Client Availability in Federated Learning: It Matters!

Dhruv Garg[*]
Georgia Tech

Debopam Sanyal[*]
Georgia Tech

Myungjin Lee
Cisco Research

Alexey Tumanov
Georgia Tech

Ada Gavrilovska
Georgia Tech

Abstract

Achieving efficient Federated Learning (FL) in large-scale cross-device deployments is challenging, in part due to stragglers and stale updates. However, another key yet often overlooked challenge is client unavailability. Some clients selected for training remain inactive and fail to participate. Unlike stragglers, which eventually return updates, unavailable clients provide no updates, causing rounds to stall. Even when they do return, they introduce significant delays, both of which increase time-to-accuracy. Existing FL approaches, including synchronous FL (SyncFL) and asynchronous FL (AsyncFL), fail to address client unavailability– SyncFL suffers from long waits, while AsyncFL experiences increased update staleness. Some recent client selection strategies assume oracular knowledge of client availability and prioritize historically high-utility clients. Not only is this an unrealistic assumption, but these strategies also rely on stale client utility values, which become increasingly stale as client unavailability rises. To enable scalable and efficient FL in real-world scenarios, we argue that it is crucial to address fluctuating and high client unavailability. This work highlights its impact on FL and underscores the need for a real-time, availability-aware mechanism that improves client selection and ensures high-quality update aggregation, ultimately reducing time-to-accuracy.

Keywords: Federated Learning, Client Availability

ACM Reference Format:
Dhruv Garg, Debopam Sanyal, Myungjin Lee, Alexey Tumanov, and Ada Gavrilovska. 2025. Client Availability in Federated Learning: It Matters!. In *The 5th Workshop on Machine Learning and Systems (EuroMLSys '25), March 30–April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3721146.3721964

[*]Equal contribution

Trace	FedScale[10]	LinkedIn[20]	Google[3]
Client Availability	10-20%	20-80%	10-60%

Table 1. Real-world client availability rates from prior works.

1 Introduction

Federated Learning (FL) has emerged as the predominant paradigm for training machine learning models in a privacy-preserving manner on distributed data across a vast number of clients in the cross-device FL setting. It enables decentralized model training without requiring raw data to leave client devices. Achieving fast *time-to-accuracy*—the time required to reach a target accuracy—is critical for accelerating model iteration and deployment in practical scenarios [3, 12, 18].

A major challenge in large-scale FL deployments is the inherent *unavailability* of clients, which significantly impacts time-to-accuracy. We define unavailability as the inability of selected devices to participate in training due to being inactive. Unlike *stragglers*—clients that are slow but eventually return updates—*unavailable* clients fail to participate in training entirely, leading to stalled rounds and slower convergence. Prior works [3, 10, 20] largely overlook this issue, often assuming that clients will consistently participate. However, real-world FL jobs span hours to days, during which clients experience fluctuations in availability due to factors such as network connectivity, device battery levels, and application activity [1]. Consequently, many clients remain unavailable for extended periods, as shown in Tab. 1, posing a significant challenge to effective training in cross-device FL settings.

Synchronous FL (SyncFL) [3] operates by selecting multiple clients to train in each round. Each client receives the latest global model weights, performs local training, and returns updates to the aggregator which combines them to produce a new global model before proceeding to the next round [15]. The primary limitation of SyncFL is the impact of *straggler clients* on the learning process. The slow clients delay training since the server must wait for all selected clients to return updates before continuing [3]. In contrast, Asynchronous FL (AsyncFL) [17, 22] aggregates updates as they arrive, thereby mitigating bottlenecks caused by slow clients. However, this approach introduces *staleness in updates*, where outdated updates from slower clients adversely affect global model convergence and accuracy.

Both SyncFL and AsyncFL fail to address the issue of unavailability. They select clients in an availability agnostic

Features	Flower[2]	FedML[6]	Flame[5]	FedScale[10]
Asynchronous FL	✗	✗	✓	✓
Availability Aware	✗	✗	✗	▲
Pluggable Selection	✗	✗	✓	✓
Intelligent Selection	✗	✗	✗	▲
Pluggable Aggregation	✗	✗	✓	✗
Intelligent Aggregation	✗	✗	✗	✗

Table 2. Features across FL Systems. ✓: Yes, ✗: No, ▲: Partial

manner and unavailable clients return no updates if selected. This results in slower and reduced number of updates for aggregation, delaying the training progress [4]. Recent works have proposed improving client selection strategies [7, 11] by prioritizing clients based on past observations of their update quality and availability. However, these methods rely on stale utility metrics, as they assume a client's utility can only be assessed post-training. Degree of utility staleness increases with longer unavailability durations of the clients. Furthermore, past observations of availability cannot accurately predict a client's current or future availability state, making these approaches ineffective in high-unavailability settings. As shown in Tab. 2, current FL systems lack accurate, real-time unavailability tracking mechanisms that can scale to large client populations. Their underlying assumption—that all clients are available for training—not only leads to suboptimal client selection but also under-utilization of currently available clients, as other transiently unavailable clients continue to be considered for training.

We highlight the need for a mechanism to address dynamic unavailability in real-world cross-device FL, thereby making it more robust and reducing time-to-accuracy. Our work demonstrates the benefits of (a) incorporating real-time availability monitoring at the systems level and (b) ensuring that the global model receives fast and high-quality updates at the algorithmic level. Specifically, we analyze the impact of unavailability, showing when and how it matters, and provide insights into potential improvements for FL systems. These improvements can pave the way for more scalable and efficient FL deployments in practical scenarios.

2 Background

2.1 SyncFL and AsyncFL Overview

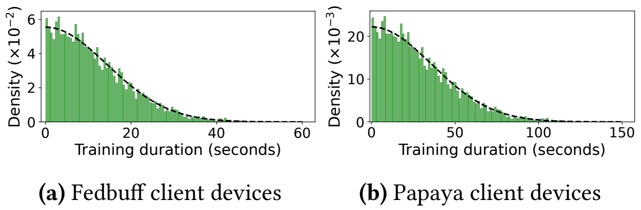
SyncFL. Synchronous Federated Learning [3] follows a synchronized approach where multiple selected clients perform training in a round, and their resulting updates are aggregated together to revise the global model. This method is inspired by large-batch synchronous SGD [15], a technique proven to be highly effective in centralized data centers. During each training round, a subset of clients, determined by a parameter known as *cohort size*, computes the gradient of the loss function based on their local data. These client-side gradients are then aggregated by the central server to perform a weighted global model update. The synchronous nature ensures that all selected clients contribute in every round, ensuring data from these clients equally influences the learning process—particularly important in heterogeneous data settings. SyncFL is also better suited for privacy, as training and aggregating updates across a large number of clients makes inference attacks ineffective. However, SyncFL is prone to the issue of stragglers as rounds proceed at the pace of the slowest client.

AsyncFL. Asynchronous Federated Learning [22] offers a promising approach to accommodate the varying compute and connectivity capabilities of the clients in the highly heterogeneous cross-device FL settings. Much of the existing research has focused on mitigating the impact of stragglers. Notably, Papaya [7] and FedBuff [17] address this issue by buffering all received model updates and incorporating them into the global model as soon as the buffer size reaches a threshold size, called the *agg-goal*. This eliminates the dependency to wait for slow devices to return updates. In cases of update delays, the global aggregation can proceed, while updates from the slower clients would be utilized in a future round. This also brings to light a major drawback of AsyncFL where slow devices risk providing highly *stale* updates. Staleness is defined as the difference between the model version number at the server (when it receives the client update) and the model version number that the client started training with. More formally, staleness (S) is calculated as $S = v_s - v_c$, where v_s is the current global model version number and v_c is the model version that the client trained on. Stale updates can be detrimental to the training progress and may even degrade the final convergence accuracy. Previous works [17, 22] incorporate staleness mitigation measures during aggregation to curb the impact of stale updates. However, these methods harshly penalize even moderately stale updates from slower devices, leading to slower overall training, and more so in heterogeneous settings. As a result, there is an increased likelihood of client participation skew in AsyncFL where faster clients participate more often than slower clients. This too is detrimental for the global training process.

2.2 Heterogeneity in cross-device FL

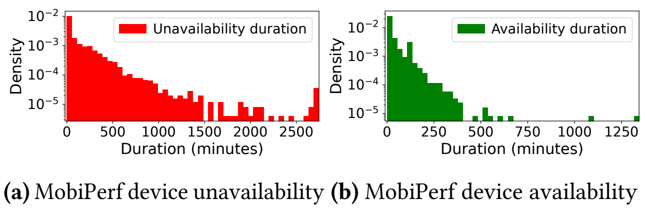
Data heterogeneity in FL arises because each client has its own data, resulting in non-identical data distributions. While there have been efforts to handle data heterogeneity [21], these approaches often assume high device participation in every round, which is often infeasible in real-world FL settings. Methods that address data heterogeneity by sharing local data [8] raise additional concerns such as high communication overhead and potential privacy violations.

Beyond data heterogeneity, *client heterogeneity* presents another important challenge. Clients differ in their computational power, storage availability, and network connectivity. This can result in stragglers and on-device training failures in the FL setting. In practice, a common strategy is to drop slower or resource-constrained clients if they fail to meet the bare minimum training requirements in a specified time



(a) Fedbuff client devices **(b)** Papaya client devices

Fig. 1. Training delays of Fedbuff and Papaya client devices. Training delays from [7, 17] are typically <120s.



(a) MobiPerf device unavailability **(b)** MobiPerf device availability

Fig. 2. Real-world device availability and unavailability durations. Availability/unavailability spans from a few minutes to several hours [14].

window [3]. However, this approach reduces the pool of active clients contributing to training and risks derailing the process if the dropped clients possess unique data distributions. Therefore, designing frameworks that robustly handle varying system capabilities while simultaneously accommodating heterogeneous data distributions is essential for practical, large-scale FL deployments.

2.3 Unavailability vs. Stragglers

Real-world cross-device FL deployments enable privacy preserving model training but face significant practical challenges. A key issue is the variability in round durations caused by device heterogeneity. Devices with latest hardware or smaller datasets complete training faster, while those with dated hardware or larger datasets require more time. Furthermore, devices communicate over wide-area networks (WANs), where bandwidth fluctuations can delay or drop the model updates. Devices impacted by such delays are referred to as *stragglers*—they contribute updates but with a lag.

Beyond stragglers, devices can also become *unavailable* due to dynamic runtime conditions. User-facing applications often take priority over FL training, resulting in interruptions when devices are in use, have low battery, lack Wi-Fi connectivity, or do not have the required application running in the foreground [3, 20]. Unlike stragglers, unavailable devices temporarily pause their participation and resume only when conditions improve, by returning to the *available* state. This transient unavailability necessitates the exploration of both system and algorithmic approaches to tackle the problem. While both stragglers and unavailability impact real-world FL training, unavailability remains an unresolved challenge. In the following section, we quantitatively differentiate these conditions to better distinguish between stragglers and unavailable devices.

Occurrence. In general, training devices exhibit minimal variability in their per-round training times. Figures 1a and 1b showcase the training duration distribution across millions of mobile devices in production. While most devices complete a single round quickly, the longer training times are relatively predictable, stemming from static factors such as slower hardware or larger local data partitions. However, unpredictability can sometimes arise due to network delays. Device availability, in contrast, is much more erratic, driven by factors like individual users' device load and various on-device conditions, which are difficult to predict. On a larger scale, device availability across regions or populations tends to follow diurnal patterns. Real-world traces [3, 10, 20] reveal that device availability fluctuates between 10% and 80% of the client population, as shown in Table 1, while still maintaining the diurnal variation.

High variability. FedBuff [17] and Papaya [7] collected per-round training times from millions of devices in real-world deployments. These runtimes follow a predictable half-normal distribution, ranging from a few seconds to less than two minutes (Fig. 1a and Fig. 1b). Slower devices in this distribution are considered stragglers. In contrast, client behavior dynamics from the real-world MobiPerf [14] trace follow a much broader distribution. The availability and unavailability durations of devices exhibit orders of magnitude higher variability, ranging from a few minutes to several hours (Fig. 2a and Fig. 2b). Similar patterns can be observed in other publicly available traces [1, 3, 10, 14, 20].

3 Impact of Client Unavailability

We now aim to quantify and analyze the detrimental effect of transient client unavailability in FL. We evaluate both synchronous and asynchronous FL paradigms across (a) synthetic and (b) real-world client unavailability traces. The collected evidence brings attention to gaps in present-day *algorithms* and *systems* for FL, and makes a case for treating client unavailability as a new fundamental concern in FL deployments.

3.1 Experiment Setup

Testbed. Our setup utilizes an emulated distributed testbed with Flame [5] and MQTT [13]. Flame replicates the cross-device FL topology with clients training on their partitioned local datasets, and a cloud-based aggregator (server). The server houses a control-plane for communication, a selector for per-round trainer selection and the aggregator to assimilate trainer updates. The experiments run on a node with 8 NVIDIA A40 GPUs, 500GB RAM and AMD EPYC 7513 32-Core processor with 128 CPUs. The FL server and all clients are hosted on it.

Task, dataset, model. We perform FL training for an image classification task on the CIFAR-10 dataset (60K 32x32 images across 10 classes), using a convolutional neural network (CNN) with 12 layers and ≈0.5M parameters.

Execution Strategies. SyncFL is represented by two strategies: OORT and OORT∗. OORT [11] is the SOTA SyncFL selection baseline. However, OORT is not inherently availability aware. An oracular version of OORT is the OORT∗ strategy, where the entire client availability trace is given a priori. With oracular knowledge, this strategy is guaranteed to select trainers only from those available to train.

For AsyncFL, FedBuff [17] is the SOTA baseline. However, its random client selection algorithm is inferior compared to OORT. Therefore, we implement A-OORT— an asynchronous strategy based on OORT. While it uses the same underlying OORT selection algorithm, we incorporate additional state management to enable fine-grained asynchronous trainer selection, compared to the once-per-round trainer selection in SyncFL OORT. To the best our knowledge, this is the most performant AsyncFL strategy yet. Since A-OORT doesn't exist in literature and instead builds upon existing methods, it is not a baseline, but rather an optimistic strategy for comparison. It combines the benefits of both FedBuff (AsyncFL aggregation) and OORT (selection). Like OORT, A-OORT also has two variant strategies: availability unaware A-OORT, and an oracular A-OORT∗.

In our experiments, we vary the client unavailability traces as detailed in §3.2 and §3.3. We denote the strategies as *S(X)*, where *S* represents the strategy and *X* represents either the average client unavailability percentage or the the unavailability trace for that experiment, as applicable. For example, OORT(20%) and OORT∗(20%) refer to the OORT strategy–availability unaware, and the oracular OORT∗ strategy respectively, on a trace with 20% client unavailability, on average.

Training workload. Experiments use a centralized FL topology with all clients directly connected to an aggregator.

Clients. We emulate 300 clients that not only exhibit data and system heterogeneity, but also exhibit varied availability. First, we simulate data heterogeneity by partitioning the training dataset using the Dirichlet(α) distribution across clients. Second, clients simulate training delays of $1 - 60$s for system heterogeneity. These delays match the training run-times of 1 million+ devices [7, 17] that exhibit performance variability due to differences in hardware, network, and datasets, as shown in Fig. 1a and Fig. 1b. The delays distinguish the fast and straggler clients in the population. Finally, clients exhibit availability patterns based on the traces in Fig. 4. Additional details about client unavailability patterns are provided in §3.2 and §3.3.

For a fair comparison of strategies in each experiment, an identical client population setup is emulated across runs, *i.e.*, the data distributions and unavailability patterns of clients are identical. Additionally, the test dataset used to evaluate the global model at the aggregator remains unchanged across all runs.

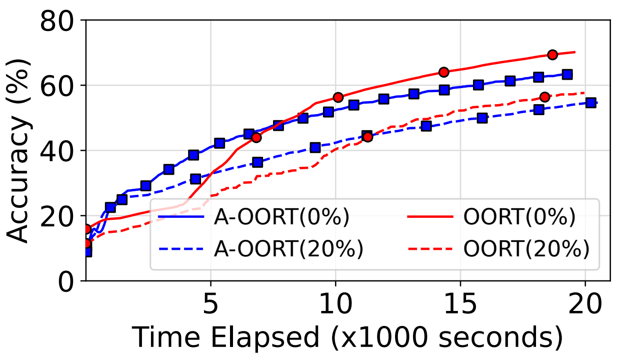
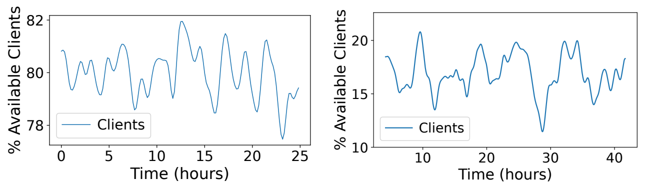


Fig. 3. Synthetic trace. The performance of both SOTA SyncFL (OORT) and AsyncFL (A-OORT) strategies degrade by > 10% w.r.t full client availability, even as the availability drops only by ∼ 20%.



(a) Synthetic trace with ∼80% client availability. **(b)** Mobiperf trace with 12-22% client availability.

Fig. 4. Synthetic and real-world client availability traces.

Metrics. We track the wall-clock time required by all strategies to achieve 70% test accuracy on the task. Production systems prefer lower time-to-accuracy. We also track low-level telemetry such as staleness and utility of updates, and aggregation stalls at the server to explain our insights.

3.2 Synthetic Availability Trace

We first evaluate FL training under a controlled unavailability setting and homogeneous data partitions.

Client behavior. Fig. 4a shows a client population averaging 80% availability, with some fluctuations. Thus, 20% clients are unavailable to train at any time. This trace was synthesized from a binomial distribution parameterized by the size of the client pool and probability for trial success. For each successful trial, the duration of trainer unavailability is 10 minutes.

Fig. 3 shows the training progress of the SOTA SyncFL OORT baseline and its AsyncFL equivalent strategy A-OORT in 0% and 20% client unavailability, respectively. Some key observations from the experiment are as follows:

Observation 1: *Significant drop in training performance in both OORT and A-OORT with modest unavailability.* This synthetic 20% unavailability is a rather lenient deployment scenario compared to high-unavailability scenarios depicted in Tab. 1. In the figure, A-OORT(0%) and OORT(0%) depict training progress in the presence of 0% unavailability *i.e.* 100% availability. Both strategies in 100% availability make steady progress towards the target accuracy. Their training curves form a reference performance upper-limit for comparison. However, Upon introducing 20% unavailability, it is seen that both OORT(20%) and A-OORT(20%) face performance drops of $> 10\%$ in their accuracy within the short experiment runtime of 6 hours. The accuracy-time elapsed gap is further expected to widen as the 0% unavailability deployments attain convergence fast, while OORT(20%) and A-OORT(20%) would continue to struggle due to transient client unavailability. Thus, we conclude that current SOTA FL algorithms cannot handle unavailability out-of-the-box, and must be improved.

Observation 2: *A-OORT completes rounds faster than OORT, but forfeits some gains with unavailability.* A-OORT, with its design of fast asynchronous aggregations, typically loses some performance due to update staleness, especially in 100% availability settings. However, high unavailability affects update staleness in two ways. First, as unavailability increases, the pace of updates and as a consequence, aggregation frequency, decreases. This lower aggregation frequency is illustrated by the wider gaps between consecutive square markers in A-OORT(20%) compared to A-OORT(0%). The square markers appear after every 200th aggregation and are visibly further apart in the run with unavailability. Fewer aggregation rounds indirectly help reduce staleness of updates, since staleness S given by $S = v_s - v_c$, now evaluates to lower values. Second, and contrary to this, unavailability can also significantly increase update staleness. This arises from trainers that receive global weights but become unavailable during or just before training begin. When such clients eventually return weights, their updates are stale by a large margin (unavailability durations are very high Fig. 2a). As a result, these client updates cannot make meaningful contributions to the global model.

Observation 3: *OORT progresses slowly but attains higher accuracy than A-OORT.* In contrast to A-OORT, OORT waits for all model updates before aggregating and distributing a new model version to clients. This synchronization step leads to longer per-round times compared to AsyncFL. This is due to stragglers as can be seen in OORT(0%) where the circle markers representing every 200th aggregation are more spread apart, representing longer per-round durations. Further, as the unavailability increases, the time taken to receive updates from trainers elongates. This leads to even slower per-round progress, as shown by the even more dispersed circle markers in OORT(20%). However, since there are no stale updates, the learning per-round in OORT is significant and the weighted averaging yields a more impactful push towards the global model minima. This is contrary to the faster, more frequent but less impactful aggregations in A-OORT.

Observation 4: *A-OORT is more resilient to unavailability than OORT.* This is despite the lower per-round learning progress in A-OORT strategy compared to the OORT baseline. Due to higher concurrency in AsyncFL—by design, A-OORT is able to tap into a larger pool of available of clients. Moreover, it is able to extract more updates at a higher frequency from these clients. Thus, it is able to gain more knowledge from the clients for global model training. This ensures that several small updates contribute to the global model in A-OORT(20%), compared to fewer but more impactful aggregations in OORT(20%). As a result, the accuracy degradation in A-OORT(20%) is observed to be lower than the accuracy degradation in the OORT(20%) baseline. For the given 6 hour experiment in Fig. 3, the accuracy degradation in A-OORT(20%) was found to be 9.5%, which is 2.5% lower than the 11% accuracy degradation seen in OORT(20%). These gaps are relative to A-OORT(0%) and OORT(0%) respectively. However, given the limitations of SyncFL in high unavailability scenarios, the time-to-accuracy gap is expected to widen further due to prolonged training stalls in SyncFL.

3.3 Real-World Availability Trace

This set of experiments evaluates SOTA FL training under real-world unavailability conditions, along with heterogeneity in data distribution.

Client Behavior. To faithfully emulate real-world FL deployments, we incorporate all three levels of heterogeneity. First, we partition the dataset across 300 clients with varying levels of $Dir(\alpha)$. Second, we continue to emulate system heterogeneity through client training runtimes. Lastly, we use a real-world client unavailability distribution 4b. These three factors enable us to comprehensively evaluate the performance of the OORT baseline and A-OORT strategy. The client unavailability is notably high, exceeding 75%. Moreover, as shown in Fig. 2a and Fig. 2b, unavailability durations are disproportionately longer than availability durations, complicating real-world FL training. For the experiment, each client simulates the behavior of a randomly selected device from the 130K+ devices in the MobiPerf device trace [14]. The original 7-day trace was slightly compressed to represent a 6-day equivalent. As specified earlier, the clients' availability may or may not be known to the aggregator. Oracular strategies, OORT$*$ and A-OORT$*$, have client availability a priori, while OORT(M) and A-OORT(M) operate without any such information.

The results of FL training in real-world settings are presented in Fig. 5, with the key observations summarized below.

Observation 1: *The SOTA OORT baseline and A-OORT strategy completely break down under high unavailability.* Fig. 5 illustrates the accuracy gap between the current SOTA algorithms: OORT baseline and A-OORT strategy, compared to their oracular counterparts, OORT$*$ and A-OORT$*$, which have prior knowledge of client availability. This accuracy

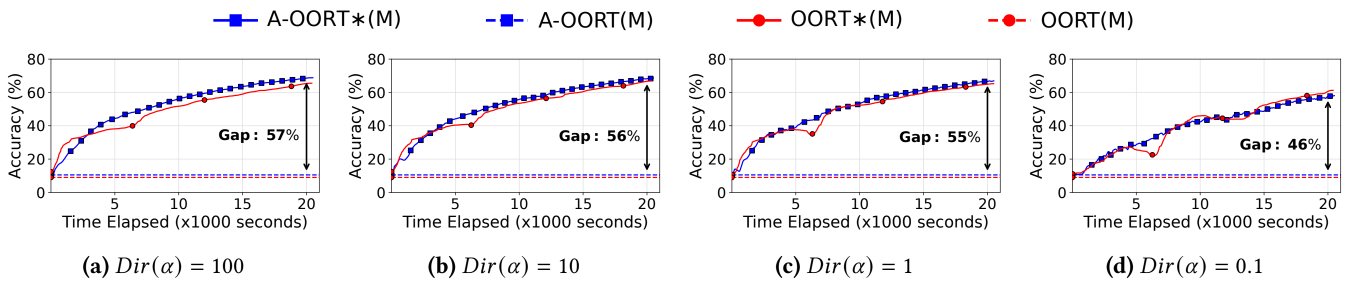


Fig. 5. Real-world trace. SyncFL and AsyncFL suffer by 46 – 57% in time-to-accuracy compared to their oracular counterparts: OORT∗ and A-OORT∗. The availability unaware strategies, A-OORT and OORT, face stalls and do not cross even 100 rounds.

gap significantly hinders the efficient deployment of FL for real-world model training on edge devices. Without availability information, OORT and A-OORT fail to make meaningful progress in training, remaining stuck at a low (10%) training accuracy for over five hours. In contrast, OORT∗ and A-OORT∗, despite using the same learning algorithms, improve training efficiency by selecting only from the pool of available trainers. This results in accuracy levels ranging from 46% to 57%, as data heterogeneity decreases from $Dir(\alpha) = 0.1$ (most heterogeneous) to $Dir(\alpha) = 100$ (most homogeneous).

Observation 2: *Oracular knowledge enables sustained training progress.* Note that the circle markers on the OORT∗ baseline are approximately equally spaced w.r.t each other and so are the square markers on the A-OORT∗ strategy w.r.t each other in Fig. 5. This indicates that the 200th round aggregations occur at nearly the same time for both strategies, suggesting similar per-round training times despite data heterogeneity and high client unavailability. In contrast, availability-unaware strategies like A-OORT and OORT experience frequent stalls and fail to reach even 100 rounds. Addressing client unavailability in FL is crucial not only for improving accuracy but also for ensuring steady training progress. A deeper analysis of A-OORT and OORT runs revealed that many weight updates sent to trainers were not utilized immediately. This occurred because selected trainers were either already unavailable or became unavailable shortly *after* receiving model weights. Thus, while the aggregator assumes that the trainer is making progress, it is not. This miscommunication between the trainers and the aggregator results in long delays and inefficient utilization of available clients.

Observation 3: *The combination of unavailability and data heterogeneity significantly hinders training.* As shown in Fig. 5, training progress becomes increasingly irregular and unstable across different strategies as deployment scenarios shift from homogeneous (easier) to heterogeneous (more challenging) data distributions.While the A-OORT∗ strategy appears to exhibit a more stable learning curve compared to OORT∗, a closer examination reveals that this stability is

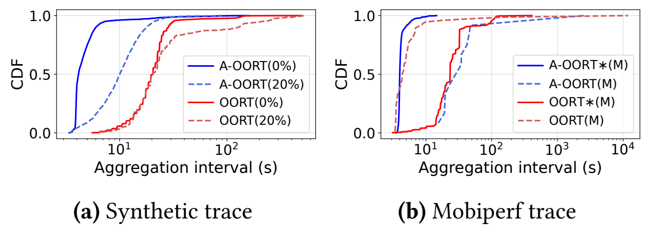


Fig. 6. Aggregators using SOTA FL algorithms experience significant stalls ranging from 10^2 to 10^3 seconds per-round in both synthetic and real-world unavailability traces.

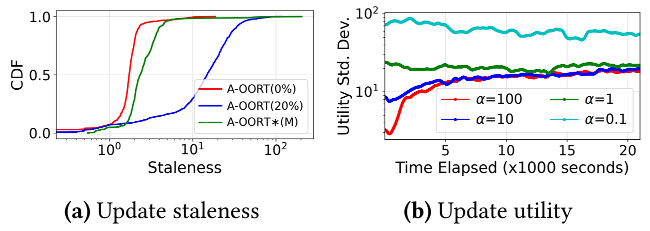


Fig. 7. The staleness and utility of trainer updates are heavily influenced by both client unavailability and data heterogeneity.

not as robust as it initially seems. The variability in training progress highlights the importance of selecting clients not only based on availability and local training speed but also on the data they possess. More informed client selection could lead to a more stable learning trajectory for the global model at the aggregator, ultimately accelerating training progress and reducing time-to-accuracy.

3.4 Analyzing the Unavailability Problem

This section provides an in-depth analysis of the end-to-end results presented in §3.2 and §3.3.

Unavailability Stalls the Aggregator. A key factor behind the lack of significant training progress for the OORT(M) baseline and the A-OORT(M) strategy in Fig. 5 is aggregator stalls. The wide distribution of these stalls is illustrated for both synthetic and MobiPerf traces in Fig. 6. In the synthetic experiment (Fig. 6a), A-OORT(0%) performs aggregations rapidly, with a median round duration of < 10 seconds, owing to its 100% trainer availability and asynchronous aggregation. In contrast, OORT(0%) experiences slightly longer round durations (10 seconds) as it waits for stragglers. However, as unavailability increases, both A-OORT(20%) and OORT(20%) frequently select clients that are either unavailable or become unavailable shortly after receiving the global model for training. This system inconsistency, *i.e.*, the aggregator waiting for an update while the trainer remains unavailable, causes stalls. These delays reach up to 100s of seconds per round in the synthetic 20% unavailability scenario, cumulatively increasing time-to-accuracy.

Moreover, in the real-world high unavailability scenario (Fig. 6b), these stalls escalate drastically to $\sim 10^4$ seconds (or up to ~ 2.8 hours), effectively halting training for OORT(M) and A-OORT(M). In contrast, their oracular counterparts perform significantly better. OORT$*$ limits its stalls to at most a few hundred seconds. These stalls are due to stragglers and clients that transiently become unavailable during training. A-OORT$*$(M) limits tail latency to approximately ~ 10 seconds by maintaining a high concurrency level from among the available trainers.

Unavailability Increases Update Staleness. Sometimes, trainers become unavailable after receiving model weights. By the time they regain availability and return updates, the global model has iterated through several versions, leading to staleness in the updates. Staleness is defined as the difference between the current global model version and the version the trainer used for training. In Fig. 7a, we observe that staleness is >10 on average for A-OORT(20%), as the strategy remains unaware of client unavailability. However, the staleness issue is mitigated in strategies with 100% and oracular availability, such as A-OORT(0%) and A-OORT$*$, where staleness is <5. The long tail for A-OORT(0%) and A-OORT$*$ is attributed to devices that temporarily become unavailable during local training.

Unavailability Exacerbates Heterogeneity Issues. Data heterogeneity causes trainers to nudge the global model towards minimizing their local loss, which can skew the model towards their specific subset of classes. Fig. 7b shows the trainer utility values observed and used over time by the A-OORT$*$ strategy on the MobiPerf availability trace. The results indicate that higher data heterogeneity leads to a higher standard deviation of trainer update utilities. This is because trainer update utilities are closely tied to their local data distributions, which become more disparate in high data heterogeneity scenarios. Moreover, with widespread unavailability as observed in the MobiPerf trace, the training process becomes further skewed in favor of the more available clients. To prevent overfitting to a subset of the data distribution, the selection strategy must consider both unavailability and data heterogeneity when making client selections.

4 Opportunities to Tackle Client Unavailability

While enhancing existing frameworks with real-time client availability tracking is essential for robust FL in high unavailability settings, it alone cannot address all the associated challenges. High client unavailability and heterogeneity introduce several issues that impact time-to-accuracy. Overall, it provides a unique opportunity to rethink not only the system design but also the selection and aggregation algorithms, to make FL more robust for real-world deployments.

Opportunity 1: Intelligent trainer selection. Our first key observation is that the selector needs additional information for improved client selection decisions. An intelligent trainer selection should not only consider current client availabilities, it should also balance it with up-to-date information about their speed, data distribution and update quality. Current frameworks [7, 11] collect such data only after training is performed, which may result in misprioritization from among the available clients, *e.g.* selecting slow or low data utility clients over ones that can expedite the training process. This highlights the need to extend client selection with a new lightweight mechanism that captures most recent client utility, along with a new algorithm that incorporates this utility when selecting trainers. By using a lightweight mechanism, the clients can reliably relay the required information even if they do not have sufficient available power or compute to train.

In current FL paradigms, there is a rigid distinction between available and unavailable clients—a client is either available to train or it is not. Since client state is a continuum w.r.t power, compute and network availability, we believe that this rigid distinction is flawed and detrimental to the model training process. It leads to *under-utilization* of clients that can be used to aid the training process, even if it is just in some limited capacity. It is known that the resource requirements for model inference are substantially lower than the full-fledged back-propagation required for training. If utilized, it could enable the low-resource clients to participate and aid FL, potentially reducing the time-to-accuracy. Thus, given prevalence of high-unavailability scenarios in real-world cross-device FL deployments, the aim should be to leverage all the clients, even the ones without capacity to train.

Opportunity 2: Effective aggregation of model updates. The second key observation is that aggregating stale updates with less penalty can sometimes benefit the global model by speeding up training. This is because the staleness of an update is not the only indicator of its informational value to the global model [9, 16, 19]. Thus, update staleness and update quality create an interesting trade-off space that is yet to be explored in AsyncFL aggregation.

Stale updates are heavily penalized in current works [17, 22]. The AsyncFL aggregation functions are parameterized by round-based staleness. A weighted average of updates is performed, where the weight of each update is inversely proportional to its staleness. However, such approaches heavily penalize updates with even moderately low staleness. For example, clients see their contributions reduced significantly (0.3× and 0.016× using `Fedbuff_poly` and `AsyncFL_hinge`, respectively, in [22]).

While it is true that highly stale updates can harm training, some client updates might still help the global model learn about unseen classes of data. This is especially true in scenarios of high data heterogeneity. Client updates from disparate data distributions are needed to make fast learning progress. Thus, there is a need for the aggregator to navigate this tradeoff space effectively, and weigh the client updates in a more nuanced manner.

Acknowledgments

This material is based upon work partially supported by the National Science Foundation (NSF) under awards CNS-2420977 and CCF-2217070, as well as a sponsored research award from Cisco Research. We would also like to sincerely thank the EuroMLSys reviewers for their insightful comments that significantly improved the quality of this paper.

References

[1] Ahmed M Abdelmoniem, Atal Narayan Sahu, Marco Canini, and Suhaib A Fahmy. 2023. Refl: Resource-efficient federated learning. In *Proceedings of the Eighteenth European Conference on Computer Systems*. 215–232.

[2] Daniel J Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Hei Li Kwing, Titouan Parcollet, Pedro PB de Gusmão, and Nicholas D Lane. 2020. Flower: A Friendly Federated Learning Research Framework. *arXiv preprint arXiv:2007.14390* (2020).

[3] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. 2019. Towards federated learning at scale: System design. *Proceedings of machine learning and systems* 1 (2019), 374–388.

[4] Zachary Charles, Zachary Garrett, Zhouyuan Huo, Sergei Shmulyian, and Virginia Smith. 2021. On large-cohort training for federated learning. *Advances in neural information processing systems* 34 (2021), 20461–20475.

[5] Harshit Daga, Jaemin Shin, Dhruv Garg, Ada Gavrilovska, Myungjin Lee, and Ramana Rao Kompella. 2023. Flame: Simplifying Topology Extension in Federated Learning. In *Proceedings of the 2023 ACM Symposium on Cloud Computing (SoCC '23)*.

[6] Chaoyang He, Songze Li, Jinhyun So, Xiao Zeng, Mi Zhang, Hongyi Wang, Xiaoyang Wang, Praneeth Vepakomma, Abhishek Singh, Hang Qiu, et al. 2020. Fedml: A research library and benchmark for federated machine learning. *arXiv preprint arXiv:2007.13518* (2020).

[7] Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, et al. 2022. Papaya: Practical, private, and scalable federated learning. *Proceedings of Machine Learning and Systems* 4 (2022), 814–832.

[8] Eunjeong Jeong, Seungeun Oh, Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. 2018. Communication-efficient on-device machine learning: Federated distillation and augmentation under non-iid private data. *arXiv preprint arXiv:1811.11479* (2018).

[9] Divyansh Jhunjhunwala, Pranay Sharma, Aushim Nagarkatti, and Gauri Joshi. 2022. Fedvarp: Tackling the variance due to partial client participation in federated learning. In *Uncertainty in Artificial Intelligence*. PMLR, 906–916.

[10] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. 2022. Fedscale: Benchmarking model and system performance of federated learning at scale. In *International conference on machine learning*. PMLR, 11814–11827.

[11] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. 2021. Oort: Efficient federated learning via guided participant selection. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*. 19–35.

[12] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. 2020. Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine* 37, 3 (2020), 50–60.

[13] Roger A Light. 2017. Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software* 2, 13 (2017), 265.

[14] M-Lab. [n. d.]. MobiPerf Data Set. https://www.measurementlab.net/tests/mobiperf/.

[15] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. PMLR, 1273–1282.

[16] Ioannis Mitliagkas, Ce Zhang, Stefan Hadjis, and Christopher Ré. 2016. Asynchrony begets momentum, with an application to deep learning. In *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*. IEEE, 997–1004.

[17] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. 2022. Federated learning with buffered asynchronous aggregation. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 3581–3607.

[18] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. 2020. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295* (2020).

[19] Angelo Rodio and Giovanni Neglia. 2024. FedStale: leveraging stale client updates in federated learning. *arXiv preprint arXiv:2405.04171* (2024).

[20] Ewen Wang, Boyi Chen, Mosharaf Chowdhury, Ajay Kannan, and Franco Liang. 2023. Flint: A platform for federated learning integration. *Proceedings of Machine Learning and Systems* 5 (2023).

[21] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K Leung, Christian Makaya, Ting He, and Kevin Chan. 2019. Adaptive federated learning in resource constrained edge computing systems. *IEEE journal on selected areas in communications* 37, 6 (2019), 1205–1221.

[22] Cong Xie, Sanmi Koyejo, and Indranil Gupta. 2019. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934* (2019).