Latency-Aware Node Selection in Federated Learning

Rustem Dautov and Erik Johannes Husom

SINTEF Digital
Oslo, Norway
{rustem.dautov, erik.johannes.husom}@sintef.no

***Abstract*—Federated learning (FL) relies on the frequent exchange of model parameters between clients and the aggregator to achieve efficient model convergence. However, network latency presents a significant challenge, particularly in congested edge/IoT scenarios, hindering the efficiency and effectiveness of distributed machine learning (ML). While existing solutions often depend on hard-coded topologies, addressing this challenge is critical to unlocking FL's full potential in real-world scenarios. This paper proposes a novel approach to mitigate network latency issues by introducing a threefold functionality: latency-aware client selection, latency-aware aggregator assignment, and consistent replication of training progress. Our proof of concept provides a scalable and robust solution to alleviate latency's impact and improve the efficiency of distributed ML operations. Through this research, we aim to advance the field of FL by offering practical solutions that enhance performance and resilience in latency-sensitive environments.**

***Index Terms*—Federated Learning, Network Latency, Node Selection, Raft Protocol, State Replication**

I. Introduction and Motivation

Federated learning (FL) offers a collaborative approach that enables multiple devices to learn a shared model while keeping data decentralised. In this paradigm, the traditional model of centralising data for training is replaced by a star topology with a central node responsible for global model aggregation and multiple client nodes performing training on their local private datasets [5]. This allows for privacy preservation, reduced communication overhead, and scalability across widely distributed networks of connected devices. However, as FL continues to evolve, challenges emerge, particularly concerning the efficiency and resilience of the system. One critical aspect that significantly impacts the performance of FL systems is network latency. The delay in communication between nodes, particularly between client devices and the central aggregator, can hinder the efficiency of model updates and ultimately degrade the quality of the learned model. As such, mitigating latency becomes paramount for ensuring the effectiveness of FL in real-world scenarios. Existing hard-coded approaches to node selection often overlook the impact of latency, leading to sub-optimal performance and inefficient resource utilisation [4]. Moreover, in dynamic network environments where latency fluctuations are commonplace, the need for adaptive strategies becomes even more important.

Our research inspires from Software-Defined Networking (SDN) and focuses on developing a latency-aware node selection mechanism that optimises the efficiency of FL by dynamically selecting client nodes based on their network latency with the aggregator. By prioritising nodes with lower latency, we aim to minimise communication overhead and expedite the model update process, thereby enhancing the overall efficiency of the FL system. Furthermore, we recognise that the aggregator node itself can sometimes be the bottleneck, leading to increased latency across the network. Detecting such scenarios is crucial for ensuring the robustness of the FL system. To address this challenge, the proposed approach involves measuring latency between node pairs and maintaining a distributed consistent table across all nodes in the FL. Leveraging Raft, a robust and efficient protocol for consistent state replication [3], each node can calculate its average latency towards other nodes and dynamically reassign the aggregator to the node with the lowest latency, thus mitigating the impact of latency bottlenecks. However, simply reassigning the aggregator node is insufficient if the system cannot seamlessly continue training without interruption. Therefore, we extend our solution with the capabilities of the Raft protocol to consistently store replicated training progress across all nodes. This ensures that a re-assigned aggregator can resume the training process without loss of previous results, thus enhancing the resilience and continuity of FL operations.

In summary, our work addresses the challenges of latency optimisation and resilience in FL systems. By developing a latency-aware node selection mechanism and leveraging the Raft protocol for dynamic aggregator reassignment and training progress replication, we aim to enhance the efficiency and robustness of FL in diverse network environments. Through a hands-on demonstration, we show the efficacy of our approach and its potential for real-world applications in various domains, especially the ones involving edge/IoT deployments.

II. Proposed Approach

Our proposed approach is designed to address the challenges posed by network latency in FL environments through a threefold functionality: *(i)* latency-aware client selection, *(ii)* latency-aware aggregator assignment, and *(iii)* consistent replication of training progress. When combined together, these components optimise efficiency and resilience in FL, ensuring smooth and uninterrupted training even in dynamic and latency-prone network conditions.

ISBN 978-3-903176-63-8 © 2024 IFIP

A. Latency-Aware Client Selection

The first component of our approach focuses on optimising client node selection based on network latency with the aggregator. Traditional FL frameworks often employ random or round-robin strategies for client node selection, disregarding the latency implications. In contrast, our approach prioritises nodes with lower latency to minimise communication overhead and speed up model updates. To achieve this, we propose a dynamic selection mechanism that continuously monitors network latency between client nodes and the aggregator. This mechanism utilises lightweight probing techniques to periodically measure round-trip latency, enabling real-time assessment of network conditions. The collected latency values are then used to dynamically adjust client node selection criteria, ensuring that nodes with the lowest latency are prioritised for participation in the FL process. Furthermore, to prevent the selection of nodes with unreliable or unstable connections, we also apply thresholding mechanisms. Nodes exceeding predefined latency thresholds are temporarily excluded from the selection pool to mitigate the risk of performance degradation due to unreliable network connections.

B. Latency-Aware Aggregator Assignment

While optimising client node selection is crucial for minimising communication latency, the effectiveness of FL is also contingent on the performance of the aggregator node. In some cases, the aggregator node itself may become a bottleneck. To address this challenge, we propose a latency-aware aggregator selection mechanism that dynamically reassigns the aggregator role to the node with the lowest average latency. Our approach leverages the Raft protocol to maintain a distributed consistent table of latency measurements between node pairs across the federated system [3]. This data consistency is extremely important to avoid potential conflicts. Each node periodically measures latency towards other nodes and updates the distributed table accordingly. Then, using these values, the nodes can be ranked based on their average latency towards other nodes, providing valuable insight into network conditions. In the event of latency spikes or prolonged periods of high latency with the current aggregator node, our framework automatically triggers a reassignment process. The node with the lowest average latency across the network automatically steps forward as the new aggregator, ensuring that the FL process remains resilient to latency fluctuations and performance bottlenecks.

C. Consistent Replication of Training Progress

In addition to optimising node selection and aggregator assignment, our framework also addresses the need for consistent replication of training progress across all nodes in the federated system. Ensuring consistent replication is essential for preserving the integrity of the training process and facilitating seamless transitions during aggregator reassignment. To achieve consistent replication of training progress, we again leverage the capabilities of the Raft protocol to maintain replicated logs of training parameters across all nodes. The aggregator periodically (typically – after each training round) updates its local training progress and propagates these updates to client nodes through Raft consensus [3]. This ensures that all nodes maintain consistent copies of the global training parameters, enabling seamless handover and continuity of the training process across aggregator transitions.

By ensuring consistent replication of training progress, our framework enhances the resilience and continuity of FL operations, enabling uninterrupted training even in the face of aggregator transitions and network fluctuations. Coupled with our latency-aware client selection and aggregator assignment mechanisms, this forms a comprehensive approach to optimising efficiency and resilience in FL environments.

III. Proof of Concept

In this section, we present a proof of concept implementation of our proposed approach to latency-aware node selection and aggregator reassignment in FL environments. Our implementation[1] leverages the *Flower* framework[2] for FL, the *PySyncObj* library[3] for Raft protocol implementation, and default Linux utilities for measuring network latency using *ping*. We showcase the functionality and effectiveness of our approach in optimising efficiency and resilience in FL systems.

A. Flower Framework for Federated Learning

Flower provides a flexible, scalable and extensible platform for distributed model training. The framework abstracts the complexities of FL, allowing developers to focus on customising strategies and algorithms tailored to specific use cases [1]. By leveraging this modular architecture, we extend the functionality of Flower by integrating a custom strategy for latency-aware node selection and aggregator reassignment into the FL pipeline. The extended strategy is essentially a Python class implementing the main steps of a FL training round, such as *fit()*, *evaluate()*, and *aggregate()*.

B. PySyncObj Library for Raft Protocol Implementation

To benefit from the Raft consensus algorithm for distributed consistency, we utilise the PySyncObj Python library. PySyncObj provides a lightweight and efficient implementation of Raft, enabling to maintain consistent state across federated nodes. Using PySyncObj, we implemented a distributed consistent key-value dictionary to store two types of values: *(i)* latency measurements between all node pairs, and *(ii)* global training progress and state. Each federated node periodically measures latency with other nodes and updates the distributed table accordingly. By leveraging Raft consensus, we ensure that all nodes maintain consistent copies of the latency table, facilitating real-time assessment of network conditions and dynamic aggregator reassignment. Whenever there is a need for a new aggregator, PySyncObj facilitates handover of the previously stored state information between the old and the new aggregator nodes. This ensures continuity of the FL process and preserves the integrity of training progress during aggregator transitions.

[1] https://github.com/SINTEF-9012/raft_flower
[2] https://flower.ai/
[3] https://github.com/bakwc/PySyncObj

C. Default Linux Utilities for Network Latency Measurement

For measuring network latency between federated nodes, we utilise ping – a simple yet powerful tool for measuring round-trip latency between two endpoints in a network. By sending ICMP echo requests and measuring response times, ping provides a straightforward method for assessing network latency in real time. In our implementation, each federated node periodically executes ping commands to measure latency with other nodes in the network. These latency measurements are then written to the distributed consistent table maintained by PySyncObj. This way, by leveraging default Linux utilities for network latency measurement, we ensure compatibility and interoperability with existing networking infrastructure, simplifying deployment and integration into FL environments.

D. Running Example

By default, client nodes in Flower are selected randomly for participation in the training process. In our proof of concept, as training progresses, latency measurements between nodes are continuously monitored using ping and updated in the distributed consistent table maintained by Raft. In the event of latency spikes or prolonged periods of high latency with the current aggregator node, a reassignment process is triggered. The node with the lowest average latency across the network is dynamically selected as the new aggregator to ensure optimal performance and resilience to network fluctuations. Furthermore, during aggregator reassignment, our prototype transfers training progress using Raft, thus achieving continuity of the FL process without interruption.

	lat_{N_1}	lat_{N_2}	lat_{N_3}	lat_{N_4}	$\overline{lat}$
N_1	–	237	256	242	245
N_2	229	–	43	51	108
N_3	252	40	–	37	110
N_4	257	64	54	–	125

→

	lat_{N_1}	lat_{N_2}	lat_{N_3}	lat_{N_4}	$\overline{lat}$
N_1	–	240	258	249	249
N_2	251	–	47	58	119
N_3	262	47	–	51	120
N_4	252	60	57	–	123

Fig. 1. Sample latency table replicated across 4 federated nodes (in *ms*).

To demonstrate the viability of our approach, we established a simple FL environment consisting of 4 Raspberry Pi boards. In the first training round (left table in Fig. 1), the initial aggregator N_1 ranks the client nodes based on their measured latency and selects only N_2 and N_3. After completing the first round, the aggregator replicates the global training results using Raft, while the rest of the nodes update their latency tables. They now realise the current aggregator's average latency exceeds 200 ms, and consistently decide to assign N_2 as the new aggregator. Next, N_2 recovers the replicated training parameters and selects N_3 and N_4 as its clients for the next training round (right table in Fig. 1). This demonstration showcases the viability of our approach in optimising efficiency and resilience in FL environments. By integrating latency-aware node selection and aggregator assignment mechanisms into the Flower framework and leveraging the PySyncObj library for consistent data replication, our prototype offers a scalable and robust solution for FL in latency-prone network environments.

IV. Further Considerations and Conclusion

Although our current network latency calculation methodology is sufficient for basic latency assessment, a more sophisticated approach could enhance the accuracy and robustness of the system. For instance, incorporating metrics such as packet loss, network congestion, or historical latency patterns could provide a more nuanced understanding of network performance. By leveraging advanced network analysis techniques, the FL system could make more informed decisions regarding node selection and aggregator reassignment, leading to improved overall efficiency and performance.

Moreover, the design allows for the incorporation of other practical metrics in the utility function beyond latency alone. For instance, considering factors such as device resource availability or data quality could provide a more comprehensive basis for selecting optimal client nodes. By extending the selection criteria with a broader range of metrics, the FL system becomes more adaptive and responsive to diverse network conditions and operational requirements.

We also acknowledge the existence of built-in leader elections in the Raft protocol itself. These elections are random by design to ensure that the protocol remains lightweight and robust – a key feature that can be used to build self-recovery mechanisms in FL [2]. In this approach, we opted for a more advanced implementation that prioritises dynamic context-awareness and explicit control over aggregator assignment.

In conclusion, our paper introduces a novel approach to addressing the challenges of network latency in FL environments. By prioritising dynamic latency-aware node selection and aggregator assignment, we demonstrated improvements in efficiency and resilience. Leveraging the Flower framework for FL, the PySyncObj library for Raft protocol implementation, and default Linux utilities for network latency measurements, our prototype offers a scalable and robust solution for real-world deployment. Furthermore, the flexibility of our approach allows for the integration of additional metrics beyond network latency, facilitating adaptability to diverse network conditions and operational requirements. Overall, our research contributes to advancing the field of FL by providing practical solutions to mitigate the impact of latency and enhance the effectiveness of distributed machine learning in latency-prone environments.

References

[1] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, et al. Flower: A friendly federated learning research framework. *arXiv preprint arXiv:2007.14390*, 2020.

[2] Rustem Dautov and Erik Johannes Husom. Raft Protocol for Fault Tolerance and Self-Recovery in Federated Learning. In *Proceedings of the 19th IEEE/ACM Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2024)*. IEEE, 2024.

[3] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX annual technical conference (USENIX ATC 14)*, pages 305–319. USENIX Association, 2014.

[4] Jiajun Wu, Fan Dong, Henry Leung, Zhuangdi Zhu, Jiayu Zhou, and Steve Drew. Topology-aware federated learning in edge computing: A comprehensive survey. *ACM Computing Surveys*, 2023.

[5] Chen Zhang, Yu Xie, Hang Bai, Bin Yu, Weihong Li, and Yuan Gao. A survey on federated learning. *Knowledge-Based Systems*, 216:106775, 2021.