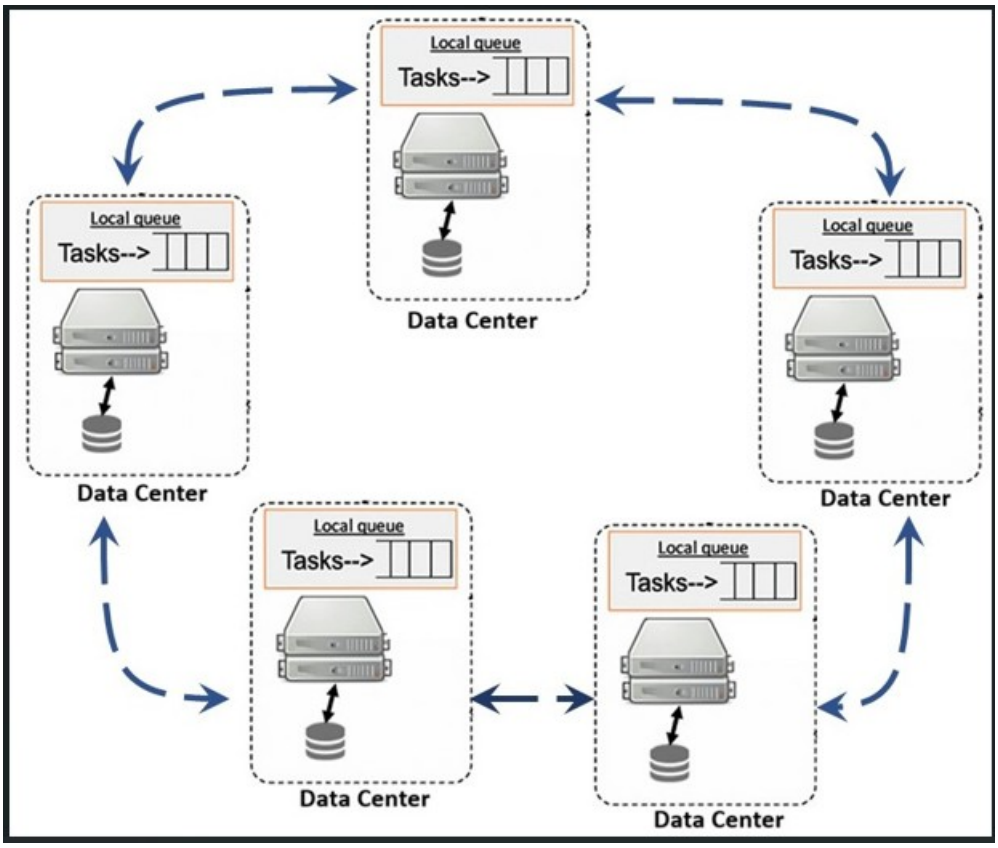
# Graphical Abstract

**Machine Learning and CPU (Central Processing Unit) Scheduling Co-Optimization over a Network of Computing Centers**

Mohammadreza Doostmohammadian, Zulfiya R. Gabidullina, Hamid R. Rabiee



# Highlights

**Machine Learning and CPU (Central Processing Unit) Scheduling Co-Optimization over a Network of Computing Centers**

Mohammadreza Doostmohammadian, Zulfiya R. Gabidullina, Hamid R. Rabiee

- Simultaneous optimization of data processing and computing resource allocation over a network of computing nodes.
- Preserving resource-demand constraint feasibility at all iterations
- Addressing log-scale quantization for limited networking traffic setups.
- Perturbation-based convergence analysis for time-varying data-sharing networks.
- Applicability in distributed support-vector machines and regression models.

# Machine Learning and CPU (Central Processing Unit) Scheduling Co-Optimization over a Network of Computing Centers

Mohammadreza Doostmohammadian[a], Zulfiya R. Gabidullina[b], Hamid R. Rabiee[c]

[a]*Mechatronics Group Faculty of Mechanical Engineering Semnan University Semnan Iran and Center for International Scientific Studies and Collaborations Tehran Iran doost@semnan.ac.ir.*
[b]*Institute of Computational Mathematics and Information Technologies Kazan Federal University Russia Zulfiya.Gabidullina@kpfu.ru.*
[c]*Computer Engineering Department Sharif University of Technology Tehran Iran rabiee@sharif.edu.*

**Abstract**

In the rapidly evolving research on artificial intelligence (AI) the demand for fast, computationally efficient, and scalable solutions has increased in recent years. The problem of optimizing the computing resources for distributed machine learning (ML) and optimization is considered in this paper. Given a set of data distributed over a network of computing-nodes/servers, the idea is to optimally assign the CPU (central processing unit) usage while simultaneously training each computing node locally via its own share of data. This formulates the problem as a co-optimization setup to (i) optimize the data processing and (ii) optimally allocate the computing resources. The information-sharing network among the nodes might be time-varying, but with balanced weights to ensure consensus-type convergence of the algorithm. The algorithm is all-time feasible, which implies that the computing resource-demand balance constraint holds at all iterations of the proposed solution. Moreover, the solution allows addressing possible log-scale quantization over the information-sharing channels to exchange log-quantized data. For some example applications, distributed support-vector-machine (SVM) and regression are considered as the ML training models. Results from perturbation theory, along with Lyapunov stability and eigen-spectrum analysis, are used to prove the convergence towards the optimal case. As compared

*Preprint submitted to Engineering Applications of Artificial Intelligence October 30, 2025*

to existing CPU scheduling solutions, the proposed algorithm improves the cost optimality gap by more than 50%.

*Keywords:* Distributed optimization, networked data centres, consensus, computing resource scheduling

## 1. Introduction

Data processing and distributed machine-learning (ML) over a network of computing centers has been recently motivated by advances in cloud/edge-computing [1], multi-agent learning systems [2, 3], Internet-of-Things (IoT) applications [4], and cyber-physical-systems (CPS) [5]. In this direction, *distributed* and *decentralized* algorithms are proposed to distribute the computational/processing loads among a group of computing nodes to advance the traditional centralized solutions in terms of scalability, robustness to single-node-of-failure, and reliability. Such algorithms allow for combining machine learning and data filtering models across a network of interconnected nodes/agents and cloud-based platforms [6], enabling parallel processing and accelerated decision-making [7, 8]. In these setups, one main problem is *how to efficiently/optimally schedule the computational resources among these computing nodes while performing optimization-based machine learning and data filtering tasks*. The synergy between CPU scheduling and learning from data offers promising solutions for optimizing computing performance in distributed ML setups.

Some related application-based works in the literature are discussed next. A relevant problem of interest is mixed-integer linear programming with applications to scheduling [9] and scalable task assignment [10], which is further extended to asynchronous coordination via primal-dual formulation in [11]. Distributed task allocation to reach consensus on duty to capability ratio [12] and facility location optimization for discrete coverage algorithms over a convex polygon [13] via swarm robotic networks are also based on distributed scheduling and optimization algorithms. On the other hand, coordination of CPUs over networked data centres using both centralized algorithms [14, 15] and quantized consensus-based distributed algorithms [16] is considered in the literature. Distributed resource allocation and optimization algorithms with applications in energy system management are also discussed in the literature; e.g., see relevant works on convex quadratic relaxations of the AC power flow [17], distributed economic dispatch to coor-

dinate generators and iteratively share local information to determine their outputs that minimizes total production cost while meeting demand and operational constraints [18], distributed generator scheduling to cooperatively manage energy production across a grid network to optimize reliability, cost, and grid services while respecting local constraints [19, 20], and distributed automatic generation control for power regulation tasks to achieve system-wide load-generation balance without centralized control [21, 22]. Another concern in the literature is non-ideal data sharing due to quantization. In this direction, a distributed subgradient method with adaptive (or dynamic) quantization is proposed in [23] while traditional static quantization with a fixed quantization level is considered for consensus [24], learning [25], and optimization [26]. One main challenge in the uniformly quantized setup is how to mitigate the quantization error. In this direction, one solution is to use *log-scale* quantization instead. Another concern is the notion of constraint feasibility in distributed optimization. All-time feasibility is crucial to ensure that the (computing) resource-demand balance always holds. This is in contrast to primal-dual-based alternating direction method of multipliers (ADMM) formulations that reach resource-demand feasibility asymptotically [27, 28, 29, 20]. What is missing in the existing literature is a unified framework to simultaneously address all-time feasible scheduling of the computing resources along with distributed learning and optimization while addressing log-scale quantization.

The main contributions of this work are as follows. Given a co-optimization problem for CPU scheduling and ML optimization, we provide a distributed algorithm to optimally allocate the computing resources while simultaneously solving the distributed optimization. This coordination algorithm is based on consensus mechanism and gradient descent tracked by introducing an auxiliary variable. We further address the logarithmic quantization of the data shared over the data-sharing network of computing-nodes/agents. Using perturbation theory and Lyapunov-based eigenspectrum analysis, we prove convergence to the optimal case in the presence of log-scale quantized information exchange. Further, the algorithm allows to consider change in the topology of the networked agents without violating the convergence of the solution. Moreover, we show that the algorithm is all-time feasible, implying that the (computing) resource-demand holds at all times along the solution. This implies that at any termination time of the algorithm, there is no violation in the balance between CPU resources and computing demand. We verify the feasibility and optimal convergence by extensive simulations

for different ML applications. Comparisons with the existing literature are also provided by the simulations.

*Paper Organization:* Section 2 formulates the problem in mathematical form. Section 3 provides the main algorithm to solve the problem. Section 4 provides the analysis of convergence, optimality, and feasibility. Section 5 presents the simulation results. Finally, Section 6 concludes the paper.

*Notations:* $I_m$ is the identity matrix of size $m$. $\mathbf{1}_n$ (or $\mathbf{0}_n$) is the column vector of all ones (or zeros) of size $n$. Similarly, $\mathbf{0}_{n\times n}$ denotes zero square matrix of size $n$. Operator $\otimes$ denotes the Kronecker product. Operator ";" in vectors implies column concatenation.

## 2. Problem Formulation

The optimization problem in this paper is two-fold: (i) optimizing the allocation of CPU resources and (ii) optimizing the loss function associated with the machine learning application.

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y}} \ & \sum_{i=1}^{n} f_i(\mathbf{x}_i) + g_i(\mathbf{y}_i) \\ \text{s.t.} \ & \mathbf{y}_1 = \mathbf{y}_2 = \cdots = \mathbf{y}_n \\ & \sum_{i=1}^{n} \mathbf{x}_i = b \\ & \mathbf{x}_i \in \Xi_i \end{aligned} \tag{1}$$

with $\mathbf{y}_i \in \mathbb{R}^m$ as the ML parameter states, $g_i : \mathbb{R}^m \to \mathbb{R}$ as the ML loss function, $\mathbf{x}_i \in \mathbb{R}$ as the assigned CPU resources, $f_i : \mathbb{R} \to \mathbb{R}$ as the cost of allocated CPU to node $i$, and $\Xi_i \subseteq \mathbb{R}$ representing a range of admissible values for CPU states, all at computing node $i$. In general, $\Xi_i = \{\mathbf{x} \in \mathbb{R} : h_i^j(\mathbf{x}) \leq 0, j = 1, \ldots, p_i\}$ with $h_i^j : \mathbb{R} \to \mathbb{R}$ as convex and smooth functions on $\Xi_i$. One simple example of the latter is the so-called box constraints $\mathbf{x}_i \in [m_i \ M_i]$. Parameter $b$ denotes the overall available CPU resources to all nodes. The global state vectors are defined as $\mathbf{x} := [\mathbf{x}_1; \ldots; \mathbf{x}_n]$ and $\mathbf{y} := [\mathbf{y}_1; \ldots; \mathbf{y}_n]$.

**Assumption 1.** *Local CPU cost function $f_i(\cdot)$ is strictly convex and smooth. The ML loss function $g_i(\cdot)$ is (possibly) non-convex and smooth, satisfying:*

$$(\mathbf{1}_n \otimes I_m)^\top H (\mathbf{1}_n \otimes I_m) \succ 0. \tag{2}$$

*with $H := diag[\nabla^2 g_i(\mathbf{y}_i)] \preceq L I_{mn}$.*

**Remark 1.** *The CPU cost is typically modelled as a quadratic function as described in [14, 15, 16], which satisfies Assumption 1. The examples for ML loss functions include Hinge loss for data classification via support-vector-machine (SVM) [30], linear and logistic regression costs [31, 32], and collaborative least square loss function for inference and estimation [33, 34, 35]. For most existing smooth cost/loss functions, Assumption 1 holds.*

The way data is distributed among the computing nodes relates the two objective functions $f_i(\cdot)$ and $g_i(\cdot)$. For example, the quantity/quality of the data accessible to each computing node defines both the share of assigned CPU resources $\mathbf{x}_i$ and the local ML parameters $\mathbf{y}_i$ associated with the data. The two optimization algorithms are genuinely coupled via the constraints, specifically the constraint $\mathbf{x}_i \in \Xi_i$, where the local constraint set $\Xi_i$ depends on the number of the data points (for ML objective) associated with the computing node $i$ (for scheduling objective) and ties the two problems. These constraints are typically addressed by the so-called box constraints in the literature [36, 37]. From this point onward in the paper, we consider a specific objective function and provide the solution. However, the solution holds for the general model (1) and can be easily extended to similar objective functions satisfying Assumption 1.

### *2.1. CPU Scheduling Model*

The CPU scheduling model in this subsection follows from [14, 15, 16]. Assume that the networked data centres consist of a set of (computing) nodes $\mathcal{V}$ of size $n$, where each node $i \in \mathcal{V}$ can also function as a resource scheduler, a common practice in modern data centres. Denote by $\mathcal{J}$ the set of all jobs to be scheduled. Each job $b_j \in \mathcal{J}$ requires $b_j$ CPU cycles to be performed, a quantity known prior to optimization. At each computing node $i$, the total workload due to arriving jobs is $l_i$. The optimization period, $T_h$, represents the time during which the current set of jobs is performed before the next reallocation. Each node's CPU capacity during this period is $\kappa_i^{\max} := c_i T_h$, with $c_i$ as the aggregate clock rate of all processing cores at node $i$, measured in cycles per second. The CPU availability at optimization step $k$ is $\kappa_i^{\text{avail}}[k] = \kappa_i^{\max} - u_i[k]$, where $u_i[k]$ represents the cycles already allocated to running tasks. Let $b[k] = \sum_{b_j[k] \in \mathcal{J}[k]} b_j[k]$ at step $k$, and $\kappa^{\text{avail}}[k] = \sum_{i \in \mathcal{V}} \kappa_i^{\text{avail}}[k]$ denote the total available capacity. The time horizon $T_h$ at step $k$ is chosen such that $b[k] \leq \kappa^{\text{avail}}[k]$, ensuring that the total demand does not exceed the available resources. Each node calculates the

optimal solution at every optimization step $k$ by executing a distributed algorithm that updates CPU utilization, accounting for (possibly log-quantized) information exchange and workloads. The algorithm ensures that each node balances its CPU utilization during task execution while meeting feasibility constraints. This *balancing strategy* requires each node $i$ to calculate the optimal workload $\mathbf{x}_i^*[k]$ such that $\forall i, j \in \mathcal{V}$:

$$\frac{\mathbf{x}_i^*[k] + u_i[k]}{\kappa_i^{\max}} = \frac{\mathbf{x}_j^*[k] + u_j[k]}{\kappa_j^{\max}} = \frac{b[k] + u_{\text{tot}}[k]}{\kappa^{\max}}, \tag{3}$$

where $\kappa^{\max} = \sum_{i\in\mathcal{V}} \kappa_i^{\max}$ and $u_{\text{tot}}[k] = \sum_{i\in\mathcal{V}} u_i[k]$. This condition (3) implies that the task allocation strategy allows every node to balance its CPU utilization during the execution of the tasks, i.e., to coordinate the given tasks such that each node utilizes the same percentage of its own capacity while meeting the constraint feasibility. For simplicity, we drop the index $k$ from this point onward in this section. Each node is associated with a scalar quadratic local cost function $f_i : \mathbb{R} \to \mathbb{R}$:

$$f_i(\varkappa) = \frac{1}{2}\alpha_i(\varkappa - b_i)^2, \tag{4}$$

where $\alpha_i > 0$, $b_i \in \mathbb{R}$ is the positive demand at node $i$, and $\varkappa = [\varkappa_1; \ldots; \varkappa_n]$ is the global optimization parameter. The goal is to minimize the global cost function (sum of local cost functions). Let the optimizer be

$$\varkappa^* = \arg\min_{\varkappa\in\mathcal{Z}} \sum_{i\in\mathcal{V}} f_i(\varkappa_i), \tag{5}$$

where $\mathcal{Z}$ is the set of feasible values for $\varkappa$, e.g., $\mathcal{Z}$ may represent the box constraints in the form $\underline{m}_i \leq \varkappa_i \leq \overline{M}_i$. The closed-form solution for (5) is:

$$\varkappa^* = \frac{\sum_{i\in\mathcal{V}} \alpha_i b_i}{\sum_{i\in\mathcal{V}} \alpha_i}. \tag{6}$$

From [16], to find the optimal workload according to (3), we need the solution of (5) to be

$$\varkappa^* = \frac{\sum_{v_i\in\mathcal{V}} \kappa_i^{\max} \frac{b_i+u_i}{\kappa_i^{\max}}}{\sum_{i\in\mathcal{V}} \kappa_i^{\max}} = \frac{\rho + u_{\text{tot}}}{\kappa^{\max}}. \tag{7}$$

From (7), we need to modify the local cost model (4) as

$$f_i(\varkappa) = \frac{1}{2}\kappa_i^{\max}\left(\varkappa - \frac{b_i + u_i}{\kappa_i^{\max}}\right)^2. \tag{8}$$

This implies that every node $i$ finds its proportion of workload, and from this proportion valueit can find the optimal workload $\mathbf{x}_i^*$ to receive, which is

$$\mathbf{x}_i^* = \frac{b + u_{\text{tot}}}{\kappa^{\max}}\kappa_i^{\max} - u_i. \tag{9}$$

However, it should be noted that the allocated workload by (9) gives the optimal allocation subject to constraint (3). From [16], a more general improved cost model in the following form can be considered

$$f_i(\varkappa) = \frac{1}{2}\alpha_i(\varkappa_i - b_i)^2, \tag{10}$$

where $\varkappa_i \neq \varkappa_j$, in general. Note the subtle difference here as the factors $\varkappa_i$ in (10) could be unequal (compared to the same $\varkappa$ in formulation (4)). Substituting $\mathbf{x}_i$ from (3),

$$f_i(\mathbf{x}_i) = \frac{1}{2\kappa_i^{\max}}(\mathbf{x}_i - b_i)^2, \tag{11}$$

This convex formulation gives lower cost by replacing the balancing constraint $\frac{\mathbf{x}_i+\rho_i}{\kappa_i^{\max}} = \frac{\mathbf{x}_j+b_j}{\kappa_j^{\max}}$ (or $\varkappa_i = \varkappa_j$) with a more general sum-preserving constraint $\sum_{i=1}^n \mathbf{x}_i = \sum_{i=1}^n \mathbf{x}_i^* = b$. This implies assigning the same workload as given by (7). Then, the modified version of (8) is

$$f_i(\mathbf{x}_i) = \frac{1}{2\kappa_i^{\max}}(\mathbf{x}_i - b_i)^2 \text{ s.t. } \sum_{i=1}^n \mathbf{x}_i = b. \tag{12}$$

To avoid exceeding server capacities and increase mean response times, we add box constraints on load-to-capacity ratios, keeping them below $70\% - 80\%$ of their capacity. This defines the local constraint $\Xi_i$. The scheduling approach ensures efficient CPU resource allocation while maintaining balanced and feasible workloads across all servers, as discussed later in Section 4.

**Remark 2.** *The existing works for CPU allocation typically consider quadratic cost models, see [14, 16] for example. However, following Assumption 1, this*

*work considers strictly convex cost models for CPU scheduling that could be non-quadratic in general. This is a more relaxed condition and allows for adding non-quadratic penalty terms (or barrier functions) to the objective function to address convex constraints or box constraints (this is discussed more in Section 3.1).*

**Remark 3.** *The CPU balancing model in [16] assumes that all CPU states reach agreement on the assigned computing loads. This balancing assumption allows to apply consensus-based algorithms directly to reach the optimal value. On the other hand, in this work, we assume sum-preserving constraint* $\sum_{i=1}^{n} \mathbf{x}_i = b$ *instead. This is a less restrictive assumption on the allocated computing resources. In general, it is easy to show that the allocation cost subject to the balancing constraint in [16] is always more than (or equal to) the allocation cost subject to sum-preserving constraint. This is because the solution by [16] assigns the resources as* $\mathbf{x}_i = \mathbf{x}_j = \frac{b}{n}$ *to reach consensus for all* $i, j$*, which is more restrictive than* $\sum_{i=1}^{n} \mathbf{x}_i = b$*. This is better illustrated in Section 5.*

### *2.2. ML Objective*

As stated in Remark (1), there are different objective functions to be optimized depending on the specific ML application. In this subsection, we consider two ML models in this paper. However, our solution holds for different ML objective functions satisfying Assumption 1.

#### *2.2.1. Linear Regression*

Linear regression is a statistical ML approach that involves fitting a hyperplane to a set of data points to model the relationship between variables. Given a set of $N$ data points $\boldsymbol{\chi}_i \in \mathbb{R}^{m-1}$, which are distributed among nodes/agents $i = \{1, \ldots, n\}$, this model predicts the variables associated with the hyperplane $\boldsymbol{\omega}^\top \boldsymbol{\chi}_i - \nu = a_i$ that is the best fit to the data. The approach to solving this problem is both centralized and distributed. In the centralized case, all the data points are gathered at the fusion centre to compute parameters $[\nu; \boldsymbol{\omega}]$ that solve the following,

$$\min_{[\boldsymbol{\omega}^\top; \nu]} \quad \sum_{i=1}^{N} (\boldsymbol{\omega}^\top \boldsymbol{\chi}_i - \nu - a_i)^2. \tag{13}$$

This problem is also referred to as the linear least-squares problem. On the other hand, the decentralized case, instead of having all the data at

one computing centre, distributes the data and computational load over a network of $n$ nodes. Each computing mode $i$ takes its own share of data, denoted by $\boldsymbol{\chi}_i$, which satisfies $\frac{N}{n} \leq N_i \leq N$. Some data might be given to two or more nodes. To solve the problem given by Eq. (13) locally and in a distributed way, every node takes its own share of data $\boldsymbol{\chi}_i$ and information of neighbouring nodes $j$. Since the data at every node is partial, the parameter values $\boldsymbol{\omega}_i$ and $\nu_i$ are different at different nodes. To reach *consensus* on these values, some key information is shared among the computing nodes. As a result, in the distributed case, the problem (13) takes the following form,

$$\begin{aligned} \min_{\boldsymbol{\omega}_1,\nu_1,\ldots,\boldsymbol{\omega}_n,\nu_n} \quad & \sum_{i=1}^{n} g_i(\boldsymbol{\omega}_i,\nu_i) = \sum_{i=1}^{n}\sum_{j=1}^{N_i}(\boldsymbol{\omega}_i^\top \boldsymbol{\chi}_j^i - \nu_i - y_j)^2 \\ \text{subject to} \quad & \boldsymbol{\omega}_1 = \cdots = \boldsymbol{\omega}_n, \nu_1 = \cdots = \nu_n \end{aligned} \tag{14}$$

This problem can be summarized in a compact form as part of problem formulation (1) with ML optimization parameter $\mathbf{y}_i = [\boldsymbol{\omega}_i^\top; \nu_i] \in \mathbb{R}^m$. Note that, in this formulation, $m$ denotes the dimension of parameters describing the hyperplane.

*2.2.2. Support-Vector-Machine*

Support-vector-machine (SVM) is a supervised learning method that classifies a given set of data by finding the best hyperplane that separates all data points into two classes. Consider $N$ data points $\boldsymbol{\chi}_i \in \mathbb{R}^{m-1}$, $i = 1, \ldots, N$ labeled by two classes $l_i \in \{-1, 1\}$. Then, the problem is to find the hyperplane $\boldsymbol{\omega}^\top \boldsymbol{\chi} - \nu = 0$, for $\boldsymbol{\chi} \in \mathbb{R}^{m-1}$, that partitions the given data into two classes (on two sides of the hyperplane). In the case that the data points are not linearly separable, a nonlinear mapping $\phi(\cdot)$ is used to map the data into another space where the data can be linearly separated. The SVM problem, then, is to minimize the following function (known as the Hinge loss [38, 39]):

$$\min_{[\boldsymbol{\omega}^\top;\nu]} \quad \boldsymbol{\omega}^\top \boldsymbol{\omega} + C \sum_{j=1}^{N} \max\{1 - l_j(\boldsymbol{\omega}^\top \phi(\boldsymbol{\chi}_j) - \nu), 0\}^p \tag{15}$$

with $p \in \mathbb{N}$ and $C \in \mathbb{R}^+$ determining the smoothness and margin size, repectively.

In the *distributed* case, the data is distributed among $n$ computing nodes, each having a partial data set $\boldsymbol{\chi}_i$ with $N_i$ data points. Similar to the previous

case (linear regression), the nodes/agents need to reach a consensus on locally found values $\boldsymbol{\omega}_i$ and $\nu_i$. Then, the distributed formulation to reach a common classifier is formulated as the following problem,

$$\begin{aligned} \min_{\boldsymbol{\omega}_1,\nu_1,\ldots,\boldsymbol{\omega}_n,\nu_n} \quad & \sum_{i=1}^{n} \boldsymbol{\omega}_i^\top \boldsymbol{\omega}_i + C \sum_{j=1}^{N} \max\{\theta_{i,j}, 0\}^p \\ \text{subject to} \quad & \boldsymbol{\omega}_1 = \cdots = \boldsymbol{\omega}_n, \qquad \nu_1 = \cdots = \nu_n, \end{aligned} \tag{16}$$

with $\theta_{i,j} = 1 - l_j(\boldsymbol{\omega}_i^\top \phi(\boldsymbol{\chi}_j^i) - \nu_i)$. Some works in the literature use another standard approximation of the above Hinge loss model by considering the following objective function for sufficiently large $\mu \in \mathbb{R}^+$ [40, 41],

$$\begin{aligned} \min_{\boldsymbol{\omega}_1,\nu_1,\ldots,\boldsymbol{\omega}_n,\nu_n} \quad & \sum_{i=1}^{n} \boldsymbol{\omega}_i^\top \boldsymbol{\omega}_i + C \sum_{j=1}^{N_i} \tfrac{1}{\mu} \log(1 + \exp(\mu\theta_{i,j})) \\ \text{subject to} \quad & \boldsymbol{\omega}_1 = \cdots = \boldsymbol{\omega}_n, \qquad \nu_1 = \cdots = \nu_n, \end{aligned} \tag{17}$$

Similarly, one can summarize the problem as part of the formulation (1) with ML optimization parameter $\mathbf{y}_i = [\boldsymbol{\omega}_i^\top; \nu_i] \in \mathbb{R}^m$ and $m$ as the dimension of the hyperplane classifier parameters.

**Remark 4.** *It is known from the literature [31, 42, 43] that both the structure of the network and distribution of the data among the nodes affect the convergence rate of the optimization algorithm. For example, exponential networks are known to result in lower optimality gap and faster convergence of distributed optimization. This is because the organized topology of the network, where each node is connected to an increasing number of neighbours at each layer, provides more connectivity, improved information exchange, and reduced communication bottleneck. On the other hand, in the case of heterogeneous data distribution among the computing nodes (in contrast to the homogeneous case where all data are available at all nodes), the optimality gap is larger and the convergence rate is slower. This is due to imbalanced information processing and data exchange among the nodes. These are also shown by simulation in Section 5.*

### *2.3. Algebraic Graph Theory*

The information-sharing among the computing nodes is modelled by a graph topology $\mathcal{G}_\gamma = \{\mathcal{V}, \mathcal{E}_\gamma\}$ including a set of $n$ nodes in $\mathcal{V}$ and links

$(i,j) \in \mathcal{E}_\gamma$ representing the information exchange between nodes $i, j$. The set of neighbours of node $i$, denoted by $\mathcal{N}_i^\gamma$, includes all nodes $j \in \mathcal{V}$ which communicate with $i$, i.e., $(j,i) \in \mathcal{E}_\gamma$. The network topology among the computing nodes might be time-varying, changing via a switching signal $\gamma : t \mapsto \Gamma$ with $\Gamma$ as the set of all possible topologies for $\mathcal{G}_\gamma$. The matrix $W_\gamma = [w_{ij}^\gamma]$ is the weighting adjacency matrix of this information-sharing network $\mathcal{G}_\gamma$ among the nodes (and depends on $\gamma$). The weight $w_{ij}^\gamma$ implies how node $i$ weights information sent by node $j$. Define the associated Laplacian matrix $\overline{W}_\gamma = [\overline{w}_{ij}^\gamma]$ as

$$\overline{w}_{ij}^\gamma = \begin{cases} -\sum_{i=1}^n w_{ij}^\gamma, & i = j \\ w_{ij}^\gamma, & i \neq j. \end{cases} \tag{18}$$

**Assumption 2.** *The network of computing nodes $\mathcal{G}_\gamma$ is time-varying, connected, and undirected with symmetric weights (i.e., $w_{ij}^\gamma(t) = w_{ji}^\gamma$ for all $t \geq 0$).*

**Lemma 1.** *[44, 45] For a network satisfying Assumption 2, all the eigenvalues of $\overline{W}_\gamma$ are real-valued and negative, except one isolated zero eigenvalue with left (and right) eigenvector $\mathbf{1}_n^\top$ (and $\mathbf{1}_n$), i.e., $\mathbf{1}_n^\top \overline{W}_\gamma = \mathbf{0}_n$ and $\overline{W}_\gamma \mathbf{1}_n = \mathbf{0}_n$.*

## 3. The Proposed Networked Algorithm

### *3.1. Linear Solution: Ideal Data-Exchange*

First, we consider the case that the information exchange among the computing nodes is ideal (not quantized) and, therefore, the solution is linear.

$$\dot{\mathbf{x}}_i = \sum_{j=1}^n w_{ij}^\gamma \Big( (\partial_{\mathbf{x}_j} f_j + \partial_{\mathbf{x}_j} f_j^\Xi) - (\partial_{\mathbf{x}_i} f_i + \partial_{\mathbf{x}_i} f_i^\Xi) \Big), \tag{19}$$

$$\dot{\mathbf{y}}_i = -\sum_{j=1}^n w_{ij}^\gamma (\mathbf{y}_i - \mathbf{y}_j) - \alpha \mathbf{z}_i, \tag{20}$$

$$\dot{\mathbf{z}}_i = -\sum_{j=1}^n w_{ij}^\gamma (\mathbf{z}_i - \mathbf{z}_j) + \partial_t \nabla g_i(\mathbf{y}_i), \tag{21}$$

where $\alpha \in \mathbb{R}^+$ denotes the gradient-tracking (GT) rate, $w_{ij}^\gamma \in \mathbb{R}^+$ is the weight on the data-sharing link between nodes $i$ and $j$ under the switching

signal $\gamma$. The local variable $\mathbf{z}_i$ is an auxiliary variable for gradient tracking at computing node $i$, and is initially set to zero, i.e., $\mathbf{z}_i(0) = 0$ for all $i$. From Eq. (20)-(21) and recalling Assumption 2, we have

$$\sum_{i=1}^{n} \dot{\mathbf{z}}_i = \sum_{i=1}^{n} \partial_t \nabla g_i(\mathbf{y}_i), \tag{22}$$

$$\sum_{i=1}^{n} \dot{\mathbf{y}}_i = -\alpha \sum_{i=1}^{n} \mathbf{z}_i. \tag{23}$$

By simple integration with respect to $t$ and recalling that $\mathbf{z}_i(0) = 0$, we get

$$\sum_{i=1}^{n} \dot{\mathbf{y}}_i = -\alpha \sum_{i=1}^{n} \mathbf{z}_i = -\alpha \sum_{i=1}^{n} \boldsymbol{\nabla} g_i(\mathbf{y}_i), \tag{24}$$

This illustrates the gradient-tracking nature of the proposed dynamics.

The auxiliary objective functions $f_i^{\Xi}(\cdot)$ address the local box constraints in (1). These are typically modelled as smooth penalty (or barrier) functions in the following form,

$$f_i^{\Xi}(\mathbf{x}_i) = \epsilon([\mathbf{x}_i - \overline{\xi}]^+ + [\underline{\xi} - \mathbf{x}_i]^+) \tag{25}$$

with $\overline{\xi}, \underline{\xi}$ as the upper/lower-bounds on $\mathbf{x}_i$ and parameter $\epsilon \in \mathbb{R}^+$ as a constant weight on the penalty term as compared to the original objective $f_i(\cdot)$. The smooth function $[u]^+$ is defined as,

$$[u]^+ = \max\{u, 0\}^\sigma, \ \sigma \in \mathbb{N}, \ \sigma \geq 2 \tag{26}$$

Another smooth penalizing function is proposed in [36],

$$[u]^+ = \frac{1}{\sigma} \log(1 + \exp(\sigma u)), \ \sigma \in \mathbb{R}^+ \tag{27}$$

where the parameter $\sigma$ is chosen sufficiently large to make the penalty function as close as possible to (26). Using this penalty formulation, one can decouple the scheduling and ML objectives. In this direction, using the laplacian matrix definition in Eq. (18), we rewrite the dynamics (20)-(21) in compact form as

$$\begin{pmatrix} \dot{\mathbf{y}} \\ \dot{\mathbf{z}} \end{pmatrix} = A(t, \alpha, \gamma) \begin{pmatrix} \mathbf{y} \\ \mathbf{z} \end{pmatrix}, \tag{28}$$

where $A(t, \alpha, \gamma)$ represents the ML system matrix defined as

$$\begin{pmatrix} \overline{W}_\gamma \otimes I_m & -\alpha I_{mn} \\ H(\overline{W}_\gamma \otimes I_m) & \overline{W}_\gamma \otimes I_m - \alpha H \end{pmatrix}, \tag{29}$$

with $H := \text{diag}[\nabla^2 g_i(\mathbf{y}_i)]$.

*3.2. Nonlinear Solution: Log-Scale Quantized Data-Exchange*

Next, we consider the case that the information exchange among the nodes is logarithmically quantized and, therefore, the solution is nonlinear.

$$\dot{\mathbf{x}}_i = \sum_{j=1}^{n} w_{ij}^{\gamma} \Big( q(\partial_{\mathbf{x}_j} f_j + \partial_{\mathbf{x}_j} f_j^{\Xi}) - q(\partial_{\mathbf{x}_i} f_i + \partial_{\mathbf{x}_i} f_i^{\Xi}) \Big), \tag{30}$$

$$\dot{\mathbf{y}}_i = -\sum_{j=1}^{n} w_{ij}^{\gamma} (q(\mathbf{y}_i) - q(\mathbf{y}_j)) - \alpha \mathbf{z}_i, \tag{31}$$

$$\dot{\mathbf{z}}_i = -\sum_{j=1}^{n} w_{ij}^{\gamma} (q(\mathbf{z}_i) - q(\mathbf{z}_j)) + \partial_t \nabla g_i(\mathbf{y}_i), \tag{32}$$

where $q(\cdot) : \mathbb{R} \mapsto \mathbb{R}$ denotes log-scale quantization as a nonlinear mapping of the information exchanged between nodes. This quantization mapping is defined as,

$$q(x) = \text{sgn}(x) \exp\left( \rho \left[ \frac{\log(|x|)}{\rho} \right] \right), \tag{33}$$

where $[\cdot]$ denotes rounding to the nearest integer and $\text{sgn}(\cdot)$ denotes the sign function. The parameter $\rho$ denotes the quantization level. As the quantization level goes to zero $\rho \to 0$, the nonlinear solution (30)-(32) converges to the linear case (19)-(21). Note that logarithmic quantization is a sector-bound nonlinearity satisfying,

$$(1 - \frac{\rho}{2})x \leq q(x) \leq (1 + \frac{\rho}{2})x \tag{34}$$

where the quantization level generally satisfies $\rho \leq 1$. Define the diagonal matrix $Q(t) = \text{diag}[\overline{q}_i(t)]$ with $\overline{q}_i(t) = \frac{q(\mathbf{x}_i)}{\mathbf{x}_i}$ (element-wise division). Then, we define a new modified laplacian matrix $\overline{W}_{\gamma,q} := \overline{W}_\gamma Q(t)$. This simply follows from $q(\mathbf{x}(t)) = Q(t)\mathbf{x}(t)$. Recall from the consensus nature of the

solution that $-\sum_{j=1}^{n} w_{ij}^{\gamma}(\mathbf{x}_i - \mathbf{x}_j)$ can be written as $(\overline{W}_{\gamma} \otimes I_m)\mathbf{x}$. Then, following Eq. (34), one can get $-\sum_{j=1}^{n} w_{ij}^{\gamma}(q(\mathbf{x}_i) - q(\mathbf{x}_j)) = (\overline{W}_{\gamma,q} \otimes I_m)\mathbf{x}$. Then, $A(t, \alpha, \gamma)$ in compact formulation (28) can be modified as

$$A(t, \alpha, \gamma) := \begin{pmatrix} \overline{W}_{\gamma,q} \otimes I_m & -\alpha I_{mn} \\ H(\overline{W}_{\gamma,q} \otimes I_m) & \overline{W}_{\gamma,q} \otimes I_m - \alpha H \end{pmatrix}, \tag{35}$$

Finally, we summarize our proposed problem-solving in Algorithm 1. The per-iteration computational complexity of this algorithm is of order $\mathcal{O}(m^2 n^3)$.

**Algorithm 1:** CPU resource scheduling and ML optimization algorithm

**Input:** $\mathcal{G}^{\gamma}$, $W_{\gamma}$, $\alpha$, $\overline{\xi}$, $\underline{\xi}$, $b$, $f_i(\cdot)$, $g_i(\cdot)$, $\sigma$, $\epsilon$, $\rho$;
**Initialization:** $\sum_{i=1}^{n} \mathbf{x}_i(0) = b$, $\mathbf{z}_i(0) = 0$, random $\mathbf{y}_i(0)$;
**while** *termination criteria NOT true* **do**
    Computing node $i$ receives state information from incoming neighbouring nodes $j \in \mathcal{N}_i^{\gamma}$;
    Node $i$ computes Eq. (30)-(32);
    Computing node $i$ shares its updated states with outgoing neighboring nodes $j$ for which $i \in \mathcal{N}_j^{\gamma}$;
**Return** Final state $\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i$ and overall cost

## 4. Analysis of Convergence, Feasibility and Optimality

In this section, we prove convergence, optimality, and feasibility for the quantized solution (30)-(32). These results can be easily extended to the linear case (19)-(21). First, we show that the proposed solution is all-time feasible. This implies that at all times the resource-demand balance denoted by the constraint $\sum_{i=1}^{n} \mathbf{x}_i = b$ holds.

**Lemma 2.** ***(all-time feasibility)*** *Given that Assumption 2 holds on the network connectivity and the computing resources are initially feasible (i.e., $\sum_{i=1}^{n} \mathbf{x}_i(0) = b$), the proposed solution* (30)-(32) *is all-time feasible.*

*Proof.* We prove that the change in the computing resources under the proposed dynamics (30)-(32) is zero, i.e., $\sum_{i=1}^{n} \dot{\mathbf{x}}_i = 0$. Recall from Eq. (30)

that,

$$\sum_{i=1}^{n} \dot{\mathbf{x}}_i = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}^{\gamma}\Big(q(\partial_{\mathbf{x}_j} f_j + \partial_{\mathbf{x}_j} f_j^{\Xi}) - q(\partial_{\mathbf{x}_i} f_i + \partial_{\mathbf{x}_i} f_i^{\Xi})\Big). \tag{36}$$

From Assumption 2 we have $w_{ij}^{\gamma}(t) = w_{ji}^{\gamma}$; and from the definition (33) it is clear that log-scale quantization is a sign-preserving and odd mapping. Therefore,

$$w_{ij}^{\gamma}\Big(q(\partial_{\mathbf{x}_j} f_j + \partial_{\mathbf{x}_j} f_j^{\Xi}) - q(\partial_{\mathbf{x}_i} f_i + \partial_{\mathbf{x}_i} f_i^{\Xi})\Big) = - w_{ji}^{\gamma}\Big(q(\partial_{\mathbf{x}_i} f_i + \partial_{\mathbf{x}_i} f_i^{\Xi}) - q(\partial_{\mathbf{x}_j} f_j + \partial_{\mathbf{x}_j} f_j^{\Xi})\Big). \tag{37}$$

This implies that the summation in Eq. (36) over all $i, j$ gives zero. As a result, by initializing from a feasible solution $\sum_{i=1}^{n} \mathbf{x}_i(0) = b$, we get

$$\sum_{i=1}^{n} \mathbf{x}_i(t) = \sum_{i=1}^{n} \mathbf{x}_i(0) = b.$$

This proves the lemma. □

**Remark 5.** *Lemma 2 proves the all-time feasibility of the computing resource-demand balance in this work, which implies that the algorithm can be terminated at any time with no constraint violation. This is in contrast to [14, 16], where no all-time constraint feasibility is given. In other words, the algorithms in [14, 16] reach the computing resource-demand balance at the convergence time, and before that time, there is no balance between the assigned resources and computing demand.*

The next lemma describes the main feature of the optimal point $[\mathbf{x}^*; \mathbf{y}^*]$. First, define

$$F(\mathbf{x}) = \sum_{i=1}^{n} f_i(\mathbf{x}_i) + f_i^{\Xi}\mathbf{x}_i,$$

$$\nabla_{\mathbf{x}} F = [\partial_{\mathbf{x}_1} f_1 + \partial_{\mathbf{x}_1} f_1^{\Xi}; \ldots; \partial_{\mathbf{x}_n} f_n + \partial_{\mathbf{x}_n} f_n^{\Xi}].$$

**Lemma 3.** ***(optimality)*** *Given the problem* (1)*, the optimal state* $\mathbf{x}^*$ *satisfies* $\nabla_{\mathbf{x}} F(\mathbf{x}^*) \in span(\mathbf{1}_n)$ *and* $\mathbf{y}^* \in span(\mathbf{1}_n)$*. Moreover, this optimal point is invariant under the solution dynamics* (30)-(32)*.*

*Proof.* The proof for $\mathbf{y}^*$ directly follows the consensus constraint $\mathbf{y}_1 = \mathbf{y}_2 = \cdots = \mathbf{y}_n$. The proof for $\mathbf{x}^*$ is a result of the Karush–Kuhn–Tucker condition for linear constraint $\sum_{i=1}^n \mathbf{x}_i = b$ and convex objective function given by Eq. (1) and (25). For more details on the latter, refer to [46, 47]. □

Note that, for the auxiliary variable $\mathbf{z} \to \mathbf{z}^* = \mathbf{0}_n$. To prove the convergence, we first recall some useful lemmas.

**Lemma 4.** *For any $\mathbf{x} \in \mathbb{R}^n$ and log-scale quantization $q(\cdot)$, under Assumption 2 we have*

$$\sum_{i=1}^{n} \mathbf{x}_i \sum_{j=1}^{n} w_{ij}^{\gamma}(q(\mathbf{x}_j) - q(\mathbf{x}_i)) = \sum_{i,j=1}^{n} \frac{w_{ij}^{\gamma}}{2}(\mathbf{x}_j - \mathbf{x}_i)q(\mathbf{x}_j) - q(\mathbf{x}_i) \tag{38}$$

*Proof.* From Assumption 2, $w_{ij}^{\gamma} = w_{ji}^{\gamma}$. Also, log-scale quantization is an odd and sign-preserving mapping. These imply that,

$$\begin{aligned} &\mathbf{x}_i w_{ij}^{\gamma}(q(\mathbf{x}_i) - q(\mathbf{x}_j)) + \mathbf{x}_j w_{ji}^{\gamma}(q(\mathbf{x}_j) - q(\mathbf{x}_i)) \\ &\qquad = w_{ij}^{\gamma}(\mathbf{x}_i - \mathbf{x}_j)(q(\mathbf{x}_j) - q(\mathbf{x}_i)) \\ &\qquad = w_{ji}^{\gamma}(\mathbf{x}_j - \mathbf{x}_i)(q(\mathbf{x}_j) - q(\mathbf{x}_i)). \end{aligned} \tag{39}$$

and the proof follows. □

**Lemma 5.** *[48, 49] Consider an $n$-by-$n$ matrix $P(\alpha)$ as a function of $\alpha \geq 0$. Assume that this matrix has $l < n$ equal eigenvalues $\lambda_1 = \ldots = \lambda_l$ with (right and left) linearly independent unit eigenvectors $\mathbf{v}_1, \ldots, \mathbf{v}_l$ and $\mathbf{u}_1, \ldots, \mathbf{u}_l$. Then, $\frac{d\lambda_i}{d\alpha}|_{\alpha=0}$ is the $i$-th eigenvalue of the following matrix:*

$$\begin{pmatrix} \mathbf{u}_1^{\top} P' \mathbf{v}_1 & \ldots & \mathbf{u}_1^{\top} P' \mathbf{v}_l \\ & \ddots & \\ \mathbf{u}_l^{\top} P' \mathbf{v}_1 & \ldots & \mathbf{u}_l^{\top} P' \mathbf{v}_l \end{pmatrix}, \; P' = \frac{dP(\alpha)}{d\alpha}|_{\alpha=0} \tag{40}$$

**Lemma 6.** *Given the compact system dynamics (28) with $A(t, \alpha, \gamma)$ as Eq. (35), the system has all negative eigenvalues except $m$ zero eigenvalues for sufficiently small $\alpha$, with $m$ as the dimension of the ML optimization variable.*

*Proof.* First, rewrite Eq. (35) as

$$A(t,\alpha,\gamma) = A_q^0 + \alpha A^1,\ A_q^0 = Q(t)A^0 \tag{41}$$

$$(1-\frac{\rho}{2})A^0 \preceq A_q^0 \preceq (1+\frac{\rho}{2})A^0 \tag{42}$$

$$(1-\frac{\rho}{2})I_n \preceq Q(t) \preceq (1+\frac{\rho}{2})I_n \tag{43}$$

where

$$\begin{aligned} A^0 &= \begin{pmatrix} \overline{W}_\gamma \otimes I_m & \mathbf{0}_{mn\times mn} \\ H(\overline{W}_\gamma \otimes I_m) & \overline{W}_\gamma \otimes I_m \end{pmatrix}, \\ A^1 &= \begin{pmatrix} \mathbf{0}_{mn\times mn} & -I_{mn} \\ \mathbf{0}_{mn\times mn} & -H \end{pmatrix}. \end{aligned}$$

One can write $(1-\frac{\rho}{2})\sigma(A^0) \leq \sigma(A_q^0) \leq (1+\frac{\rho}{2})\sigma(A^0)$, where $\sigma(A^0) = \sigma(\overline{W}\otimes I_m) \cup \sigma(\overline{W}\otimes I_m)$. Then, from Lemma 1, $A^0$ has $m$ set of eigenvalues which satisfy

$$\lambda_{2n,j} \leq \ldots \leq \lambda_{3,j} < \lambda_{2,j} = \lambda_{1,j} = 0,\ j = \{1,\ldots,m\}.$$

Then, using Lemma 5, one can check how the perturbation $\alpha A^1$ affects $\lambda_{1,j}$ and $\lambda_{2,j}$ in $\sigma(A_q^0)$. Following Lemma 5,

$$\mathbf{v} = [\mathbf{v}_1\ \mathbf{v}_2] = \begin{pmatrix} \mathbf{1}_n & \mathbf{0}_n \\ \mathbf{0}_n & \mathbf{1}_n \end{pmatrix} \otimes I_m,$$

and $\mathbf{u} = \mathbf{v}^\top$. From (35), we have $\frac{dA(\alpha)}{d\alpha}|_{\alpha=0} = A^1$. Then,

$$\mathbf{v}^\top A^1 \mathbf{v} = \begin{pmatrix} \mathbf{0}_{m\times m} & \mathbf{0}_{m\times m} \\ \ldots & -(\mathbf{1}_n \otimes I_m)^\top H(\mathbf{1}_n \otimes I_m) \end{pmatrix}, \tag{44}$$

which has $m$ zero eigenvalues and $m$ other eigenvalues are negative. This follows Assumption 1 since

$$-(\mathbf{1}_n \otimes I_m)^\top H(\mathbf{1}_n \otimes I_m) = -\sum_{i=1}^{n} \nabla^2 f_i(\mathbf{x}_i) \prec 0, \tag{45}$$

Therefore, $\frac{d\lambda_{1,j}}{d\alpha}|_{\alpha=0} = 0$ and $\frac{d\lambda_{2,j}}{d\alpha}|_{\alpha=0} < 0$. This implies that after perturbation of $A_q^0$ by $\alpha A^1$, $m$ zero eigenvalues $\lambda_{2,j}(\alpha,t)$ become negative while $\lambda_{1,j}(\alpha,t)$ remain zero for all $j$. Further, we need to show that for sufficiently

small $\alpha$ other negative eigenvalues still remain in the left-half-plane. First, we recall from [50, Appendix] to relate the spectrum of $A(t,\alpha,\gamma)$ in (41) to $\alpha$. For simplicity of the proof analysis, from this point onward we set $m=1$, but the proof holds for any $m>1$. By proper row/column permutations, following [50, Eq. (18)], $\sigma(A(t,\alpha,\gamma))$ can be defined as,

$$\det(\alpha I_n)\det(H(\overline{W}_\gamma)+(\overline{W}_\gamma-\alpha H-\lambda I_n)(\frac{1}{\alpha})(\overline{W}_\gamma-\lambda I_n))=0.$$

This can be simplified as follows,

$$\det(I_n)\det((\overline{W}_\gamma-\lambda I_n)(\overline{W}_\gamma-\lambda I_n)+\alpha\lambda H)=0 \tag{46}$$

For stability/convergence, we need to find the admissible $\alpha$ values such that $\lambda$ remains in the left-half-plane, except for one zero eigenvalue[1]. Clearly, $\alpha=0$ satisfies (46) and leads to the eigen-spectrum following as $\sigma(A(t,\alpha,\gamma))=\sigma(\overline{W}_\gamma)\cup\sigma(\overline{W}_\gamma)$ which has two zero roots for all switching topologies $\gamma$. The other root $\overline{\alpha}>0$ needs to be defined for the admissible range $0<\alpha<\overline{\alpha}$ for the stability of $A(t,\alpha,\gamma)$. For any $\lambda<0$, with some abuse of notation, Eq. (46) can be reformulated as

$$\det((\overline{W}_\gamma-\lambda I_n\pm\sqrt{\alpha|\lambda|H})(\overline{W}_\gamma-\lambda I_n\mp\sqrt{\alpha|\lambda|H}))=0$$

Then, we have,

$$\det(\overline{W}_\gamma-\lambda I_n\mp\sqrt{\alpha|\lambda|H})=\det(\overline{W}_\gamma-\lambda(I_n\mp\sqrt{\frac{\alpha H}{|\lambda|}}))=0$$

This gives $\lambda(1\pm\sqrt{\frac{\alpha H}{|\lambda|}})$ as perturbed eigenvalue of $\lambda\in\sigma(\overline{W}_\gamma)$. Therefore, for $\lambda\neq 0$, the minimum $\overline{\alpha}$ making this term zero is,

$$\overline{\alpha}=\underset{\alpha}{\operatorname{argmin}}\,|1-\sqrt{\frac{\alpha H}{|\lambda|}}|\geq\frac{\min\{|\lambda|\neq 0\}}{\max\{H_{ii}\}}=\frac{|\lambda_2|}{L} \tag{47}$$

where $H\preceq LI_n$ is recalled from Assumption 1. Therefore, the admissible range for the convergence of the linear dynamics (19)-(21) is

$$0<\alpha<\overline{\alpha}:=\frac{|\lambda_2|}{L} \tag{48}$$

[1]The eigenspectrum analysis in this section follows the fact that the eigenvalues are continuous functions of the matrix elements [48].

Considering the log-quantization nonlinearity in dynamics (30)-(32) and from Eq. (34), we have

$$0 < \alpha < \overline{\alpha} := \frac{|\lambda_2|}{L(1+\frac{\rho}{2})} \tag{49}$$

This gives the admissible range of $\alpha$ for which all the eigenvalues are in the left-half-plane (except one set of $m$ zero eigenvalues). This completes the proof. □

**Theorem 1.** *(**convergence**) Suppose Assumptions 1-2 hold. Then, under initial conditions* $\sum_{i=1}^{n} \mathbf{x}_i(0) = b$, $\mathbf{z}(0) = \mathbf{0}_{nm}$ *and sufficiently small* $\alpha$ *satisfying* $0 < \alpha < \overline{\alpha} := \frac{|\lambda_2|}{L(1+\frac{\rho}{2})}$, *Algorithm 1 solves the problem* (1).

*Proof.* We need to show that the algorithm converges to the optimal point given by Lemma 3. We prove this by defining the proper residual-based Lyapunov function. Define the state residual vector for the ML objective as

$$\delta_m = \begin{pmatrix} \mathbf{y} \\ \mathbf{z} \end{pmatrix} - \begin{pmatrix} \mathbf{y}^* \\ \mathbf{0}_{nm} \end{pmatrix} \tag{50}$$

and the residual for scheduling as

$$\delta_c = F(\mathbf{x}) - F(\mathbf{x}^*) \tag{51}$$

For Lyapunov analysis, we consider the following function,

$$V = \frac{1}{2}\|\delta_m\|_2^2 + \delta_c \tag{52}$$

This function is positive-definite with $V \to 0$ as $\delta \to \mathbf{0}$. We have

$$\dot{V} = \delta_m^\top \dot{\delta}_m + \dot{\delta}_c = \delta_m^\top A \delta_m + \nabla_{\mathbf{x}} F \dot{\mathbf{x}}. \tag{53}$$

Then, from [44, Sections VIII-IX], for the first term we get

$$\delta_m^\top A \delta_m \leq \lambda_2 \delta_m^\top \delta_m, \tag{54}$$

From Lemma 6 we know that $\lambda_2$ is negative, which implies that this term in $\dot{\mathcal{V}}$ is negative-definite for $\delta_m \neq \mathbf{0}$.

For the second term in (53) we have

$$\nabla_{\mathbf{x}} F\dot{\mathbf{x}} = \sum_{i=1}^{n} \dot{\mathbf{x}}_i(\partial_{\mathbf{x}_i} f_i + \partial_{\mathbf{x}_i} f_i^{\Xi}). \tag{55}$$

Replacing $\dot{\mathbf{x}}_i$ from Eq. (30), we obtain

$$\begin{aligned}\nabla_{\mathbf{x}} F\dot{\mathbf{x}} = &\sum_{i=1}^{n}(\partial_{\mathbf{x}_i} f_i + \partial_{\mathbf{x}_i} f_i^{\Xi}) \\ &\sum_{j=1}^{n} w_{ij}^{\gamma}\Big(q(\partial_{\mathbf{x}_j} f_j + \partial_{\mathbf{x}_j} f_j^{\Xi}) - q(\partial_{\mathbf{x}_i} f_i + \partial_{\mathbf{x}_i} f_i^{\Xi})\Big).\end{aligned} \tag{56}$$

Using Lemma 4 and Assumption 2, we see that

$$\begin{aligned}\nabla_{\mathbf{x}} F\dot{\mathbf{x}} = -\sum_{i,j=1}^{n} \frac{w_{ij}^{\gamma}}{2}\Big((\partial_{\mathbf{x}_i} f_i + \partial_{\mathbf{x}_i} f_i^{\Xi}) - (\partial_{\mathbf{x}_j} f_j + \partial_{\mathbf{x}_j} f_j^{\Xi})\Big) \\ \Big(q(\partial_{\mathbf{x}_i} f_i + \partial_{\mathbf{x}_i} f_i^{\Xi}) - q(\partial_{\mathbf{x}_j} f_j + \partial_{\mathbf{x}_j} f_j^{\Xi})\Big)\Big).\end{aligned} \tag{57}$$

Recall that log-scale quantization is odd and monotonically non-decreasing. This implies that $\nabla_{\mathbf{x}} F\dot{\mathbf{x}} < 0$ for $\delta_c \neq \mathbf{0}$ and $\dot{V}$ is negative-definite. Then, recalling Lyapunov stability theorem [51], the residual $\delta_m + \delta_c$ under the dynamics (30)-(32) is decreasing toward zero and, therefore, $F(\mathbf{x}) \to F(\mathbf{x}^*)$ (and $\mathbf{x} \to \mathbf{x}^*$), $\mathbf{y} \to \mathbf{y}^*$, and $\mathbf{z} \to \mathbf{0}_{nm}$. This completes the proof. □

It should be noted that the proof of Lemma 6 and Theorem 1 follows from Assumption 1 and is only based on the constraint $(\mathbf{1}_n \otimes I_m)^\top H(\mathbf{1}_n \otimes I_m) \succ 0$ with no other assumption on the convexity of the $g_i(\cdot)$ function. Therefore, the convergence holds for possibly non-convex $g_i(\cdot)$ satisfying Assumption 1. This is shown later via simulations in Section 5.1.

**Remark 6.** *The proof analysis for convergence and feasibility in Lemma 2, 6, and Theorem 1 is irrespective of the structure of the network topology and only requires the network to be connected. Therefore, the network could be time-varying, and the proofs of Lemma 2, 6, and Theorem 1 still hold. This is in contrast to [14, 16], which consider time-invariant (static) network topology.*

**Remark 7.** *Theorem 1 clearly proves convergence subject to log-scale quantization on data-sharing considered in the nodes' dynamics* (30)-(32)*. This is*

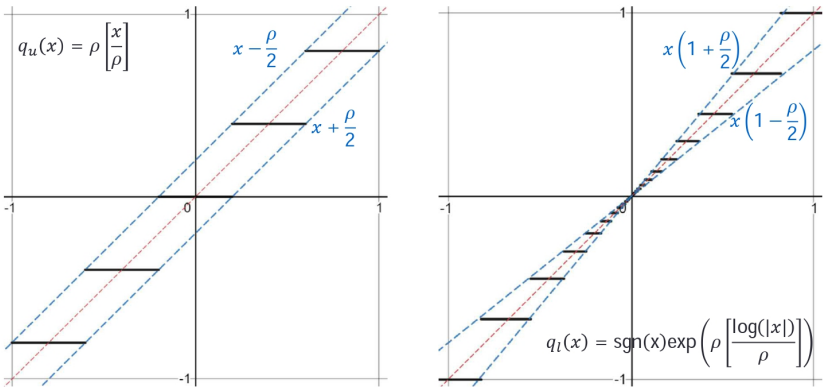



Figure 1: Comparison between uniform quantization (left) and log-scale quantization (right): uniform quantization is not a sector-bound nonlinearity, while logarithmic quantization is sector-bounded, assigning more bits to represent smaller values and fewer bits to larger values.

*in contrast to uniform quantization, which in general may result in some optimality gap (or steady-state residual) which is often proportional to the quantization level, e.g., see existing literature on distributed learning/optimization with uniformly quantized data [16, 52, 53, 54, 55]. This is because log-scale quantization is a sector-bound nonlinearity, while uniform quantization is not sector-bounded (with some bias at the origin), see Fig. 1. In fact, logarithmic quantization assigns more bits to represent smaller values and fewer bits to larger values; this is advantageous for gradient-tracking as smaller gradient values near the optimal point (and more critical for convergence) are represented with higher precision. This may increase the complexity of the distributed optimization setup, but reduces the optimality gap.*

## 5. Simulations

The simulations are performed in an Intel Core-i5 laptop with 2.50 GHz CPU and 8 GB RAM. The simulations in Section 5.1 are performed in MATLAB R2022, and simulations in Section 5.2 are performed in Python.

### *5.1. Academic Examples*

In contrast to traditional centralized computing, in this section, we apply parallel data processing of data over many computing nodes while cost-optimally assigning the computing resources. For simulation, we consider

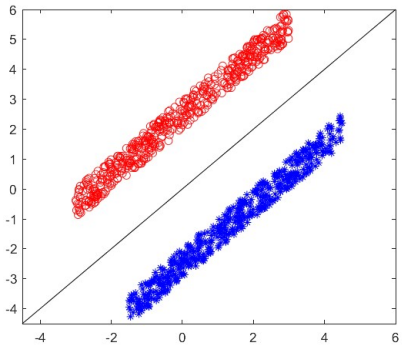



Figure 2: The data points in 2D used for the SVM classification and the associated SVM classifier line.

a network of $n = 20$ computing nodes communicating over a time-varying Erdos-Renyi graph topology with 40% linking probability. The link weights $w_{ij}^{\gamma}$ are randomly set in the range $(0, 1]$. First, we compare the optimal cost of the proposed CPU scheduling algorithm with the CPU balancing strategy in [16] in Table 1. As stated in Remark 3, considering our sum-preserving constraint results in less resource allocation cost as compared to the consensus-based balancing strategy in [16].

Table 1: Comparing the optimality performance with uniformly-quantized algorithm [16] in terms of the cost of assigned CPU resources for different values of overall demand $b$

| The demand | $b = 1000$ | $b = 3000$ | $b = 5000$ | $b = 8600$ |
|---|---|---|---|---|
| This work | 84 | 288 | 338 | 417 |
| Ref. [16] | 1387 | 2015 | 2386 | 2453 |

For the rest of this section, we set overall resources $b = 8600$ to meet the workload demand and the box constraints as $0 < \mathbf{x}_i < 700$ (i.e., $\underline{\zeta} = 0$, $\overline{\zeta} = 700$ in Algorithm 1). We use these computing resources to solve two ML problems as follows.

**Distributed SVM:** Consider $N = 1000$ data points in 2D as shown in Fig. 2 to be classified by the SVM line shown in the figure. The data is distributed among the computing nodes such that every node has access to 75% of the data points. The parameters in Algorithm 1 are set as $\alpha = 0.05$, $\mu = 2$, $C = 5$, $\sigma = 2$, $\epsilon = 1$, and $\rho = 0.125$. Every computing node minimizes its SVM loss function described in (17) and CPU resource allocation

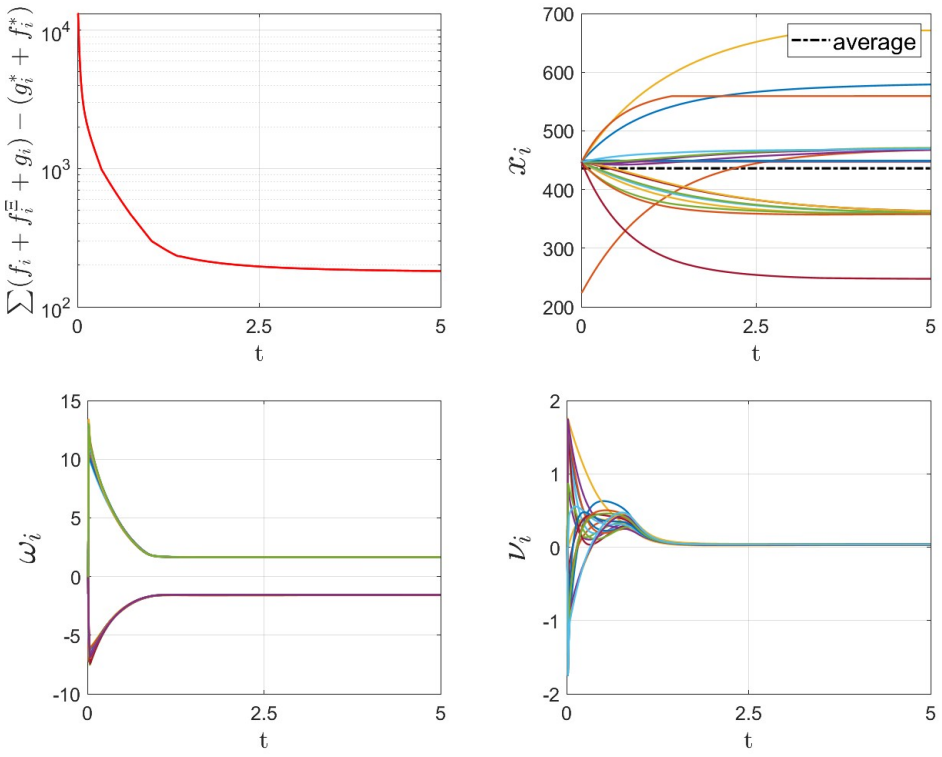



Figure 3: Time-evolution of the overall cost residual (optimality gap), assigned CPU resources $\mathbf{x}_i$, and SVM parameters $\omega_i, \nu_i$ under the proposed Algorithm 1.

cost (12) to find the optimal variables $\mathbf{x}_i$, $\omega_i$, and $\nu_i$. The time-evolution of the parameter $\mathbf{x}_i$ denoting the assigned CPU resources, the SVM classifier parameters $\omega_i \in \mathbb{R}^2, \nu_i \in \mathbb{R}$, and the overall optimality gap (or cost residual) are shown in Fig. 3. As it is clear from the figure, the average value of the assigned resources $\mathbf{x}_i$ is constant over time, implying resource-demand feasibility at all times. The SVM parameters (both dimensions of vector $\omega_i$ and scalar $\nu_i$) at all nodes reach consensus. The overall residual function $\sum_{i=1}^{n}(f_i(\mathbf{x}_i) + f_i^{\Xi}(\mathbf{x}_i) + g_i(\mathbf{y}_i)) - (f_i(\mathbf{x}_i^*) + g_i(\mathbf{y}_i^*))$ is decreasing over time; however, the convergence rate is slow since the data is distributed *heterogeneously* among the computing nodes (see Remark 4) and only a portion of data is available at each computing node.

**Distributed linear regression:** We consider a set of $N = 1000$ randomly generated 2D data points to be fitted with a regressor line. Each computing node has access to 70% of the data points and locally finds the

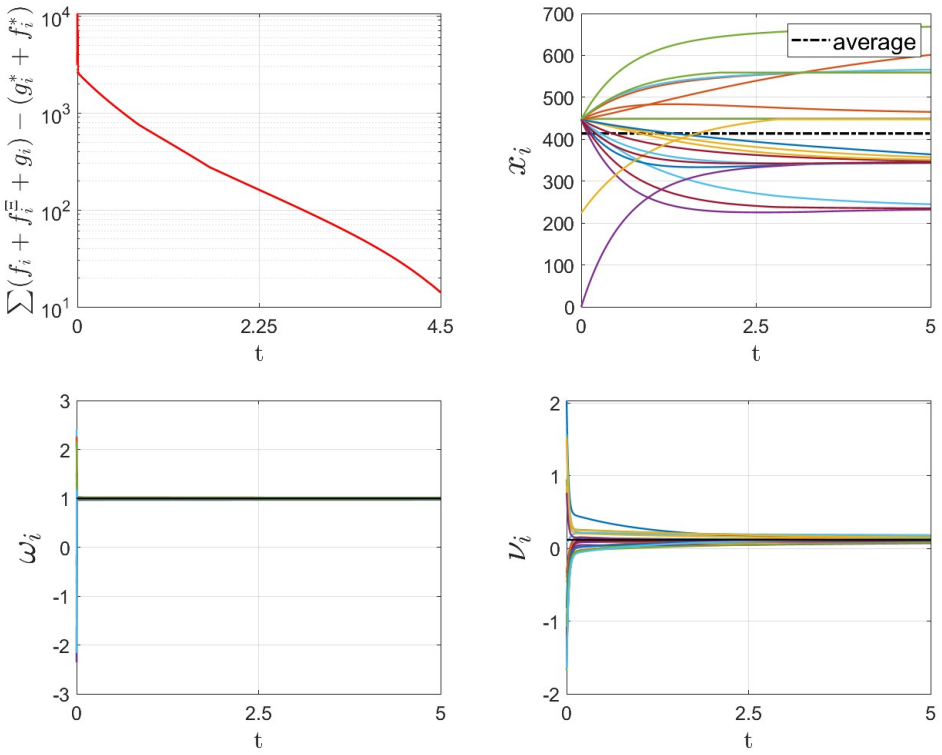



Figure 4: Time-evolution of the overall cost residual (optimality gap), assigned CPU resources $\mathbf{x}_i$, and regression parameters $\beta_i, \nu_i$ under the proposed Algorithm 1.

parameters of the regressor line; then, it shares this information with the neighbouring nodes over the network to reach a consensus on the regression parameters. The parameters in Algorithm 1 are set as $\alpha = 0.1$, $\epsilon = 1$, $\sigma = 2$, and $\rho = 0.0625$. Every computing node minimizes its regression loss function described in (14) and CPU resource allocation cost (12) to find the optimal variables $\mathbf{x}_i$, $\omega_i$, and $\nu_i$. The time-evolution of the CPU parameter $\mathbf{x}_i$, the regression line parameters $\omega_i, \nu_i$, and the cost residual are shown in Fig. 4. Clearly, the residual is decreasing toward the optimal point, resource-demand feasibility of CPU resources $\mathbf{x}_i$ holds at all times, and the regressor line parameters reach consensus at all nodes.

**Non-convex objective:** Next, the simulation is performed for a synthetic non-convex local objective function (taken from [56]) defined as:

$$g_{i,j}(y_i) = 2y_i^2 + 3\sin^2(y_i) + a_{i,j}\cos(y_i) + b_{i,j}y_i, \tag{58}$$

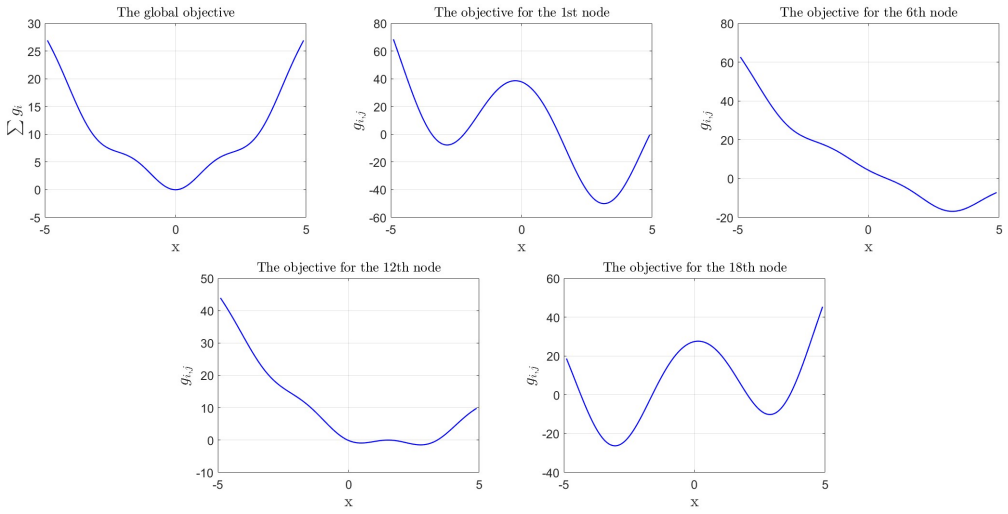



Figure 5: The top-left figure shows the global objective function $\frac{1}{n}\sum_{i=1}^{n}\frac{1}{m}\sum_{j=1}^{N} g_{i,j}(y_i)$. The other four figures show the local non-convex objective functions at four sample nodes. This shows an example non-convex local objective function $g_i(\cdot)$ satisfying Assumption 1.

with $\sum_{i=1}^{n}\sum_{j=1}^{N} a_{i,j} = 0$ and $\sum_{i=1}^{n}\sum_{j=1}^{N} b_{i,j} = 0$ such that $a_{i,j}, b_{i,j} \neq 0$ and randomly chosen in the range $(-5, 5)$ with $N = 60$ sample data points. As illustrated in Fig. 5, these local objectives are not convex, i.e., $\nabla^2 f_i(\mathbf{x})$ is negative at some points, while the global objective satisfies Assumption 1. We set the gradient-tracking parameter as $\alpha = 0.1$ and the log-scale parameter as $\rho = 0.03125, 0.125$. The CPU scheduling parameters are $\epsilon = 2$, $\sigma = 2$. The simulation result is shown in Fig. 6.

### *5.2. Real Data-Set Example*

Next, we consider the MNIST image data set and compare the performance of our ML setup with some existing literature for image classification. We use $N = 12000$ labelled images from this data set and classify them using logistic regression with a convex regularizer over a network of $n = 16$ nodes. The objective optimization model is defined as

$$\min_{\mathbf{b},c} F(\mathbf{b}, c) = \frac{1}{n}\sum_{i=1}^{n} f_i, \tag{59}$$

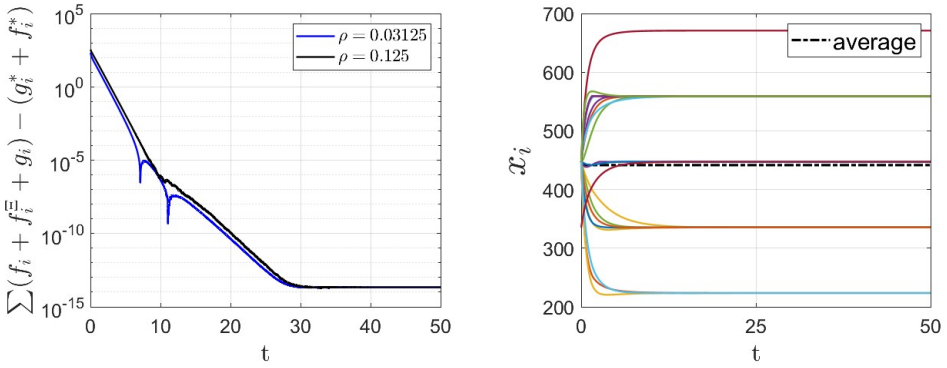



Figure 6: These figures show the time-evolution of the locally non-convex cost-function (58) plus CPU costs (left) and assigned CPU resources (right) subject to log-scale quantization.

with every node $i$ accessing a batch of $m_i = \frac{N}{n} = 750$ sample images. Then, every node locally minimizes the following training loss function:

$$f_i(\mathbf{x}) = \frac{1}{m_i} \sum_{j=1}^{m_i} \ln(1 + \exp(-(\mathbf{b}^\top \mathcal{X}_{i,j} + c)\mathcal{Y}_{i,j})) + \frac{\theta}{2}\|\mathbf{b}\|_2^2, \tag{60}$$

which is smooth because of the addition of the regularizer $\theta$. The $j$-th sample at node $i$ is defined as a tuple $\mathcal{X}_{i,j}, \mathcal{Y}_{i,j} \subseteq \mathbb{R}^{784} \times \{+1, -1\}$ and $\mathbf{b}, c$ are the regression objective parameters to be optimized. This training objective model is convex and satisfies Assumption 1 for finite number of bounded-value data points. The network of computing nodes is considered as an *exponential* graph, where, following Remark 4, this type of network structure gives a lower optimality gap and faster convergence. Some existing (non-quantized) algorithms are used for comparison: *GP* [57], *SGP* [58], *S-ADDOPT* [59], *ADDOPT* [60], and *PushSAGA* [61]. For all algorithms, the step sizes are hand-tuned to achieve the best performance. Fig. 7 shows the comparison results on the MNIST data set. Clearly, the proposed log-quantized algorithm (with $\alpha = 4.1$ and $\rho = 0.0625$) shows good performance as compared to these works.

## 6. Conclusions

### *6.1. Concluding Remarks*

This work provides a distributed co-optimization algorithm for CPU scheduling and different ML applications over dynamic networks. One advantage is

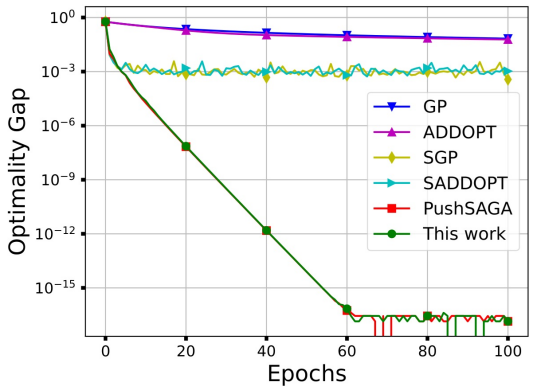



Figure 7: The performance comparison of the proposed ML optimization algorithm on the MNIST dataset with some existing literature in terms of optimality gap (cost residual) and rate of convergence.

that the algorithm optimally assigns the computing resources with all-time resource-demand feasibility and can be terminated at any time depending on the termination criteria. Moreover, the proof is based on eigenspectrum perturbation analysis and the Lyapunov stability theorem, which allows convergence over time-varying networks. On the other hand, although Assumption 2 assumes undirected networks, the optimization of the ML objective alone may converge over weight-balanced directed networks. A limitation is solving the general co-optimization objective (1) over directed networks, which is a direction of our current research. Both simulation results and mathematical analysis show the optimal convergence and all-time feasibility of the algorithm.

### *6.2. Future Directions*

One direction of our future research is to relax Assumption 1 to more general non-convex objective models such that the algorithm can be applied to more complex models, e.g., deep neural networks. As other future research directions, the results can be used for CPU scheduling and computing resource management in different distributed optimization, learning, and filtering applications. For example, distributed optimization and filtering over large transportation networks, computing resource allocation for detection and estimation over large-scale CPS and IoT environments, and resource scheduling for multi-target tracking over large-scale sensor networks. Considering node failure or loss of communications by adding redundancy mechanisms in the

network of computing nodes is another interesting research direction.

## References


[1] J. A. S. Aranda, R. dos Santos Costa, V. W. de Vargas, P. R. da Silva Pereira, J. L. V. Barbosa, M. P. Vianna, Context-aware edge computing and internet of things in smart grids: A systematic mapping study, Computers and Electrical Engineering 99 (2022) 107826.

[2] H. Zamani, H. Zayyani, F. Marvasti, An iterative dictionary learning-based algorithm for DOA estimation, IEEE Communications Letters 20 (9) (2016) 1784–1787.

[3] J. Wang, Multi agent system based smart grid anomaly detection using blockchain machine learning model in mobile edge computing network, Computers and Electrical Engineering 121 (2025) 109825.

[4] B. A. Yosuf, M. Musa, T. Elgorashi, J. Elmirghani, Energy efficient distributed processing for IoT, IEEE Access 8 (2020) 161080–161108.

[5] E. A. Capota, C. S. Stangaciu, M. V. Micea, D. Curiac, Towards mixed criticality task scheduling in cyber physical systems: Challenges and perspectives, Journal of Systems and Software 156 (2019) 204–216.

[6] A. Meloni, P. A. Pegoraro, L. Atzori, A. Benigni, S. Sulis, Cloud-based iot solution for state estimation in smart grids: Exploiting virtualization and edge-intelligence technologies, Computer Networks 130 (2018) 156–165.

[7] P. Di Lorenzo, S. Barbarossa, S. Sardellitti, Distributed signal processing and optimization based on in-network subspace projections, IEEE Transactions on Signal Processing 68 (2020) 2061–2076.

[8] M. Doostmohammadian, A. Aghasi, Accelerated distributed allocation, IEEE Signal Processing Letters 31 (2024) 651–655.

[9] A. Liu, P. B. Luh, B. Yan, M. A. Bragin, A novel integer linear programming formulation for job-shop scheduling problems, IEEE Robotics and Automation Letters 6 (3) (2021) 5937–5944.

[10] M. A. Bragin, P. B. Luh, B. Yan, X. Sun, A scalable solution methodology for mixed-integer linear programming problems arising in automation, IEEE Transactions on Automation Science and Engineering 16 (2) (2018) 531–541.

[11] M. A. Bragin, B. Yan, P. B. Luh, Distributed and asynchronous coordination of a mixed-integer linear system via surrogate lagrangian relaxation, IEEE Transactions on Automation Science and Engineering 18 (3) (2020) 1191–1205.

[12] H. Sayyaadi, M. Moarref, A distributed algorithm for proportional task allocation in networks of mobile agents, IEEE Transactions on Automatic Control 56 (2) (2011) 405–410. doi:10.1109/TAC.2010.2089653.

[13] M. Doostmohammadian, H. Sayyaadi, M. Moarref, A novel consensus protocol using facility location algorithms, in: IEEE Conf. on Control Applications & Intelligent Control, 2009, pp. 914–919.

[14] A. Grammenos, T. Charalambous, E. Kalyvianaki, CPU scheduling in data centers using asynchronous finite-time distributed coordination mechanisms, IEEE Transactions on Network Science and Engineering 10 (4) (2023) 1880–1894.

[15] E. Kalyvianaki, T. Charalambous, S. Hand, Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters, in: Proceedings of the 6th international conference on Autonomic computing, 2009, pp. 117–126.

[16] A. I. Rikos, A. Grammenos, E. Kalyvianaki, C. N. Hadjicostis, T. Charalambous, K. H. Johansson, Optimal CPU scheduling in data centers via a finite-time distributed quantized coordination mechanism, in: 60th IEEE Conference on Decision and Control (CDC), IEEE, 2021, pp. 6276–6281.

[17] N. C. Koutsoukis, D. O. Siagkas, P. S. Georgilakis, N. D. Hatziargyriou, Online reconfiguration of active distribution networks for maximum integration of distributed generation, IEEE Transactions on Automation Science and Engineering 14 (2) (2016) 437–448.

[18] M. Zholbaryssov, D. Fooladivanda, A. D. Dominguez-Garcia, Resilient distributed optimal generation dispatch for lossy AC microgrids, Systems & Control Letters 123 (2019) 47–54.

[19] M. Doostmohammadian, Distributed energy resource management: All-time resource-demand feasibility, delay-tolerance, nonlinearity, and beyond, IEEE Control Systems Letters 7 (2023) 3423 – 3428.

[20] L. Cheng, S. Zhang, Y. Wang, Distributed optimal capacity allocation of integrated energy system via modified ADMM, Applied Mathematics and Computation 465 (2024) 128369.

[21] M. Doostmohammadian, A. Aghasi, M. Vrakopoulou, H. R. Rabiee, U. A. Khan, T. Charalambous, Distributed delay-tolerant strategies for equality-constraint sum-preserving resource allocation, Systems & Control Letters 182 (2023) 105657.

[22] M. Doostmohammadian, H. R. Rabiee, Distributed automatic generation control subject to ramp-rate-limits: Anytime feasibility and uniform network-connectivity, Digital Signal Processing (2025) 105576.

[23] T. T. Doan, S. T. Maguluri, J. Romberg, Fast convergence rates of distributed subgradient methods with adaptive quantization, IEEE Transactions on Automatic Control 66 (5) (2020) 2191–2205.

[24] A. I. Rikos, C. N. Hadjicostis, K. H. Johansson, Non-oscillating quantized average consensus over dynamic directed topologies, Automatica 146 (2022) 110621.

[25] J. Wu, W. Huang, J. Huang, T. Zhang, Error compensated quantized sgd and its applications to large-scale distributed optimization, in: International Conference on Machine Learning, PMLR, 2018, pp. 5325–5333.

[26] Y. Kajiyama, N. Hayashi, S. Takai, Linear convergence of consensus-based quantized optimization for smooth and strongly convex cost functions, IEEE Transactions on Automatic Control 66 (3) (2020) 1254–1261.

[27] L. Jian, J. Hu, J. Wang, K. Shi, Distributed inexact dual consensus ADMM for network resource allocation, Optimal Control Applications and Methods 40 (6) (2019) 1071–1087.

[28] A. Falsone, M. Prandini, Augmented lagrangian tracking for distributed optimization with equality and inequality coupling constraints, Automatica 157 (2023) 111269.

[29] K. Gong, L. Zhang, Primal-dual algorithm for distributed optimization with coupled constraints, Journal of Optimization Theory and Applications 201 (1) (2024) 252–279.

[30] M. Doostmohammadian, W. Jiang, M. Liaquat, A. Aghasi, H. Zarrabi, Discretized distributed optimization over dynamic digraphs, IEEE Transactions on Automation Science and Engineering 22 (2024) 2758–2767.

[31] R. Xin, S. Kar, U. A. Khan, Decentralized stochastic optimization and machine learning: A unified variance-reduction framework for robust performance and fast convergence, IEEE Signal Processing Magazine 37 (3) (2020) 102–113.

[32] S. Sundhar Ram, A. Nedić, V. V. Veeravalli, A new class of distributed optimization algorithms: Application to regression of distributed data, Optimization Methods and Software 27 (1) (2012) 71–88.

[33] A. G. Dimakis, S. Kar, J. M. F. Moura, M. G. Rabbat, A. Scaglione, Gossip algorithms for distributed signal processing, Proceedings of the IEEE 98 (11) (2010) 1847–1864.

[34] S. Kar, J. M. F. Moura, Distributed consensus algorithms in sensor networks with imperfect communication: Link failures and channel noise, IEEE Transactions on Signal Processing 57 (1) (2008) 355–369.

[35] X. Zhang, M. M. Vasconcelos, Top-k data selection via distributed sample quantile inference, in: Learning for Dynamics and Control Conference, PMLR, 2023, pp. 813–824.

[36] Y. Nesterov, Introductory lectures on convex programming, volume I: Basic course, Lecture notes 3 (4) (1998) 5.

[37] D. P. Bertsekas, Necessary and sufficient conditions for a penalty method to be exact, Mathematical programming 9 (1) (1975) 87–99.

[38] O. Chapelle, Training a support vector machine in the primal, Neural computation 19 (5) (2007) 1155–1178.

[39] U. Dogan, T. Glasmachers, C. Igel, A unified view on multi-class support vector classification, The Journal of Machine Learning Research 17 (1) (2016) 1550–1831.

[40] D. Jurafsky, J. H. Martin, Speech and Language Processing, Prentice Hall, 2020.

[41] J. Zhang, R. Jin, Y. Yang, A. G. Hauptmann, Modified logistic regression: An approximation to SVM and its application in large-scale text categorization, in: Proceeding 20th International Conference on Machine Learinig (ICML), Washington, DC, 2003.

[42] M. I. Qureshi, R. Xin, S. Kar, U. A. Khan, A decentralized variance-reduced method for stochastic optimization over directed graphs, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2021, pp. 5030–5034.

[43] B. McMahan, E. Moore, D. Ramage, S. Hampson, B. A. Arcas, Communication-efficient learning of deep networks from decentralized data, in: Artificial intelligence and statistics, PMLR, 2017, pp. 1273–1282.

[44] R. Olfati-Saber, R. M. Murray, Consensus problems in networks of agents with switching topology and time-delays, IEEE Transactions on Automatic Control 49, no. 9 (2004) 1520–1533.

[45] R. Olfati-Saber, J. A. Fax, R. M. Murray, Consensus and cooperation in networked multi-agent systems, IEEE Proceedings 95 (1) (2007) 215–233.

[46] S. Boyd, L. Vandenberghe, Convex Optimization, Cambridge University Press, New York, NY, USA, 2004.

[47] D. P. Bertsekas, A. Nedic, A. E. Ozdaglar, Convexity, duality, and Lagrange multipliers, Lecture Notes, MIT Press.

[48] G. W. Stewart, J. Sun, Matrix perturbation theory, Academic Press, 1990.

[49] K. Cai, H. Ishii, Average consensus on general strongly connected digraphs, Automatica 48 (11) (2012) 2750–2761.

[50] M. Doostmohammadian, M. Pirani, U. A. Khan, T. Charalambous, Consensus-based distributed estimation in the presence of heterogeneous, time-invariant delays, IEEE Control Systems Letters 6 (2021) 1598 – 1603.

[51] J. E. Slotine, W. Li, Applied nonlinear control, Prentice hall Englewood Cliffs, NJ, 1991.

[52] S. Zhu, M. Hong, B. Chen, Quantized consensus ADMM for multi-agent distributed optimization, in: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2016, pp. 4134–4138.

[53] A. Danaee, R. de Lamare, V. Nascimento, Energy-efficient distributed learning with coarsely quantized signals, IEEE Signal Processing Letters 28 (2021) 329–333.

[54] O. A. Hanna, Y. H. Ezzeldin, C. Fragouli, S. Diggavi, Quantization of distributed data for learning, IEEE Journal on Selected Areas in Information Theory 2 (3) (2021) 987–1001.

[55] N. Bastianello, A. I. Rikos, K. H. Johansson, Online distributed learning with quantized finite-time coordination, in: 62nd IEEE Conference on Decision and Control (CDC), IEEE, 2023, pp. 5026–5032.

[56] R. Xin, U. A. Khan, S. Kar, A fast randomized incremental gradient method for decentralized nonconvex optimization, IEEE Transactions on Automatic Control 67 (10) (2021) 5150–5165.

[57] A. Nedić, A. Olshevsky, Distributed optimization over time-varying directed graphs, IEEE Transactions on Automatic Control 60 (3) (2014) 601–615.

[58] A. Spiridonoff, A. Olshevsky, I. Paschalidis, Robust asynchronous stochastic gradient-push: Asymptotically optimal and network-independent performance for strongly convex functions, Journal of machine learning research 21 (58).

[59] M. I. Qureshi, R. Xin, S. Kar, U. A. Khan, S-ADDOPT: Decentralized stochastic first-order optimization over directed graphs, IEEE Control Systems Letters 5 (3) (2020) 953–958.

[60] C. Xi, R. Xin, U. A. Khan, ADD-OPT: accelerated distributed directed optimization, IEEE Transactions on Automatic Control 63 (5) (2017) 1329–1339.

[61] M. I. Qureshi, R. Xin, S. Kar, U. A. Khan, Push-SAGA: a decentralized stochastic algorithm with variance reduction over directed graphs, IEEE Control Systems Letters 6 (2021) 1202–1207.