Optimizing Asynchronous Federated Learning: A Delicate Trade-Off Between Model-Parameter Staleness and Update Frequency

Abdelkrim Alahyane, Céline Comte, Matthieu Jonckheere, Éric Moulines

Optimizing Asynchronous Federated Learning: A Delicate Trade-Off Between Model-Parameter Staleness and Update Frequency

Abdelkrim Alahyane[a,b,*], **Céline Comte**[b], **Matthieu Jonckheere**[b] **and Éric Moulines**[c]

[a]EMINES, Mohammed VI Polytechnic University, Ben Guerir, Morocco
[b]LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
[c]Ecole Polytechnique, Palaiseau, France

Abstract. Synchronous federated learning (FL) scales poorly with the number of clients due to the straggler effect. Algorithms like `FedAsync` and `GeneralizedFedAsync` address this limitation by enabling asynchronous communication between clients and the central server. In this work, we rely on stochastic modeling and analysis to better understand the impact of design choices in asynchronous FL algorithms, such as the concurrency level and routing probabilities, and we leverage this knowledge to optimize loss. Compared to most existing studies, we account for the joint impact of heterogeneous and variable service speeds and heterogeneous datasets at the clients. We characterize in particular a fundamental trade-off for optimizing asynchronous FL: minimizing gradient estimation errors by avoiding model parameter staleness, while also speeding up the system by increasing the throughput of model updates. Our two main contributions can be summarized as follows. First, we prove a discrete variant of Little's law to derive a closed-form expression for relative delay, a metric that quantifies staleness. This allows us to efficiently minimize the average loss per model update, which has been the gold standard in literature to date, using the upper-bound of Leconte et al. as a proxy. Second, we observe that naively optimizing this metric drastically slows down the system by overemphasizing staleness at the expense of throughput. This motivates us to introduce an alternative metric that also accounts for speed, for which we derive a tractable upper-bound that can be minimized numerically. Extensive numerical results show these optimizations enhance accuracy by 10% to 30%.

1 INTRODUCTION

Federated learning (FL) is a learning paradigm that enables distributed model training across multiple clients under the supervision of a central server (CS), without requiring data sharing [19, 27]. In synchronous FL, stochastic gradient descent (SGD) is executed in rounds; during each round, the CS sends the current model parameters to a subset of clients, waits until all of them return a new stochastic gradient estimate, and then updates the model parameters before sending them to another batch of clients. Unfortunately, the performance of synchronous FL is often hindered by the variability of computational speeds among clients, leading to the straggler effect [9].

Asynchronous algorithms such as `FedAsync` [37, 8, 38], `FedBuff` [29], `AsGrad` [14], and `AsyncSGD` [18, Algorithm 2] aim to tackle these challenges by allowing clients and the CS to communicate asynchronously. In particular, the CS may update the model parameters while clients estimate gradients (possibly based on outdated model parameters). Asynchrony is implemented by allowing tasks (i.e., requests for gradient estimates) to be queued at the clients [18]. Intuitively, asynchronous FL has the potential to speed up the system by circumventing the straggler effect, possibly at the cost of errors due to the staleness of the model parameters used to estimate gradients.

Consequently, a significant research effort has been devoted to analyze the performance of asynchronous FL in both homogeneous and heterogeneous data settings [8, 9, 38, 18]. However, as we will review in Section 1.1, most existing studies overlook the critical impact of queueing dynamics on the actual performance of asynchronous FL. In contrast, [21] recently showed that the performance of asynchronous FL depends critically on these queueing dynamics via the staleness of the model parameters used by clients to estimate gradients. Staleness is captured by the *relative delay*, defined as the (stochastic) number of times the CS updates the model parameters while a gradient-estimation task is being held at a client (either queued or being processed). While existing analyses assume this relative delay is bounded irrespective of the system parameters, the results of [21] imply that the relative delay depends heavily on these parameters and may grow arbitrarily large even in realistic scenarios. This stands in sharp contrast to existing studies such as [34, 35, 25, 26].

In this paper, we derive fundamental insights and actionable tools for optimizing performance in asynchronous FL. Unlike most works from the literature, our results apply to systems where clients are *heterogeneous*, both in terms of computation speeds and of datasets. First, building on [21], we derive an explicit expression for the mean relative delay and its gradient by leveraging the framework of Jackson networks [15], which in turn allows us to design a gradient-descent algorithm that optimizes performance.

Next, we observe that minimizing the average gradient norm per update, as commonly done in the literature, can be counterproductive in asynchronous FL, as it ignores update throughput and slows the system to the pace of the slowest client, thereby underutilizing the others. Although we focus on an extension of `AsyncSGD` called `Generalized AsyncSGD`, we believe our results are relevant to other asynchronous FL algorithms.

* Corresponding Author. Email: abdelkrim.alahyane@um6p.ma.

1.1 Related work

Synchronous FL has major pitfalls [36, 30, 23, 24, 33]: the straggler effect leads to important delays, and synchronization becomes challenging when the number of clients increases [37]. These limitations spurred the development of asynchronous FL algorithms, such as `FedAsync` [37, 8, 38], `FedBuff` [29], `AsGrad` [14], and `AsyncSGD` [18, Algorithm 2], which incorporate memory-based updates, adaptive learning rate adjustments, and strategies to reduce staleness and handle varying computation speeds. [28, 18, 11] showed in a simplified model that Asynchrounous SGD is provably faster in terms of wall-clock time than Minibatch SGD.

Despite these advances and the plethora of asynchronous FL algorithms that have been proposed, there is still little understanding of the impact of system design on performance. In real-world distributed learning, compute times are unpredictable and heterogeneous due to hardware failures, preemptions, GPU delays, and network issues, making fixed-time assumptions unrealistic [13, 7, 16]. Instead, compute times should be treated as dynamic client-dependent random variables. Furthermore, in many applications, datasets are heterogeneous across clients. In contract, as we see now, most existing analyses fail to provide solutions that work well when clients are heterogeneous both in terms of compute speeds and datasets.

Several works have attempted to account for client heterogeneity in asynchronous FL. The analysis of the celebrated `FedBuff` algorithm [29] allows for heterogeneous datasets, but it assumes that the next client completing a gradient computation is sampled uniformly at random, which is especially unrealistic when service speeds are heterogeneous. [1] designs an asynchronous FL algorithm robust to dataset heterogeneity and variability in compute times, but it assumes that compute times are sampled from the same distribution for all clients, so that it does not accommodate heterogeneity of client speeds. [34, 35, 25, 26, 11] attempt to model client speed heterogeneity by allowing for fixed but heterogeneous delays. However, their approach enforces synchronous updates by setting a time threshold and discarding clients that exceed it. This biases the global model against underrepresented data and wastes near-complete computations, reducing overall system efficiency. Furthermore, by assuming that delay is fixed, these works implicitly assume that the system parameters have a bounded impact on the delay, which is no longer true when queueing dynamics are taken into account. [18] explicitly accounts for dataset heterogeneity, but it again makes the unrealistic assumption that delay is bounded irrespective of the system parameters.

This discussion reveals a major gap in the literature: existing analyses fail to fully capture the interplay between, on the one hand, the unpredictable and heterogeneous nature of client speeds and network conditions, and, on the other hand, heterogeneous datasets. This is all the more critical since these properties motivated the introduction of asynchronous FL in the first place.

A preliminary attempt is made in [21], which introduces `Generalized AsyncSGD`, an algorithm that utilizes non-uniform client selection to address queueing dynamics, heterogeneous client speeds, and heterogeneous datasets. However, their analysis falls short of providing explicit performance bounds, instead relying on scaling regimes to approximate the system behavior.

1.2 Contributions

Our findings stem from key theoretical insights that allow an explicit characterization of the impact of queueing dynamics on the performance of asynchronous FL. They can be summarized as follows:

Derive tractable bounds on loss gradients. Using the framework of queueing theory, we derive exact and tractable formulas for the mean relative delay and its gradient. In general, these formulas can be estimated in time $\mathcal{O}(n^2m^2)$, where n is the number of clients and m the concurrency level, or they can be estimated through Monte Carlo simulations. Further simplifications allow us to compute them in time $\mathcal{O}(n)$ for simple routing strategies.

Optimize performance. Leveraging this result, we design an algorithm that improves the performance of `Generalized AsyncSGD` by using the bound from [21] as a proxy objective for routing optimization. Our result also allows us to gauge the bound's sensitivity to crucial system parameters, such as the ratio of the concurrency level to the number of clients, and the clients' service speeds. These insights are relevant to other asynchronous FL algorithms and underscore that routing strategies should be adapted based on application-specific bottlenecks.

Account for clock-time performance. We observe both analytically and numerically that minimizing the average norm-square of the gradient per update actually leads us to slow down the system considerably by underutilizing all clients but (the slowest) one. Roughly speaking, since this metric ignores the throughput of model updates, it is optimized by minimizing staleness, which is achieved by giving priority to the slowest client. This motivates us to introduce an alternative metric that explicitly accounts for throughput, and for which we derive an upper bound that can again be optimized tractably.

Our findings provide not only qualitative insights into the impact of queueing dynamics, but also efficient numerical methods to optimize performance. We show in particular that routing strategies should be adapted based on the specific bottlenecks imposed by applications, such as the number of computation rounds versus actual training time. Our experiments on real-world datasets show that tuning the routing strategy and/or the concurrency level can improve accuracy by 10% to 30%.

1.3 Notations

$\mathbb{Z}, \mathbb{N}, \mathbb{N}_{>0}, \mathbb{R}, \mathbb{R}_{\geq 0}, \mathbb{R}_{>0}$ denote the sets of integers, non-negative integers, positive integers, real numbers, non-negative real numbers, and positive real numbers. Let $|\cdot|$ denote the ℓ_1-norm and $\mathbf{1}[\cdot]$ the indicator function. For each $n, m \in \mathbb{N}_{>0}$, let $\mathcal{X}_{n,m} = \{x \in \mathbb{N}^n : |x| = m\}$ denote the set of n-dimensional natural-number-valued vectors with ℓ_1-norm m. For every $n \in \mathbb{N}_{>0}$, let $\mathcal{P}_n = \{p \in \mathbb{R}^n : 0 < p_i < 1 \text{ for } i \in \{1, 2, \ldots, n\} \text{ and } |p| = 1\}$.

2 MODEL AND PRIOR RESULTS

2.1 Asynchronous federated learning

The goal in federated learning (FL) is to optimize the average performance of a model across multiple clients under the supervision of a central server (CS): $\min_{w \in \mathbb{R}^d} f(w)$, where $f(w) = \frac{1}{n}\sum_{i=1}^n f_i(w)$, and

$$f_i(w) = \mathbb{E}_{(x,y)\sim\mathcal{D}_i}[\ell_i(\mathrm{NN}(x, w), y)], \quad i \in \{1, 2, \ldots, n\}.$$

Here, w denotes the parameters of a deep neural network, d the number of parameters (including weights and biases), $\mathrm{NN}(x, w)$ the prediction function of the neural network, n the number of clients, ℓ_i the local loss function of client i, and $\mathcal{D}_i$ the data distribution at client i. Each client i approximates the gradient $\nabla_w f_i(w)$ of its local loss function using a stochastic gradient denoted by $g_i(w)$. The computation of such a stochastic gradient by a client is called a *task*.

Algorithm 1 `Generalized AsyncSGD (CS)`

1: **Input:** Numbers T, n, and m of rounds, clients, and tasks; routing p; learning rate η
2: Initialize parameters w_0 randomly
3: Initialize state vector $\xi \in \mathcal{X}_{n,m}$ randomly
4: **for** $i = 1, 2, \ldots, n$ **do**
5: Send ξ_i times model parameter w_0 to client i
6: **end for**
7: **for** $t = 0, \ldots, T$ **do**
8: CS receives stochastic gradient $g_{C_t}(w_{I_t})$ from a client C_t
9: Update $w_{t+1} \leftarrow w_t - \frac{\eta}{np_{C_t}} g_{C_t}(w_{I_t})$
10: Sample a new client A_{t+1} according to p
11: Send model parameter w_{t+1} to client A_{t+1}
12: **end for**

Algorithm 2 `Generalized AsyncSGD (Client i)`

1: **Input:** Queue of received parameters, local dataset
2: **if** Queue is not empty **then**
3: Take the received parameter w from the queue using a FIFO policy
4: Compute the gradient estimate $g_i(w)$
5: Send the gradient to the CS
6: Repeat
7: **end if**

`Generalized AsyncSGD` [21] is shown in Algorithms 1 (CS) and 2 (client i). A task assigned to a busy client is queued according to the first-in-first-out (FIFO) policy. As we will see, performance depends critically on two parameters: p, the routing probability vector of tasks to clients; and m, the concurrency [18], defined as the number of tasks that are concurrently dispatched to the clients (either queued or being processed). `Generalized AsyncSGD` simplifies to the classical `AsyncSGD` algorithm [18, Algorithm 2] when $p = p^{\text{uniform}}$ with $p_i^{\text{uniform}} = \frac{1}{n}$ for each $i \in \{1, 2, \ldots, n\}$.

Let us focus on Algorithm 1 (CS perspective). The model parameter is initialized to a random vector w_0, and the system state is initialized to a random vector $\xi = (\xi_1, \xi_2, \ldots, \xi_n) \in \mathcal{X}_{n,m}$, where ξ_i is the number of tasks initially dispatched to client i, for each $i \in \{1, 2, \ldots, n\}$. After this initialization, each iteration $t \in \{0, 1, \ldots, T\}$ of the `for` loop (Line 7) proceeds as follows: whenever a client C_t completes a task and reports a gradient estimate $g_{C_t}(w_{I_t})$ (Line 8; I_t will be defined shortly), the CS immediately updates the model parameter (Line 9) and sends it to the next client A_{t+1}, where $\mathbb{P}(A_{t+1} = i) = p_i$ for each $i \in \{1, 2, \ldots, n\}$ (Lines 10 and 11). Recall that a client can be chosen even if it is processing a task. Observe that the step size in the model-parameter update step is divided by the routing probability to avoid biasing the model towards clients that are sampled more often.

Time indices are such that, for each $t \in \{1, 2, \ldots, T\}$, A_t and C_t correspond to the same (model-parameter update) round, where a round is defined as the time between the assignment of a task to a client and the next task completion (leading to a model update). In Section 4, we will see that throughput, defined as the inverse of the (clock-time) duration of a typical round, has a crucial impact on performance. For each $t \in \{0, 1, \ldots, T\}$, we let $X_t = (X_{1,t}, X_{2,t}, \ldots, X_{n,t}) \in \mathcal{X}_{n,m-1}$ denote the state at the end of round t, so that, for each $i \in \{1, 2, \ldots, n\}$, we have $X_{i,0} = \xi_i - \mathbf{1}[C_0 = i]$ and, for each $t \in \{1, 2, \ldots, T\}$,

$$X_{i,t} = X_{i,t-1} + \mathbf{1}[A_t = i] - \mathbf{1}[C_t = i]. \tag{1}$$

The processing times of successive tasks at client i are assumed to be independent and exponentially distributed with rate $\mu_i > 0$, for each $i \in \{1, 2, \ldots, n\}$. In particular, for each $t \in \{1, 2, \ldots, T\}$, we have $\mathbb{P}[C_t = i | X_{t-1}, A_t] \propto \mu_i$ for all $i \in \{1, 2, \ldots, n\}$ such that $X_{i,t-1} + \mathbf{1}[A_t = i] > 0$. As we will show in later sections, this assumption enables the derivation of tractable performance bounds while still capturing queueing dynamics, phenomena that were not addressed in prior work before [21].

Critically, the gradient estimate $g_{C_t}(w_{I_t})$ returned at the end of an iteration t is based on the model parameters w_{I_t} known to the CS at the beginning of round $I_t = \sum_{s=0}^{t} s\mathbf{1}[s + D_{A_s,s} = t]$ when the task was assigned to client C_t, where $D_{i,t}$ is called the *relative delay* and is defined as the number of rounds that get completed during the sojourn of a task at a client:

$$D_{i,t} = \mathbf{1}[A_t = i] R_{i,t}, \text{ where} \tag{2}$$

$$R_{i,t} = \min \left\{ r \in \mathbb{N} : \sum_{s=t}^{t+r} \mathbf{1}[C_s = i] = X_{i,t-1} + \mathbf{1}[A_t = i] \right\}. \tag{3}$$

Consistent with the literature on decentralized learning, we make the following assumptions: a uniform lower bound f^* on the objective function f; L-Lipschitz continuity of the gradients ∇f_i to ensure smoothness ($\|\nabla f_i(w) - \nabla f_i(\mu)\| \leq L\|w - \mu\|$); an upper bound σ^2 on the variance of the stochastic gradients ($\mathbb{E}[\|g_i(w) - \nabla f_i(w)\|^2] \leq \sigma^2$); and an upper bound M^2 on the gradient dissimilarity across clients ($\|\nabla f(w) - \nabla f_i(w)\|^2 \leq M^2$). These are detailed in Assumptions **A1**–**A4** in Section A of the supplementary material [2].

2.2 *Queueing dynamics*

The next result is a variant of [21, Proposition 2] and will be instrumental throughout the paper. It relates the queueing dynamics to the routing and service-rate vectors $p = (p_1, p_2, \ldots, p_n)$ and $\mu = (\mu_1, \mu_2, \ldots, \mu_n)$. Recall that the state space is $\mathcal{X}_{n,m-1}$ (and not $\mathcal{X}_{n,m}$) because we consider model-parameter-update times. Buzen's algorithm was introduced in [6].

Proposition 2.1. *In the framework of Section 2.1, the sequence $(X_t, t \in \mathbb{N})$ defines an irreducible positive recurrent Markov chain with stationary distribution*

$$\pi_{n,m-1}(x) = \frac{1}{Z_{n,m-1}} \prod_{i=1}^{n} \left(\frac{p_i}{\mu_i} \right)^{x_i}, \; x \in \mathcal{X}_{n,m-1}, \tag{4}$$

where the normalizing constant $Z_{n,m-1}$ can be computed by applying Buzen's recursive algorithm:

- $Z_{\mathfrak{n},0} = 1$ *for each* $\mathfrak{n} \in \{1, 2, \ldots, n\}$,
- $Z_{1,\mathfrak{m}} = (\frac{p_1}{\mu_1})^{\mathfrak{m}}$ *for each* $\mathfrak{m} \in \{0, 1, 2, \ldots, m\}$,
- $Z_{\mathfrak{n},\mathfrak{m}} = Z_{\mathfrak{n}-1,\mathfrak{m}} + \frac{p_{\mathfrak{n}}}{\mu_{\mathfrak{n}}} Z_{\mathfrak{n},\mathfrak{m}-1}$ *for* $\mathfrak{n} \in \{2, 3, \ldots, n\}$ *and* $\mathfrak{m} \in \{1, 2, \ldots, m\}$.

Proof. See Section B in the supplementary material [2]. □

In the rest of the paper, we will assume that the system starts in steady state. This assumption is reasonable when the total number T of updates is sufficiently large, as the distribution of X_t converges exponentially fast with t towards the stationary distribution regardless of the initial distribution (see Theorem 13.4.14 in [4]). Concretely, we will assume that $X \triangleq X_0$ follows the stationary distribution $\pi_{n,m-1}$, and we will often drop the time index, so that for instance D_i will be a random variable distributed like $D_{i,t}$ for any $t \in \mathbb{N}_{>0}$.

3 OPTIMIZE MODEL UPDATES

Consistently with the literature on asynchronous FL, in this section, we assume the CS has a fixed budget T of model parameter updates, and we try to make the best of these updates. This is particularly relevant in contexts, such as cellular networks with costly data plans or satellite internet services, where data transmission costs are high.

3.1 Bound, delay, and gradient descent

[21, Theorem 1] gave the following upper bound on the ergodic mean of the norm-square of the gradient of f: there exists $\eta_{\max} > 0$ dependent on p [1] such that, for any $\eta \in (0, \eta_{\max})$,

$$\frac{1}{T+1}\sum_{t=0}^{T}\mathbb{E}[\|\nabla f(w_t)\|^2] \leq 8G, \text{ where}$$

$$G = \frac{A}{\eta(T+1)} + \frac{\eta LB}{n^2}\sum_{i=1}^{n}\frac{1}{p_i} + \frac{\eta^2 L^2 Bm}{n^2}\sum_{i=1}^{n}\frac{\mathbb{E}[D_i]}{p_i^2}, \quad (5)$$

with $A = f(w_0) - f^*$ and $B = \sigma^2 + 2M^2$ (see Section A of the supplementary material [2] for details on these constants). If needed, the dependency of G on the routing vector p will be made explicit by writing $G(p)$. In the remainder, we use G as a proxy objective to minimize the ergodic norm-squared gradient of f.

The first term in G follows a classical pattern, capturing the influence of initialization on convergence. The next two terms reflect the effects of stochastic gradient estimation (through σ) and data heterogeneity across clients (through M). The second term also captures the variance of gradient updates induced by the routing-dependent learning rate and is minimized by uniform routing. Anticipating the discussion in Section 3.2, the third term quantifies gradient staleness via the relative delay $\mathbb{E}[D_i]$. The last two terms in G depend heavily on the routing vector p and the service rate vector μ, both explicitly and implicitly through the mean relative delays $\mathbb{E}[D_i]$.

A priori, expected relative delays are complex functions of the system dynamics, as a task's delay may depend on an unbounded number of future rounds. Theorem 3.1, our first main contribution, bypasses this difficulty by expressing them as functions of the mean numbers of stationary tasks, which, in turn, allows us to derive closed-form expressions for the mean relative delays and their gradient.

Theorem 3.1. *In the framework of Section 2.1, we have*

$$\mathbb{E}[D_i] = \mathbb{E}[X_i], \quad i \in \{1, 2, \ldots, n\}, \quad (6)$$

$$\frac{\partial \mathbb{E}[D_i]}{\partial p_j} = \frac{1}{p_j}\operatorname{Cov}[X_i, X_j], \quad i, j \in \{1, 2, \ldots, n\}, \quad (7)$$

where for each $i, j \in \{1, 2, \ldots, n\}$,

$$\mathbb{E}[X_i] = \sum_{k=1}^{m-1}\left(\frac{p_i}{\mu_i}\right)^k \frac{Z_{n,m-1-k}}{Z_{n,m-1}}, \quad (8)$$

$$\mathbb{E}[X_i X_j] = \sum_{\substack{k,\ell=1 \\ k+\ell\leq m-1}}^{m-1}\left(\frac{p_i}{\mu_i}\right)^k\left(\frac{p_j}{\mu_j}\right)^\ell \frac{Z_{n,m-1-k-\ell}}{Z_{n,m-1}}, \quad (9)$$

and the constants $Z_{n,\mathfrak{m}}$ *for* $\mathfrak{m} \in \{0, 1, \ldots, m-1\}$ *are computed as in Proposition 2.1.*

[1] The original proof of the bound G in [21] incorrectly assumes independence between $D_{i,k}$ and $\nabla f(w_k)$ for all $k \in \{1, \ldots, T\}$, which does not generally hold in asynchronous settings. We provide a correction based on a stochastic bound in Section C of the supplementary material [2], which yields a revised expression for $\eta_{\max}$.

Proof. The proof of Theorem 3.1 is in Section D of the supplementary material [2]. Here we briefly give the intuition behind (6), which is analogous to Little's law. If each task at client i pays \$1 each time a task gets completed at another client, there are two ways of collecting payments: either we receive an upfront payment of $\$R_i$ when a task is assigned to client i (yielding $\mathbb{E}[D_i]$), or we earn $\$X_i$ each time a task gets completed at another client (yielding $\mathbb{E}[X_i]$). Equations (7)–(9) follow by direct computations, after observing that the distribution (4) is an exponential family. □

In the remainder, we will apply (6)–(7) to compute $\mathbb{E}[D_i]$ and $\nabla_p \mathbb{E}[D_i]$. However, these equations can also be used to estimate these quantities through Monte Carlo simulations. Section G of the supplementary material [2] gives a gradient-descent algorithm to optimize the routing vector p in view of minimizing G, which will be applied in Section 3.3 to run extensive numerical results.

3.2 Discussion

Theorem 3.1 allows us to gain insight into the impact on G of the system parameters.

How to minimize G? One can verify that the second term in G is minimized by applying the uniform routing p^{uniform}, given by $p_i^{\text{uniform}} = \frac{1}{n}$ for each $i \in \{1, 2, \ldots, n\}$. Figure 1 shows that minimizing the third term in G, involving the mean relative delays, is more challenging: even in a toy 2-client example, the third term is non-monotonic and is minimized by assigning almost all tasks to the slowest client.

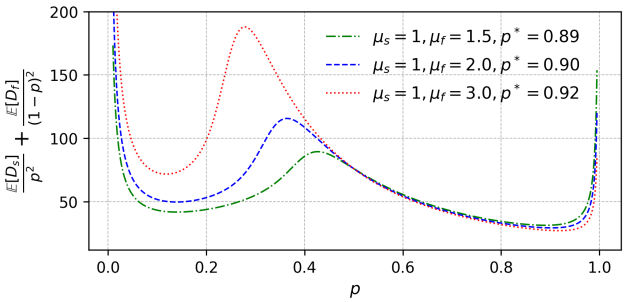

Figure 1. Third term of the bound G given in (5) vs. the routing probability to the slowest client, in a toy example with $n = 2$ clients and $m = 20$ tasks, for various speed vectors $\mu = (\mu_s, \mu_f)$.

To gain more insight into this third term, observe that Equation (6) from Theorem 3.1 yields the following simple relation between the mean relative delays:

$$\sum_{i=1}^{n}\mathbb{E}[D_i] = \sum_{i=1}^{n}\mathbb{E}[X_i] = m - 1. \quad (10)$$

This relation has several consequences, in particular: (i) the sum of the mean relative delays depends only on the numbers n and m of clients and tasks, while p and μ impact only how the relative delay is distributed across clients; (ii) decreasing the relative delay at a client necessarily comes at the cost of an increased relative delay at another client; and (iii) since the routing probabilities p_i and relative delays $\mathbb{E}[D_i]$ both have constant sums over the clients i, minimizing the third term in G requires finding a vector p so that a client i with a high relative delay $\mathbb{E}[D_i]$ also has a relatively large routing probability p_i.

Dependency on the number m of tasks Another consequence of Theorem 3.1 is that the bound G is an increasing function of the number m of tasks (by combining (6) with the observation that $\mathbb{E}[X_i]$ is a non-decreasing function of m [32, Lemma 2]). In particular, keeping all other system parameters fixed, the performance is optimized when only $m = 1$ task circulates in the network! In this case, (10) implies that the third term in G is equal to zero. This is intuitive because, with a single task circulating in the network, the staleness issue is trivially eliminated, and the system works like a synchronous FL system in which the CS samples a single client at each round. This observation motivates the alternative metric we introduce in Section 4.

Simple routing strategies Our result allows us to simplify the bound G for two noteworthy routing vectors, that serve as baselines in our experiments. First, under uniform routing p^{uniform}, (10) yields

$$G(p^{\text{uniform}}) = \frac{A}{\eta(T+1)} + \eta LB + \eta^2 L^2 Bm(m-1).$$

Note that `Generalized AsyncSGD` with uniform routing reduces to the standard `AsyncSGD` algorithm.

Another routing strategy that admits explicit analysis is the *balanced* routing vector $p^{\propto\mu}$, defined by $p_i^{\propto\mu} = \frac{\mu_i}{\sum_j \mu_j}$, i.e., each client is selected with probability proportional to its service speed. This strategy is well-known in queueing theory for "balancing the load" across clients, in the sense that $\mathbb{E}[D_i] = \mathbb{E}[X_i] = (m-1)/n$ for all $i \in \{1, 2, \ldots, n\}$ (see (4), (6), and (10)). Under this policy, all clients experience the same average relative delay, regardless of individual speeds. Moreover, this strategy is commonly used as a heuristic to maximize the throughput of closed Jackson networks. It follows that

$$G(p^{\propto\mu}) = \frac{A}{\eta(T+1)} + \frac{\eta LB|\mu|}{n^2} \sum_{i=1}^{n} \frac{1}{\mu_i} \left(1 + \frac{\eta Lm(m-1)|\mu|}{n\mu_i}\right).$$

3.3 Numerical results

In this section, we numerically evaluate the impact of the optimizations proposed in Section 3.1 on the accuracy and loss performance of `Generalized AsyncSGD` across several image classification tasks. The objective function G is used as a proxy to minimize the average squared norm of the gradient of f per update round. We consider mainly two data-distribution scenarios:

Homogeneous: Data is distributed independently and identically across clients, and each client receives an equal number of data points from each class.

Heterogeneous: Datasets are heterogeneous across clients, both in terms of distribution and volume. For each class k, we sample a vector $q_k \sim \text{Dir}_n(0.5)$, where $q_{k,j}$ is the proportion of class-k instances allocated to client j and $\text{Dir}_n(\beta)$ the Dirichlet distribution with dimension n and concentration parameter $\beta > 0$ [22].

Given a system as described in Section 2.1, the G-optimized routing vector p_G^* is computed by minimizing G using the Adam gradient-descent algorithm with standard hyperparameters from the literature [17], initialized with the uniform routing vector p^{uniform}. Since G is non-convex, the resulting routing vector may be a local minimum. The gradient of G is computed in closed form using the results of Theorem 3.1, as detailed in Section G of the supplementary material [2].

Given a routing vector p, we simulate the system of Section 2.1 and evaluate `Generalized AsyncSGD` on image classification tasks using the Fashion-MNIST [12], CIFAR-10, and CIFAR-100 [20] datasets. We employ the standard multi-class cross-entropy loss and evaluate performance on an unseen, label-balanced test dataset. Additional experimental details are provided in Section I of the supplementary material [2].

We consider a network of $n = 20$ clients managing $m = 100$ tasks. For each $i \in \{1, \ldots, 20\}$, the service speed of client i is set to $\mu_i = \exp(i/100)$, such that the fastest client is approximately 20% faster than the slowest. The learning rate is $\eta = 0.01$, and $L = 1$. As discussed in Section 3.2, this high-concurrency setting poses a significant challenge for minimizing gradient staleness. To assess the robustness of p_G^* to the stationarity assumption, we initialized the system out of stationarity by assigning $\frac{m}{n}$ tasks to each client instead of sampling the initial state from the stationary distribution. We compare the performance of `Generalized AsyncSGD` under the optimized routing vector p_G^* with two baselines: (i) `AsyncSGD`, corresponding to `Generalized AsyncSGD` with the uniform routing vector p^{uniform}; (ii) `Generalized AsyncSGD` with the balanced routing vector $p^{\propto\mu}$ described in the previous section.

Optimizing G. The optimized vector p_G^* obtained by minimizing G selects the slowest client over 40% of the time, while the routing probabilities to faster clients are all of the same magnitude but decrease as the client speed increases. This result is consistent with [21, Section 5] but counterintuitive *a priori*. Such a skewed routing vector is obtained because it significantly reduces the third term of G (see Section 3.2). Intuitively, p_G^* synchronizes the system to the pace of the slowest client, while routing tasks to faster clients in inverse proportion to their speeds, to reduce errors caused by stale gradients.

Additional numerical results (not shown here) reveal that the skewness of the optimized routing vector p_G^* towards the slowest client is accentuated when the concurrency m is large, as in the scenario we consider here. More specifically, the vector p_G^* is closer to the uniform vector p^{uniform} when m is small relative to the number n of clients, while the routing probability to the slowest client rises significantly as m increases. This is in line with Section 3.2, where we observed that the relative weight of third term of G tends to increase with m.

Performance on datasets. Figure 2 shows that `Generalized AsyncSGD` with the optimized routing strategy p_G^* consistently outperforms the baseline methods, namely, `AsyncSGD` and the balanced routing strategy $p^{\propto\mu}$, across all experiments and throughout the learning process. Notably, although assigning a higher routing probability to slower clients might seem to skew the model toward their local data distributions, our simulations show that p_G^* achieves both superior and more stable performance. This robustness is partly due to the use of an adaptive learning rate, which ensures that gradient updates remain unbiased despite routing asymmetries. In contrast, the standard deviation of the loss is significantly higher under uniform and balanced strategies, which we attribute to increased gradient staleness.

Additional experiments in Section J.1.1 of the supplementary material [2] confirm that the results hold under more heterogeneous data distributions, where clients have highly imbalanced and disjoint label sets, i.e., each client's dataset contains only a subset of image labels, possibly disjoint from those of other clients. Further experiments with deterministic and lognormal (heavy-tailed) computation times, reported in Section J.1.2 of the supplementary material [2], show similar performance trends, indicating robustness to the assumption of exponentially distributed computation times.

Looking ahead to Section 4, note that the performance advantage of p_G^* is specific to the round-based metric used in Figure 2. If the x-axis represented wall-clock time instead of update rounds, the ranking would be reversed. As discussed in Section 3.2, minimizing G reduces staleness by favoring slower clients, but comes at the cost of lower throughput. Concretely, in Figure 2, the average wall-clock

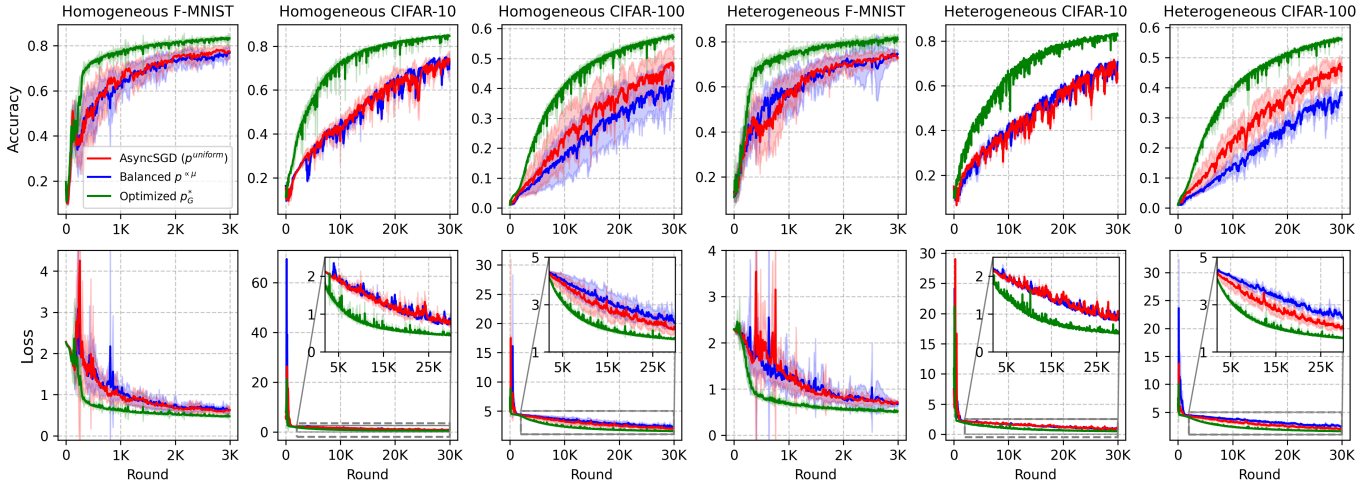

Figure 2. Performance on the test set at the CS in the scenario of Section 3.3, with $n = 20$ clients and $m = 100$ tasks, under homogeneous and heterogeneous data distributions. Solid lines show averages over independent runs; shaded areas denote standard deviations. For Fashion-MNIST, we ran 10 simulations of 3,000 rounds, recording accuracy and loss every 5 rounds. For CIFAR-10 and CIFAR-100, we applied standard normalization and data augmentation, and ran 3 simulations of 30,000 rounds, logging performance every 50 rounds. All routing strategies use the same model initialization.

time to complete 3,000 update rounds was 7 to 8 times larger under p_G^* compared to p^{uniform} and $p^{\propto\mu}$. We believe this trade-off may be prohibitive in practice, even when wall-clock time is not the primary concern. This motivates the development of alternative metrics that better balance staleness reduction and update frequency to optimize convergence in terms of wall-clock time rather than round count.

4 OPTIMIZE WALL-CLOCK TIME

In this section, we aim to optimize performance with respect to wall-clock time, explicitly accounting for the duration of model-parameter update rounds. In Section 4.1, we introduce a new performance metric that incorporates these durations and derive an upper bound H, which serves as the wall-clock-time counterpart to G. Additionally, we provide a proxy for the expected wall-clock time required to reach ϵ-accuracy. Section 4.2 highlights the key differences between this analysis and that of Section 3. Finally, in Section 4.3, we present numerical results demonstrating improved model performance in wall-clock time when optimizing H rather than G.

4.1 Bound, delay, and gradient descent

Proposition 4.1 below provides an upper bound on the ergodic mean of the squared norm of the gradient of f, weighted by the expected duration of each corresponding round. This quantity can be interpreted as a per-round cost, where the cost is defined as the product of the squared gradient norm and the average duration of that round. For each $t \in \{0, 1, \dots, T\}$, let τ_t denote the duration of round t in the model described in Section 2.1, and define $\bar{\tau}_t = \mathbb{E}[\tau_t]$.

Proposition 4.1. *In the framework of Section 2.1, there exists $\eta_{max} > 0$ such that for all $\eta \in (0, \eta_{max})$, the following bound holds:*

$$\frac{1}{T+1}\sum_{t=0}^{T} \bar{\tau}_t \mathbb{E}\left[\|\nabla f(w_t)\|^2\right] \leq 8H = 8\frac{G}{\lambda}. \tag{11}$$

Here, λ is the throughput of the Jackson network, that is, the number of rounds per unit of wall-clock time, given as follows, with $Z_{n,m}$ and $Z_{n,m-1}$ as defined in Proposition 2.1 and $\xi \sim \pi_{n,m}$:

$$\lambda = \sum_{i=1}^{n} \mu_i \mathbb{P}(\xi_i > 0) = \frac{Z_{n,m-1}}{Z_{n,m}}. \tag{12}$$

Proof. See Section E of the supplementary material [2]. □

The bound H in Proposition 4.1 can be viewed as the throughput-aware counterpart to the bound G. As discussed in the previous section, minimizing G alone (which reflects the average model error) tends to route most tasks to slower clients, significantly reducing system throughput. This occurs because prioritizing slow clients helps reduce staleness, thereby improving model accuracy per iteration.

In contrast, the quantity $H = G/\lambda$ captures the trade-off between two competing objectives: minimizing G to improve per-iteration model quality, and maximizing λ to increase the frequency of updates in wall-clock time. These two goals are often in conflict, as reducing one typically increases the other. Thus, H provides a principled objective that balances staleness reduction with practical training speed, making it more realistic in asynchronous FL settings.

Another useful metric for evaluating performance over wall-clock time is the expected time to reach ϵ-accuracy. Proposition 4.2 gives its expression within the framework of Section 2.1. Let $\tilde{\tau}_T = \sum_{t=0}^{T-1} \tau_t$ denote the time required to complete T rounds.

Proposition 4.2 (Time to achieve an ϵ-accuracy)**.** *Assume that the learning rate $\eta = \frac{C}{T^\alpha}$ is a function of the number of rounds T, where $\alpha, C \in \mathbb{R}_{>0}$, such that $\alpha < 1$ and $\eta < \eta_{max}$. Then it holds that $\frac{1}{T+1}\sum_{t=0}^{T} \mathbb{E}[\|\nabla f(w_t)\|^2] \leq \epsilon$ whenever $T \geq T_\epsilon$, where*

$$T_\epsilon = \mathcal{O}\left(\left(\frac{A}{C\epsilon}\right)^{\frac{1}{1-\alpha}} + \left(\frac{CLB}{\epsilon n^2}\sum_{i=1}^{n}\frac{1}{p_i}\right)^{\frac{1}{\alpha}} + \left(\frac{C^2L^2Bm}{\epsilon n^2}\sum_{i=1}^{n}\frac{\mathbb{E}[D_i]}{p_i^2}\right)^{\frac{1}{2\alpha}}\right)$$

rounds with expected wall-clock time $\mathbb{E}[\tilde{\tau}_{T_\epsilon}] = T_\epsilon/\lambda$.

Proof. See Section F of the supplementary material [2]. □

Each term in T_ϵ coincides with a term in G. The metric $\mathbb{E}[\tilde{\tau}_\epsilon] = T_\epsilon/\lambda$ further highlights the trade-off between minimizing per-round error and maximizing update frequency. Minimizing the number of rounds T_ϵ to reach ϵ-accuracy may slow the system by favoring slower clients to reduce staleness, while maximizing throughput λ increases update frequency but can worsen staleness. Like the proxy H, the expected time to reach ϵ-accuracy captures this fundamental trade-off.

4.2 *Discussion*

Part of the discussion in Section 3.2 can be adapted to H with minor modification. One fundamental difference between G and H is that H is generally *not* a non-decreasing function of the number m of tasks. Figure 3 shows the bound $H(p^{\text{uniform}})$ as a function of m under different step sizes η, revealing the existence of an optimal number $m^* > 1$ of tasks that minimizes H. Our intuition is that when $m < m^*$, throughput is insufficient and clients are underutilized; conversely, when $m > m^*$, throughput increases but so does staleness, which ultimately hinders convergence, as already observed with G. As a side remark, observe that the optimal number m^* of tasks decreases (albeit slowly) with η. Our intuition is that, with a higher η, the impact of stale gradients becomes more significant, as each gradient carries more weight in the model-parameter updates.

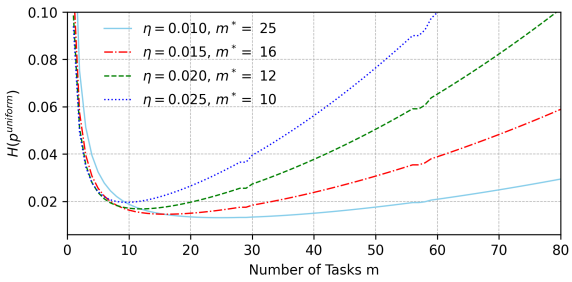

Figure 3. Bound $H(p^{\text{uniform}})$ as a function of the number m of tasks for different values of the step size η. The system consists of 50 clients, with speeds given by $\mu_i = \exp(i/100)$ for each $i \in \{1, \ldots, n\}$.

4.3 *Numerical results*

In this section, we follow the same experimental procedure as in Section 3.3, but now we optimize the performance metric H introduced in Section 4.1, which accounts for wall-clock time. The routing vector p_H^* is obtained by minimizing H using the same Adam-based optimization setup described previously. We evaluate the performance of `Generalized AsyncSGD` under this routing strategy on the KMNIST dataset [10], using the same simulation and evaluation framework as in the first set of experiments.

We analyze a network of $n = 30$ clients with concurrency level $m = 30$. Clients are organized into three clusters of 10. The slowest cluster has an average service time of 100 time units, the medium cluster 10 time units, and the fastest 1 time unit. This low-concurrency, high-speed-heterogeneity scenario is particularly challenging for optimizing throughput. We set the parameters as follows: $L = 1$, $\sigma = 3$, $M = 10$, $A/T = 15$, and $\eta = 0.01$. More details appear in Sections H and I of the supplementary material [2].

Optimizing H The H-optimized routing probabilities are (per client) $p^*_{\text{H, slow}} = 0.0068$, $p^*_{\text{H, medium}} = 0.0449$, and $p^*_{\text{H, fast}} = 0.0487$. In contrast to Section 3.3, faster clients receive a larger fraction of tasks, but $p^*_{H,\text{medium}}$ and $p^*_{H,\text{fast}}$ remain of the same order, so that p_H^* seems to achieve a trade-off between minimizing staleness (like p_G^*)

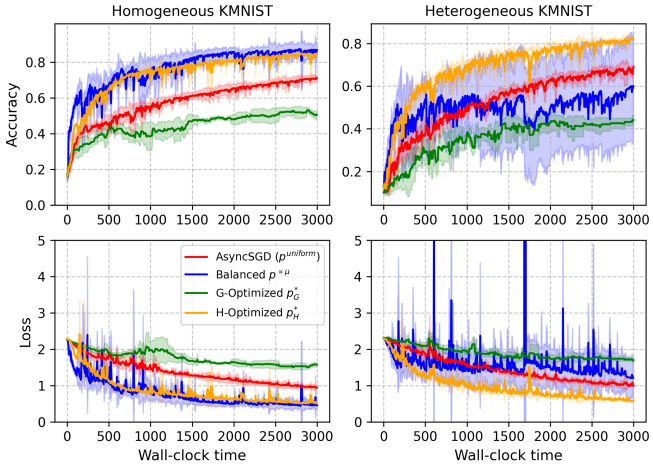

Figure 4. Performance on the test set with respect to wall-clock time at the CS in the scenario of Section 4.3, with $n = 30$ clients and $m = 30$ tasks on homogeneous and heterogeneous KMNIST datasets. Simulations ran for 3,000 wall-clock time units and were repeated 10 times. Solid lines show means; shaded areas indicate standard deviations.

and maximizing throughput (like $p^{\propto\mu}$, a common heuristic to maximize throughput). Concretely, within the wall-clock time frame of 3,000 units plotted in Figure 4, $p^{\propto\mu}$ completes 17,000 rounds, which sets it apart from p_H^* (3,200 rounds), p^{uniform} (690), and p_G^* (145).

Performance on datasets Figure 4 compares the accuracy and loss of `Generalized AsyncSGD` under these four routing strategies. Focusing first on the average performance, we observe that in the homogeneous scenario, $p^{\propto\mu}$ performs slightly better than p_H^*, which in turn outperforms p_G^* and p^{uniform}, while in the heterogeneous scenario p_H^* outperforms all strategies. Furthermore, the balanced strategy $p^{\propto\mu}$ exhibits sharp spikes in the loss trajectory in all scenarios, and a particularly large standard deviation in the heterogeneous scenario. This contrasts with the low and stable standard deviation exhibited by both p_G^* and p_H^*. All in all, these numerical results confirm that when wall-clock time is a performance criterion, the H-optimized strategy provides a suitable trade-off between minimizing staleness (overemphasized by p_G^*) and maximizing throughput (overemphasized by $p^{\propto\mu}$), even though the bound H serves only as a proxy for actual performance. Section J.2 of the supplementary material [2] includes additional plots for alternative image classification tasks, heterogeneous data distributions, and computation time distributions.

5 CONCLUSION

We provide novel insights into the impact of queueing dynamics on asynchronous FL, enabling both performance optimization and the identification of fundamental limitations in existing performance objectives. Our experiments on real datasets show that queueing effects can significantly affect performance, and that our proposed optimizations can improve accuracy by 10% to 30%. These insights are relevant not only to `Generalized AsyncSGD`, our focus here, but also to algorithms such as `FedBuff`, which introduce additional control dimensions and could similarly benefit from our stochastic modeling framework. For example, we believe that adding a buffering mechanism at the CS to aggregate gradients every K service completion could reduce the average relative delay by a factor of $\frac{1}{K}$ under similar network dynamics. Modeling client unavailability is also a promising direction for future work.

Acknowledgements

This research was supported in part by ANR EPLER, Projet E2CC, and the "Data Science & Processus Industriels" chair funded by École Polytechnique, the Mohammed VI Polytechnic University, and Fondation de l'X. This research was also facilitated by the support of LabEx CIMI via the SOLACE project-team and the "Stochastic control and learning for complex networks" thematic semester.

References

[1] A. Agarwal, G. Joshi, and L. Pileggi. Fedecado: A dynamical system model of federated learning. *arXiv preprint arXiv:2410.09933*, 2024.

[2] A. Alahyane, C. Comte, M. Jonckheere, and É. Moulines. Optimizing asynchronous federated learning: A delicate trade-off between model-parameter staleness and update frequency. *arXiv preprint arXiv:2502.08206*, 2025. Full version of this paper.

[3] F. Baccelli and P. Brémaud. *Elements of queueing theory: Palm Martingale calculus and stochastic recurrences*, volume 26. Springer Science & Business Media, 2013.

[4] P. Brémaud. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*, volume 31. Springer Science & Business Media, 2013.

[5] P. Brémaud. *Probability theory and stochastic processes*. Springer, 2020.

[6] J. P. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Communications of the ACM*, 16:527–531, 1973.

[7] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz. Revisiting distributed synchronous sgd. *arXiv preprint arXiv:1604.00981*, 2016.

[8] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 15–24. IEEE, 2020.

[9] Z. Chen, W. Liao, K. Hua, C. Lu, and W. Yu. Towards asynchronous federated learning for heterogeneous edge-powered internet of things. *Digital Communications and Networks*, 7(3):317–326, 2021.

[10] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha. Deep learning for classical japanese literature. *arXiv preprint arXiv:1812.01718*, 2018.

[11] A. Cohen, A. Daniely, Y. Drori, T. Koren, and M. Schain. Asynchronous stochastic optimization robust to arbitrary delays. *Advances in Neural Information Processing Systems*, 34:9024–9035, 2021.

[12] L. Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[13] S. Dutta, G. Joshi, S. Ghosh, P. Dube, and P. Nagpurkar. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd. In *International Conference on Artificial Intelligence and Statistics*, pages 803–812. PMLR, 2018.

[14] R. Islamov, M. Safaryan, and D. Alistarh. Asgrad: A sharp unified analysis of asynchronous-sgd algorithms. In *International Conference on Artificial Intelligence and Statistics*, pages 649–657. PMLR, 2024.

[15] J. R. Jackson. Networks of waiting lines. *Operations research*, 5(4): 518–521, 1957.

[16] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1–2):1–210, 2021.

[17] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] A. Koloskova, S. U. Stich, and M. Jaggi. Sharper convergence guarantees for asynchronous sgd for distributed and federated learning. *Advances in Neural Information Processing Systems*, 35:17202–17215, 2022.

[19] J. Konečnỳ, B. McMahan, and D. Ramage. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575*, 2015.

[20] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[21] L. Leconte, M. Jonckheere, S. Samsonov, and E. Moulines. Queuing dynamics of asynchronous federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1711–1719. PMLR, 2024.

[22] Q. Li, Y. Diao, Q. Chen, and B. He. Federated learning on non-iid data silos: An experimental study. In *2022 IEEE 38th international conference on data engineering (ICDE)*, pages 965–978. IEEE, 2022.

[23] M. Makarenko, E. Gasanov, R. Islamov, A. Sadiev, and P. Richtarik. Adaptive compression for communication-efficient distributed training. *arXiv preprint arXiv:2211.00188*, 2022.

[24] Y. Mao, Z. Zhao, G. Yan, Y. Liu, T. Lan, L. Song, and W. Ding. Communication-efficient federated learning with adaptive quantization. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 13(4): 1–26, 2022.

[25] A. Maranjyan, O. S. Omar, and P. Richtárik. Mindflayer: Efficient asynchronous parallel sgd in the presence of heterogeneous and random worker compute times. *arXiv preprint arXiv:2410.04285*, 2024.

[26] A. Maranjyan, A. Tyurin, and P. Richtárik. Ringmaster asgd: The first asynchronous sgd with optimal time complexity. *arXiv preprint arXiv:2501.16168*, 2025.

[27] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[28] K. Mishchenko, F. Bach, M. Even, and B. E. Woodworth. Asynchronous sgd beats minibatch sgd under arbitrary delays. *Advances in Neural Information Processing Systems*, 35:420–433, 2022.

[29] J. Nguyen, K. Malik, H. Zhan, A. Yousefpour, M. Rabbat, M. Malek, and D. Huba. Federated learning with buffered asynchronous aggregation. In *International conference on artificial intelligence and statistics*, pages 3581–3607. PMLR, 2022.

[30] L. Qu, S. Song, and C.-Y. Tsui. Feddq: Communication-efficient federated learning with descending quantization. In *GLOBECOM 2022-2022 IEEE Global Communications Conference*, pages 281–286. IEEE, 2022.

[31] R. Serfozo. *Introduction to stochastic networks*, volume 44. Springer Science & Business Media, 2012.

[32] R. Suri. A concept of monotonicity and its characterization for closed queueing networks. *Operations Research*, 33(3):606–624, 1985. doi: 10.1287/opre.33.3.606.

[33] A. Tyurin and P. Richtárik. Dasha: Distributed nonconvex optimization with communication compression, optimal oracle complexity, and no client synchronization. *arXiv preprint arXiv:2202.01268*, 2022.

[34] A. Tyurin and P. Richtárik. Optimal time complexities of parallel stochastic optimization methods under a fixed computation model. *Advances in Neural Information Processing Systems*, 36:16515–16577, 2023.

[35] A. Tyurin, M. Pozzi, I. Ilin, and P. Richtárik. Shadowheart sgd: Distributed asynchronous sgd with optimal time complexity under arbitrary computation and communication heterogeneity. *Advances in Neural Information Processing Systems*, 37:3717–3780, 2024.

[36] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor. Tackling the objective inconsistency problem in heterogeneous federated optimization. *Advances in neural information processing systems*, 33:7611–7623, 2020.

[37] C. Xie, S. Koyejo, and I. Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

[38] C. Xu, Y. Qu, Y. Xiang, and L. Gao. Asynchronous federated learning on heterogeneous devices: A survey. *Computer Science Review*, 50:100595, 2023.

Optimizing Asynchronous Federated Learning: A Delicate Trade-Off Between Model-Parameter Staleness and Update Frequency

Supplementary Materials

A Assumptions

Our analysis is grounded in the assumptions presented in Section 2.1, which align with those established in [21]. These assumptions are detailed as follows:

A1 Lower Boundedness: The objective function f is bounded from below by some real number f^*, meaning $f(w) \geq f^*$ for all $w \in \mathbb{R}^d$.

A2 Gradient Smoothness: Each client's local function f_i has an L-Lipschitz continuous gradient, where $L > 0$. Mathematically, for any vectors $w, \mu \in \mathbb{R}^d$:

$$\|\nabla f_i(w) - \nabla f_i(\mu)\| \leq L\|w - \mu\|.$$

A3 Stochastic Gradient Properties: For each client i, the stochastic gradient $g_i(w)$ is an unbiased estimator of the gradient $\nabla f_i(w)$ with bounded variance $\sigma^2 > 0$. That is, for all $w \in \mathbb{R}^d$:

$$\mathbb{E}[g_i(w) - \nabla f_i(w)] = 0 \text{ and } \mathbb{E}[\|g_i(w) - \nabla f_i(w)\|^2] \leq \sigma^2.$$

A4 Bounded Client Heterogeneity: There exist constant $M > 0$, such that for all $w \in \mathbb{R}^d$:

$$\|\nabla f(w) - \nabla f_i(w)\|^2 \leq M^2.$$

A5 Stationary Regime: The dynamics of the closed Jackson network are assumed to be in their stationary regime. Specifically, the random n-dimensional vector X_t, where $t \in \mathbb{N}$, follows its stationary distribution $\pi_{n,m-1}$.

B Proof of Proposition 2.1

The result follows by observing that $(X_t, t \in \mathbb{N})$ tracks the state at departure times of a Jackson network [15, 31] with n clients (usually called *servers*) and m tasks (often called *customers* or *jobs*), with service rate vector μ and relative arrival rate vector p. In particular, (4) follows from Chapter 1 (Definition 1.8 and Theorem 1.12) and Section 4.8 (Definition 4.34 and Example 4.38) in [31].

C Revised Expression for $\eta_{\max}$

In [21], there is a technical issue in the proof establishing the upper bound G. The authors assume that for all $k \in \{0, \ldots, T\}$, the relative delay $D_{i,k}$ is independent of $\nabla f(w_k)$. However, this assumption does not hold in the case of closed Jackson network dynamics and leads to an incorrect expression for $\eta_{\max}$.

To address this issue, we propose a stochastic upper bound on the relative delay $D_{i,k}$ using another random variable that is independent of $\nabla f(w_k)$, and we derive a corrected expression for $\eta_{\max}$. Specifically, we consider the worst-case sojourn time scenario for the task sent to client i at round k, which occurs when this task finds $m - 1$ tasks already queued at client i.

Due to the memoryless property of exponential service times, the sojourn time of this task at client i is then distributed as an Erlang random variable Er with parameters m and μ_i. The quantity $D_{i,k}$, representing the number of service completions during the sojourn of this task, can be stochastically bounded by the number of events generated by an independent Poisson process N with intensity $\sum_{j=1}^n \mu_j$ over a time interval distributed as an Erlang with parameters m and μ_i.

Therefore, for all $k \in \{0, \ldots, T\}$:

$$\begin{aligned}
\mathbb{E}[D_{i,k}\|\nabla f(w_k)\|^2] &\leq \mathbb{E}[N(Er)\|\nabla f(w_k)\|^2] \\
&= \mathbb{E}[N(Er)] \cdot \mathbb{E}[\|\nabla f(w_k)\|^2] \\
&= \mathbb{E}[\|\nabla f(w_k)\|^2] \cdot \frac{m}{\mu_i} \sum_{j=1}^n \mu_j,
\end{aligned}$$

where $N(Er)$ denotes the number of events in a Poisson process with rate $\sum_{j=1}^n \mu_j$ over a time period following an independent Erlang distribution with parameters (m, μ_i).

Using this upper bound and following the structure of the proof in [21], we derive the following corrected expression for $\eta_{\max}$:

$$\eta_{\max} = \frac{1}{4L} \min \left\{ \left(\frac{m^2}{n^2} \sum_{j=1}^n \mu_j \sum_{i=1}^n \frac{1}{\mu_i p_i^2} \right)^{-1/2}, \frac{2}{\sum_{i=1}^n \frac{1}{n^2 p_i}} \right\}.$$

D Proof of Theorem 3.1

Consider the asynchronous federated learning framework of Section 2.1. After proving preliminary results in Lemma D.1 and Corollary D.2, we prove (6) in Appendix D.1 and (7) in Appendix D.2.

Lemma D.1. *For each $i \in \{1, 2, \dots, n\}$, for each $t \in \mathbb{N}_{>0}$ we have*

$$R_{i,t+1} = \begin{cases} \max(0, R_{i,t} - 1) & \text{if } A_{t+1} \neq i, \\ \min\{r \geq R_{i,t} : C_{t+1+r} = i\} & \text{if } A_{t+1} = i. \end{cases}$$

Proof. Let $i \in \{1, 2, \dots, n\}$ and $t \in \mathbb{N}_{>0}$.

The following preliminary results stem from the definition (3) of $R_{i,t}$ and the observation that the sequence $r \in \mathbb{N} \mapsto \sum_{s=t}^{t+r} \mathbf{1}[C_s = i]$ is nondecreasing with increments 0 or 1 and takes value $\mathbf{1}[C_t = i] \in \{0, 1\}$ at $r = 0$ (equal to 0 if $X_{i,t-1} + \mathbf{1}[A_t = i] = 0$ by definition of C_t):

- We can replace the equal sign with a larger-than-or-equal sign in the definition of $R_{i,t}$:

$$R_{i,t} = \min \left\{ r \in \mathbb{N} : \sum_{s=t}^{t+r} \mathbf{1}[C_s = i] \geq X_{i,t-1} + \mathbf{1}[A_t = i] \right\}. \tag{13}$$

- We have

$$\sum_{s=t}^{t+R_{i,t}} \mathbf{1}[C_s = i] = X_{i,t-1} + \mathbf{1}[A_t = i]. \tag{14}$$

- Lastly, we can verify that $A_t = i$ implies $C_{t+R_{i,t}} = i$.

Now focusing on $R_{i,t+1}$, we have successively:

$$\begin{aligned} R_{i,t+1} &\overset{\text{(a)}}{=} \min \left\{ r \in \mathbb{N} : \sum_{s=t+1}^{t+1+r} \mathbf{1}[C_s = i] \geq X_{i,t} + \mathbf{1}[A_{t+1} = i] \right\}, \\ &\overset{\text{(b)}}{=} \min \left\{ r \in \mathbb{N} : \sum_{s=t}^{t+1+r} \mathbf{1}[C_s = i] \geq X_{i,t-1} + \mathbf{1}[A_t = i] + \mathbf{1}[A_{t+1} = i] \right\}, \\ &\overset{\text{(c)}}{=} \min \left\{ r \geq \max(0, R_{i,t} - 1) : \sum_{s=t}^{t+1+r} \mathbf{1}[C_s = i] \geq X_{i,t-1} + \mathbf{1}[A_t = i] + \mathbf{1}[A_{t+1} = i] \right\}, \\ &\overset{\text{(d)}}{=} \min \left\{ r \geq \max(0, R_{i,t} - 1) : \sum_{s=t+R_{i,t}+1}^{t+1+r} \mathbf{1}[C_s = i] \geq \mathbf{1}[A_{t+1} = i], \right\} \end{aligned}$$

where (a) and (c) follow from (13), (b) from (1), and (d) by injecting (14). Consistently with the result announced in the lemma, we conclude by making a case disjunction:

- If $A_{t+1} \neq i$: The right-hand side of the inequality in (d) is 0, hence the inequality is already satisfied by choosing the smallest authorized value for $R_{i,t}$.
- If $A_{t+1} = i$: The right-hand side of the inequality in (d) is 1. Since the sum $\sum_{s=t+R_{i,t}+1}^{t+1+r} \mathbf{1}[C_s = i]$ is 0 (empty) when $r = R_{i,t} - 1$, remains 0 as long as $C_{t+1+r} \neq i$, and increases to 1 whenever $C_{t+r+1} = i$, the result follows. □

Corollary D.2. *For each $i \in \{1, 2, \dots, n\}$, the function $t \in \mathbb{N}_{>0} \mapsto t + D_{i,t}$ defines a bijective increasing mapping from the set $\{t \in \mathbb{N}_{>0} : A_t = i\}$ into the set $\{t \in \mathbb{N}_{>0} : C_t = i\}$.*

Proof. Let $i \in \{1, 2, \dots, n\}$. By definition of $D_{i,t}$, we have $D_{i,t} = R_{i,t}$ for each $t \in \mathbb{N}_{>0}$ so that $A_t = i$, hence it suffices to prove the result for $R_{i,t}$ instead of $D_{i,t}$.

For each $t \in \mathbb{N}_{>0}$, $t + R_{i,t}$ is the round at the end of which client i finishes processing the last task that has arrived at this client by the beginning of round t. Lemma D.1 implies that, for each $t \in \mathbb{N}_{>0}$, we have

$$t + 1 + R_{i,t+1} = \begin{cases} t + \max(1, R_{i,t}) & \text{if } A_{t+1} \neq i, \\ \min\{s > t + R_{i,t} : C_s = i\} & \text{if } A_{t+1} = i. \end{cases}$$

It follows directly that the function $t \in \mathbb{N}_{>0} \mapsto t + R_{i,t}$ is non-decreasing and defines an injection from $\{t \in \mathbb{N}_{>0} : A_t = i\}$ onto $\{t \in \mathbb{N}_{>0} : C_t = i\}$. Furthermore, to prove this injection is also a surjection, it suffices to verify that $A_{t+1} \neq i$ implies either $t + 1 + R_{i,t+1} = t + R_{i,t}$ or $C_{t+1+R_{i,t+1}} \neq i$. If $A_{t+1} \neq i$, the only way that $t + 1 + R_{i,t+1} > t + R_{i,t}$ is when $R_{i,t} = 0$, so that the maximum is attained at 1 and $t + 1 + R_{i,t+1} = t + 1$. Using (3), we can verify that $R_{i,t} = 0$ implies $X_{i,t} = 0$ which, combined with $A_{t+1} \neq i$, in turn implies $C_{t+1} (= C_{t+1+R_{i,t+1}}) \neq i$ by definition of C_{t+1}. □

D.1 Proof of Equation (6)

Let $i \in \{1, 2, \dots n\}$. Our goal in this section is to prove that $\mathbb{E}[D_i] = \mathbb{E}[X_i]$. In a nutshell, we will prove that

$$\mathbb{E}[D_i] \overset{(a)}{=} \lim_{T \to +\infty} \frac{1}{T} \sum_{t=1}^{T} D_{i,t} \overset{(b)}{=} \lim_{T \to +\infty} \frac{1}{T} \sum_{t=1}^{T} X_{i,t} \overset{(c)}{=} \mathbb{E}[X_i],$$

where all equal signs hold almost surely. Equation (a) will be shown in Lemma D.3 using an ergodicity argument. Equation (b) will be proved using a squeeze argument formulated in Lemmas D.4 and D.5. Lastly, Equation (c) follows from the classical ergodic theorem for irreducible positive-recurrent Markov chains. These arguments are combined at the end of the section to prove (6).

Lemma D.3. *For each $i \in \{1, 2, \dots, n\}$, we have*

$$\lim_{T \to +\infty} \frac{1}{T} \sum_{t=1}^{T} D_{i,t} = \mathbb{E}[D_i] \quad \textit{almost surely}.$$

Proof of Lemma D.3. Equations (2) and (3) show that, for each $i \in \{1, 2, \dots, n\}$, there exists a deterministic function $g_i : (\mathbb{N} \times \{1, 2, \dots, n\} \times \{1, 2, \dots, n\})^{\mathbb{N}} \to \mathbb{N}$ such that we can write $D_{i,t} = g_i((X_{t+s}, A_{t+s}, C_{t+s}), s \in \mathbb{N})$ for each $t \in \mathbb{N}$. Since the sequence $((X_t, A_t, C_t), t \in \mathbb{N})$ is ergodic, the conclusion follows from [5, Remark 16.1.11]. □

Lemma D.4. *Let $i \in \{1, 2, \dots, n\}$. If $X_{i,0} = 0$, then for each $T \in \{1, 2, 3, \dots\}$, we have*

$$\sum_{t=1}^{T} X_{i,t} \leq \sum_{t=1}^{T} D_{i,t} \leq \sum_{t=1}^{T+R_{i,T}} X_{i,t}. \tag{15}$$

Proof of Lemma D.4. Let $i \in \{1, 2, \dots, n\}$ and $T \in \mathbb{N}_{>0}$, and assume that $X_{i,0} = 0$. If the set $\{t \in \{1, 2, \dots, T\} : A_t = i\}$ is empty, then all three sums are zero, and the inequalities are trivially satisfied. Therefore, in the remainder, we focus on sample paths for which this set is nonempty.

First observe that, for each $t \in \mathbb{N}$, we have

$$X_{i,s} = \sum_{t=1}^{+\infty} \mathbf{1}[A_t = i, t \leq s < t + D_{i,t}]. \tag{16}$$

Indeed, we have successively:

$$X_{i,s} \overset{(a)}{=} \sum_{t=1}^{s} \mathbf{1}[A_t = i] - \sum_{t=1}^{s} \mathbf{1}[C_t = i] \overset{(b)}{=} \sum_{t=1}^{s} \mathbf{1}[A_t = i] - \sum_{t=1}^{s} \mathbf{1}[A_t = i, t + D_{i,t} \leq s],$$

where (a) follows by unfolding (1) and using our assumption that $X_{i,0} = 0$, and (b) follows by observing that, by Corollary D.2, the function $t \mapsto t + D_{i,t}$ defines a bijective (increasing) mapping from the set $\{t \in \mathbb{N}_{>0} : A_t = i\}$ onto the set $\{t \in \mathbb{N}_{>0} : C_t = i\}$. Equation (16) then follows by rearranging the terms.

Now, we can also use the following trick to rewrite $\sum_{t=1}^{T} D_{i,t}$ in a manner that is similar to (16):

$$\begin{aligned} \sum_{t=1}^{T} D_{i,t} &= \sum_{t=1}^{T} \mathbf{1}[A_t = i] D_{i,t} = \sum_{t=1}^{T} \mathbf{1}[A_t = i] \sum_{s=1}^{+\infty} \mathbf{1}[t \leq s < t + D_{i,t}], \\ &= \sum_{s=1}^{+\infty} \sum_{t=1}^{T} \mathbf{1}[A_t = i, t \leq s < t + D_{i,t}]. \end{aligned} \tag{17}$$

The lower bound in (15) follows by capping the outer sum at $s \in \{1, 2, \dots, T\}$ in (17) and injecting (16). The upper-bound in (15) follows in a similar spirit:

$$\sum_{t=1}^{T} D_{i,t} \overset{(a)}{=} \sum_{s=1}^{T+R_{i,T}} \sum_{t=1}^{T} \mathbf{1}[A_t = i, t \leq s < t + D_{i,t}] \overset{(b)}{\leq} \sum_{s=1}^{T+R_{i,T}} X_{i,s},$$

where (a) is equivalent to (17), after recalling that tasks leave clients in the same order as they arrive, and (b) follows by expanding the inner sum to $t \in \mathbb{N}_{>0}$. □

Lemma D.5. *For each $i \in \{1, 2, \dots, n\}$, we have almost surely that $R_{i,T} = o(T)$ as $T \to +\infty$.*

Proof of Lemma D.5. Let $i \in \{1, 2, \ldots, n\}$. For each $t \in \mathbb{N}$, we have $0 \leq R_{i,t} \leq E_{i,t}$, where $E_{i,t} = \min\{r \in \mathbb{N} : X_{i,t+r} = 0\}$ denotes the number of rounds, starting from round t, until client i first becomes empty.

Since the Markov chain $(X_t)_{t\in\mathbb{N}}$ is irreducible and positive recurrent, the ergodic theorem for Markov chains [5, Remark 16.1.11] implies that

$$\lim_{t\to+\infty} \frac{1}{t} \sum_{k=1}^{t} E_{i,k} = \mathbb{E}[E_i], \text{ almost surely.}$$

Where $\mathbb{E}[E_i]$ is the expected hitting time, under the stationary distribution of $(X_t)_{t\in\mathbb{N}}$, to the set of states where client i is empty.

This yields

$$\lim_{t\to+\infty} \frac{E_{i,t}}{t} = \lim_{t\to+\infty} \left(\frac{1}{t} \sum_{k=1}^{t} E_{i,k} - \frac{1}{t} \sum_{k=1}^{t-1} E_{i,k} \right) = \mathbb{E}[E_i] - \mathbb{E}[E_i] = 0.$$

Since $0 \leq \frac{R_{i,t}}{t} \leq \frac{E_{i,t}}{t}$, we conclude that $\lim_{t\to+\infty} \frac{R_{i,t}}{t} = 0$, almost surely. □

Proof of Equation (6). Let $i \in \{1, 2, \ldots, n\}$. Let us assume for now that the initial distribution of the Markov chain $(X_t, t \in \mathbb{N})$ is such that $X_{i,0} = 0$ (with probability 1).

Let us first prove that the upper and lower bound in Lemma D.4 converge almost surely to the same limit, and that this limit is $\mathbb{E}[X_i]$, that is:

$$\lim_{T\to+\infty} \frac{1}{T} \sum_{t=1}^{T} X_{i,t} \overset{(a)}{=} \mathbb{E}[X_i] \overset{(b)}{=} \lim_{T\to+\infty} \frac{1}{T} \sum_{t=1}^{T+R_{i,T}} X_{i,t}. \tag{18}$$

Since $(X_t, t \in \mathbb{N})$ is an irreducible positive-recurrent Markov chain, (a) follows directly from the ergodic theorem [5, Theorem 3.3.2]. The argument for (b) is in a similar spirit, with the extra-complication that the upper bound of summation $(T+R_{i,T})$ is different from the denominator (T). We start by rewriting the upper bound as follows:

$$\frac{1}{T} \sum_{t=1}^{T+R_{i,T}} X_{i,t} = \left(1 + \frac{R_{i,T}}{T}\right) \frac{1}{T + R_{i,T}} \sum_{t=1}^{T+R_{i,T}} X_{i,t}.$$

Equation (b) then follows by combining two arguments: (i) Lemma D.5 implies that $\lim_{T\to+\infty} 1 + \frac{R_{i,T}}{T} = 1$ almost surely; (ii) since $\lim_{T\to+\infty} T + R_{i,T} = +\infty$, the ergodic theorem for irreducible positive-recurrent Markov chains again implies that

$$\lim_{T\to+\infty} \frac{1}{T + R_{i,T}} \sum_{t=1}^{T+R_{i,T}} X_{i,t} = \mathbb{E}[X_i], \quad \text{almost surely.}$$

Now, combining (18) with Lemma D.4 and the squeeze theorem allows us to conclude that

$$\lim_{T\to+\infty} \frac{1}{T} \sum_{t=1}^{T} D_{i,t} = \mathbb{E}[X_i], \quad \text{almost surely.}$$

Equation (6) then follows from Lemma D.3.

To prove that (6) also holds without the assumption that $X_{i,0} = 0$ with probability 1, it suffices to recall that the expectations $\mathbb{E}[D_i]$ and $\mathbb{E}[X_i]$ do not depend on the initial distribution. □

D.2 Proof of Equation (7)

Let $i, j \in \{1, 2, \ldots, n\}$. Our goal is to prove (7), which by (6) is equivalent to

$$\frac{\partial \mathbb{E}[X_i]}{\partial(\log p_j)} = \mathrm{Cov}[X_i, X_j].$$

Recall that the vector X follows the stationary distribution (4), which we can rewrite as

$$\log \mathbb{P}(X = x) = \log \pi_{n,m-1}(x) = \sum_{i=1}^{n} x_i \left(\log p_i - \log \mu_i\right) - \log Z_{n,m-1}, \quad x \in \mathcal{X}_{n,m-1}, \tag{19}$$

where $Z_{n,m-1}$ follows by normalization:

$$\log Z_{n,m-1} = \log \left(\sum_{x\in\mathcal{X}_{n,m-1}} \exp\left(\sum_{i=1}^{n} x_i \left(\log p_i - \log \mu_i\right) \right) \right). \tag{20}$$

Let us first prove the following intermediary result:

$$\frac{\partial \log Z_{n,m-1}}{\partial(\log p_j)} = \mathbb{E}[X_j], \qquad \frac{\partial \log \pi_{n,m-1}(x)}{\partial(\log p_j)} = x_j - \mathbb{E}[X_j], \quad x \in \mathcal{X}_{n,m-1}. \tag{21}$$

The first part of (21) follows by taking the partial derivative of (20) and rearranging the terms to retrieve the definition of $\pi_{n,m-1}$:

$$\begin{aligned}\frac{\partial \log(Z_{n,m-1})}{\partial(\log p_j)} &= \frac{1}{Z_{n,m-1}} \frac{\partial Z_{n,m-1}}{\partial(\log p_j)} = \frac{1}{Z_{n,m-1}} \sum_{x \in \mathcal{X}_{n,m-1}} x_j \exp\left(\sum_{i=1}^{n} x_i(\log p_i - \log \mu_i)\right), \\ &= \sum_{x \in \mathcal{X}_{n,m-1}} x_j \exp\left(\sum_{i=1}^{n} x_i(\log p_i - \log \mu_i) - \log Z_{n,m-1}\right) = \sum_{x \in \mathcal{X}_{n,m-1}} x_j \pi_{n,m-1}(x) = \mathbb{E}[X_j].\end{aligned}$$

Now, the second part of (21) follows by taking the partial derivative of (19) and injecting the previous result:

$$\frac{\partial \log \pi_{n,m-1}(x)}{\partial(\log p_j)} = x_j - \frac{\partial \log Z_{n,m-1}}{\partial(\log p_j)} = x_j - \mathbb{E}[X_j], \quad x \in \mathcal{X}_{n,m-1}.$$

To conclude, it suffices to inject the second part of (21) into the definition of expectation:

$$\begin{aligned}\frac{\partial \mathbb{E}[X_i]}{\partial(\log p_j)} &= \sum_{x \in \mathcal{X}_{n,m-1}} x_i \frac{\partial \pi_{n,m-1}(x)}{\partial(\log p_j)} = \sum_{x \in \mathcal{X}_{n,m-1}} \pi_{n,m-1}(x) x_i \frac{\partial \log \pi_{n,m-1}(x)}{\partial(\log p_j)}, \\ &= \sum_{x \in \mathcal{X}_{n,m-1}} \pi_{n,m-1}(x) x_i (x_j - \mathbb{E}[X_j]) = \mathrm{Cov}[X_i, X_j].\end{aligned}$$

D.3 Proof of Equations (8) *and* (9)

Equation (8) was proved in [6, Equation (8)]. Buzen's algorithm [6] as described in the proposition was introduced and shown to yield the correct result in [6, Paragraph "Computation of $G(N)$"]. (Note that, in [6], N corresponds to our m and M to our n.) All that remains is to prove (9), which we do in a similar way to the proof of (8) in [6]. To simplify notation, we can focus without loss of generality on the pair $(i, j) = (1, 2)$. We have successively:

$$\begin{aligned}\mathbb{E}[X_1 X_2] &= \sum_{x \in \mathcal{X}_{n,m-1}} x_1 x_2 \pi_{n,m-1}(x) = \sum_{x \in \mathcal{X}_{n,m-1}} \sum_{k=1}^{x_1} \sum_{\ell=1}^{x_2} \pi_{n,m-1}(x) = \sum_{\substack{k,\ell=1 \\ k+\ell \le m-1}}^{m-1} \sum_{\substack{x \in \mathcal{X}_{n,m-1} \\ x_1 \ge k, x_2 \ge \ell}} \pi_{n,m-1}(x), \\ &= \frac{1}{Z_{n,m-1}} \sum_{\substack{k,\ell=1 \\ k+\ell \le m-1}}^{m-1} \sum_{\substack{x \in \mathcal{X}_{n,m-1} \\ x_1 \ge k, x_2 \ge \ell}} \prod_{i=1}^{n} \left(\frac{p_i}{\mu_i}\right)^{x_i} \overset{(*)}{=} \frac{1}{Z_{n,m-1}} \sum_{\substack{k,\ell=1 \\ k+\ell \le m-1}}^{m-1} \left(\frac{p_1}{\mu_1}\right)^k \left(\frac{p_2}{\mu_2}\right)^\ell \sum_{y \in \mathcal{X}_{n,m-1-k-\ell}} \prod_{i=1}^{n} \left(\frac{p_i}{\mu_i}\right)^{y_i}, \\ &= \frac{1}{Z_{n,m-1}} \sum_{\substack{k,\ell=1 \\ k+\ell \le m-1}}^{m-1} \left(\frac{p_1}{\mu_1}\right)^k \left(\frac{p_2}{\mu_2}\right)^\ell Z_{n,m-1-k-\ell},\end{aligned}$$

where $(*)$ follows by making the change of variable $y = x - ke_1 - \ell e_2$, where e_i is the n-dimensional vector with one in component i and zero elsewhere.

E Proof of Proposition 4.1

Proof of Equation (12) We start by proving Equation (12). Let the sequence $\{t_k, k \in \mathbb{N}\}$ denote the wall-clock time instants at which round k starts, and let $\{\xi_i(t), t \in \mathbb{R}_{\ge 0}\}$ represent the number of tasks at client i at (wall-clock) time t, for all $i \in \{1, \dots, n\}$. The throughput λ is defined as the average number of rounds completed per unit of (wall-clock) time. Mathematically, for a given time window $s \in \mathbb{R}_{>0}$, the throughput is expressed as:

$$\begin{aligned}\lambda &= \mathbb{E}\left[\frac{1}{s} \sum_{k \ge 0} \mathbf{1}[t_k \le s]\right] = \mathbb{E}\left[\frac{1}{s} \int_0^s \sum_{i=1}^{n} \mu_i \mathbf{1}[\xi_i(t) > 0] dt\right], \\ &= \sum_{i=1}^{n} \mu_i \mathbb{P}(\xi_i > 0) = \sum_{i=1}^{n} \mu_i \frac{p_i}{\mu_i} \frac{Z_{n,m-1}}{Z_{n,m}} = \frac{Z_{n,m-1}}{Z_{n,m}}.\end{aligned}$$

The first equality follows from the definition of throughput. The second applies the stochastic intensity formula [3, Section 1.8.3, "Stochastic Intensity Integration Formula"] to the point process $\{t_k\}_{k \in \mathbb{N}}$, whose stochastic intensity is $\sum_{i=1}^{n} \mu_i \mathbf{1}[\xi_i(t) > 0]$. The third equality follows by interchanging expectation and integration, using the linearity of expectation and the stationarity of ξ. The fourth uses the expression for $\mathbb{P}(\xi_i > 0)$ from [6, Equation (6)]. This completes the proof.

Proof of Equation (11) Let $Y_k = (Y_{1,k}, Y_{2,k}, \ldots, Y_{n,k})$, where $k \in \mathbb{N}$, represents the state of the network at round k. The sequence $\{Y_k, k \in \mathbb{N}\}$ can be recursively defined as follows:

- Y_0 represents the post-jump state at $t = 0$, i.e., for all $i \in \{1, \ldots, n\}$, $Y_{i,0} = X_{i,0} + \mathbf{1}[C_0 = i]$.
- For each $k \in \mathbb{N}_{>0}$ and $i \in \{1, \ldots, n\}$, $Y_{i,k} = Y_{i,k-1} + \mathbf{1}[A_k = i] - \mathbf{1}[C_{k-1} = i]$.

We know that the sequence $\{\tau_k | Y_k\}_{k \in \{0,\ldots,T\}}$ consists of independent random variables, such that $\tau_k | Y_k$ is exponentially distributed with a parameter $\sum_{i=1}^n \mu_i \mathbf{1}\{Y_{i,k} \geq 1\}$, for each $k \in \{0, 1, \ldots, T\}$. Conditioning on Y_k, we can write for all $k \in \{0, 1, \ldots, T\}$:

$$\begin{aligned}\mathbb{E}[\tau_k] &= \sum_{x \in \mathcal{X}_{n,m}} \mathbb{E}[\tau_k | Y_k = x] \mathbb{P}(Y_k = x), \\ &= \sum_{x \in \mathcal{X}_{n,m}} \frac{1}{\sum_{i=1}^n \mu_i \mathbf{1}\{x_i \geq 1\}} \mathbb{P}(Y_k = x).\end{aligned}$$

The sequence $\{Y_k, k \in \mathbb{N}\}$ forms an ergodic, discrete-time, homogeneous Markov chain. Its stationary distribution can be derived by noting that it corresponds to the jump chain of the ergodic, continuous-time Markov chain $\{\xi(t), t \geq 0\}$, whose stationary distribution is $\pi_{n,m}$. Using Equation (13.56) from Theorem 13.4.5 in [4], the stationary distribution of $\{Y_k, k \in \mathbb{N}\}$ is given by:

$$\mathbb{P}(Y_k = x) = \frac{1}{V_{n,m}} \sum_{i=1}^n \mu_i \mathbf{1}\{x_i \geq 1\} \prod_{j=1}^n \left(\frac{p_j}{\mu_j}\right)^{x_j}, \quad x \in \mathcal{X}_{n,m}, \tag{22}$$

where $V_{n,m}$ is a normalizing constant.

Under the stationarity assumption (**A5**) and substituting (22) into the expression previously derived for $\mathbb{E}[\tau_k]$, we obtain:

$$\mathbb{E}[\tau_k] = \sum_{x \in \mathcal{X}_{n,m}} \frac{1}{V_{n,m}} \prod_{j=1}^n \left(\frac{p_j}{\mu_j}\right)^{x_j} \overset{(*)}{=} \frac{Z_{n,m}}{V_{n,m}}. \tag{23}$$

Next, we demonstrate that $V_{n,m} = Z_{n,m-1}$. By definition, we have:

$$V_{n,m} = \sum_{x \in \mathcal{X}_{n,m}} \sum_{\substack{i=1 \\ x_i \geq 1}}^n \mu_i \prod_{j=1}^n \left(\frac{p_j}{\mu_j}\right)^{x_j} \overset{(a)}{=} \sum_{i=1}^n \mu_i \sum_{\substack{x \in \mathcal{X}_{n,m} \\ x_i \geq 1}} \prod_{j=1}^n \left(\frac{p_j}{\mu_j}\right)^{x_j} \overset{(b)}{=} \sum_{i=1}^n p_i \sum_{\substack{x \in \mathcal{X}_{n,m} \\ x_i > 0}} \prod_{j=1}^n \left(\frac{p_j}{\mu_j}\right)^{(x-e_i)_j},$$

where, for each $i \in \{1, \ldots, n\}$, e_i denotes the n-dimensional vector with one in component i and zero elsewhere. (a) is obtained by rearranging the order of summation, while (b) follows from factoring out $\frac{p_i}{\mu_i}$. Applying the variable substitution $y = x - e_i$, the expression simplifies as:

$$V_{n,m} = \underbrace{\sum_{i=1}^n p_i}_{=1} \underbrace{\sum_{y \in \mathcal{X}_{n,m-1}} \prod_{j=1}^n \left(\frac{p_j}{\mu_j}\right)^{y_j}}_{=Z_{n,m-1}} = Z_{n,m-1}.$$

Thus, incorporating this result into equality $(*)$ of (23), and using (12), we conclude that for all $k \in \{0, 1, \ldots, T\}$:

$$\mathbb{E}[\tau_k] = \frac{Z_{n,m}}{Z_{n,m-1}} = \frac{1}{\lambda}. \tag{24}$$

Finally, we get:

$$\frac{1}{T+1} \sum_{t=0}^T \bar{\tau}_t \mathbb{E}\left[\|\nabla f(w_t)\|^2\right] = \frac{1}{\lambda} \frac{1}{T+1} \sum_{t=0}^T \mathbb{E}\left[\|\nabla f(w_t)\|^2\right] \leq \frac{1}{\lambda} 8G$$

F Proof of Proposition 4.2

As recalled in Section 3.1, [21, Theorem 1] provides the following upper bound on the ergodic mean of the squared gradient norm of f: there exists a constant $\eta_{\max} > 0$ (which depends on p) such that, for any $\eta \in (0, \eta_{\max})$,

$$\frac{1}{T+1} \sum_{t=0}^T \mathbb{E}[\|\nabla f(w_t)\|^2] \leq 8G, \quad \text{where} \quad G = \frac{A}{\eta(T+1)} + \frac{\eta LB}{n^2} \sum_{i=1}^n \frac{1}{p_i} + \frac{\eta^2 L^2 Bm}{n^2} \sum_{i=1}^n \frac{\mathbb{E}[D_i]}{p_i^2}.$$

Assume the learning rate is a function of the number of rounds T, i.e., $\eta = \frac{C}{T^\alpha}$ for some constants $\alpha, C \in \mathbb{R}_{>0}$ such that $\alpha < 1$ and $\eta < \eta_{\max}$. Then we can express G as:

$$G = \frac{A}{CT^{-\alpha}(T+1)} + \frac{CLB}{n^2 T^\alpha} \sum_{i=1}^n \frac{1}{p_i} + \frac{C^2 L^2 Bm}{n^2 T^{2\alpha}} \sum_{i=1}^n \frac{\mathbb{E}[D_i]}{p_i^2}.$$

Therefore, to obtain

$$\frac{1}{T+1}\sum_{t=0}^{T}\mathbb{E}[\|\nabla f(w_t)\|^2] \leq \epsilon,$$

it suffices to take $T \geq T_\epsilon$, where

$$T_\epsilon = \mathcal{O}\left(\left(\frac{A}{C\epsilon}\right)^{\frac{1}{1-\alpha}} + \left(\frac{CLB}{\epsilon n^2}\sum_{i=1}^{n}\frac{1}{p_i}\right)^{\frac{1}{\alpha}} + \left(\frac{C^2L^2Bm}{\epsilon n^2}\sum_{i=1}^{n}\frac{\mathbb{E}[D_i]}{p_i^2}\right)^{\frac{1}{2\alpha}} \right) \text{ rounds.}$$

Using Equation (24), the expected wall-clock time to reach this accuracy is

$$\mathbb{E}[\tilde{\tau}_\epsilon] = \sum_{t=0}^{T_\epsilon - 1}\mathbb{E}[\tau_t] = \frac{T_\epsilon}{\lambda}.$$

G Compute G and $\nabla_p G$

Proposition G.1. *In the framework of Section 2.1, the expression of the upper bound $G(p)$ in terms of routing probabilities is given by:*

$$G(p) = \frac{A}{\eta(T+1)} + \frac{\eta LB}{n^2}\sum_{i=1}^{n}\frac{1}{p_i} + \frac{\eta^2L^2Bm}{n^2}\sum_{i=1}^{n}\sum_{k=1}^{m-1}\frac{p_i^{k-2}}{\mu_i^k}\frac{Z_{n,m-1-k}}{Z_{n,m-1}}, \tag{25}$$

where the normalizing constants $Z_{n,\mathfrak{m}}$ for $\mathfrak{m} \in \{0,1,\ldots,m-1\}$ can be computed explicitly with $\mathcal{O}(nm)$ time and $\mathcal{O}(m)$ memory complexity, as shown in Proposition 2.1.

To identify the best routing strategy, we aim to minimize this upper bound using a gradient descent algorithm. This requires computing the gradient of the function $G(p)$ with respect to the routing probability vector, as detailed in the following proposition, which follows from Theorem 3.1.

Proposition G.2. *For each $j \in \{1,\ldots,n\}$, we have*

$$\frac{\partial G}{\partial p_j} = \frac{\eta LB}{n^2 p_j}\left(-\frac{1}{p_j} + \eta mL\left(\sum_{i=1}^{n}\frac{\mathbb{E}[X_iX_j]}{p_i^2} - \mathbb{E}[X_j]\sum_{i=1}^{n}\frac{\mathbb{E}[X_i]}{p_i^2} - \frac{2\mathbb{E}[X_j]}{p_j^2}\right)\right),$$

where the random vector $X \in \mathcal{X}_{n,m-1}$ is distributed according to $\pi_{n,m-1}$, as defined in Equation (4). Moreover, for all $(i,j) \in \{1,\ldots,n\}^2$, the quantities $\mathbb{E}[X_j]$ and $\mathbb{E}[X_iX_j]$ can be computed using the formulas from Theorem 3.1, where the normalizing constants are efficiently computed in $\mathcal{O}(nm)$ time and $\mathcal{O}(m)$ memory complexity using Buzen's algorithm, as described in Proposition 2.1.

To enforce the probability constraints on $p = (p_i)_{i=1}^n$, we introduce auxiliary parameters $\Theta = (\theta_i)_{i=1}^n$ and apply the softmax transformation:

$$p_j = \frac{\mathrm{e}^{\theta_j}}{\sum_{i=1}^{n}\mathrm{e}^{\theta_i}}, \quad j \in \{1,\ldots,n\}.$$

This guarantees that the resulting p forms a valid probability vector.

The gradient of the upper bound G with respect to Θ is given by:

$$\frac{\partial G}{\partial \theta_j} = \left\langle \nabla_p G, \frac{\partial p}{\partial \theta_j} \right\rangle, \quad \text{where } \frac{\partial p}{\partial \theta_j} = p_j(e_j - p), \text{ for each } j \in \{1,\ldots,n\}.$$

Here, e_j denotes the n-dimensional unit vector with 1 in coordinate j, and $\langle\cdot,\cdot\rangle$ denotes the dot product in $\mathbb{R}^n$.

H Compute H and $\nabla_p H$

Proposition H.1. *In the framework of Section 2.1, the expression of the upper bound $H(p)$ in terms of routing probabilities is given by:*

$$H(p) = \frac{Z_{n,m}}{Z_{n,m-1}}\left(\frac{A}{\eta(T+1)} + \frac{\eta LB}{n^2}\sum_{i=1}^{n}\frac{1}{p_i} + \frac{\eta^2L^2Bm}{n^2}\sum_{i=1}^{n}\sum_{k=1}^{m-1}\frac{p_i^{k-2}}{\mu_i^k}\frac{Z_{n,m-1-k}}{Z_{n,m-1}}\right), \tag{26}$$

where the normalizing constants $Z_{n,\mathfrak{m}}$ for $\mathfrak{m} \in \{0,1,\ldots,m\}$ can be computed explicitly with $\mathcal{O}(nm)$ time and $\mathcal{O}(m)$ memory complexity, as shown in Proposition 2.1.

To find the optimal routing strategy, we aim to minimize this upper bound using a gradient descent algorithm. This requires the gradient of $H(p)$ with respect to the routing probabilities, given in the following proposition.

Proposition H.2. *For each $j \in \{1, \ldots, n\}$, the gradient $\nabla_p H$ with respect to the routing probability p_j is given by:*

$$\frac{\partial H}{\partial p_j} = \frac{A\mathbb{E}[\xi_j - X_j]}{(T+1)\eta\lambda p_j} + \frac{\eta LB}{n^2 p_j \lambda}\left(-\frac{1}{p_j} + \mathbb{E}[\xi_j - X_j]\sum_{i=1}^{n}\frac{1}{p_i} + \eta m L\left(-\frac{2\mathbb{E}[X_j]}{p_j^2} + \sum_{i=1}^{n}\frac{\mathbb{E}[X_i X_j]}{p_i^2} + \mathbb{E}[\xi_j - 2X_j]\sum_{i=1}^{n}\frac{\mathbb{E}[X_i]}{p_i^2}\right)\right),$$

where the random vectors $\xi \in \mathcal{X}_{n,m}$ and $X \in \mathcal{X}_{n,m-1}$ are distributed according to $\pi_{n,m}$ and $\pi_{n,m-1}$, respectively, as defined in Equation (4)*.*

Moreover, for all $(i,j) \in \{1, \ldots, n\}^2$, the quantities $\mathbb{E}[X_j]$, $\mathbb{E}[\xi_j]$, and $\mathbb{E}[X_i X_j]$ can be computed using the formulas from Theorem 3.1, where the normalizing constants are efficiently computed in $\mathcal{O}(nm)$ time and $\mathcal{O}(m)$ memory complexity using Buzen's algorithm, as described in Proposition 2.1.

The proxy objective H is optimized using the same softmax reparameterization as in Section G.

I Experiments Details

I.1 Neural Networks Architectures

The Fashion-MNIST and KMNIST datasets each contain 70,000 grayscale images, with 60,000 designated for training and 10,000 reserved for testing. These images, sized at 28×28 pixels, are evenly distributed across 10 classes. On the other hand, the CIFAR-10 and CIFAR-100 datasets feature 60,000 color images (RGB) with a resolution of 32×32 pixels. Of these, 50,000 are used for training, while 10,000 are allocated for testing. CIFAR-10 is organized into 10 classes, while CIFAR-100 expands to 100 classes. Notably, all these datasets are class-balanced, ensuring each class has an equal number of images.

For the Fashion-MNIST and KMNIST datasets, we employ a convolutional neural network (CNN) with the following structure:

- Two convolutional layers with 7x7 filters, each followed by a ReLU activation function. The first convolutional layer has 20 channels, while the second has 40 channels.
- A 2x2 max pooling layer.
- A final fully connected layer with 10 neurons, concluded by a softmax activation function.

For the CIFAR-10 and CIFAR-100 datasets, we employ a convolutional neural network (CNN) with the following architecture:

- **Three sequential convolutional blocks**, each consisting of two 3×3 convolutional layers. Each layer is followed by ReLU activation and Group Normalization. The configuration of these blocks is as follows:
  - The first block contains convolutional layers with 32 channels.
  - The second block contains convolutional layers with 64 channels.
  - The third block contains convolutional layers with 128 channels.

  Additionally, each block includes 2×2 max pooling and a dropout layer with a probability of 0.25.

- **A classification block** comprising:
  - A flattening layer.
  - A fully connected layer with 128 neurons.
  - A dropout layer with a probability of 0.25.
  - A final fully connected layer with the number of neurons corresponding to the number of classes (10 for CIFAR-10 and 100 for CIFAR-100), followed by a softmax activation function.

In all experiments, the stochastic gradient for each task is computed using a batch size of 512 data points. The numerical implementation is carried out in PyTorch, and the experiments are performed on an NVIDIA Tesla P100 GPU.

I.2 Closed Jackson Network Simulation

For each $t \in \{0, \ldots, T\}$, recall that Y_t denote the n-dimensional random variable representing the queue lengths during round t, and τ_t represent the duration (in wall-clock time) of round t. The Jackson network simulation proceeds by initializing Y_0, then iterating through the following steps for each round $t \in \{0, \ldots, T\}$:

- Sample τ_t from an exponential distribution with rate parameter $\sum_{j=1}^{n} \mu_j \mathbf{1}\{Y_{j,t} > 0\}$, where μ_j represents the processing speed of client j and $Y_{j,t}$ denotes the queue length of client j at round t.
- Select a client k to complete a task at the end of round t from the set of non-empty queues. The probability of selecting client k is proportional to the processing speeds of the clients, given by the distribution:

$$\left(\frac{\mu_i \mathbf{1}\{Y_{i,t} > 0\}}{\sum_{j=1}^{n} \mu_j \mathbf{1}\{Y_{j,t} > 0\}}, i \in \{1, \ldots, n\}\right).$$

- Reassign the task to another client l based on the routing distribution $(p_i, i \in \{1, \ldots, n\})$. The queue lengths are then updated as follows:

$$Y_{t+1} = Y_t - e_k + e_l,$$

where e_k and e_l are unit vectors indicating that the task is removed from client k and added to client l, respectively. This concludes round t and initiates the next round.

J Additional Experiments

J.1 Additional Experiments for Section 3.3

To evaluate the robustness of the optimized routing strategy p_G^* with respect to the proxy objective G under challenging heterogeneous data distributions, and to assess its sensitivity to different computation time distributions, we conducted additional experiments on the Fashion-MNIST, CIFAR-10, and CIFAR-100 datasets.

J.1.1 Robustness to Data Distribution Heterogeneity

Highly heterogeneous data distribution This scenario follows the setup of Section 3.3, but with a different data partitioning across clients. For Fashion-MNIST and CIFAR-10, each client was assigned 3 unique image labels out of 10, distributed sequentially and cyclically. For example, the first client received labels 0, 1, and 2; the second, labels 3, 4, and 5; and so on, wrapping around after label 9. The same procedure was applied to CIFAR-100, but with 5 labels per client.

This setup creates a non-independent and identically distributed (iid) data distribution, as each client can only access a small subset of labels. All clients were given an equal number of images.

We compare the test accuracy and loss trajectories of the optimized, uniform, and balanced routing strategies. To eliminate the impact of initialization, all neural networks were initialized with the same weights across routing strategies. For each dataset and strategy, we performed multiple independent simulations using different random seeds and recorded test performance.

The results in Figure 5 show that the optimized routing strategy ultimately achieves the highest accuracy, with a notably faster performance gain compared to the other methods. In contrast, the uniform and balanced routing strategies exhibit greater volatility across independent simulations, along with more frequent loss spikes.

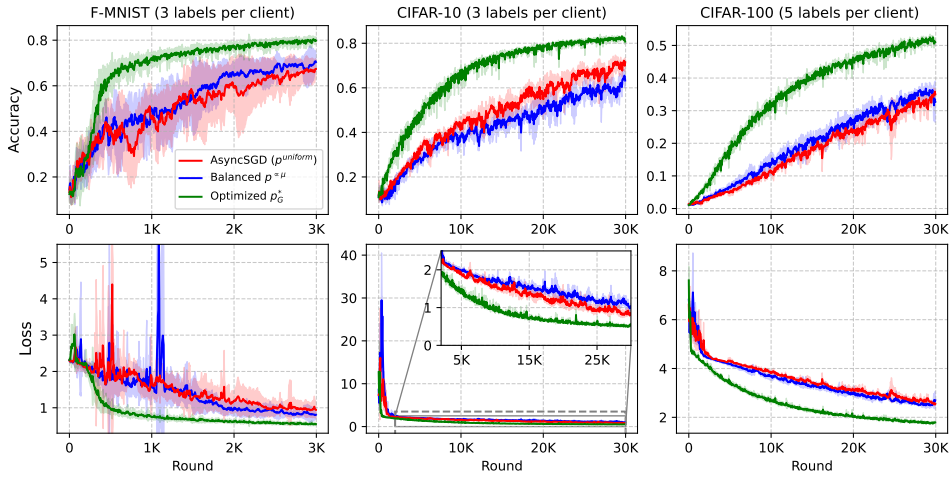

Figure 5. Performance on the test set at the CS in the scenario of Section 3.3, with $n = 20$ clients and $m = 100$ tasks under highly heterogeneous data splits. For Fashion-MNIST, we simulated training over 3,000 rounds, repeated 10 times, recording accuracy and loss every 5 rounds. For CIFAR-10 and CIFAR-100, we applied standard normalization and data augmentation, ran each simulation for 30,000 rounds, and repeated it three times, logging metrics every 50 rounds on an unseen test set. Solid lines show metrics averaged over independent runs; shaded areas represent standard deviations.

Disjoint datasets To further validate these findings, we examined another challenging data heterogeneous setting involving a system with 10 clients and 100 tasks. In this scenario, each image label from the 10 labels in the Fashion-MNIST dataset is assigned exclusively to a single client, ensuring that each client has a completely distinct data distribution. Additionally, the service speed of each client i is defined as $\mu_i = \mathrm{e}^{i/50}$, resulting in the fastest client being approximately 20% faster than the slowest. The learning rate is set to $\eta = 0.005$, and the smoothness constant is $L = 1$.

In Figure 6, we compare the evolution of accuracy and loss under the optimized routing strategy against the uniform and balanced routing strategies in this more complex environment. Similar to the previous experiment, training spans 3,000 rounds, with metrics recorded on a test dataset at intervals of 5 rounds.

Our results further validate that the optimized routing strategy p_G^* consistently surpasses the alternatives p^{uniform} and $p^{\propto\mu}$, achieving superior accuracy with a consistently faster performance gain. Furthermore, it demonstrates greater robustness and stability in this highly heterogeneous environment, showing a smaller standard deviation in both loss and accuracy across the 10 independent simulations, along with fewer loss

spikes compared to the uniform and balanced strategies. Additionally, although the optimized routing often selects the slowest client more frequently, the learning process does not develop a bias toward optimizing the local loss of this client. Instead, it preserves balanced global performance, effectively avoiding overfitting to the image labels associated with any specific client.

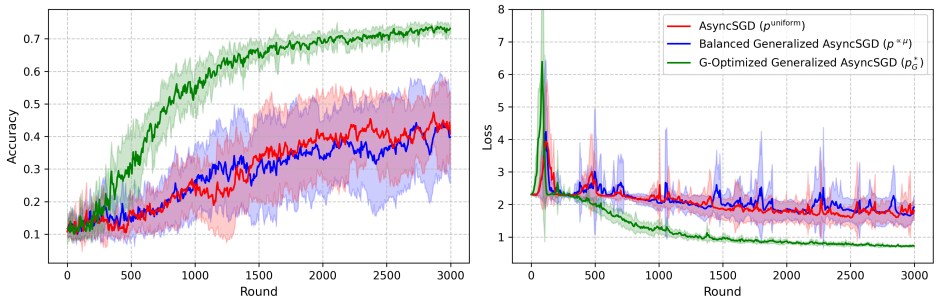

Figure 6. Performance on the test set at the CS with 10 clients and 100 tasks under heterogeneous data splits. Solid lines show metrics averaged over independent simulations; shaded areas represent standard deviations.

J.1.2 Robustness to Computation Time Distributions

To evaluate the robustness of the proposed approach under different computation time distributions, we retain the setup from Section 3.3 but modify the distribution, while maintaining the same average task completion rate at each client.

We study two scenarios: (i) deterministic computation times, where computations at each client i have a fixed duration equal to $\frac{1}{\mu_i}$; and (ii) lognormal computation times, a heavy-tailed distribution used to model realistic processing delays in distributed systems. In the latter case, the mean is set to $\frac{1}{\mu_i}$ for each client i, and the standard deviation of the underlying normal distribution is fixed at $\sigma_s = 1$, ensuring a constant coefficient of variation across clients. For all $i \in \{1, \ldots, n\}$, the values of μ_i match those used in the exponential setting of Section 3.3.

We compare the optimized routing strategy p^*_G (computed as in Section 3.3) with the uniform and balanced strategies under both homogeneous and heterogeneous data settings. Heterogeneity is introduced using Dirichlet-based sampling ($\text{Dir}_n(0.5)$) or disjoint label partitions across clients. For Fashion-MNIST, training was simulated for 3,000 rounds and repeated 10 times, with accuracy and loss recorded every 5 rounds. For CIFAR-10 and CIFAR-100, we applied standard normalization and data augmentation, ran each simulation for 30,000 rounds, repeated three times, and logged performance every 50 rounds on an unseen test set. All neural networks were initialized with the same weights across routing strategies to ensure fair comparison.

The results are shown in Figures 7 and 8, where solid lines indicate averages over independent runs and shaded regions represent standard deviations. They indicate that performance is largely insensitive to the choice of computation time distribution, with trends closely matching those under the exponential assumption. The G-optimized routing consistently achieves the highest accuracy, with faster performance gain and lower volatility in both accuracy and loss trajectories.

J.2 Additional Experiments for Section 4.3

To further assess the effectiveness of the H-optimized routing strategy with respect to wall-clock performance, we conduct additional experiments. Section J.2.1 evaluates its robustness across different datasets (CIFAR-10 and CIFAR-100) and under more challenging heterogeneous data distributions. Section J.2.2 examines its sensitivity to different computation time distributions.

J.2.1 Robustness to Data Distribution Heterogeneity

This experiment builds on the setup of Section 4.3, extending the evaluation to the CIFAR-10 and CIFAR-100 datasets. We assess the performance of the H-optimized routing strategy in terms of wall-clock time across three data distribution scenarios:

- The first two match those in Section 4.3: a **homogeneous** setting where data are i.i.d. across clients, and a **heterogeneous** setting where data are distributed according to a Dirichlet distribution ($\text{Dir}_n(0.5)$). KMNIST results for these settings are already presented in Section 4.3.
- The third is a **highly heterogeneous** scenario, similar to that in Section J.1.1, where each client sees only a limited subset of labels. Specifically, for KMNIST and CIFAR-10, each client receives 3 unique labels out of 10; for CIFAR-100, 31 labels out of 100 are assigned per client, distributed sequentially and cyclically.

We adopt the same network dynamics as in Section 4.3 and compare the evolution of test accuracy and loss under four routing strategies: uniform, balanced, G-optimized, and H-optimized. To control for initialization effects, all models were initialized with the same weights across strategies. For each dataset and routing strategy, we ran 5 independent simulations with different random seeds and evaluated performance on an unseen, label-balanced test set.

The results in Figure 9 support the conclusions of Section 4.3. The H-optimized and balanced strategies perform roughly similarly in homogeneous settings. However, when data are heterogeneous, the H-optimized routing consistently outperforms the alternatives, achieving higher accuracy and demonstrating greater robustness and stability. In contrast, the balanced routing strategy becomes unstable in both heterogeneous and highly heterogeneous settings for KMNIST and CIFAR-10.

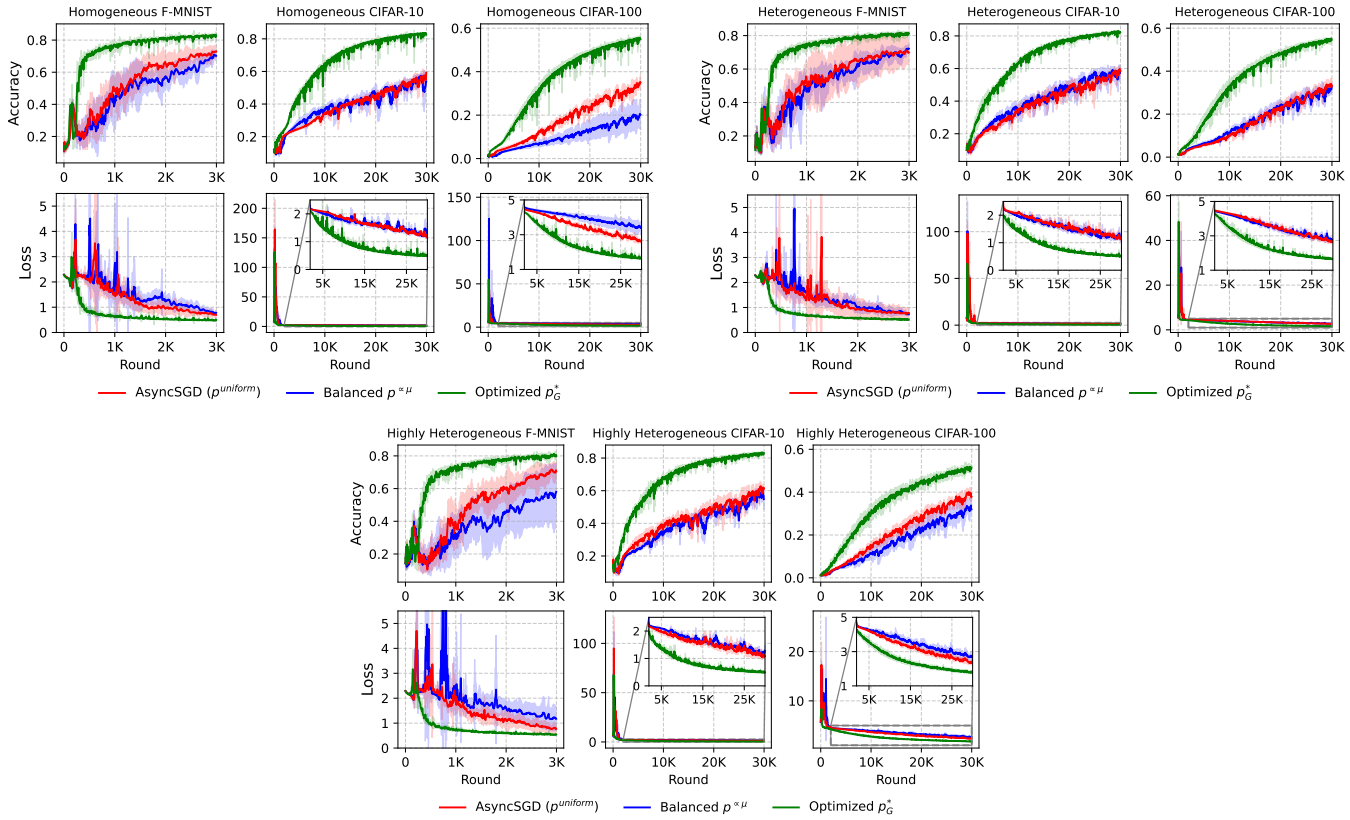

Figure 7. Performance on the test set at the CS in the scenario of Section 3.3, with $n = 20$ clients and $m = 100$ tasks. Results are reported under **deterministic computation times** for three data distribution regimes: homogeneous, heterogeneous (Dirichlet), and highly heterogeneous.

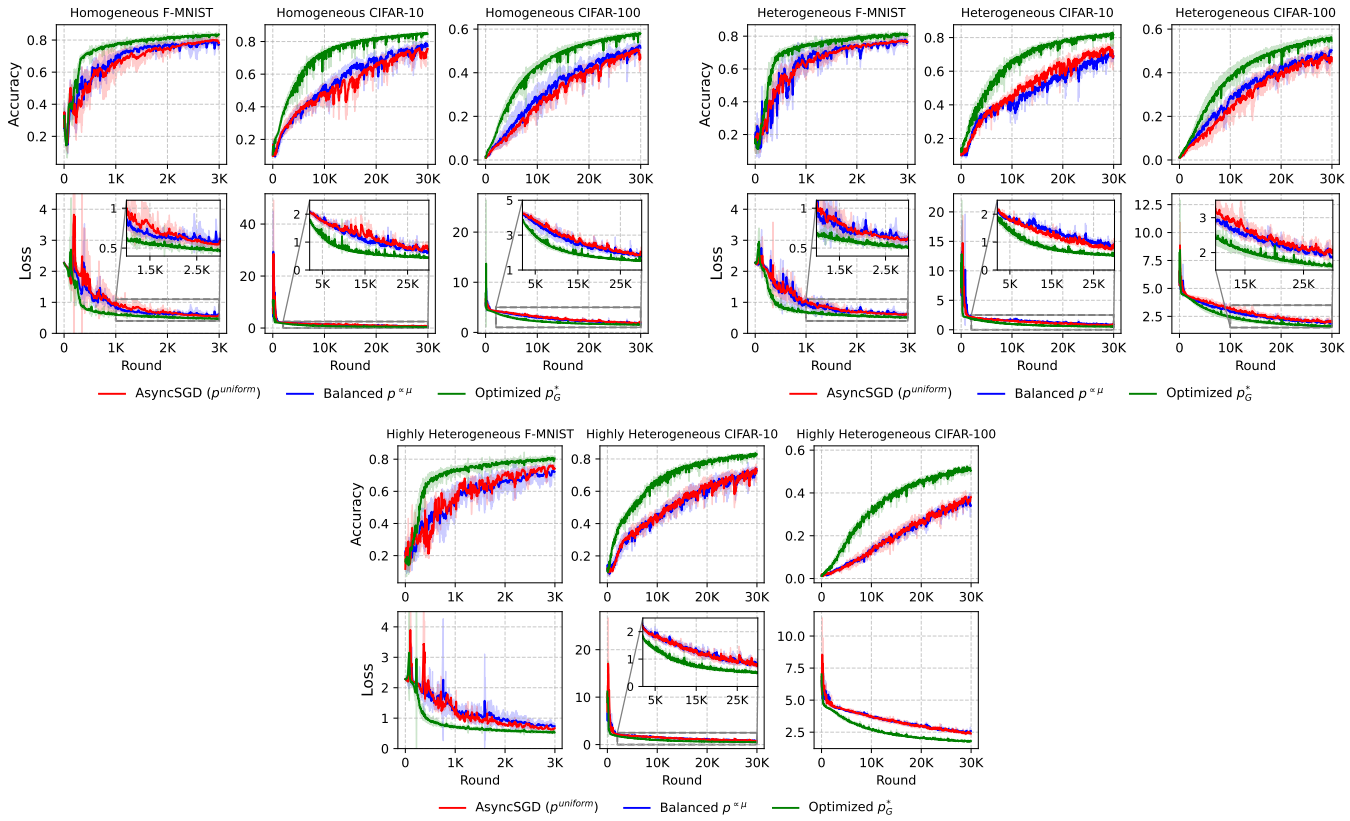

Figure 8. Performance on the test set at the CS in the scenario of Section 3.3, with $n = 20$ clients and $m = 100$ tasks. Results are reported under **lognormal computation times** for three data distribution regimes: homogeneous, heterogeneous (Dirichlet), and highly heterogeneous.

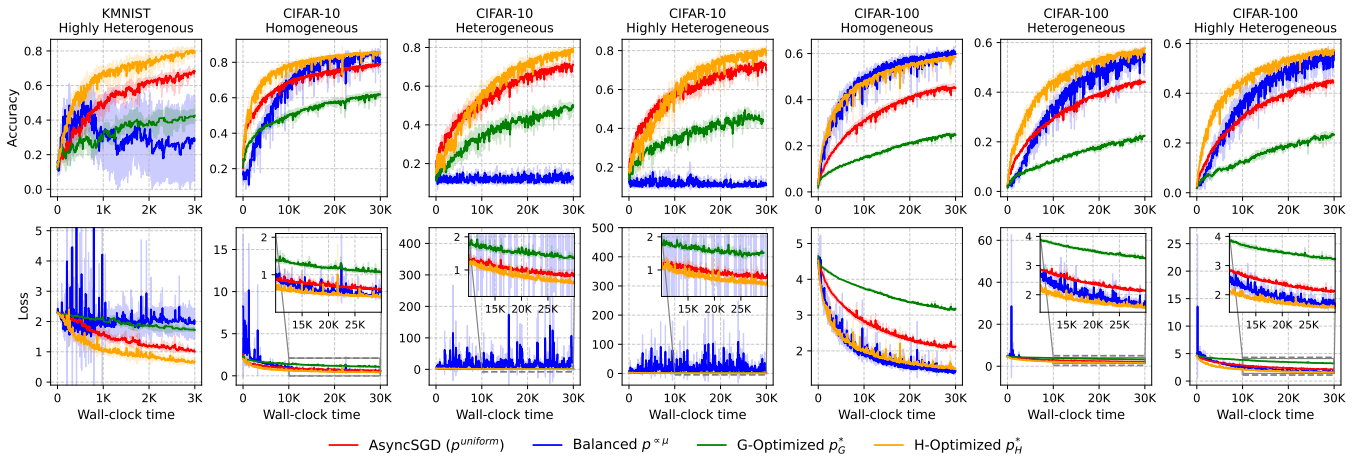

Figure 9. Performance on the test set with respect to wall-clock time at the CS in the scenario of Section 4.3, using $n = 30$ clients and $m = 30$ tasks, under homogeneous, heterogeneous, and highly heterogeneous data distributions. Solid lines denote averages over independent runs; shaded areas indicate standard deviations.

J.2.2 Robustness to Computation Time Distributions

To evaluate the robustness of H-optimized routing under varying computation time distributions, we retain the setup from Section 4.3, modifying only the distribution of computation times while keeping the average task completion rate per client fixed.

As before, we study two scenarios: (i) deterministic times, where each client i has fixed duration $\frac{1}{\mu_i}$; and (ii) lognormal times, a heavy-tailed distribution modeling realistic delays. In the lognormal case, the mean is set to $\frac{1}{\mu_i}$ and the standard deviation of the underlying normal distribution is fixed at $\sigma_s = 1$ to ensure constant coefficient of variation across clients. The μ_i values match those from the exponential case in Section 4.3.

We compare the H-optimized routing strategy p_H^* (computed as in Section 4.3) with uniform, balanced, and G-optimized strategies under homogeneous and heterogeneous data settings for KMNIST and CIFAR-100. Heterogeneity is introduced via Dirichlet sampling ($\mathrm{Dir}_n(0.5)$) or disjoint label partitions (see Section J.2.1). KMNIST is trained for 3,000 wall-clock time units over 5 runs; CIFAR-100 uses standard preprocessing, is trained for 30,000 wall-clock time units, and repeated 3 times. Accuracy and loss are logged on a label-balanced test set. All models share the same initialization to ensure fair comparison.

The results in Figures 10 and 11 show that performance is largely insensitive to the choice of computation time distribution, with trends closely aligning with those observed under the exponential assumption. Across all configurations, H-optimized routing effectively balances update frequency and gradient staleness.

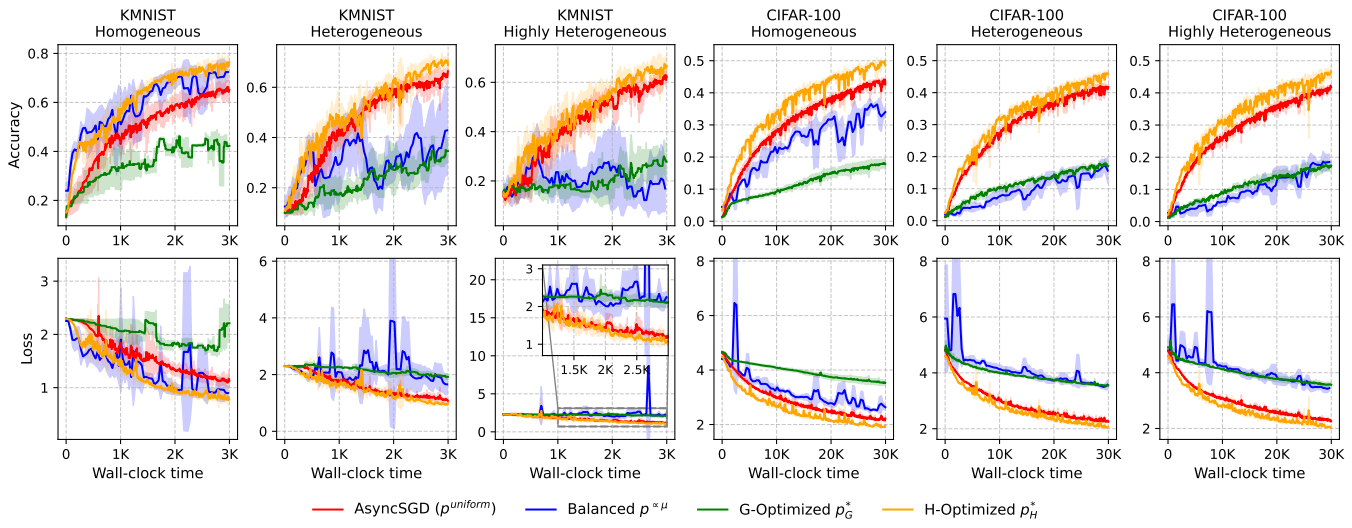

Figure 10. Performance on the test set with respect to wall-clock time at the CS in the scenario of Section 4.3, with $n = 30$ clients and $m = 30$ tasks. Results are reported for the KMNIST and CIFAR-100 datasets under **deterministic computation times**, across three data distribution regimes: homogeneous, heterogeneous, and highly heterogeneous. Solid lines denote averages over independent runs; shaded areas indicate standard deviations.

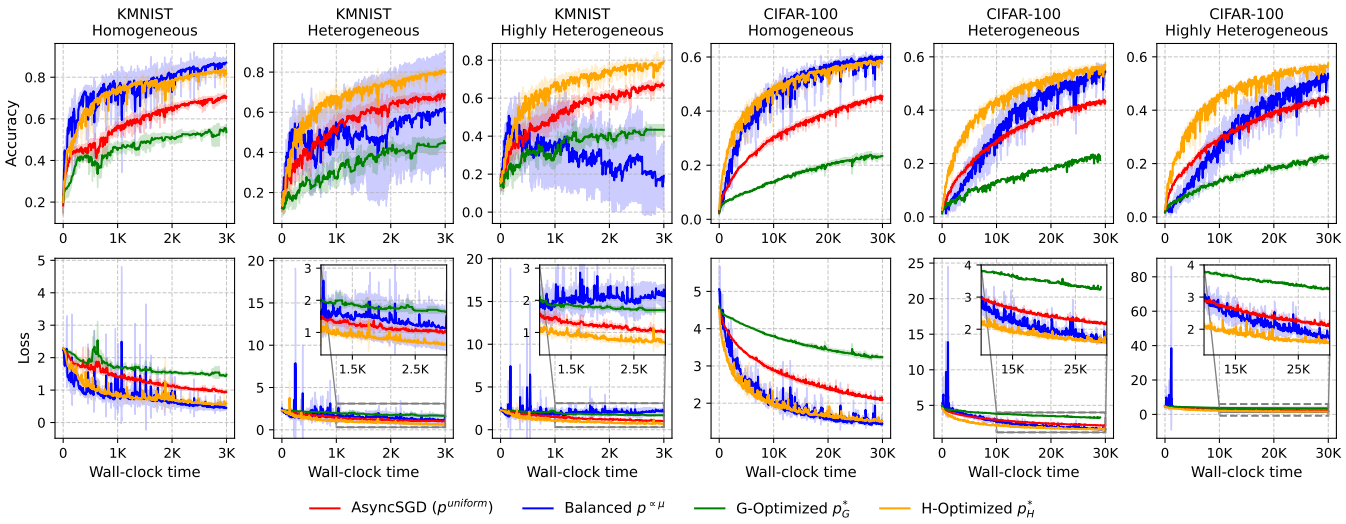

Figure 11. Performance on the test set with respect to wall-clock time at the CS in the scenario of Section 4.3, with $n = 30$ clients and $m = 30$ tasks. Results are reported for the KMNIST and CIFAR-100 datasets under **lognormal computation times**, across three data distribution regimes: homogeneous, heterogeneous, and highly heterogeneous. Solid lines denote averages over independent runs; shaded areas indicate standard deviations.