TFD: Fault Detection and Localization for Efficient Large-Scale LLM Training

Tianshi Wang[1,2], Yiran Zhang[1,2*], Ao Zhou[1,2], and Shangguang Wang[1,2]

[1] Beijing University of Posts and Telecommunications, Beijing 100876, China
[2] Beiyou Shenzhen Institute, Shenzhen 518000, China
{tswang,yiranzhang,aozhou,sgwang}@bupt.edu.cn

Abstract. Large language model training on massive GPU clusters has become increasingly prevalent, yet communication inefficiencies and performance issues pose significant challenges to training reliability and efficiency. Through comprehensive measurements of GPU and RDMA NIC metrics, we provide detailed analysis of network performance issues and their root causes. Our main findings and contributions are threefold: (1) We select GPU algorithm bandwidth as the primary metric to monitor LLM training cluster performance and reveal the GPU algorithm bandwidth jitter phenomenon. (2) We design and implement TFD, a practical fault detection tool that integrates fine-grained GPU and RDMA NIC metrics for real-time monitoring and precise fault localization. (3) In a real-world LLM training cluster, TFD achieves 92% accuracy in fault detection. Our work provides valuable insights for optimizing communication efficiency in large-scale LLM training clusters and offers a practical solution for automated fault detection.

Keywords: Large Language Model; Distributed Training; Fault Detection; Fault Localization; GPU Cluster; RDMA

1 Introduction

In recent years, the advent of large language model (LLM) has revolutionized the field of natural language processing (NLP), bringing unprecedented capabilities to tasks ranging from text generation to semantic understanding. Despite their impressive applications, training and deploying LLMs present significant challenges. The training of LLMs is computationally intensive, typically requiring vast computational resources and complex distributed systems to manage enormous datasets and model parameters [1]. To achieve more powerful model capabilities, industries and institutions have resorted to training large-scale models with hundreds of billions or even trillions of parameters on infrastructures comprising tens of thousands of GPUs [2].

As a result, LLM training on massive GPU clusters necessitates the synchronization of model parameters across GPUs. As model size increases, inter-GPU

* Corresponding Author

communication overheads rise significantly. Optimizing the coordination between computation and communication becomes a central challenge in improving training efficiency. Current distributed training strategies, including data parallelism (DP), tensor parallelism (TP), and pipeline parallelism (PP), are combined to synchronize parameters and aggregate gradients, thereby maximizing computational and communication efficiency. These strategies rely on collective communication libraries (e.g., all-reduce, all-gather) to efficiently synchronize parameters and accelerate model convergence [6, 7].

Recent studies have explored monitoring performance issues in LLM training clusters. For instance, ByteDance's Megascale [3] provides fault diagnosis and parallelism communication measurement visualization. R-Pingmesh [4] monitors RoCE network performance by measuring network RTT and host processing latency. Hostping [5] focuses on diagnosing in-host network bottlenecks through loopback tests. These systems play crucial roles in the normal operations of LLM training. However, existing systems primarily focus on fault diagnosis for specific components within LLM clusters, yet lack comprehensive fine-grained observation of multi-component metrics (e.g., GPU algorithm bandwidth, end-to-end latency) and the ability to analyze correlations between these metrics. This gap restricts our capacity to holistically understand performance interdependencies and promptly identify and mitigate bottlenecks in large-scale GPU clusters.

In this paper, we first conduct a comprehensive measurement study of GPU and RDMA NIC performance in one large-scale LLM training cluster. We measure and analyze the rail-based network topology, traffic patterns in LLM training iterations, and key performance metrics. Through extensive measurements, we identify the GPU algorithm bandwidth as a crucial indicator of training efficiency and reveal the GPU algorithm bandwidth jitter phenomenon. Based on the analysis of GPU algorithm bandwidth, we reveal that LLM training clusters suffer from various performance-impacting faults, motivating the need for an effective fault detection and localization system.

Further, we design an automated fault detection tool tailored for LLM training clusters called TFD (Training Fault Detection). TFD employs GPU algorithm bandwidth, RDMA NIC metrics, and system-level diagnostics to continuously monitor cluster training performance in real time. Upon fault occurrence, TFD promptly detects anomalies, leverages fine-grained metric information to precisely localize the fault's origin, and analyzes underlying causes, thereby enabling engineers to efficiently resolve issues. The key advantage of TFD is that it can be seamlessly integrated into existing training workflows without hardware modifications, ensuring practical applicability in large-scale environments.

Finally, we evaluate the fault detection tool in a real-world LLM training cluster, demonstrating its efficacy. Our extensive evaluation shows that TFD achieves a fault detection accuracy of 92% and a fault localization accuracy of 79%, demonstrating its effectiveness in identifying and diagnosing performance issues in production environments. These results validate TFD's capability in enhancing the reliability and efficiency of LLM training operations.

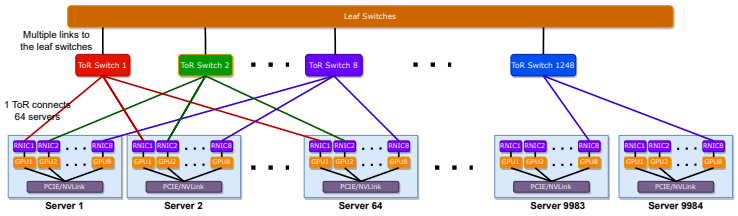


Fig. 1: Network topology architecture of the LLM training cluster with rail-based design.

2 Data Collection and Measurement Results

In this section, we first describe the network characteristics of one large-scale GPU cluster where we conduct measurement (Section 2.1). Next, we analyze the traffic patterns observed during LLM training iterations (Section 2.2). Then, we present the measurement metrics, including RDMA NIC bandwidth, GPU algorithm bandwidth, latency, and congestion signals (Section 2.3). Finally, we investigate the phenomenon of GPU algorithm bandwidth jitter and its correlation with network performance in cluster failure analysis (Section 2.4).

2.1 Network Topology

The datacenter network employs a rail-based topology, typically comprising 2 tiers of switches reminiscent of a CLOS topology. As shown in Figure 1, each Top-of-Rack (ToR) switch is equipped with 64 uplink ports and 64 downlink ports, connecting approximately 10,000 GPUs. For each tier of switches, the bandwidth ratio between downlinks and uplinks is maintained at 1:1.

GPUs are partitioned into multiple groups, forming high-bandwidth domains (HB domains) with fast interconnects. Each server constitutes an HB domain, comprising eight GPUs and eight RDMA NICs. GPUs that communicate frequently are grouped into the same HB domain. Within an HB domain, GPUs communicate via NVSwitch, ensuring high-speed connectivity. Cross-domain GPU communication is facilitated through RDMA NICs, with each RDMA NIC in a server connecting to a different Top-of-Rack (ToR) switch. The link capacity for RDMA NICs is 100Gbps.

This design ensures that each GPU can communicate with any other GPU within a limited number of hops, either within the same server or across servers, thus maintaining efficient and scalable communication throughout the cluster.

2.2 Traffic Pattern

In each server, each GPU communicates across HB domains using a single RDMA NIC for sending and receiving data, with each RDMA NIC exclusively handling the communication data of a single GPU. As illustrated in Figure 2, during each iteration, most of the time is spent overlapping computation and communication, including DP, TP, and PP. The higher the overlap between computation and communication within each iteration, the greater the utilization of GPUs, leading to more efficient training [8].

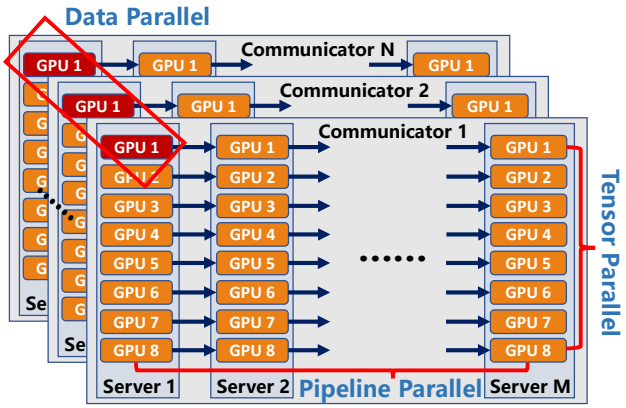
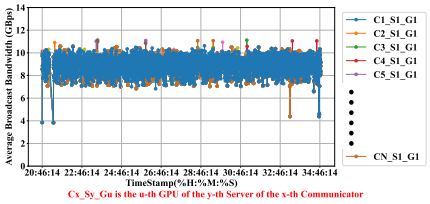
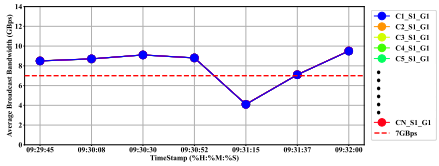


Fig. 2: Parallelism strategies: data parallel, pipeline parallel, and tensor parallel.



(a) GPU algorithm bandwidth over 14 hours



(b) GPU algorithm bandwidth jitter period and its adjacent periods

Fig. 3: GPU algorithm bandwidth of a group of data parallelism GPUs

Considering the communication pattern of model training tasks, the network traffic characteristics of each iteration exhibit a similar pattern based on three parallelism strategies. DP replicates the entire model across multiple GPUs, with each GPU processing different data batches and synchronizing gradients through collective operations. TP partitions individual layers horizontally across GPUs within the same server, leveraging high-bandwidth NVSwitch for frequent synchronization. PP divides the model vertically into stages across servers, with micro-batches flowing sequentially through the pipeline. Towards the end of each iteration, DP dominates the network resources as TP exploits intra-server bandwidth and PP traffic becomes negligible after pipeline stages complete. We define a group of GPUs that have deployed a complete replica of the model as a communicator. Within a single communicator, there only exists TP and PP. Each communicator possesses an identical number of servers and GPUs, and the traffic pattern is also largely consistent. DP is conducted between different communicators, enabling scalable training while TP and PP reduce per-GPU memory requirements. This design allows us to maintain the simplicity of communication within a single communicator while achieving parallelism processing across communicators.

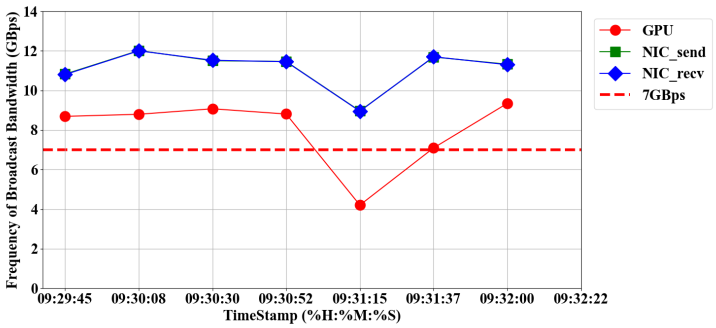


Fig. 4: Correlation between network bandwidth and GPU algorithm bandwidth jitter.

DP employs a ring-allreduce [9] architecture, where GPUs in the same position from different communicators form a ring to communicate and synchronize parameters. The specific process is as follows: each GPU completes mini-batch training, calculates gradients, and passes them to the next GPU in the ring, while simultaneously receiving gradients from the previous GPU. For a ring consisting of N GPUs, each GPU needs to receive gradients from the other $N-1$ workers before updating the model parameters.

This process typically involves multiple collective communication operations, specifically ReduceScatter and AllGather in the traffic pattern. In a ring, all GPUs simultaneously transmit data during each DP communication, and the message size is consistent across all transmissions. Consequently, the send and receive bandwidth of each GPU should exhibit similar characteristics.

2.3 Measurement Metrics and Results

In LLM training clusters, we observe periodic degradation in training efficiency that significantly impacts overall performance. To investigate these performance issues, we first identify GPU algorithm bandwidth as a key performance indicator, as it directly reflects the efficiency of collective communications during training iterations.

GPU algorithm bandwidth. The GPU algorithm bandwidth is a critical metric for evaluating the performance of collective communications during training iterations. For each GPU, the algorithm bandwidth is calculated by dividing the message size of the collective communication by the time the GPU spends participating in the communication. For instance, the GPU algorithm bandwidth during an iteration's AllGather operation is calculated by dividing the message size of the AllGather by the time the GPU spends on the AllGather. A higher algorithm bandwidth indicates more effective data handling and communication, resulting in faster training iterations. We deploy GPU Log Collector on the servers to capture the algorithm bandwidth from each training iteration.

These logs include the message size and the time the GPU spends participating in the communication for each type of collective communication, with message size recorded down to the byte and completion time measured in microseconds.

Figure 3a shows the AllGather algorithm bandwidth of all GPUs on a ring in DP. The algorithm bandwidth of AllGather is generally 8-10GBps, but occasionally it drops significantly, suggesting a low training efficiency in those iterations. We refer to this phenomenon as **GPU algorithm bandwidth jitter**, and set the jitter determination threshold at 7GBps (approximately 80% of the normal algorithm bandwidth). Our observations indicate that over 5000 GPUs experience an average of 3-5 instances of GPU algorithm bandwidth jitter per hour. These jitters could stem from multiple sources: host machine issues, network bottlenecks, or issues in the application and training framework.

To investigate whether these issues are related to network problems, we further analyze the network bandwidth through RDMA NIC measurements.

Send and receive bandwidth of RDMA NICs. The send and receive bandwidth of RDMA NICs directly indicates network bandwidth performance across servers. We deployed a software tool at the application and training framework, called RDMA NIC Collector, to collect information from RDMA NICs and compute the network bandwidth. The collected data includes the amount of data sent and received by the RDMA NICs. To manage storage overhead, this data is recorded four times per second, with bandwidth measured at sub-second granularity and data volume accurate to the byte.

We measure the network send bandwidth of over 5000 RDMA NICs, and the network bandwidth characteristics during LLM training exhibit periodic traffic patterns, with each period corresponding to one iteration. In each iteration, the overlap time between computation and communication exceeds 90%, demonstrating that the current parallelism strategy combination achieves high GPU utilization. When GPUs require minimal compute resources, specifically during the AllGather step of DP in an iteration, if there is no network congestion, the network send bandwidth generally ranges from 10-12 GBps, reaching 80%-96% of the full-speed bandwidth of the RDMA NICs. This indicates that under the current traffic pattern, the RDMA NICs achieve high network utilization.

However, we observe an occasional phenomenon where the network send bandwidth for DP drops to 4-5 GBps, which is half of the normal send bandwidth, and this condition persists for several hours until the task is restarted. This significantly reduces the training efficiency of the cluster.

2.4 Results Analysis

To diagnose the causes of GPU algorithm bandwidth jitter in LLM training clusters, we first investigate whether GPU hardware faults are the primary source. We analyze the GPU algorithm bandwidth of iterations exhibiting jitter alongside their adjacent iterations. As shown in Figure 3b, the iterations immediately preceding and following jitter events show no significant reduction in GPU algorithm bandwidth, suggesting that the jitter does not primarily stem from GPU hardware defects but rather arises during computation or communication processes. To further assess the role of GPU hardware, we examine GPU hardware

after the GPU algorithm bandwidth jitter occurred. The GPU algorithm bandwidth jitter correlates with GPU hardware faults in 40% of cases. This jitter is not exclusively indicative of hardware issues, as it also occurs in the absence of GPU defects.

To determine whether network performance contributes to these jitters, we investigate the GPU communication performance corresponding to RDMA NICs. As shown in Figure 4, when GPU algorithm bandwidth jitter occurs, the send and receive bandwidth of the corresponding RDMA NIC also significantly drops, indicating that GPU communication performance degradation contributes to the observed jitter. GPU algorithm bandwidth and RDMA NIC bandwidth alone, however, are insufficient to precisely identify the root causes of performance issues or accurately locate faults.

Addressing the challenge of diagnosing performance issues in large-scale LLM training clusters is complex due to deployment constraints and the confidentiality requirements of large language models. While some practices utilize CUDA events to monitor GPU execution time, this method only captures operation-level timing with microsecond resolution, failing to reflect the actual data transmission process during collective communications. Typically, operations staff only have access to high-level metrics such as GPU algorithm bandwidth and RDMA NIC bandwidth, without visibility into proprietary model details or fine-grained system states. Moreover, the sheer scale of LLM clusters, with thousands of GPUs generating massive data volumes, poses significant challenges. Operations staff must spend considerable time analyzing data from routine monitoring tools to pinpoint specific issues and determine fault locations. This delay in diagnosis hinders the rapid recovery of training processes, resulting in substantial economic losses. In this work, we aim to develop a practical solution for real-time monitoring and fault localization in LLM clusters, enabling efficient and timely resolution of performance issues.

To address the challenges of fault detection and precise fault localization in LLM clusters, we design and deploy a profiling system tailored for one LLM cluster environment. This system should meet the following core requirements: (1) the ability to perform real-time monitoring and fault detection without introducing significant overhead in large-scale GPU clusters; (2) the ability to provide precise fault localization and root cause analysis by correlating multi-layer performance metrics to minimize training downtime.

To fulfill these requirements, we propose a profiling framework called TFD, which works together to enable rapid and accurate fault diagnosis. In the following sections, we will elaborate on the design of this profiling framework (Section 3.1) and detail its mechanisms for real-time monitoring (Section 3.2), fault localization, and root cause analysis (Section 3.3).

3 System Overview

3.1 TFD Design

Figure 5 illustrates the design of our real-time system for monitoring and fault localization in LLM training clusters. The system integrates GPU and network

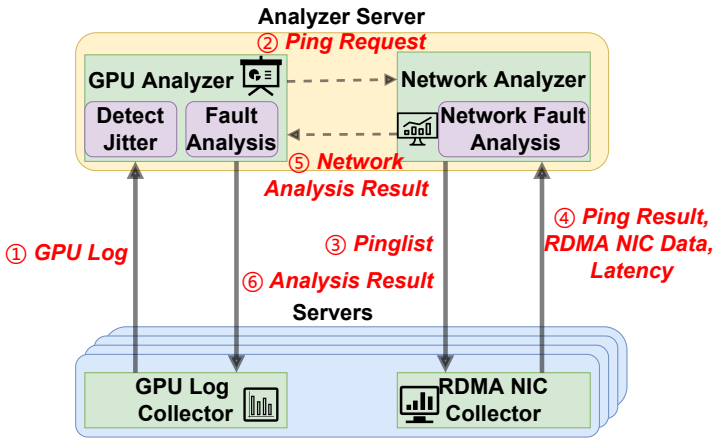


Fig. 5: TFD design.

metric collection, and cluster failure analysis. They enable rapid detection and diagnosis of performance issues in large-scale GPU clusters.

TFD deploys lightweight data collection across servers to capture critical metrics at runtime for monitoring LLM training clusters. For GPU performance, it uses GPU Log Collector to measure algorithm bandwidth during collective communications with microsecond precision. For network performance, it employs the RDMA NIC Collector to record RDMA NIC send/receive bandwidth at subsecond granularity. Additionally, to enable more comprehensive network performance analysis, RDMA NIC Collector also captures end-to-end latency between servers and network congestion signals. These collectors ensure comprehensive, low-overhead monitoring of GPU and network behavior.

TFD performs cluster failure analysis on collected metrics to identify and localize faults. The operator selects a server as the TFD master server. It analyzes GPU and network data, correlating GPU algorithm bandwidth, RDMA NIC performance, latency, and congestion signals to detect abnormalities and determine their causes.

Alternative metrics such as RDMA NIC bandwidth, end-to-end latency, and congestion signals provide partial views of system performance. GPU algorithm bandwidth offers distinct advantages: (1) it directly measures collective communication efficiency, (2) it can be collected with minimal overhead using existing NVIDIA Collective Communications Library (NCCL) [12] profiling interfaces, and (3) it exhibits clear patterns distinguishing normal operation from various fault conditions. We therefore choose GPU algorithm bandwidth as the primary metric for anomaly detection. Our measurement study reveals that GPU algorithm bandwidth exhibits strong correlations with other performance metrics: when GPU algorithm bandwidth drops below normal levels, it often indicates issues in either the host machine, network performance, or the application and training framework.

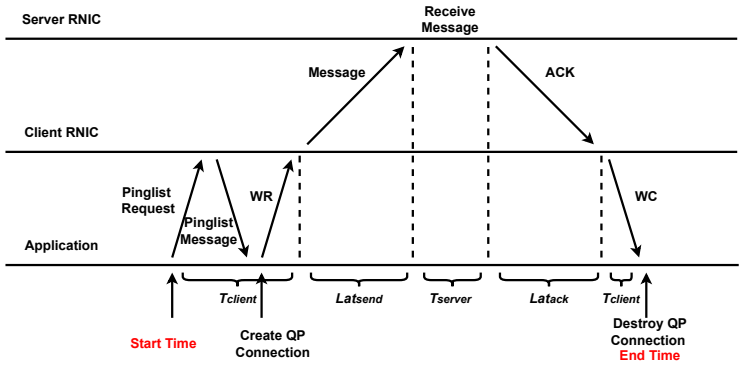


Fig. 6: Latency calculation.

3.2 GPU and RDMA Network Metric Collection

TFD efficiently collects performance metrics across the LLM training cluster while maintaining minimal overhead. It utilizes two main components: the GPU Log Collector and RDMA NIC Collector.

GPU Log Collector. This component is deployed on each server to capture detailed performance metrics of GPU collective communications. Specifically, it records the following for each collective communication operation:

- Message size, representing the data volume transferred during the operation.
- Execution time, measuring the duration of the collective communication.
- Parallelism type and collective communication operation involved.

These metrics are collected asynchronously using lightweight logging mechanisms compatible with NCCL profiling interfaces to minimize impact on training performance. The collected data is periodically aggregated monitoring packets per GPU, which are then transmitted to the GPU Analyzer on the master server.

RDMA NIC Collector. This component monitors network performance by collecting RDMA NIC statistics. It samples the send and receive data volumes of each RDMA NIC four times per second, enabling precise bandwidth calculations at sub-second granularity. To ensure comprehensive network monitoring while managing storage overhead, we implement an efficient data sampling and storage strategy.

To enable fine-grained monitoring and precise fault localization in the network, we extend RDMA NIC Collector with latency measurements. As shown in Figure 6, we implement two types of latency monitoring:

- Communication latency: This metric focuses solely on the underlying network latency, including NIC forwarding latency, switch forwarding latency, and propagation latency. We employ UDP ping detection with network card hardware timestamps to exclude host internal latencies, providing an accurate measure of network infrastructure performance.

$$Lat_{\text{Communication}} = Lat_{\text{send}} + T_{\text{server}} + Lat_{\text{ack}} \tag{1}$$

• End-to-end latency: This metric provides a comprehensive view of system latencies by measuring both network latency and host internal latency using RDMA UD ping method.

$$Lat_{\text{End-to-end}} = T_{\text{client}} + Lat_{\text{send}} + T_{\text{server}} + Lat_{\text{ack}} \tag{2}$$

By measuring both types of latency, we can accurately differentiate between latencies occurring within the host system and those in the network infrastructure, enabling more precise diagnosis of performance issues.

To further enhance network performance monitoring, RDMA NIC Collector captures congestion signals, including Congestion Notification Packet (CNP) and Priority-based Flow Control (PFC) [10] signals from the DCQCN congestion control algorithm [11] and flow control mechanisms. These congestion indicators are also analyzed to identify factors affecting GPU communication performance.

When instructed by Network Analyzer to perform ping operations, RDMA NIC Collector probes latency information and aggregates its recently collected bandwidth and congestion signal data for transmission to the Network Analyzer.

3.3 Cluster Failure Analysis

TFD utilizes two components to analyze cluster failures: the GPU Analyzer to detect and locate GPU-related faults and the Network Analyzer to diagnose network issues.

GPU Analyzer. GPU Analyzer periodically collects bandwidth information from all GPUs in the cluster and monitors their communication patterns to detect potential anomalies. Every period T, GPU Analyzer processes the collected GPU messages and calculates the actual bandwidth for each GPU (①in Figure 5). When the calculated bandwidth falls below a pre-configured *jitter threshold*, it indicates the fault.

To precisely locate the fault, GPU Analyzer employs a hierarchical detection approach. First, it examines other GPUs in the same Ring to determine if cross-server collective communication is affected. If Ring GPUs show normal behavior, GPU Analyzer determines that the fault is likely within the host machine or at the application and training framework. However, if other GPUs in the Ring show anomalies, the fault might occur in the network, prompting GPU Analyzer to send ping requests to Network Analyzer for further network diagnostics (②).

After receiving network analysis results from Network Analyzer, GPU Analyzer makes its final diagnosis. If network issues are detected, GPU Analyzer correlates network analysis results with GPU logs, identifies network issues and can distinguish hardware problems such as RDMA NIC failures and network link issues, or protocol-level problems such as Priority-based Flow Control (PFC) deadlocks. If the available information is insufficient for precise fault localization, GPU Analyzer requests additional network information by sending ping requests to Network Analyzer to gather network metrics from other hosts to assist in fault localization. If the network analysis indicates issues within the host machine, GPU Analyzer correlates this information with GPU logs to distinguish between GPU hardware failures, PCIe link issues, or NVLink failures.

If both host machine and network analyzes show normal behavior, GPU Analyzer will identify the problem as application and training framework issues or other issues. It can distinguish GPU scheduling inefficiencies, traffic isolation misconfigurations, or application-specific mechanisms like garbage collection.

Network Analyzer. Network Analyzer conducts network diagnostics by coordinating with RDMA NIC Collectors on the affected servers. Network Analyzer maintains a pre-configured pinglist provided by operators. Upon receiving ping requests from GPU Analyzer, Network Analyzer instructs all NICs on the faulty server to send ping messages to these targets (③). RDMA NIC Collectors gather latency, bandwidth metrics, and congestion signals during this process, and sends them to Network Analyzer (④).

Network Analyzer first calculates host-internal latency by comparing end-to-end latency with communication latency. If the host-internal latency exceeds its pre-configured threshold, Network Analyzer determines that the fault occurs within the host machine. If no host-internal issues are detected, Network Analyzer proceeds to examine end-to-end latency, send/receive bandwidth, and congestion signals. Network Analyzer maintains two threshold levels for each of these metrics. If any metric exceeds its high threshold (or falls below low threshold), Network Analyzer identifies that the fault occurs in the network. If no severe network issues are found but three or more metrics exceed their low thresholds (or fall below the high threshold for bandwidth), Network Analyzer identifies the faulty server as a slow node. If none of these issues are detected, Network Analyzer concludes that the fault is not a network issue.

Finally, Network Analyzer sends its comprehensive analysis results back to GPU Analyzer for final fault diagnosis (⑤).

4 Evaluation

4.1 Cluster Environment

We deploy TFD in the real training cluster as mentioned in Section 2, where LLM training jobs coexist. The cluster employs DCQCN for congestion control and PFC for flow control, with all RDMA NIC parameters set to their default values. We configure the *jitter threshold* to 7 GBps and the monitoring period T to five model iteration periods. The pinglist includes all servers in the cluster for comprehensive network monitoring.

4.2 TFD in Large-scale Cluster

Table 1 presents fault types observed in our LLM training clusters, organized into three categories: host machine issues, network issues, and application & training framework issues. For each fault type, we compare the detection capabilities of TFD against existing approaches, including Hostping [5] and R-Pingmesh [4].

We define detection accuracy as TFD's ability to correctly identify the occurrence of faults, while localization accuracy refers to TFD's ability to precisely identify the root cause and specific component responsible for the detected fault.

Table 1: Common Fault Types in LLM Training Clusters

Category	**Root Causes**	**Detectable by**
Host Machine Issues	1) PCIe link faults	1) TFD, Hostping
	2) NVLink connection failures	2) TFD
	3) GPU hardware malfunctions	3) TFD
	4) Host memory subsystem problems	4) TFD, Hostping
	5) PCIe port faults	5) TFD, Hostping
Network Issues	1) RDMA NIC faults	1) TFD, R-Pingmesh
	2) Network cable defects	2) TFD, R-Pingmesh
	3) Switch hardware failures	3) TFD
	4) Network link degradation	4) TFD, R-Pingmesh
	5) PFC pause deadlock	5) TFD
Application & Training Framework Layer Issues	1) Resource allocation misconfigurations	1) TFD
	2) Communication pattern errors	2) TFD
	3) Task scheduling inefficiencies	3) TFD
	4) Framework parameter settings	4) TFD
	5) Memory management issues	5) TFD

Localization is inherently more challenging than detection, as multiple components may exhibit similar symptoms, requiring correlation analysis across GPU, network, and application layers to distinguish between different failure modes. Using TFD, we successfully diagnosed and localized 15 distinct fault types across these categories. Specifically, host machine issues, which constitute 40.3% of all failure cases, are detected by TFD with an 85.2% accuracy. Network issues, accounting for 26.6% of total failures, achieve a 97.3% detection accuracy. Application and training framework issues, representing 33.1% of cases, are identified with a 96.4% detection accuracy. These results contribute to an overall system detection accuracy of 92.1%, demonstrating TFD's effectiveness in identifying and localizing diverse failure patterns in LLM clusters.

The fault localization capabilities of TFD show varying degrees of accuracy across different fault types. For network issues, TFD achieves 99% localization accuracy, while host machine issues are located with 81.5% accuracy. The localization accuracy for application and training framework issues reaches 60.3%. TFD achieves an overall localization accuracy of 79%. After enriching the fault signature database with more application and training framework issues, TFD achieves over 90% accuracy in identifying all known fault patterns.

During one task execution, TFD detects a significant GPU algorithm bandwidth drop at a specific node, decreasing from 8.4 GBps to 3.5 GBps. The communication latency in the LLM cluster generally ranges from 15 μs to 25 μs. TFD's Network Analyzer detects a normal communication latency of 17 μs for single-switch communication using UDP ping with hardware timestamps. However, the measured end-to-end latency reaches 45 μs. This indicates a host-internal latency of 28 μs, which exceeds the high threshold.

By comparing end-to-end latency and communication latency measurements, TFD identifies potential issues within the host machine. To further locate the

specific component, we examine the RDMA NIC's performance. The NIC's send and receive bandwidth drops significantly from normal 10-12GBps to 4-5GBps while congestion signals remain stable. Then we examine different communication patterns. The GPU logs show normal bandwidth for TP communications. This rules out GPU and NVLink failures. However, both DP and PP communications show abnormal GPU algorithm bandwidth. Based on these comprehensive observations, TFD concludes that the PCIe link between the faulty server's GPU and its RDMA NIC is the root cause. This precise diagnosis approach demonstrates TFD's capability to accurately identify host machine issues through systematic analysis of multi-layer metrics.

Besides, TFD helps identify previously unknown failure causes in LLM training clusters. In one case, with TFD's performance monitoring data, operators detect performance degradation caused by suboptimal TP scheduling, where GPUs belonging to the same TP group were incorrectly allocated across different servers, forcing cross-machine communication instead of utilizing efficient NVLink connections. In another instance, TFD's monitoring metrics help identify traffic priority misconfiguration where TP communication and storage traffic were interfering with each other, despite the intended higher priority setting for TP over DP traffic.

Furthermore, TFD helps identify and characterize the "slow node" problem [3], where certain nodes exhibit longer initialization times for reduce-scatter operations compared to others. We observed that even with identical task configurations and settings, training speed could vary significantly across multiple launches. Through TFD's multi-metric correlation analysis, we discovered that this performance variation often stems from subtle hardware issues: NIC instability manifesting as frequent link down/up events, and PCIe link speed degradation leading to reduced NIC performance. These degradations, while difficult to detect as conventional failures, frequently impact training performance.

5 Conclusion

In this paper, we present TFD, a fault detection and localization system for large-scale LLM training clusters. TFD leverages GPU algorithm bandwidth as a key indicator and employs a hierarchical analysis approach to identify and locate faults in LLM training systems. Without requiring modifications to existing training frameworks or additional monitoring overhead, TFD effectively detects and localizes three major types of faults, including host issues, network issues, and application and training framework issues. By analyzing the correlation between GPU algorithm bandwidth and training performance, TFD provides a practical solution for maintaining the reliability of LLM training clusters.

Acknowledgments. This work is supported in part by the National Key Research and Development Program of China under Grant No. 2024YFB2907000, the National Natural Science Foundation of China (NSFC) under Grant No. 62302055, the Beijing Natural Science Foundation under Grant No. L253005, and the Fundamental Research Funds for the Central Universities under Grant No. 2024ZCJH11.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. X. Ma, G. Fang, and X. Wang, "Llm-pruner: On the structural pruning of large language models," *Advances in neural information processing systems*, vol. 36, pp. 21 702–21 720, 2023.
2. S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, "Zero: Memory optimizations toward training trillion parameter models," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–16.
3. Z. Jiang, H. Lin, Y. Zhong, Q. Huang, Y. Chen, Z. Zhang, Y. Peng, X. Li, C. Xie, S. Nong *et al.*, "Megascale: Scaling large language model training to more than 10,000 gpus," *arXiv preprint arXiv:2402.15627*, 2024.
4. K. Liu, Z. Jiang, J. Zhang, S. Guo, X. Zhang, Y. Bai, Y. Dong, F. Luo, Z. Zhang, L. Wang *et al.*, "R-pingmesh: A service-aware roce network monitoring and diagnostic system," in *Proceedings of the ACM SIGCOMM 2024 Conference*, 2024, pp. 554–567.
5. K. Liu, Z. Jiang, J. Zhang, H. Wei, X. Zhong, L. Tan, T. Pan, and T. Huang, "Hostping: Diagnosing intra-host network bottlenecks in rdma servers," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 15–29.
6. A. Shah, V. Chidambaram, M. Cowan, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, O. Saarikivi, and R. Singh, "Taccl: Guiding collective algorithm synthesis using communication sketches," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 593–612.
7. L. Zheng, Z. Li, H. Zhang, Y. Zhuang, Z. Chen, Y. Huang, Y. Wang, Y. Xu, D. Zhuo, E. P. Xing *et al.*, "Alpa: Automating inter-and intra-operator parallelism for distributed deep learning," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, 2022, pp. 559–578.
8. S. Rajbhandari, O. Ruwase, J. Rasley, S. Smith, and Y. He, "Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning," in *Proceedings of the international conference for high performance computing, networking, storage and analysis*, 2021, pp. 1–14.
9. S. Li and T. Hoefler, "Near-optimal sparse allreduce for distributed deep learning," in *Proceedings of the 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2022, pp. 135–149.
10. C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn, "Rdma over commodity ethernet at scale," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 202–215.
11. Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang, "Congestion control for large-scale rdma deployments," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 523–536, 2015.
12. NVIDIA, "NCCL: NVIDIA collective communication library," https://github.com/NVIDIA/nccl, 2016.