constraints or assumptions that help to narrow down the set of possible topologies. These constraints might include prior knowledge about the network's design, such as typical routing behaviors, known physical constraints, or historical performance data. By tailoring the inference process to the specific characteristics and requirements of the application, the accuracy of topology inference can be significantly improved, making it more reliable for real-world deployment.

Moreover, leveraging contextual information can also assist in mitigating the effects of measurement noise and inaccuracies, which are common in practical network environments. This approach not only enhances the robustness of the inference but also provides a more meaningful and actionable understanding of the network's structure, which is essential for tasks such as fault diagnosis, traffic engineering, and network optimization. Therefore, while the inherent limitations of topology inference present significant challenges, careful consideration of the application context and the introduction of appropriate constraints can greatly improve the feasibility and effectiveness of this approach.

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.

3) The reconstructed path must be valid and must have the same measurements as the path in the ground truth topology.

Table 2.1: 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 (2.4) and (3.3)
Γ_A	A set of edges with the same category A, defined in (3.1)
w_A	The sum of the weights of all edges belonging to the same category A
δ_A	Amendment to category weight w_A

Chapter 3

A New Topology Inference Algorithm

3.1 An Amendment to Category Weight

There are very few topology inference algorithms based on general graphs. Lin et al. [24] 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

$$\Gamma_A = \{e \in E : e \in p_i \text{ iff } i \in A\}, \tag{3.1}$$

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 [24, 25].

By relating the concept of category weight to the measurement model introduced in Section 2.2, we can derive the following equations:

$$w_A = \sum_{e \in \Gamma_A} u_e \tag{3.2}$$

$$\phi_C = \sum_{A \cap C \neq \emptyset} w_A \tag{3.3}$$

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 [23–25]. 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. It is worth noting that at the application-layer overlay networks, non-simple path is often necessary, e.g., a neural network module may be called repeatedly by different machine training tasks. 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. 3.1, where p_1 traverses e_2 twice. According to the definition (3.1), 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 (2.4), $\phi_1 = u_{e_1} + 2 \times u_{e_2} + u_{e_3} + u_{e_4}$, but based on (3.3), $\phi_1 = w_{1,2} + w_1 = u_{e_1} + u_{e_2} + u_{e_3} + u_{e_4}$. There is an inconsistency here.

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 (3.2). 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 (3.4), 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

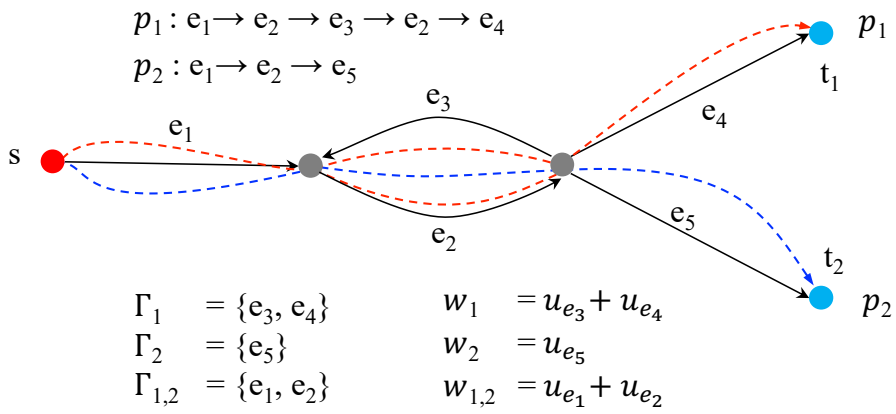


Figure 3.1: Network topology with non-simple paths.

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.

$$\delta_A = \sum_{e \in p_A} (k_e - 1) \cdot u_e \quad \text{where} \quad |A| = 1 \tag{3.4}$$

$$\hat{w}_A = \begin{cases} \delta_A + \sum_{e \in \Gamma_A} u_e & |A| = 1 \\ \sum_{e \in \Gamma_A} u_e & \text{Otherwise} \end{cases} \tag{3.5}$$

$$\phi_C = \sum_{A \cap C \neq \emptyset} \hat{w}_A \tag{3.6}$$

3.2 Topology Inference for Cloud Computing

Studies on algorithms that use category weights for topology inference are relatively limited. Lin et al. [23] 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 2.3.

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 2.3, 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, 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 [25]. 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. 3.2 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.

The proposed algorithm has the following advantages:

1) **Wide Applicability:** *Flow Tracking* is suitable for both general graphs and tree-like structures, making it more versatile compared to other topology inference algorithms that are typically limited to tree structures.

2) **Effective Path Reconstruction:** Every path reconstructed by *Flow Tracking* is valid, as it reconstructs paths by tracking the category information along each route. This approach eliminates the need for adding dummy nodes and zero-weight

edges to reconstruct valid paths, as required by algorithms like Clique Embedding.

3) **Optimized for Hierarchical, Non-Tree Topologies:** The algorithm is particularly well-suited for hierarchical but non-tree network topologies, such as those found in data centers. This is because the reconstruction order follows the sequence of category sizes from largest to smallest, which also partially reconstructs the hierarchical information inherent in the ground truth topology.

4) **Weight Transfer Enhancement:** By incorporating weight transfer, *Flow Tracking* shifts from reconstructing node-weighted graphs to edge-weighted graphs. This alignment with the ground truth topology facilitates the application of results in topology inference scenarios where performance metrics are evaluated based on weights.

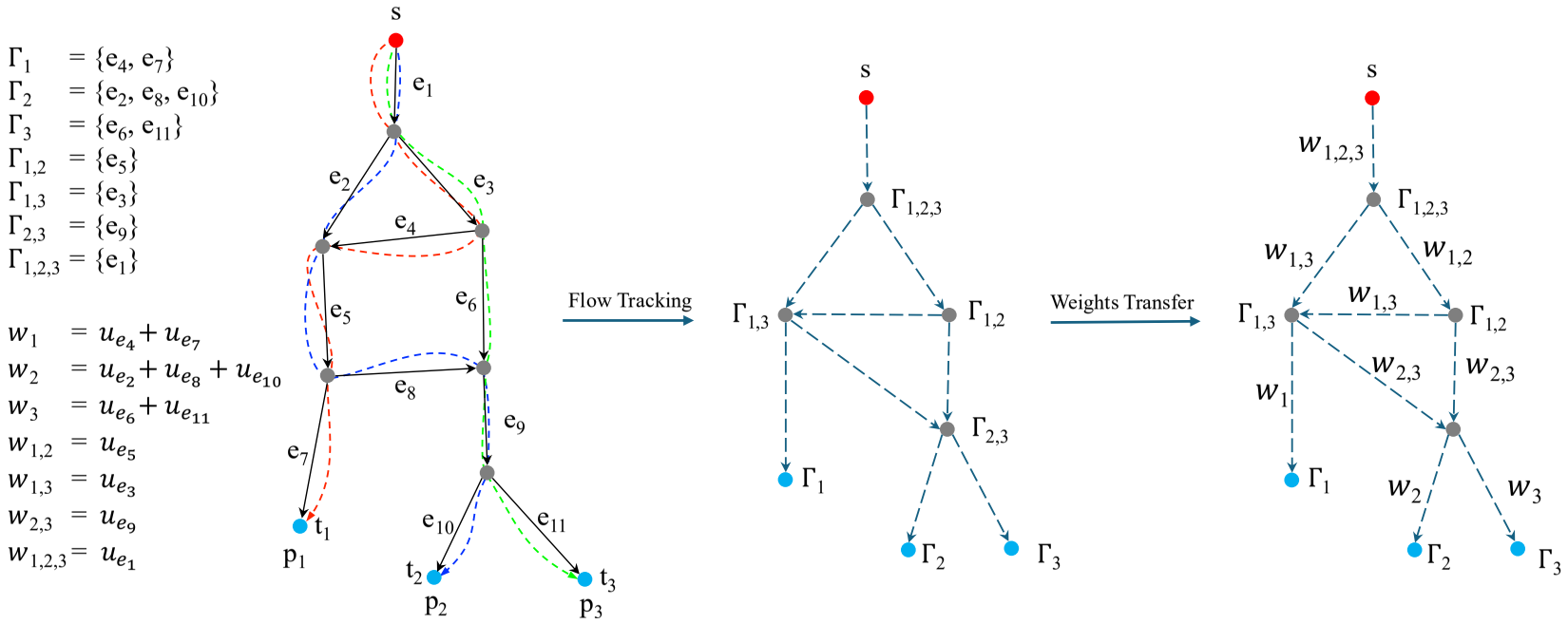


Figure 3.2: An example of topology inference using Flow Tracking (FT).

Chapter 4

Task Allocation for Model Training

4.1 Task Allocation Problem

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 [33] and All-Reduce [34]. 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.*

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.

4.2 Algorithms to Solve TAP

Task scheduling problems of the same kind have been proven to be NP-hard [27], 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 (4.1), 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**.

$$\min \quad \sum_{i,i'=1}^{m} \sum_{j,j'=1}^{n} X_{ij} \cdot X_{i'j'} \cdot M_{ii'} \cdot D_{jj'} \tag{4.1}$$

$$\text{s.t.} \quad X_{ij} \in \{0,1\} \quad \forall 1 \leq i \leq m, 1 \leq j \leq n, \tag{4.2}$$

$$m \leq n \cdot K, \tag{4.3}$$

$$\sum_{i=1}^{m} X_{ij} \leq K \quad \forall 1 \leq j \leq n. \tag{4.4}$$

$$\sum_{j=1}^{n} X_{ij} = 1 \quad \forall 1 \leq i \leq m, \tag{4.5}$$

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 (4.3) and (4.4). Constraint (4.5) means that a module can be assigned to only one VM.

Algorithm 2 Cluster Embedding (CLE)

Input: Overlay network G_o; Underlay network G
Output: An assignment of modules in the overlay to VMs in the underlay

```
Sort the edges of G_o by weight in descending order
Find clusters in G and sort the clusters by capacity in descending order
current cluster ← the cluster with the highest capacity
while There are unassigned modules in the overlay do
    Fetch the edge e_o of the highest weight in the overlay
    if current cluster has space then
        Assign the endpoints of e_o to the current cluster
        Remove the edge e_o from the overlay
    else
        the next cluster ← max{capacity/hopCount}
        current cluster ← next cluster
```

Due to the NP-hardness of **TAP**, the computation time for the MINLP problem increases significantly as the number of machine learning modules and virtual machines grows. In this case, existing optimization tools would not return a result in a reasonable time. To address this, 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. Our idea is to 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. The pseudocode is shown in Algorithm 2. In Line 10, *capacity* denotes the sorted cluster capacity and *hopCount* means the smallest hop count of the next cluster under consideration to all the clusters that have been processed, i.e., those have been assigned modules and have no space.

Complexity Analysis: The time complexity of Algorithm 2 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.

Algorithm 3 Average-Based Matching (ABM)

Input: Complete weighted graph G_1, connected unweighted graph G_2
Output: Total data transmission between modules and matching result

1: **Initialization:**
2: Calculate the average weight for each node in G_1, denoted as $weight\_avg$
3: Calculate the average (minimal) hop count for each node in G_2, denoted as hop_avg
4: **Sorting:**
5: Sort nodes of G_1 in descending order of $weight\_avg$
6: Sort nodes of G_2 in ascending order of hop_avg
7: **Matching:**
8: Initialize an empty list *matching*
9: **for** each node i in G_1 **do**
10: Select the unmatched node i in G_1 with the highest $weight\_avg$
11: Select the unmatched node j in G_2 with the lowest hop_avg
12: Match node i to node j
13: Add the pair (i, j) to *matching*
14: Update the data transmission calculation
15: **Handling Special Cases:**
16: **if** There are multiple nodes in G_2 with the same minimum hop count **then**
17: Calculate the impact on data transmission for different choices
18: **for** each pair of nodes (i, j) and (k, l) in *matching* **do**
19: Calculate the impact $I_{i,j}$ for each match:
20: $I_{i,j} = x_{i,j} \times k_{i,j}$ where $x_{i,j}$ is the weight between nodes i and j in G_1, and $k_{i,j}$ is the (minimal) hop count between nodes i and j in G_2
21: Choose the matching with the minimal total impact $\sum I_{i,j}$
22: **Output:**
23: Calculate and output the total data transmission
24: Output the matching result *matching*

We also propose another greedy algorithm called *Average-Based Matching*. The core idea of this algorithm is to 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. The pseudocode is shown in Algorithm 3.

Complexity Analysis: The time complexity of Algorithm 3 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.

Chapter 5

Performance Evaluation

5.1 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. 5.1a depicts the 4-ary Fat-tree network topology, which consists of three layers of switches designed to serve large-scale data centers. 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, where core layer switches are responsible for processing and distributing data from the aggregation layer. Fig. 5.1b shows an example of 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.

In the aforementioned network topologies, 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. It is worth noting that at the application-layer overlay networks, non-simple path is often necessary, e.g., a neural network module may be called repeatedly by different machine training tasks; 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.

5.2 Performance in Topology Inference

Baselines: We selected two algorithms as the baselines. One is Clique Embedding (CE) [23], which is also the state-of-the-art topology inference algorithm for general graphs. The other is Rooted Neighbor-Joining (RNJ) [32], which achieves highly accurate results if the ground truth topology is a tree.

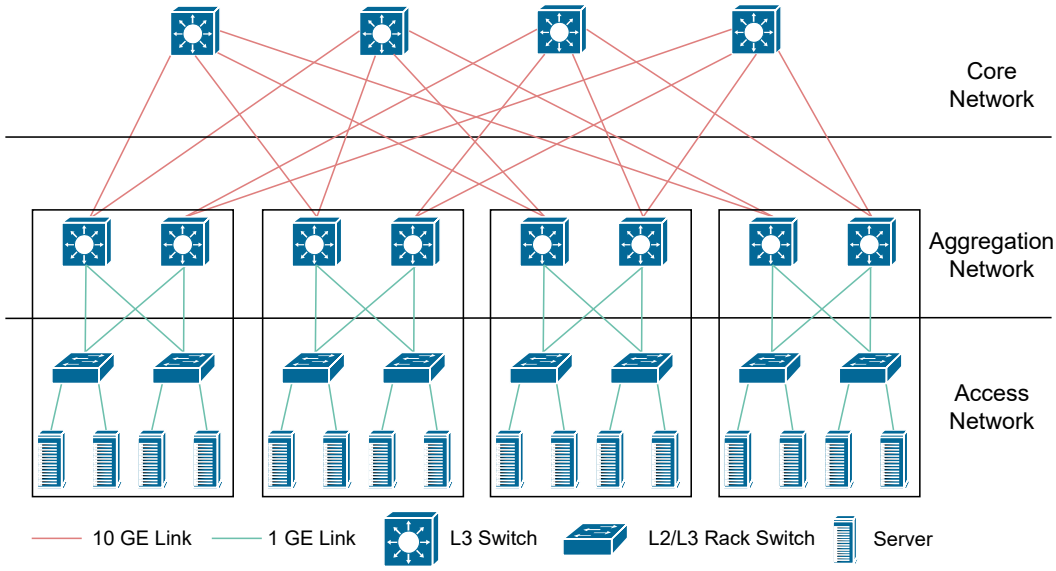
Performance Metrics: We evaluate the performance of topology inference algorithms with the following metrics:

- *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.

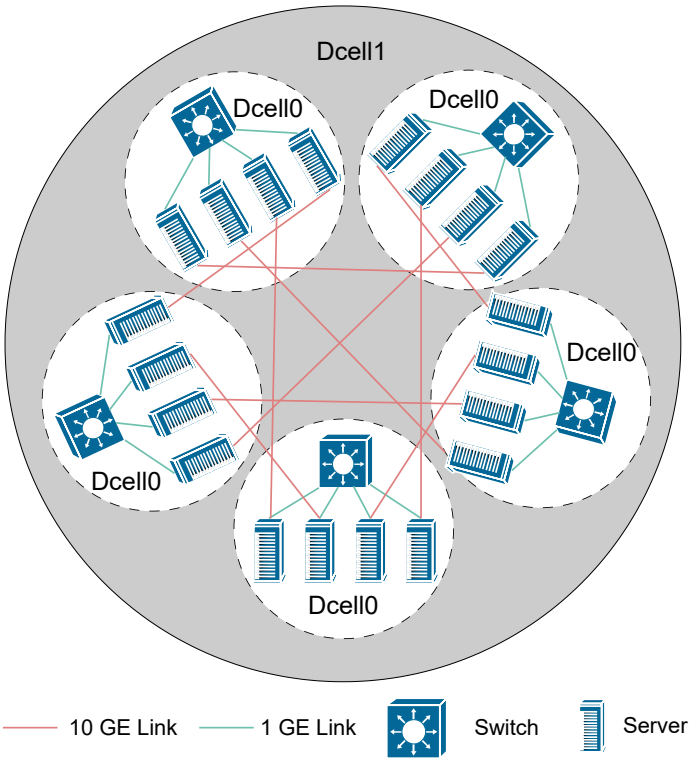
- *Degree error:* We use two types of degree errors: the first is global degree error (GDE), which is the discrepancy between the average node degree in the inferred topology and that in ground truth topology. The second is path degree error (PDE), which measures the difference between the average node degree along the measured paths in the inferred topology and that in the corresponding paths in the ground truth topology. GDE is proposed because nodes with high degrees can indicate hub nodes; PDE is proposed because the degrees of nodes traversed by a path can reflect the congestion level of that path.

- *Normalized graph edit distance (NGED):* Graph edit distance (GED) quantifies the minimum number of edit operations required to transform one graph into another. These operations typically include the insertion, deletion, and substitution of nodes and edges. To account for the impact of network size on GED results, we normalize the GED between the inferred and ground truth topologies by dividing it by the size of the ground truth network, which is determined by the total number of edges and nodes.

- H_2 *index:* In data center networks, we care about the VM clusters, i.e., those VMs connecting to the same intermediate router. As such, we define an H_2 index to measure the accuracy of topology inference. Specifically, the H_2 index



(a) 4-ary Fat-tree Topology



(b) Dcell 1-4 Topology

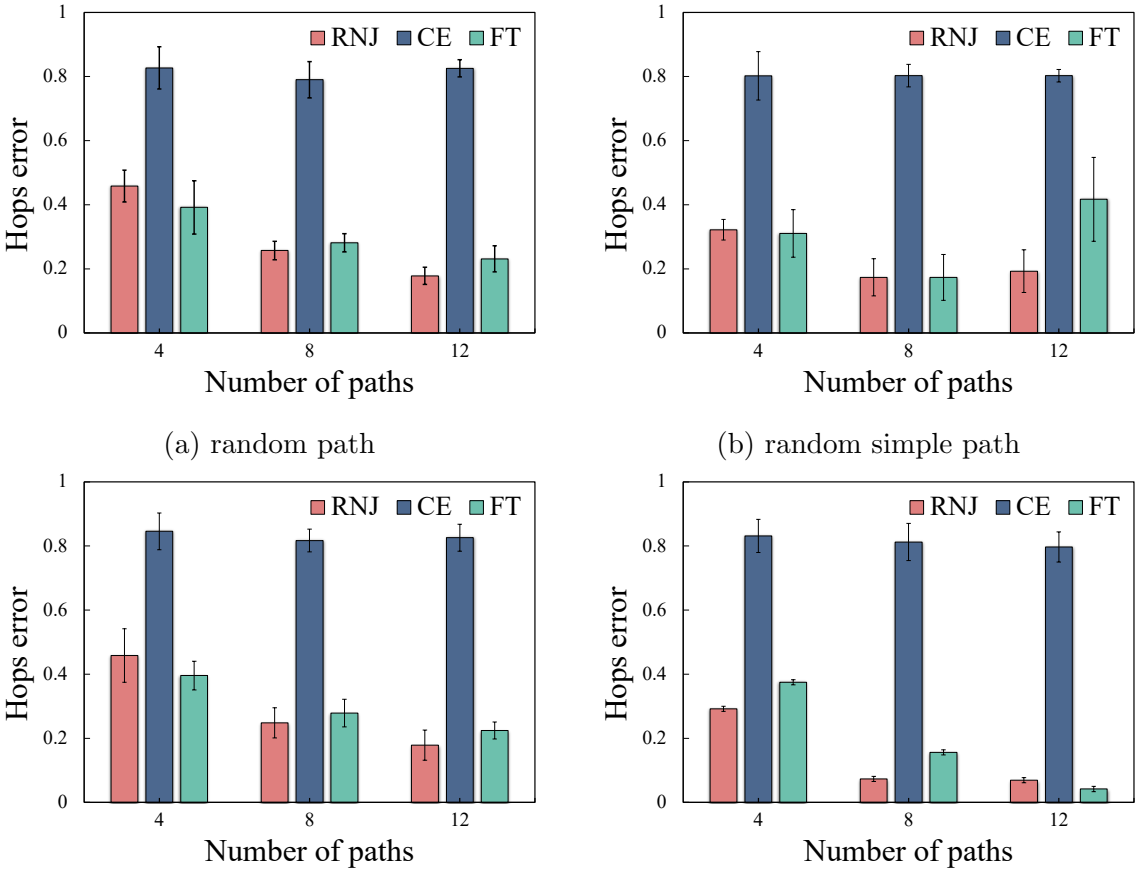
Figure 5.1: Ground truth topologies.

refers to the proportion of servers within 2 hops in the ground truth topology that also appear within 2 hops in the reconstructed topology.

Hops error results: Figs. 5.2 and 5.3 illustrate the performance of various inference algorithms regarding hops error across different number 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 method yields superior inference accuracy, which is further enhanced as the quantity of measurement paths increases. This enhancement can be attributed to the acquisition of additional categorical information through the shortest path routing method and extensive measurement data, facilitating more precise topology reconstruction. It is worth noting that the RNJ algorithm outperforms the FT algorithm in some experiments. When the Fat-tree serves as the underlying topology, the structure of the ground truth topology is more similar to a tree, causing the paths formed in this topology to also resemble a tree structure. As a result, the tree-based RNJ algorithm achieves better performance. Similarly, in cases where Dcell is the ground truth topology, the RNJ algorithm performs well when there are fewer paths, as the routing paths formed are closer to a tree structure. However, when the routing topology begins to resemble a more general graph, our proposed FT algorithm yields lower hops errors.

Degree error results: Figs. 5.4 and 5.5 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. This is particularly evident in scenarios utilizing random paths and simple paths, where FT significantly outperforms the RNJ algorithm. 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. 5.4 and 5.5 to make the figures visually clear.

NGED results: It is significant that our proposed algorithm achieved very small errors using Dijkstra's shortest path, especially as the number of measurement paths increased, with errors progressively approaching 0. This prompts the question: Have we completely reconstructed the ground truth topology? To answer this, we show the



(a) random path

(b) random simple path

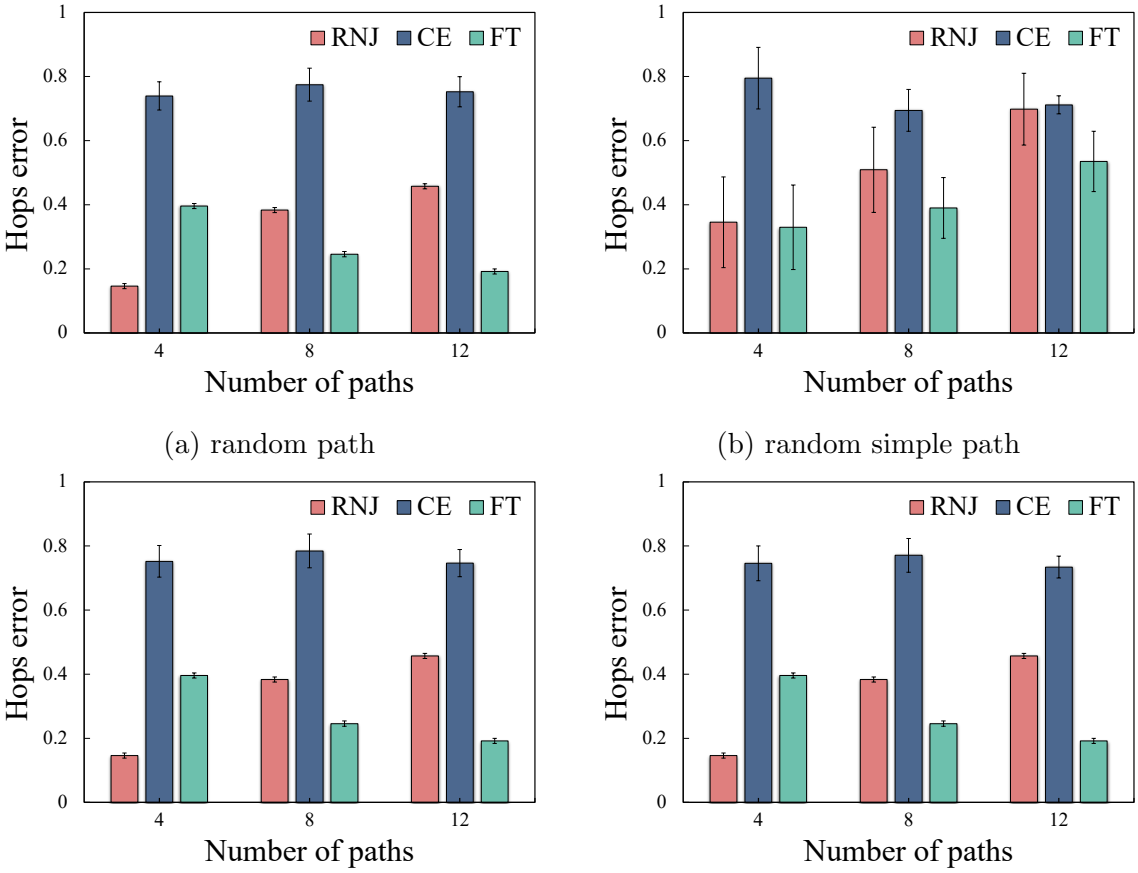
(c) random shortest path

(d) Dijkstra's shortest path

Figure 5.2: Hops error in Fat-tree topology as ground truth.

NGED results in Fig. 5.6. As the number of measurement paths increased, NGED gradually decreased and finally approached zero. This indicates 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, and in this case, FT can fully utilize the category information for topology reconstruction.

H_2 index results: As shown in Fig. 5.7, both 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 achieving 100% accuracy. This trend suggests that transitioning from complex to simpler measurement paths enhances clustering accu-



(a) random path

(b) random simple path

(c) random shortest path

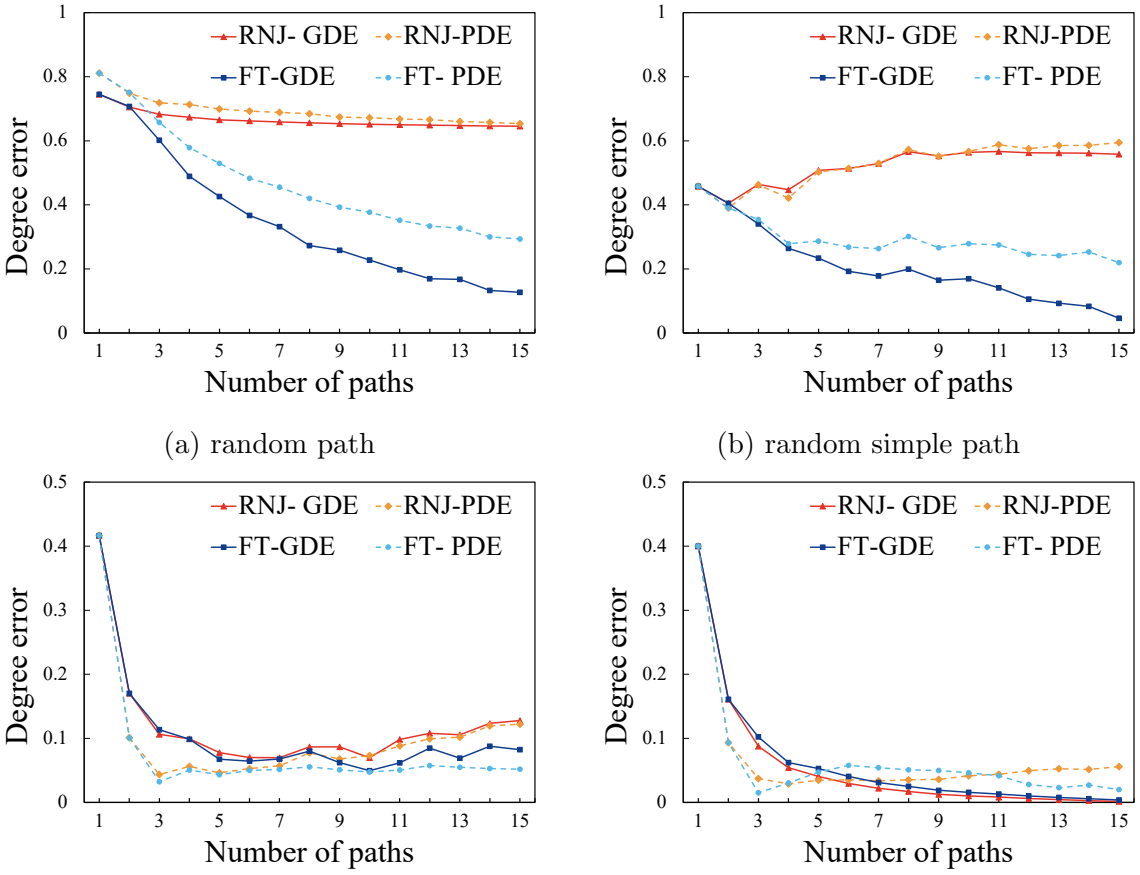
(d) Dijkstra's shortest path

Figure 5.3: Hops error in Dcell topology as ground truth.

racy, achieving the best performance under Dijkstra's shortest path routing strategy. In contrast, CE exhibits varying performance and has the lowest clustering accuracy in several cases.

5.3 FT Helps Model Training

We evaluate how our topology inference results can accelerate model training by distributing tasks across multiple computing nodes (workers). Common communication schemes for distributed model training include Parameter Servers and All-Reduce. In the parameter server scheme, workers send gradients to the parameter server for parameter updates. In the All-Reduce scheme, all nodes directly exchange and synchronize gradients among them, avoiding the single point of failure issue. In this section, we aim to determine whether deploying modules using inferred topology in-



(a) random path

(b) random simple path

(c) random shortest path

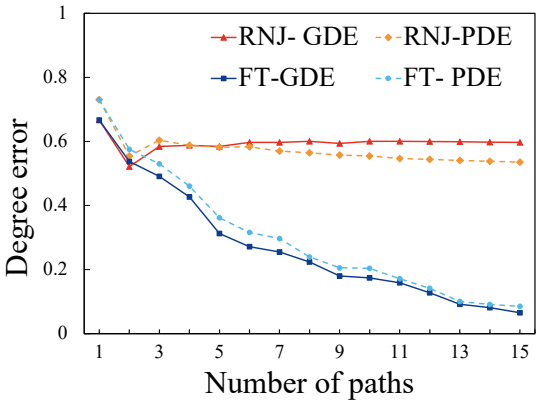
(d) Dijkstra's shortest path

Figure 5.4: Degree error in Fat-tree topology as ground truth.

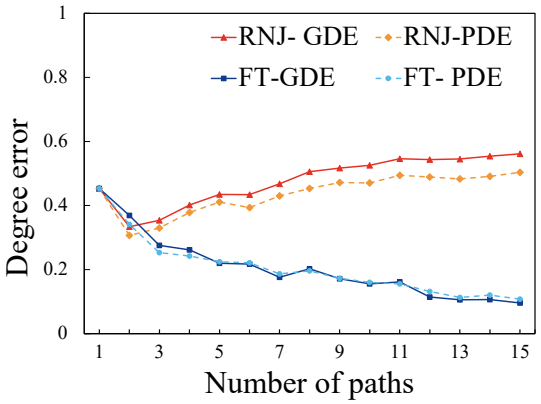
formation enhances communication efficiency compared to using random allocation.

We simulated a distributed machine learning model consisting of 8 computational modules. The communication load between these modules was randomly assigned values ranging from 0 to 10, indicating the intensity of communication between the modules. We set the K value in Equation (4.3) and (4.4) to 1, allowing each virtual machine to host only one machine learning module. Note that using different K values within the constraint (4.3) and (4.4) will not affect our conclusion. We will utilize the MINLP, *Cluster Embedding* and *Average-Based Matching* methods, along with a random approach, to distribute these 8 modules across a varying number of virtual machines within the Fat-tree and Dcell topologies. This setup will enable us to assess the influence of topology awareness on machine learning communication cost, defined in (4.1), and the effectiveness of the two task allocation algorithms.

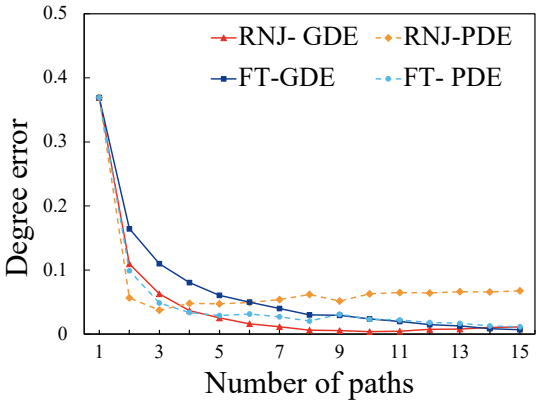
Experimental results in Fig. 5.8 reveal that, compared to random allocation,



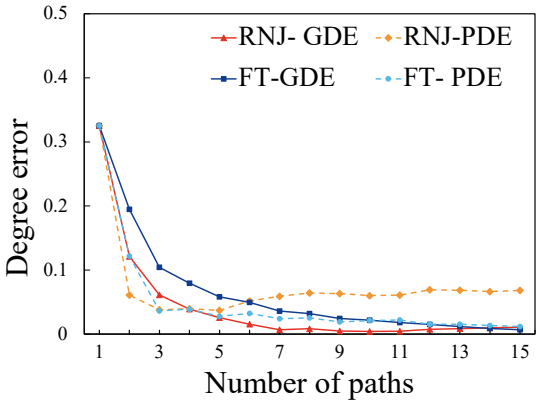
(a) random path



(b) random simple path

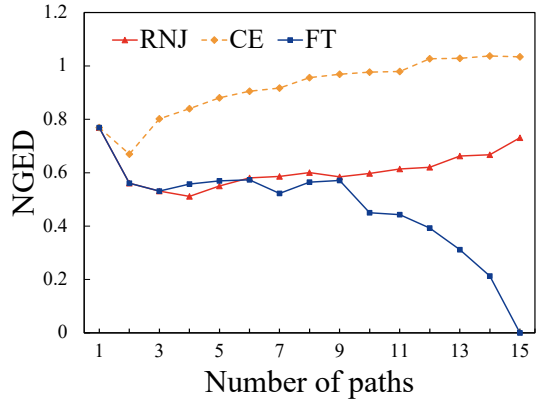


(c) random shortest path

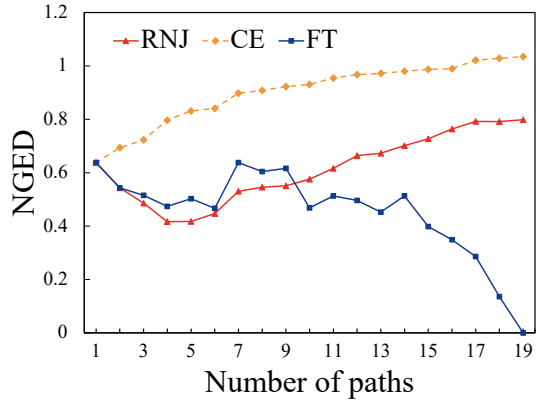


(d) Dijkstra's shortest path

Figure 5.5: Degree error in Dcell topology as ground truth.

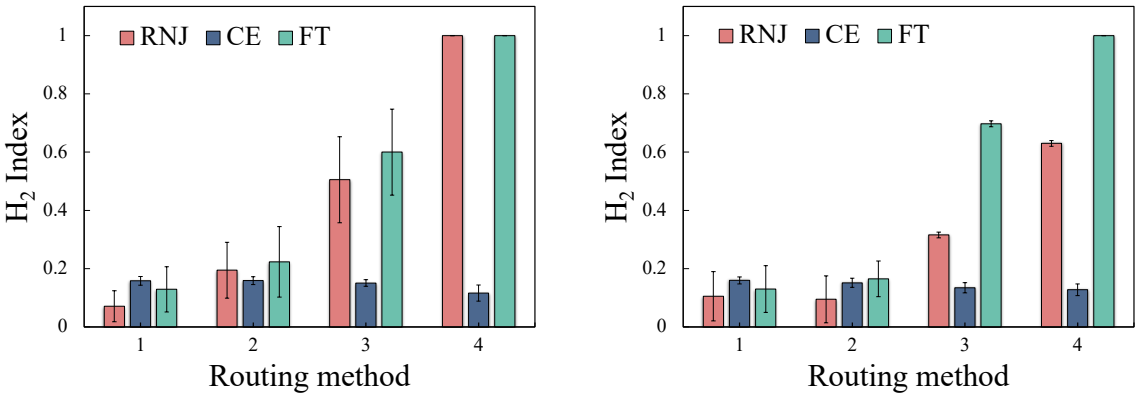


(a) Fat-tree



(b) Dcell

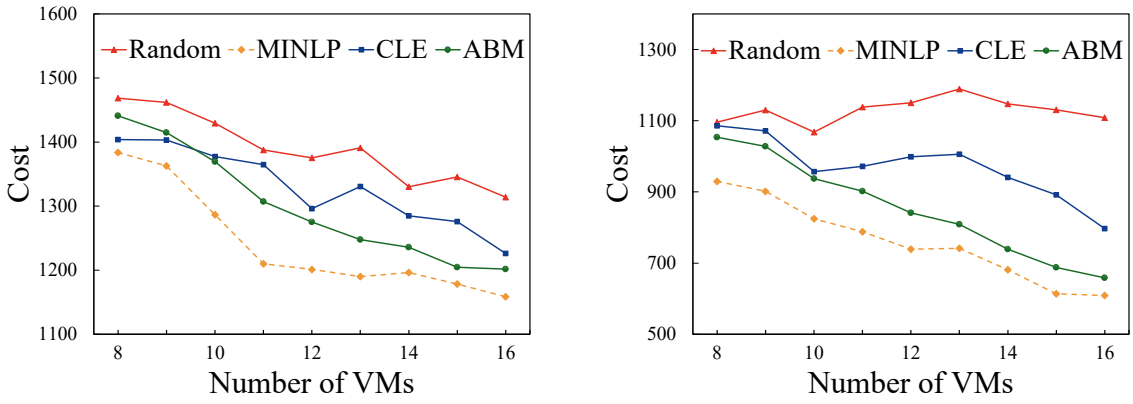
Figure 5.6: NGED in two different ground truth topologies.



(a) Fat-tree (b) Dcell

Figure 5.7: H_2 index in two different ground truth topologies.

MINLP allocation results in the lowest communication costs within the two ground truth topologies. This is attributed to Gurobi deriving precise solutions in small-scale models. Both *Cluster Embedding* and *Average-Based Matching* also reduce communication costs, with the latter outperforming the former. This difference arises because CLE ignores the communication costs within clusters, whereas ABM accounts for them by utilizing an averaging method. While neither algorithm reduces communication costs as extensively as MINLP, they significantly excle it in terms of computation time. Running over a commodity computer (macOS 14.5, Intel Core i5, 16 GB RAM), MINLP requires around 10^4 times the computation time of *Cluster Embedding* and *Average-Based Matching*, with this ratio increasing as the task scale grows. This demonstrates that the proposed two algorithms are more feasible for large-scale machine learning task allocation. Additionally, the benefits of utilizing topology awareness for task allocation become increasingly apparent as the number of VMs grows. With more VMs, there are more options for task allocation. Thus, random allocation has a much smaller chance of finding an effective allocation scheme.



(a) Fat-tree (b) Dcell

Figure 5.8: Communication cost in two types of topology.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The underlying network topology has a crucial impact on the efficiency of distributed machine learning model training. However, due to the information isolation between end users and cloud service providers, this topology is often invisible to end users, limiting the optimal use of cloud computing resources purchased or leased by users.

In this thesis, we address this significant challenge from two perspectives:

First, we tackle a long-standing unresolved issue: topology inference for general graphs. Through a comprehensive survey and analysis of related research, we propose that inferring the topology of general graphs solely based on external measurements is challenging because different topologies can yield identical measurement results. This phenomenon has been demonstrated in various studies, necessitating the development of topology inference criteria tailored to specific network structures. Inspired by category weight information obtained from end-to-end measurements, we introduce a general graph-based algorithm called *Flow Tracking (FT)*. This algorithm reconstructs the entire graph's category information based on category information from each measurement path. Additionally, by modifying the original category weight definition, we add an amendment, extending the definition's applicability to non-simple paths. We validated our algorithm through extensive simulations on two ground truth topologies, Dcell and Fattree. The experimental results demonstrate that our *FT* algorithm achieves superior performance across four key metrics. Notably, when Dijkstra's shortest path algorithm is used for routing, our proposed algorithm can fully reconstruct the ground truth topology with 100% accuracy.

Second, we address the problem of optimizing the allocation of machine learning modules when the topology information between virtual machines is fully or partially known. By analyzing the characteristics of distributed machine learning, we generalize this as a task allocation problem and further formulate it as a Mixed-Integer Linear Programming (MILP) problem. The task allocation problem, aimed at minimizing costs across multiple objectives, is proven to be NP-hard. While small-scale distributed models can be solved precisely using existing tools like Gurobi, the computation time significantly increases for larger models. To address this, we propose two heuristic algorithms: the *Cluster Embedding* algorithm, based on the clustering characteristics of data center topologies, and the *Average-based Matching* algorithm, based on the concept of average matching. We compared the performance of these algorithms with the allocation solutions obtained using Gurobi and random allocation on a distributed machine learning model with eight modules, using two ground truth topologies, Dcell and Fattree. The experimental results indicate that both proposed algorithms reduce communication costs to varying degrees. Although the reduction is less significant than that achieved by Gurobi, the computation time of both algorithms is significantly lower, demonstrating their potential applicability to larger-scale distributed machine learning models.

6.2 Future Work

Topology inference represents a highly challenging and complex research area, particularly when it involves the inference of general graphs based solely on end-to-end measurements. The inherent difficulties stem from the fact that multiple topological structures can yield identical measurement results, making it hard to accurately infer the underlying network structure. Despite these challenges, this field remains critically important, as accurate topology inference plays a fundamental role in optimizing network performance, enhancing security, and improving resource allocation within distributed systems.

The integration of topology inference with distributed machine learning introduces a particularly promising research direction. As distributed machine learning continues to gain popularity due to its ability to leverage large-scale data and computational resources across multiple nodes, the underlying network topology becomes a key factor influencing the efficiency and effectiveness of model training and deployment. By making distributed machine learning models topology-aware, it is possible to opti-

mize communication costs, reduce latency, and improve overall system performance, leading to more scalable and robust learning frameworks.

Future research in this domain holds significant potential and could explore several key areas. One such area is the development of more sophisticated and accurate metrics for topology inference. Improving these metrics could enhance the precision of inferred topologies, especially in complex and dynamic network environments. Additionally, extending the applicability of topology inference methods to a broader range of ground-truth topologies, beyond traditional data center architectures like Dcell and Fattree, could lead to more versatile and generalized solutions.

Moreover, there is a growing opportunity to develop distributed machine learning models that are inherently topology-aware. Such models could adapt to the underlying network structure, optimizing data flow and computation across nodes based on real-time topology information. This approach could minimize bottlenecks and enhance the scalability of machine learning applications in diverse network settings, including edge computing, Internet of Things (IoT) networks, and multi-cloud environments.

Bibliography

[1] Bernard Fortz, Jennifer Rexford, and Mikkel Thorup. Traffic engineering with traditional ip routing protocols. *IEEE communications Magazine*, 40(10):118–124, 2002.

[2] Neil Spring, Ratul Mahajan, and David Wetherall. Measuring isp topologies with rocketfuel. *ACM SIGCOMM Computer Communication Review*, 32(4):133–145, 2002.

[3] Bin Yao, Ramesh Viswanathan, Fangzhe Chang, and Daniel Waddington. Topology inference in the presence of anonymous routers. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*, volume 1, pages 353–363. IEEE, 2003.

[4] Yehuda Vardi. Network tomography: Estimating source-destination traffic intensities from link data. *Journal of the American statistical association*, 91(433):365–377, 1996.

[5] AHeroIII Coates, Alfred O Hero III, Robert Nowak, and Bin Yu. Internet tomography. *IEEE Signal processing magazine*, 19(3):47–65, 2002.

[6] Mark Coates, Rui Castro, Robert Nowak, Manik Gadhiok, Ryan King, and Yolanda Tsang. Maximum likelihood network topology identification from edge-based unicast measurements. *ACM SIGMETRICS Performance Evaluation Review*, 30(1):11–20, 2002.

[7] TS Eugene Ng and Hui Zhang. Predicting internet network distance with coordinates-based approaches. In *Proceedings. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 1, pages 170–179. IEEE, 2002.

[8] David L Donoho. Compressed sensing. *IEEE Transactions on information theory*, 52(4):1289–1306, 2006.

[9] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S Rellermeyer. A survey on distributed machine learning. *Acm computing surveys (csur)*, 53(2):1–33, 2020.

[10] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Marc'aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, et al. Large scale distributed deep networks. *Advances in neural information processing systems*, 25, 2012.

[11] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on operating systems design and implementation (OSDI 14)*, pages 583–598, 2014.

[12] Giovanni Neglia, Gianmarco Calbi, Don Towsley, and Gayane Vardoyan. The role of network topology for distributed machine learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2350–2358. IEEE, 2019.

[13] Yufu Wang, Qiaozhi Bao, Jiufan Wang, Guangze Su, and Xiaonan Xu. Cloud computing for large-scale resource computation and storage in machine learning. *Journal of Theory and Practice of Engineering Science*, 4(03):163–171, 2024.

[14] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[15] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280, 2010.

[16] Raj Jain and Subharthi Paul. Network virtualization and software defined networking for cloud computing: a survey. *IEEE Communications Magazine*, 51(11):24–31, 2013.

[17] Sardar Khaliq Uzaman, Junaid Shuja, Tahir Maqsood, Faisal Rehman, Saad Mustafa, et al. A systems overview of commercial data centers: initial energy

and cost analysis. *International Journal of Information Technology and Web Engineering (IJITWE)*, 14(1):42–65, 2019.

[18] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. *ACM SIGCOMM computer communication review*, 38(4):63–74, 2008.

[19] R Caceres, NG Duffield, J Horowitz, F Lo 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)*, volume 3, pages 3065–3070. IEEE, 1999.

[20] Sylvia Ratnasamy and Steven 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)*, volume 1, pages 353–360. IEEE, 1999.

[21] Nick G Duffield, Joseph Horowitz, and F Lo 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)*, volume 3, pages 1636–1645. IEEE, 2001.

[22] Nick G Duffield, Joseph Horowitz, F Lo Presti, and Don Towsley. Multicast topology inference from measured end-to-end loss. *IEEE Transactions on Information Theory*, 48(1):26–45, 2002.

[23] Yilei Lin, Ting He, Shiqiang Wang, Kevin Chan, and Stephen Pasteris. Looking glass of nfv: Inferring the structure and state of nfv network from external observations. *IEEE/ACM Transactions on Networking*, 28(4):1477–1490, 2020.

[24] Yilei Lin, Ting He, Shiqiang Wang, Kevin Chan, and Stephen Pasteris. Multicast-based weight inference in general network topologies. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2019.

[25] Yudi Huang and Ting He. Overlay routing over an uncooperative underlay. In *Proceedings of the Twenty-fourth International Symposium on Theory, Al-*

gorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing, pages 151–160, 2023.

[26] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann De Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888–1906, 2013.

[27] Jeffrey D. Ullman. Np-complete scheduling problems. *Journal of Computer and System sciences*, 10(3):384–393, 1975.

[28] Rui M Castro, Mark J Coates, and Robert D Nowak. Likelihood based hierarchical clustering. *IEEE Transactions on signal processing*, 52(8):2308–2321, 2004.

[29] Michael A Kozuch, Michael P Ryan, Richard Gass, Steven W Schlosser, David O'Hallaron, James Cipar, Elie Krevat, Julio López, Michael Stroucken, and Gregory R Ganger. Tashi: location-aware cluster management. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, pages 43–48, 2009.

[30] Dominic Battré, Natalia Frejnik, Siddhant Goel, Odej Kao, and Daniel Warneke. Evaluation of network topology inference in opaque compute clouds through end-to-end measurements. In *2011 IEEE 4th International Conference on Cloud Computing*, pages 17–24. IEEE, 2011.

[31] Abdallah Saad and Ahmed El-Mahdy. Network topology identification for cloud instances. In *2013 International Conference on Cloud and Green Computing*, pages 92–98. IEEE, 2013.

[32] Jian Ni, Haiyong Xie, Sekhar Tatikonda, and Yang Richard Yang. Efficient and dynamic routing topology inference from end-to-end measurements. *IEEE/ACM transactions on networking*, 18(1):123–135, 2009.

[33] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. *Advances in Neural Information Processing Systems*, 27, 2014.

[34] Yixin Bao, Yanghua Peng, Yangrui Chen, and Chuan Wu. Preemptive all-reduce scheduling for expediting distributed dnn training. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 626–635. IEEE, 2020.