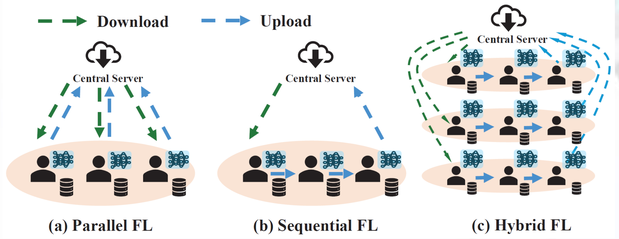
FedHusky: Accelerating Hybrid Federated Learning with Client Hopping

Fangtong Zhou, Yi Shi, Wenjing Lou, Y. Thomas Hou (Virginia Tech)

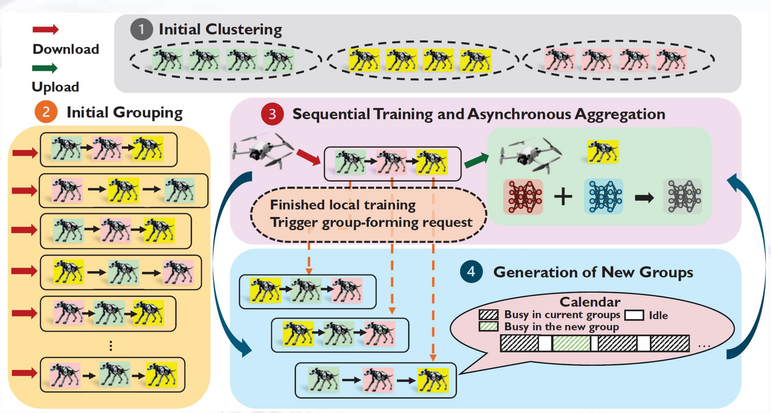
Background



- **Parallel FL**
  - **Pros**: Parallelism shorten the round duration
  - **Cons**: Performance degraded by small-scale and heterogenous clients' local datasets
- **Sequential FL**
  - **Pros**: Each client's information is captured without destructive cancellation
  - **Cons**: There is only one client busy training at any time
- **Hybrid FL**
  - **Pros**: General form for both parallel and sequential FL
  - **Cons**: Each group still inherits client idleness

Problem: Accelerate Hybrid FL

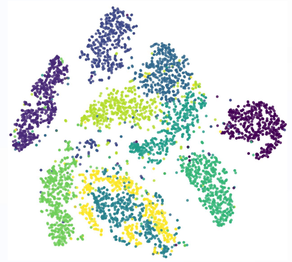
FedHusky: Overview



- **Step 1: Initial Clustering**
  - Estimates the processing time of each client
  - Groups clients into clusters based on the similarity of the locally trained models: clients in the same cluster share similar data labels
- **Step 2: Generating Initial Groups**
  - Each group contains one client from every cluster
  - Generate as many simultaneous groups as possible
- **Step 3 : Sequential Training in Each Group and Aggregation**
  - Local training in each group follows a sequential manner
  - The process stops until the last client in the group completes its training and uploads its trained model back to the server
- **Step 4 : Generation of New Groups**
  - Once a client completes its training in the group, the client will be released from the group
  - Once the last client in the group completes its training, the group ceases to exist (dies)
  - Whenever a client completes its task and is released from its current group, it triggers a new group forming request at the server
- **Step 5 : Termination**
  - The algorithm terminates once a pre-defined training time limit is reached

FedHusky: Step 1

- Determine **how many** clusters will be formed
  - Too few clusters may put dissimilar clients in the same group
  - Too many clusters may excessively fragment the data labels



Gap Statistic and **K-means algorithms**

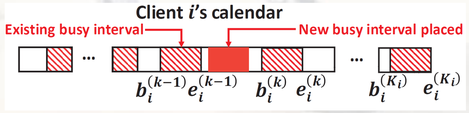
- **Rebalance Step**
  - We should have a cluster size of either $\lceil \frac{N}{C} \rceil$ or $\lfloor \frac{N}{C} \rfloor$
  - For a cluster with more than $\lceil \frac{N}{C} \rceil$ clients, the server discards the **"farthest"** clients and reassigns them to the cluster that is **"nearest"** that currently has fewer clients than $\lfloor \frac{N}{C} \rfloor$

FedHusky: Step 2

- **Goal**: Form as many groups as possible by allowing a client to participate in as many groups as possible, if it is **not double-booked** at the same time

The server iteratively considers each client from 1 to N as the first member of a potential group

- Generate the **FIRST** group
  - First spot (client#1), rest spots (Rest $C-1$ clusters randomly pick 1 for each)
  - Each client maintains a **calendar** $\mathcal{B}_i = \{[b_i^{(k)}, e_i^{(k)}], k = 1,2,..\}$ recording its busy interval
- Generate the **SECOND** group
  - First spot (client#2)
  - Rest spots: Need to model an optimization problem to ensure no double-booking
  - Decision variables
    - **Client selection**: x_i, **position placement**: $y_{i\mapsto p}$, **starting time of position** p: $h[p]$, **starting time of client** i: s_i, **calendar gap insertion**: $z_i^{(k)}$
  - Constraints
    - Each cluster selects one client, each position places one client, each client placed at one spot, new busy interval can be inserted into one gap in calendar
  - Objective
    - Define a busy ratio of client i: θ_i. If i is not selected in group 1: $\theta_i = 0$. If i is selected in group 1: $\theta_i = (e_i^{(1)} - b_i^{(1)})/T$
    - OPT-2: $\max \min \theta_i + \Delta_i x_i / T$
  - After solving, selected clients for group 2 update each of their calendars
- Generate the ***i*-th** group
  - Solve OPT-i
  - For OPT-i
    - If optimal: update calendars, group formed
    - If infeasible: Move on to OPT-$(i+1)$
  - After solving OPT-N, step 2 completed



FedHusky: Step 3

- **Sequential Training**

Server transmits the global to the first client in the group → After finished training, the first client passes the model to the next client → Last client completes its training and uploads the final model back to the server

  - Wireless relay
    - Ad Hoc networking or Use server as a relay, whichever is faster
- **Asynchronous Aggregation**
  - Whenever the last client in a group completes its training, it sends its model to the server for aggregation
  - Use an aggregation coefficient so that larger gap, smaller aggregation weight

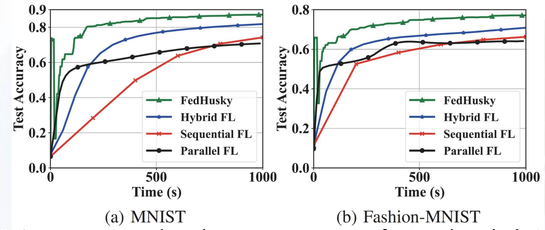
FedHusky: Step 4

- Each generated group is used only once
  - When a client finishes training and relaying, it is released
  - When the last client finishes, the group dies
- Solving new optimization problem OPT-k, $k>N$
  - No need the specify the first position of the group: ~~$x_2 = 1$~~, ~~$y_{2\mapsto 1}$~~
  - t_0 is shifted to the current wall-clock time: $h[1] = t_0$
  - If there is a feasible solution of OPT-k, a new group formed, otherwise, no new group
- Server waits until the next client notification arrives and then starts the same process for OPT-$(k+1)$

Publication

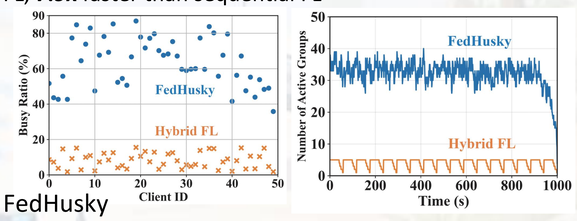
F. Zhou, Y. Shi, W. Lou, and Y. T. Hou, "FedHusky: Accelerating Hybrid Federated Learning with Client Hopping," in Proc. International Symposium on Modeling and Optimization in Mobile, Adhoc, and Wireless Networks (WiOpt), Columbus, Ohio, USA, June 2026.

Results: Case Study



(a) MNIST (b) Fashion-MNIST

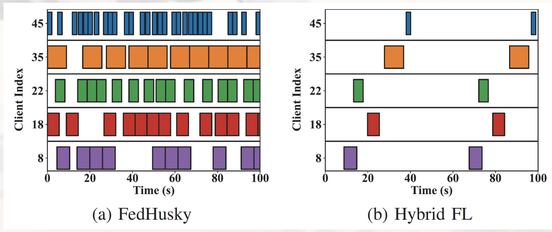
- 80% accuracy: FedHusky converges **5.9x** faster than hybrid FL, **7.9x** faster than sequential FL



- FedHusky
  - Busy ratio varies from 35.8% to 86.9% over the 50 clients, with the average ratio being 63.4%

6.7x Acceleration

- Hybrid FL
  - Busy ratio varies from 1.8% to 15.5%, with the average ratio being 8.2%



(a) FedHusky (b) Hybrid FL

- Examine the calendar of each individual client
  - Each client under FedHusky clearly **works much harder** (more efficiently) than it works under hybrid FL