Fault-Tolerant Decentralized Distributed Asynchronous Federated Learning with Adaptive Termination Detection

Phani Sahasra Akkinepally[1], Manaswini Piduguralla[1], Sushant Joshi[1], Sathya Peri[1], and Sandeep Kulkarni[2]

[1] Indian Institute of Technology Hyderabad, India
[2] Michigan State University, USA

Abstract. Federated Learning (FL) facilitates collaborative model training across distributed clients while ensuring data privacy. Traditionally, FL relies on a centralized server to coordinate learning, which creates bottlenecks and a single point of failure. Decentralized FL architectures eliminate the need for a central server and can operate in either synchronous or asynchronous modes. Synchronous FL requires all clients to compute updates and wait for one another before aggregation, guaranteeing consistency but often suffering from delays due to slower participants. Asynchronous FL addresses this by allowing clients to update independently, offering better scalability and responsiveness in heterogeneous environments.

Our research[3] develops an asynchronous decentralized FL approach in two progressive phases. (a) In Phase 1, we develop an asynchronous FL framework that enables clients to learn and update independently, removing the need for strict synchronization. (b) In Phase 2, we extend this framework with fault tolerance mechanisms to handle client failures and message drops, ensuring robust performance even under unpredictable conditions. As a central contribution, we propose *Client-Confident Convergence* and *Client-Responsive Termination* novel techniques that provide each client with the ability to autonomously determine appropriate termination points. These methods ensure that all active clients conclude meaningfully and efficiently, maintaining reliable convergence despite the challenges of asynchronous communication and faults.

Keywords: Federated Learning · Decentralized Framework · Asynchronous System · Crash Tolerance · Termination Detection

1 Introduction

Federated learning is a branch of Distributed Machine Learning (DML), in which nodes collaborate without sharing raw data. Clients conduct local model training and share parameters for aggregation, effectively preserving privacy. Centralized

[3] Code can be viewed here: https://github.com/PDCRL/CFTFedML

Fig. 1. Centralized to decentralized federated learning architecture. (a) Centralized approach with central server. (b) Decentralized peer-to-peer approach.

FL relies on a central server to coordinate client communication through two-way interactions between the server and each client, clients share the local model with the server while the server shares the global aggregrated model with all clients, as shown in Figure 1(a). In contrast, decentralized FL uses peer-to-peer communication among clients, improving resilience, similar to centralized model here too peers share models with each other, as illustrated in Figure 1(b).

Traditionally, FL is implemented using a centralized setup, as described earlier. While conceptually straightforward, centralized FL inherently suffers from scalability issues and is vulnerable to single points of failure due to its reliance on a central coordinating server. A natural alternative is decentralized synchronous FL, where clients collaboratively train without relying on a central coordinator.

However, even in controlled environments such as machines connected in the same Local Area Network (LAN), message delay and relative speeds can make it difficult to assume synchronous mode of execution. Delays, packet reordering, and device variability lead to divergence and inconsistent progress across clients. Moreover, simulations often fail to accurately capture such real-world complexities, resulting in overly optimistic assumptions about system behavior. This realization motivates our exploration of multi-machine, multi-client real-world setups, allowing us to study asynchronous FL more rigorously under realistic conditions.

An equally critical aspect of real-world FL deployments is *fault tolerance* which is the ability of the system to continue delivering correct results despite failures, such as clients dropping out or disconnecting. In this work, we specifically focus on benign crash faults, excluding more adversarial scenarios such as Byzantine faults [6].

In decentralized asynchronous setups, especially in multi-machine environments, there is no global notion of completion or termination of a work, known as *termination detection* (TD). Termination detection in a distributed system [16] is the problem of determining whether a distributed computation has completed—that is, all processes are idle and no messages are in transit. This can be challenging because each process only has partial information about the global system state, and local inactivity does not guarantee global completion.

In distributed federated learning, the absence of a clear termination condition can lead to two significant issues: clients may either terminate prematurely before sufficient convergence is achieved, or continue training unnecessarily, wasting

computational resources. To mitigate these risks, we introduce two key mechanisms: *Client-Confident Convergence* and *Client-Responsive Termination*.

The *Client-Confident Convergence* mechanism ensures that a client initiates termination only after observing sufficient stability in the training process. In response, the *Client-Responsive Termination* mechanism ensures that any client receiving this signal updates its own termination flag and propagates the signal through its subsequent model broadcasts. This enables a distributed yet coordinated shutdown across all clients, improving robustness and avoiding indefinite training. (For detailed design, refer to Section 2.)

In this work, we aim to contribute practical insights into designing federated learning systems that can work effectively (fault-tolerant) despite being the system being asynchronous and crash-prone. Specifically, in this work, we: (a) propose a fully decentralized convergence mechanism where each client autonomously monitors local model stability without centralized coordination in Section 2. (b) introduce a client-responsive termination protocol that ensures termination signals are reliably propagated, and decided when training had converged in Section 2. (c) demonstrate the practical effectiveness of the approach through experiments in realistic, multi-machine federated learning environments in Section 3.

2 Proposed Framework

2.1 System Model

We consider a decentralized, federated machine learning (FedML) system comprising a set of N clients, denoted as $C = \{c_1, c_2, \ldots, c_n\}$, where each client c_i possesses their own local dataset D_i. These datasets are non-identical and independently distributed (non-IID) and may vary in size and distribution, reflecting practical heterogeneous data scenarios. Unlike traditional centralized FL setups, our system does not rely on a central server for coordination. Instead, the clients are connected in a peer-to-peer (P2P) network topology where each client can communicate directly with all of the other clients.

Each client maintains a local model and iteratively updates it using its local data. Periodically, clients exchange model updates with their neighbors through message passing. The model exchanges are in the form of model parameters. Messages sent over the network are assumed to be reliable. We assume no message loss, corruption, or duplication. However, communication delays are allowed and may vary between different client pairs.

Asynchronous Model: The system follows a asynchronous communication model, where clients exchange messages over the network using sockets. Communication is fully decentralized, and the system is asynchronous, i.e., clients proceed with their local computations and communication at their speed without a common clock.

Failure Model: The system assumes a crash fault model, where clients may become unresponsive due to software crashes, network disconnections, or resource limitations. These faults are benign—clients stop functioning without sending incorrect or malicious messages (i.e., no Byzantine behavior is considered). The

model also supports temporary and intermittent failures, allowing clients to rejoin after transient faults.

2.2 Approach

To systematically address the challenges identified in the traditional FL, this research is structured into two distinct phases, each tailored to tackle a specific dimension of complexity in Federated Learning (FL) systems namely, asynchronous behavior and fault tolerance. By adopting this phased methodology, we ensure that both the coordination challenges of decentralized systems and the resilience requirements of fault-prone environments are addressed in a gradual and controlled manner.

Phase 1: Managing Asynchronous Behavior Using Round-Based Coordination

In the first phase, we focus on mitigating the effects of asynchronous inherent in distributed FL settings by leveraging a round-based synchronization strategy. Each client, during its communication phase, broadcasts its current round number along with its model updates to all other participating clients in the system. This exchange of round numbers provides a lightweight coordination mechanism, enabling clients to maintain awareness of the collective progress of the system despite differences in local update schedules or temporary communication delays.

This strategy not only facilitates smoother convergence of the global model but also prevents divergence due to inconsistent or out-of-order updates. To complement the synchronization mechanism, we implement a termination protocol, ensuring that all active clients reach a mutual agreement on when to halt training. This eliminates premature or inconsistent termination across the network, establishing a foundation for reliable learning outcomes in distributed environments.

Phase 2: Achieving Fault Tolerance in Fully Asynchronous FL Systems

The second phase of our research transitions to a fully asynchronous FL setup, where round-based coordination is removed, allowing clients to send updates entirely independently of one another.

To gracefully handle delays and failures, we introduce timeout-based mechanisms to detect potentially failed or unresponsive clients. In this approach, a client C_i will wait for a message m from another C_j until the timeout expires. After which C_i proceeds to the next round as shown in Algorithm 1. So, if C_j had failed, then C_i will not receive m. So, C_i will proceed with the execution while marking C_j as crashed. However, if the message m is delayed then C_i will consider m in whatever round it receives and change the status of C_j as alive. Unlike rigid synchronization, this approach allows the system to differentiate between slow and genuinely failed clients, minimizing unnecessary blocking and promoting progress even in degraded conditions.

To ensure effective learning despite faults, we propose the development of a client confidence-based convergence architecture and additionally, a client-responsive termination mechanism is incorporated to prevent indefinite waiting

Algorithm 1 Client Logic

Require: ClientID `id`, ClientsList $P = \{p_1, p_2, \cdots p_n\}$

Initialize model, optimizer, train/test data loaders
Initialize tracking variables (rounds, weights, flags)
while $current\_round < R\_PRIME$ **do** // Local Training
    for $epoch \leftarrow 1$ to $EPOCHS\_PER\_ROUND$ **do**
        Train model on local data batch-wise
    end for
    Extract model weights $localWeights$
    if termination flag received **then**
        Broadcast $localWeights$ with terminate flag
        break
    end if
    // Broadcast and Wait
    Broadcast $localWeights$ to peers
    Wait $TIMEOUT$ seconds for incoming weights
    // Crash Detection
    for all peer p in ClientsList **do**
        if no message from p_i **and** p_i not marked crashed **then**
            Mark p_i as crashed
            Log the event
        end if
    end for
    // Model Update
    Aggregate all received weights
    Update model with aggregated weights
    Evaluate accuracy on test set
    // Termination Criteria Check
    if $current\_round \geq MINIMUM\_ROUNDS$ **then**
        if $curr\_weight - prev\_weight > threshold$ **and** $recent\_crashes = None$ **then**
            Increment convergence counter
        else
            Reset convergence counter
        end if
        if $convergence\_counter \geq COUNT\_THRESHOLD$ **then**
            Log termination condition met
            Broadcast $weights$ with terminate flag
            break
        end if
    end if
    Store current weights
    round++
    Clear message buffer
end while
if maximum rounds reached **then** // Termination Finalization
    Log and broadcast final weights
end if
Broadcast termination message to all peers

on failed or lagging clients. This enables the system to dynamically adapt to fluctuating participation levels while safeguarding convergence properties.
Client-Confident Convergence: It is a decentralized mechanism where each client independently executes the same termination logic during the federated learning process. Specifically, every client continuously monitors the progress of training by evaluating two key conditions: (a) The client has observed 'x' (convergence threshold) consecutive rounds without any detected crashes in the system. (b) The difference between the previous global model average and the current global model average falls below a predefined threshold, indicating diminishing model improvement.

When both conditions are satisfied, the client broadcasts a termination signal to all other active clients in the system. The rationale here is that the client encountering this stable state first has already witnessed x stable rounds, suggesting that further training is unlikely to yield significant improvements. Importantly, this process is fully decentralized and can be triggered by any client in the system at runtime, making termination adaptive to dynamic network conditions and learning convergence.
Client-Responsive Termination Protocol: While signaling termination is essential, simply broadcasting a termination message is not sufficient in decentralized systems. Without a structured termination protocol, some clients might mistakenly interpret missing updates as client failures, leading to ambiguity regarding which clients have genuinely terminated versus those that might have crashed or disconnected.

To address this, we introduce the Client-Responsive Termination Protocol. In this mechanism, whenever a client receives a termination signal from another client, it updates its own internal termination flag in real time. From that point onward, the client continues participating in communication by broadcasting its model updates along with the termination flag enabled. This ensures that the termination signal propagates reliably throughout the network, even reaching clients that may have temporarily missed earlier signals due to delays or intermittent disconnections.

This approach adds an additional layer of robustness to the termination process, ensuring that even clients that were disconnected during the initial termination broadcast eventually receive the signal and terminate gracefully. By doing so, the system avoids false assumptions of crashes and maintains a clear, coordinated shutdown procedure across the distributed environment. Through these two phases, our research presents a comprehensive approach to building federated learning systems that are not only capable of handling the natural asynchronous of decentralized computation but also robust enough to sustain learning in fault-prone, real-world environments. The pseudocode of the algorithm incorporating these techniques is shown in Algorithm 1.

3 Experiments and Analysis

To validate the effectiveness and robustness of the proposed federated learning framework, we conducted a comprehensive set of experiments under various simulated distributed learning conditions. These experiments were designed to

assess system behavior in the presence of real-world challenges such as client speed, communication delays, and potential client failures.

Experimental setup: The experimental evaluation was conducted with the number of clients ranging from 4 to 12, increasing in steps of 2. The system was developed in Python, leveraging threads to spawn individual client processes and sockets to enable communication between them in a fully decentralized, peer-to-peer manner. All experiments were executed on CPU-only environments. Two distinct phases were tested under different environmental assumptions. In Phase 1, the system operates through synchronization over an asynchronous system, with no client crashes permitted. In contrast, Phase 2 adopts an asynchronous setting where client crashes. The experiments were deployed across a multi-machine, multi-client setup, to simulate realistic distributed environments and test the robustness and scalability of the proposed approach.

System specifications: The experiments were conducted using three high-performance machines in a distributed, multi-client, multi-machine environment. All machines communicated over a LAN (local area network), and experiments were executed entirely on CPUs without GPU acceleration as shown in Table 1.

Machine	**Operating System**	**RAM**	**Physical Cores**	**Clock Speed**
Machine 1	Ubuntu 18.04.6	376 GiB	56	4.0 GHz
Machine 2	Ubuntu 22.04.5	251 GiB	112	2.0 GHz
Machine 3	Ubuntu 22.04.5	188 GiB	52	3.5 GHz

Table 1. System specifications of the machines used.

Data specifications: Preliminary experiments were conducted using both the CIFAR-10 and MNIST datasets under IID (independent and identically distributed) and Non-IID conditions to establish baseline performance. However, the primary focus of this study is to analyze the behavior of federated learning systems under extreme and adverse conditions. **As such, the subsequent experiments presented in this work specifically focus on Non-IID data distributions using the CIFAR-10 dataset, which better reflect real-world data heterogeneity and serve to stress-test the robustness of the proposed system.** The dataset comprises 60,000 color images distributed across 10 classes, with 50,000 images used for training and 10,000 for testing. Each image has dimensions of $32 \times 32 \times 3$ pixels. To simulate Non-IID conditions, client-specific data partitions were generated using a Dirichlet distribution-based sampling strategy with $\alpha = 0.6$, which enables controlled variation in data skew by adjusting the concentration parameter. The Convolutional Neural Network (CNN) model architecture used throughout the experiments comprises two convolutional layers followed by two fully connected layers, resulting in approximately 225,034 parameters (about 0.44 MB). Consequently, each communication round between clients involved the transmission of roughly 0.44 MB of data for each client, representing model updates.

Phase 1 - Fault-Free System Experiments: To establish baseline performance metrics, we initially evaluated single-client configuration to understand the lower bounds of our federated learning system. Table 2 summarizes the classification accuracy achieved by individual clients under varying data distribution settings.

Table 2. Baseline Performance Results

Scenario	**Accuracy (%)**
Non-IID Single Client (Fixed data chunk)	26.23
IID Single Client (Fixed data chunk)	37.48
Single Client (Full dataset)	70.82

To understand the impact of data distribution and collaboration in federated learning, we evaluated three single-client training configuration.

In the first case (Non-IID Sgl), each client was assigned a fixed chunk of 5000 data points drawn from a highly *Non-IID distribution*, representing real-world data heterogeneity. These clients trained independently without any communication, and the average accuracy achieved across all clients was **26.23%**, highlighting the limitations of learning from skewed, isolated data. In the second scenario (IID Sgl), each client again received a fixed 5000 data point chunk, but this time the data was sampled in an *IID manner*, ensuring a balanced and representative distribution. This improved the average accuracy to **37.48%**, demonstrating that even without collaboration, better data diversity significantly enhances performance. The third case (Sgl Full) represents the *ideal baseline*, where a single client is given access to the **entire training dataset**. Without any distribution or communication overhead, this setup yielded an accuracy of **70.82%**, indicating the upper-bound performance that could be achieved in a centralized setting.

These results collectively emphasize the importance of collaboration in federated learning, particularly under Non-IID conditions, where isolated training yields significantly suboptimal outcomes.

Results on CIFAR-10 Dataset: The results in Figure 2 validate the effectiveness of the round-based synchronization in Phase 1, demonstrating proper coordination among clients.

As the number of clients increases from 2 to 10, accuracy steadily improves due to the inclusion of more data in each experiment. For Non-IID data, the accuracy rises from 59.78% to 67.47%, while for IID data it increases from 61.10% to 70.50%.

The system achieved consistent convergence with perfect synchronization across all runs. Termination was reliably detected using the designed consensus mechanisms. The results reveal the following insights:

- The IID scenario consistently achieves higher accuracy compared to non-IID, with a maximum accuracy of 70.50% versus 67.47%.
- Both IID and non-IID scenarios demonstrate scalability, with accuracy improving as the number of clients increases.

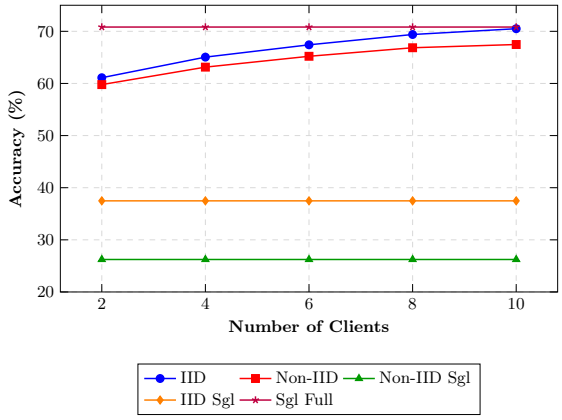

Fig. 2. Accuracy vs. Number of Clients for IID and Non-IID CIFAR-10 Settings with Single Client Baselines

These results provide a strong baseline for evaluating the fault-tolerant mechanisms implemented in Phase 2.

Phase 2: Fault Tolerance Experiments To systematically analyze the impact of client failures on federated learning, a series of targeted experiments were conducted under varying fault conditions. These experiments were designed to capture both the *progressive degradation* and the *scalability limits* of the system in the presence of crashes.To simulate a realistic system, we tested with three machine as shown in Table 1 with these machines were communicating with each other through the network. The 12 clients equally distributed among these machines. In experiments with lesser number of machines (such as 1 or 2), the clients were accordingly distributed.

Experiment 1: Variable Crash Analysis evaluates system robustness by gradually increasing the number of crashed clients from 0 to 11 out of 12, revealing how performance metric, accuracy degrade with increasing fault intensity. Fig. 4 illustrates the variation in model accuracy with an increasing number of crash faults, while Fig. 3 presents the total training time under the same fault conditions. Both figures compare the system behavior under single-machine, two-machine, and three-machine setups.

As expected, a gradual decline in accuracy is observed across all three configurations with increasing fault severity, demonstrating the system's sensitivity to reduced client participation. However, Figure 3 provides additional insight into the practical execution characteristics. In the case of zero faults, the single-machine setup incurs significantly higher training time compared to the multi-machine configurations. This elevated duration can be attributed to resource contention, since all 12 clients are hosted on a single physical machine, they compete heavily for CPU and memory, thereby introducing delays.

In contrast, when the same experiment is distributed across two and three machines, the total training time becomes more stable and comparable. This

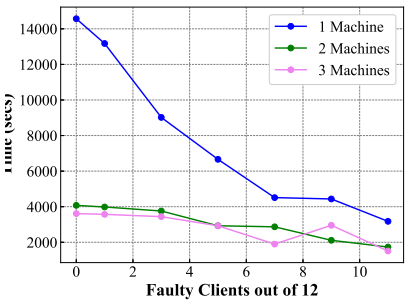

Fig. 3. Time taken with 12 Clients under Variable Fault Conditions

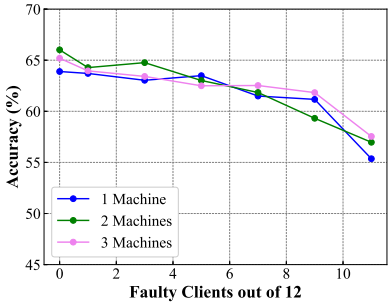

Fig. 4. Performance with 12 Clients under Variable Fault Conditions

outcome emphasizes the importance of deploying federated systems in realistic, distributed environments rather than simulating all clients on a single machine. Additionally, involving machines with differing hardware specifications and clock speeds increases the system's overall asynchrony, allowing for a more accurate evaluation of the framework's performance and robustness in heterogeneous, real-world conditions.

Experiment 2: Proportional Fault Analysis maintains a consistent failure rate (33%) across different total client counts. Here the clients fail during the system execution at regular intervals. This setup helps assess whether the system can *scale gracefully under consistent stress* and still converge effectively. These results are shown in Figures 5, 6. Here the baseline refers to the non-faulty case executing with $\lfloor(2 * n/3)\rfloor$ clients and are executing the learning algorithm outlined in phase1. When n is 4 and 12, the actual number of clients in the baseline is 3 and 8 respectively.

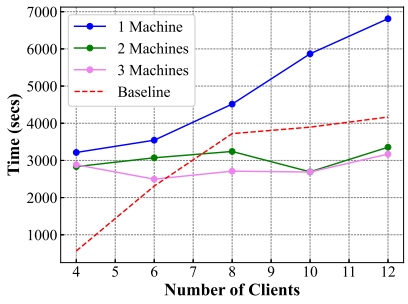

Fig. 5. Time taken with N/3 Faults and variable number of clients

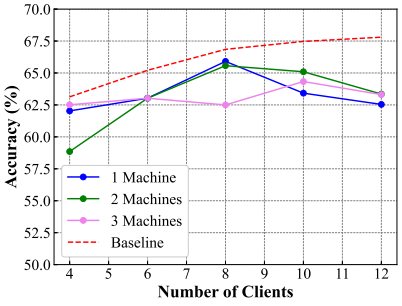

Fig. 6. Performance with N/3 Faults and variable number of clients

Fig. 6 illustrates that the three variants - single machine, two machine, and three machine setups. The experiment shows that the accuracy in case of the faulty environment is comparable to the baseline fault-free case even when handling $n/3$ client failures in an asynchronous environment. This showcases the robustness and adaptability of the proposed approach under partial client participation and non-deterministic execution patterns.

Fig. 5 highlights the time-related impact of these configurations. While the single-machine setup exhibits a noticeable increase in training time due to higher resource contention, both the two-machine and three-machine setups manage to outperform the baseline in terms of computation time. This is because the client failures in these system are configured to occur sometime in the middle of the system execution and the failed clients help with the learning process till they fail. While this is not the case with the baseline which only has lesser number of clients throughout the execution.

For example, in the case of 12 clients, an $n/3$ failure corresponds to 4 client crashes, leaving 8 functional clients in the system. When comparing with a fault-free setup involving only 8 clients (the baseline), we observe that the time taken by the baseline is higher than the time required by the system operating with 12 clients and 4 faults. This clearly highlights the efficiency of the proposed asynchronous and fault-tolerant design, which is capable of leveraging partial participation more effectively than an asynchronous fault-free configuration with fewer active clients.

Experiment 3: Maximum Fault Analysis represents the *worst-case scenario*, where only one client remains active. This setup allows for evaluating the system's capability to tolerate extreme isolation and still make learning progress under minimal participation.

As shown in Figure 7, the accuracy observed in the presence of $n - 1$ faults is significantly lower than that of the synchronized no-fault system, which is expected due to the lack of client diversity and aggregation. However, it is noteworthy that the performance is still superior to the baseline case of a single client training independently on a fixed chunk of Non-IID data, as reported in Table 2. This highlights the benefit of collaborative learning—even with limited participation—over isolated learning on skewed datasets. When considering the time behavior shown in Figure 8, we observe a natural decline in total training duration. This reduction is attributed to the smaller number of participating clients, leading to fewer communication rounds and significantly reduced coordination overhead.

Observations from the experiments: The three fault experiments collectively validate the robustness, adaptability, and practical efficiency of our proposed asynchronous, fault-tolerant federated learning framework.

Experiment 1 demonstrated the system's graceful degradation under increasing crash faults. Even as the number of failed clients rose from 0 to 11, the system maintained reasonable accuracy and convergence time, particularly when deployed in a multi-machine setup. This highlighted the effectiveness of

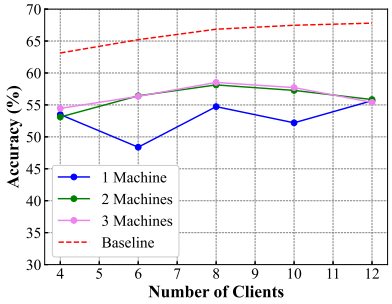

Fig. 7. Performance with N-1 Faults and variable number of clients

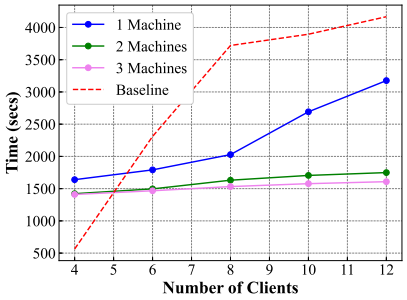

Fig. 8. Time taken with N-1 Faults and variable number of clients

distributing clients across machines to reduce contention and better reflect real-world asynchronous.

Experiment 2 examined the system's behavior under a consistent failure rate of $n/3$ across varying client counts. The results showed that despite substantial asynchronous and partial failures, the proposed approach performed comparably to a fault-free baseline. This confirmed that the system can scale under fixed-stress conditions without significant accuracy or efficiency loss.

Experiment 3 tested the extreme boundary of the system—when only one client remains active. While the accuracy dropped significantly, as expected, it still outperformed the Non-IID single-client baseline (Table 2), underlining the benefit of initial collaborative updates and the residual value of even limited federation.

4 Related Work

Federated Learning was first introduced as a practical method for training machine learning models across distributed devices while preserving data privacy [12]. The seminal work by McMahan et al. [12] presented **Federated Averaging (FedAvg)**, which operates through iterative model averaging where clients perform local stochastic gradient descent updates followed by server-side aggregation. FedAvg demonstrated significant communication efficiency, reducing required communication rounds by 10-100x compared to synchronized stochastic gradient descent.

The theoretical foundations of FedAvg convergence have been extensively studied under various conditions [15, 18]. Wang and Ji [17] provided a unified convergence analysis for federated learning with arbitrary client participation, introducing a generalized version of federated averaging that amplifies parameter updates at intervals of multiple FL rounds. Their analysis captures the effect of client participation in a single term, obtaining convergence upper bounds for both non-stochastic and stochastic participation patterns. Li et al. [18] estab-

lished convergence rates for strongly convex and smooth problems in non-IID data settings, revealing that data heterogeneity slows convergence and necessitates decaying learning rates. Recent work has challenged pessimistic theoretical predictions, showing that FedAvg can achieve identical convergence rates in both homogeneous and heterogeneous data settings under certain conditions [2].

Synchronous Centralized Federated Learning Systems

Synchronous federated learning has been extensively studied from system optimization perspectives, with comprehensive surveys identifying key bottlenecks in client selection, configuration, and reporting phases [7]. Communication-efficient approaches for synchronous FL have focused on adaptive aggregation strategies in client-edge-cloud architectures [10]. Luo et al. [10] propose theoretical convergence analysis under aggregation frequency control, enabling dynamic resource allocation while maintaining model convergence guarantees. Their FedAda method demonstrates up to 4% improvement in test accuracy, 6.8× shorter training time and 3.3× less communication overhead compared to prior solutions.

Feng et al. [4, 5] studied federated learning efficiency over wireless networks, deriving analytical expressions to characterize FL convergence rates accounting for transmission reliability, scheduling policies, and momentum methods. Their analysis reveals that delicately designed user scheduling policies or expanding bandwidth can expedite model training in reliable networks, but these methods become ineffective when connections are erratic. The incorporation of momentum method into model training algorithms accelerates convergence rate and provides greater resilience against transmission failures [4].

Real-World Asynchrony in Federated Learning

The transition from theoretical synchronous models to practical asynchronous implementations addresses critical real-world challenges including device heterogeneity, intermittent connectivity, and varying computational resources [3,9]. Liu et al. [9] propose an adaptive asynchronous federated learning (AAFL) mechanism for resource-constrained edge computing, where a certain fraction of local updates are aggregated by arrival order at the parameter server in each epoch.

Asynchronous federated learning eliminates the need for synchronized communication, allowing devices to contribute updates at their own pace [3]. Recent advances focus on addressing efficiency challenges through prospective momentum aggregation and fine-grained correction techniques [20]. The work by Liao et al. [8] contributes to improved robustness in federated learning through decentralization and asynchronous updates. While this approach effectively addresses device heterogeneity, it does not explicitly implement fault tolerance mechanisms, leaving gaps in handling system failures during training. Similarly, MPLS [19] is a decentralized federated learning method that speeds up training by letting devices share and combine important parts of their models asynchronously, helping handle different device capabilities in edge environments; however, it does not address fault tolerance in the system.

Morell et al. [13] introduces a dynamic and adaptive fault-tolerant asynchronous federated learning framework utilizing volunteer edge devices. While this demonstrates the feasibility of distributed training across platforms like web

browsers and terminal processes, such proof-of-concept environments fail to fully capture the computational heterogeneity, network variability, and rich data distributions encountered in real-world deployments. This limits their practical applicability for evaluating fault-tolerant learning at scale. A centralized approach with sequential model combination using first-come-first-serve aggregation is introduced by Ma et al. [11]. This work demonstrates practical applications of asynchronous FL in industrial fault diagnosis scenarios, though its centralized nature may present single points of failure.The potential of decentralized architectures to enhance system resilience over centralized alternatives has been extensively discussed in the literature [21]. However, existing approaches often suffer from high communication overhead and significant message complexity, which limit their practicality in real-time, fault-tolerant federated learning scenarios [14]. This highlights the pressing need for more efficient, scalable solutions that can deliver fault tolerance without compromising responsiveness.

The reviewed literature reveals a clear progression from early synchronous centralized approaches to more sophisticated asynchronous and decentralized methods [7, 21]. However, existing work focuses on robustness through decentralization without explicit fault tolerance or addresses fault tolerance with prohibitive computational overhead [14]. The challenge of achieving fault-tolerant federated learning in real time even with simple averaging mechanisms to start with, while maintaining efficiency remains largely unaddressed [4, 9]. This gap motivates research on lightweight fault tolerance mechanisms that can operate effectively in distributed learning environments in real time without compromising the fundamental benefits of federated learning paradigms [10, 17]. However, in the context of comparing performance with state-of-the-art frameworks, there is currently no practical implementation to enable direct execution-level comparison.

5 Conclusion

In this work, we presented a fully asynchronous, fault-tolerant federated learning framework designed to operate effectively under realistic deployment conditions characterized by client heterogeneity, network instability, and the absence of centralized control. Through systematic experimentation, we demonstrated the framework's robustness to crash faults, its ability to scale under partial failures, and its resilience even in extreme scenarios with minimal active clients. The proposed Client-Confident Convergence and Client-Responsive Termination mechanisms address a long-standing challenge in decentralized asynchronous FL achieving reliable, efficient, and autonomous termination without global coordination. By combining these contributions with practical fault-tolerance strategies, our framework enables scalable and efficient federated learning across distributed, failure-prone environments. Further details on the algorithmic design and comprehensive performance comparisons are presented in the archived version of this paper [1]. In future we will focus on extending these guarantees to tolerate Byzantine faults, where clients may behave arbitrarily or maliciously

due to software bugs, or adversarial manipulation. Incorporating Byzantine resilience is essential for deploying FL in open, untrusted environments such as cross-organizational collaborations or public networks

References

1. Akkinepally, P.S., Piduguralla, M., Joshi, S., Peri, S., Kulkarni, S.: Fault-tolerant decentralized distributed asynchronous federated learning with adaptive termination detection (2025), https://arxiv.org/abs/2509.02186
2. Beikmohammadi, Y., Pillutla, K., Karimireddy, S.P., Stich, S.U.: On the convergence of federated averaging with cyclic client participation. arXiv preprint arXiv:2402.16520 (2024)
3. Chen, X., Li, Q., Wu, Q., Zhang, X.: A survey on asynchronous federated learning. IEEE Communications Surveys & Tutorials (2024)
4. Feng, Y., Wan, S., Liu, M., Chen, S.: Towards efficient federated learning over wireless networks: A convergence analysis. IEEE Transactions on Communications (2024)
5. Feng, Y., Wan, S., Liu, M., Chen, S., Poor, H.V.: Understanding federated learning efficiency over wireless networks. IEEE Transactions on Wireless Communications (2024)
6. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS) **4**(3), 382–401 (1982)
7. Li, T., Sahu, A.K., Talwalkar, A., Smith, V.: Federated learning: Challenges, methods, and future directions. ACM Computing Surveys **55**(6), 1–35 (2023)
8. Liao, Y., Xu, Y., Xu, H., Chen, M., Wang, L., Qiao, C.: Asynchronous decentralized federated learning for heterogeneous devices. IEEE/ACM Transactions on Networking **32**(5), 4535–4550 (2024)
9. Liu, J., Xu, H., Wang, L., Xu, Y., Qian, C., Huang, J., Huang, H.: Adaptive asynchronous federated learning in resource-constrained edge computing. IEEE Transactions on Mobile Computing **21**(12), 4515–4528 (2021)
10. Luo, L., Zhang, C., Yu, H., Sun, G., Luo, S., Dustdar, S.: Communication-efficient federated learning with adaptive aggregation for heterogeneous client-edge-cloud network. IEEE Transactions on Services Computing **17**(6), 3241–3254 (2024)
11. Ma, X., Wen, C., Wen, T.: An asynchronous and real-time update paradigm of federated learning for fault diagnosis. IEEE Transactions on Industrial Informatics **17**(12), 8531–8540 (2021)
12. McMahan, H.B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Proceedings of the 20th International Conference on Artificial Intelligence and Statistics. vol. 54, pp. 1273–1282. PMLR (2017)
13. Ángel Morell, J., Alba, E.: Dynamic and adaptive fault-tolerant asynchronous federated learning using volunteer edge devices. Future Generation Computer Systems **133**, 53–67 (2022)
14. Ranellucci, S., Dov, N., Orsini, E., Rotaru, D., Smart, N.P.: Learning from failures: Secure and fault-tolerant aggregation for federated learning. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 2657–2670 (2022)
15. Sun, G., Luo, L., Zhang, C., Li, J., Chen, D., Yu, H.: Decentralized federated learning: Fundamentals, state of the art, frameworks, trends, and challenges. IEEE Communications Surveys & Tutorials **25**(4), 2983–3013 (2022)

16. Tel, G.: Introduction to Distributed Algorithms. Cambridge University Press, 2nd edn. (2000), chapter 8: Termination Detection
17. Wang, S., Ji, M.: A unified analysis of federated learning with arbitrary client participation. Advances in Neural Information Processing Systems **35**, 2323–2335 (2022)
18. Wang, X., Li, G., Zhang, J., Wang, Z., Zhang, Y.: A novel federated learning approach with local adaptive differential privacy. Neurocomputing **472**, 103–115 (2021)
19. Xu, Y., Yao, Z., Xu, H., Liao, Y., Xie, Z.: MPLS: Stacking Diverse Layers Into One Model for Decentralized Federated Learning. In: Euro-Par 2025: Parallel Processing. pp. 190–204. Springer Nature Switzerland (2026)
20. Zang, Y., Xue, Z., Ou, S., Chu, L., Du, J., Long, Y.: Efficient asynchronous federated learning with prospective momentum aggregation and fine-grained correction. Proceedings of the AAAI Conference on Artificial Intelligence **38**(15), 16642–16650 (2024)
21. Zhang, W., Li, T., Lu, J., Liu, Y., Chen, D.O.: Decentralized federated learning: A survey and perspective. IEEE Internet of Things Journal (2023)