
This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Rikos, Apostolos I.; Grammenos, Andreas; Kalyvianaki, Evangelia; Hadjicostis, Christoforos N.; Charalambous, Themistoklis; Johansson, Karl H.

Distributed Optimization for Quadratic Cost Functions With Quantized Communication and Finite-Time Convergence

Published in:
IEEE Transactions on Control of Network Systems

DOI:
[10.1109/TCNS.2024.3431413](https://doi.org/10.1109/TCNS.2024.3431413)

Published: 01/01/2025

Document Version
Publisher's PDF, also known as Version of record

Published under the following license:
CC BY

Please cite the original version:
Rikos, A. I., Grammenos, A., Kalyvianaki, E., Hadjicostis, C. N., Charalambous, T., & Johansson, K. H. (2025). Distributed Optimization for Quadratic Cost Functions With Quantized Communication and Finite-Time Convergence. *IEEE Transactions on Control of Network Systems*, 12(1), 930-942.
<https://doi.org/10.1109/TCNS.2024.3431413>

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.

Distributed Optimization for Quadratic Cost Functions With Quantized Communication and Finite-Time Convergence

Apostolos I. Rikos , *Member, IEEE*, Andreas Grammenos , Evangelia Kalyvianaki ,
 Christoforos N. Hadjicostis , *Fellow, IEEE*, Themistoklis Charalambous , *Senior Member, IEEE*,
 and Karl H. Johansson , *Fellow, IEEE*

Abstract—In this article, we propose two distributed iterative algorithms that can be used to solve the distributed optimization problem for quadratic local cost functions over large-scale networks in finite time. The first algorithm exhibits synchronous operation while the second one exhibits asynchronous operation. Both algorithms operate exclusively with quantized values. This means that the information stored, processed, and exchanged between neighboring nodes is subject to deterministic uniform quantization. The algorithms rely on event-driven updates in order to reduce energy consumption, communication bandwidth, network congestion, and/or processor usage. Finally, once the algorithms converge, nodes distributively terminate their operation. We prove that our algorithms converge in a finite number of iterations to the exact optimal solution depending on the quantization level, and we present applications of our algorithms to, first, optimal task scheduling for data centers, and second, global model aggregation for distributed federated learning. We provide simulations of these applications to illustrate the operation, performance, and advantages of the proposed algorithms. In addition, it is shown that our proposed algorithms compare favorably to algorithms in the current literature.

Index Terms—Distributed algorithms, federated learning, finite-time, optimization, quantization, resource allocation.

I. INTRODUCTION

IN VARIOUS applications, such as cloud computing, power systems, and traffic networks, the conventional approach to optimize performance involves gathering data from all users at a central processor, which handles the resource-intensive computations and then distributes the results back to the users. Centralized optimization requires data transfer to the central processor (e.g., [2], [3], [4], and [5]), and leads to increased network traffic and bottlenecks. They also lack scalability, demand significant computational power for the central processor, and are susceptible to a single point of failure (as highlighted in [6]).

In recent years, rapid advancements in telecommunication systems have spurred the emergence of new networked applications, including distributed localization, estimation, and target tracking. These applications involve multiple agents collaborating through wired or wireless channels to achieve common objectives. Consequently, there is a growing interest in the control and coordination of networked systems, with distributed optimization being a key focus [7], [8]. Distributed optimization addresses the limitations of centralized algorithms and enables innovative applications. It offers benefits, such as scalability, resilience to failures, enhanced performance, and efficient utilization of network resources.

Related Works: Most distributed optimization algorithms rely on real-valued message exchanges [7]. For instance, in [8], nodes update using subgradients and communicate over doubly stochastic networks. Tsianos et al. [9] performed subgradient descent and coordinate using push-sum consensus. In [10], nodes operate asynchronously over stochastic networks, employing surrogate gradients and dynamic average consensus. The algorithm in [11] uses ratio consensus and combines received information with gradients over column stochastic networks. In [12], nodes coordinate over row-stochastic and column-stochastic matrices, employing gradient information, while in [13], nodes perform local Bayesian updates and average log beliefs. Finally, in [14], nodes exchange messages with neighbors and combine them with gradients, achieving linear convergence in

Manuscript received 2 November 2023; accepted 6 January 2024. Date of publication 19 July 2024; date of current version 20 March 2025. The work of Themistoklis Charalambous was supported in part by MINERVA, a European Research Council (ERC) project funded under the European Union's Horizon 2022 research and innovation programme under Grant 101044629. Recommended by Associate Editor S. Patterson. An earlier version of this paper was presented in part at the 2021 60th IEEE Conference on Decision and Control (CDC) [DOI: 10.1109/CDC45484.2021.9683763]. (*Corresponding author: Themistoklis Charalambous.*)

Apostolos I. Rikos and Karl H. Johansson are with the Division of Decision and Control Systems, KTH Royal Institute of Technology, 11428 Stockholm, Sweden, and also with Digital Futures, 11428 Stockholm, Sweden (e-mail: rikos@kth.se; kallejg@kth.se).

Andreas Grammenos and Evangelia Kalyvianaki are with the Department of Computer Science and Technology, University of Cambridge, CB3 0FD Cambridge, U.K., and also with Alan Turing Institute, NW1 2DB London, U.K. (e-mail: ag926@cl.cam.ac.uk; ek264g@cl.cam.ac.uk).

Christoforos N. Hadjicostis is with the Department of Electrical and Computer Engineering, School of Engineering, University of Cyprus, 1678 Nicosia, Cyprus (e-mail: hadjicostis.christoforos@ucy.ac.cy).

Themistoklis Charalambous is with the Department of Electrical and Computer Engineering, School of Engineering, University of Cyprus, 1678 Nicosia, Cyprus, and also with the Department of Electrical Engineering and Automation, School of Electrical Engineering, Aalto University, 02150 Espoo, Finland (e-mail: charalambous.themistoklis@ucy.ac.cy).

Digital Object Identifier 10.1109/TCNS.2024.3431413

synchronous and asynchronous ways over row- and column-stochastic matrices.

The aforementioned algorithms require real-valued messages (i.e., channels of infinite capacity) and either achieve asymptotic convergence or terminate near optimality when operating for a limited time frame [6], [15]. Specifically, the work in [6] combines finite-time consensus and distributed stopping to compute a closed-form solution. In [15], nodes employ the conjugate gradient method over a doubly stochastic network model, achieving linear convergence. In [16], nodes combine gradient descent with a finite-time protocol, ensuring finite-time convergence without error. However, they require high-precision real number exchanges, assuming unlimited channel capacity.

Recently, there has been an increasing interest within the control and machine learning communities for distributed optimization algorithms that exhibit efficient communication and involve the exchange of quantized values. Lee et al. [17] introduced a method where nodes combine gradient tracking with perturbed quantized consensus, operating over an undirected communication topology and achieving geometric convergence. The work in [18] focuses on nodes exchanging quantized messages and updating their states through a linear combination of gradients and received messages, achieving a vanishing mean solution error. Koloskova et al. [19] proposed a linearly converging gossip algorithm that supports compressed messages, and relies on stochastic gradient descent and local coordination. In [20], the proposed algorithm relies on periodic averaging, partial device participation, and quantized messages, and is able to achieve near-optimal convergence guarantees. Doostmohammadian et al. [21] proposed an algorithm that relies on the Laplacian-gradient model and allows nodes to exchange logarithmically quantized messages. In [22], the approach centers around quantized distributed gradient tracking. The authors establish lower bounds for the number of quantization levels for achieving linear convergence.

It is important to note that since the aforementioned approaches are mainly quantizing the values of a real-valued distributed optimization algorithm, they maintain the algorithm's asymptotic convergence behavior. To the best of the authors' knowledge, no existing algorithm in the literature simultaneously utilizes quantized values for efficient processing and communication, and achieves finite-time convergence to either the exact optimal solution or one in close proximity depending on the quantization level (ensuring that the difference between the calculated and exact solutions remains smaller than the quantization level).

The aim of this article is to introduce a pioneering approach that integrates the above characteristics, thereby paving the way for the implementation of finite-time algorithms operating exclusively with quantized values to address distributed optimization challenges within directed networks. Note here that unlike prior approaches, our algorithm exploits the structure of quadratic cost functions, enabling the calculation of an optimal solution without the need for gradient calculations. This operational characteristic not only enhances efficiency but also leads to fast convergence rates, as it will be shown later in this article.

Main contributions: In this article, we focus on distributed optimization over quadratic local cost functions. We take into

consideration that the exchanged messages consist of quantized values, and propose two distributed algorithms—one synchronous and one asynchronous—that solve the optimization problem in a finite number of steps. The algorithms terminate their operation once they have agreed on the solution, which is either optimal or in the proximity of the optimal solution, depending on the quantization level. The main contributions of this article are as follows.

- 1) We present a novel synchronous distributed algorithm that solves the optimization problem in a finite number of time steps using quantized values (see Algorithm 1). A distributed stopping mechanism is deployed in order to terminate the algorithm in a finite number of time steps. We provide an upper bound on the number of time steps needed for convergence based on properties of primitive matrices. The convergence time relies on the network connectivity as determined by the diameter of the network, rather than the number of network nodes (see Theorem 1).
- 2) We present an asynchronous version of our synchronous algorithm, which solves the optimization problem in a finite number of steps using quantized values (see Algorithm 2). Then, we discuss a modified asynchronous algorithm for the case where each node requires a random number of time steps to process the received information. We provide an upper bound on the number of time steps needed for convergence of our asynchronous algorithm (see Theorem 2).
- 3) Although the proposed algorithms could be used in many applications, here, we discuss them within the context of 1) resource management in cloud infrastructures [6], and 2) global model aggregation in distributed federated learning systems [13], [20], [23], [24], [25], [26] (see Section V). The first application is task scheduling, which is a problem that optimally allocates tasks to server machines. The goal is to balance the central processing unit (CPU) utilization across data center servers in a distributed fashion [1], [6], [21]. The second application is global model aggregation, over a federated learning system. The goal is to calculate the global model parameters by aggregating the local model parameters of each node (obtained by local training on each node) in a distributed fashion [23], [25], [26].

Although influenced by works, such as in [27], [28], [29], and [30], our algorithms offer a unique approach due to their quantized operation, finite-time convergence, and the ability for operation termination. In addition, while our stopping mechanism is inspired from Giannini et al. [30], it is adjusted to the quantized nature of our algorithms, using two parallel linear iterations.

Article organization: The rest of this article is organized as follows. Section II introduces the notations used in this article. Section III provides the problem formulation. Section IV presents the proposed algorithms and their convergence analysis. Section V covers applications and algorithm comparisons. Finally, Section VI concludes this article.

II. NOTATION AND PRELIMINARIES

Notations: The sets of real, rational, integer, and natural numbers are denoted by \mathbb{R} , \mathbb{Q} , \mathbb{Z} , and \mathbb{N} , respectively. Symbol

$\mathbb{Z}_{\geq 0}$ ($\mathbb{Z}_{>0}$) denotes the set of nonnegative (positive) integer numbers, while $\mathbb{Z}_{\leq 0}$ ($\mathbb{Z}_{<0}$) denotes the set of nonpositive (negative) integer numbers. For any real number $a \in \mathbb{R}$, the floor $\lfloor a \rfloor$ denotes the greatest integer less than or equal to a while the ceiling $\lceil a \rceil$ denotes the least integer greater than or equal to a . Vectors are denoted by small letters, matrices are denoted by capital letters, and the transpose of a matrix A is denoted by A^T . For a matrix $A \in \mathbb{R}^{n \times n}$, the entry at row i and column j is denoted by A_{ij} . By $\mathbf{1}$, we denote the all-ones column vector, and by I , we denote the identity matrix (of appropriate dimensions).

Network preliminaries: The communication topology of a network of n ($n \geq 2$) nodes communicating only with their immediate neighbors can be captured by a directed graph (digraph) defined as $\mathcal{G}_d = (\mathcal{V}, \mathcal{E})$. In digraph \mathcal{G}_d , $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$ is the set of nodes, whose cardinality is denoted as $|\mathcal{V}| = n$, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} - \{(v_j, v_j) \mid v_j \in \mathcal{V}\}$ is the set of edges (self-edges excluded) whose cardinality is denoted as $m = |\mathcal{E}|$. A directed edge from node v_i to node v_j is denoted by $m_{ji} \triangleq (v_j, v_i) \in \mathcal{E}$, and captures the fact that node v_j can receive information from node v_i (but not the other way around). We assume that the given digraph $\mathcal{G}_d = (\mathcal{V}, \mathcal{E})$ is *strongly connected*, i.e., for each pair of nodes $v_j, v_i \in \mathcal{V}$, $v_j \neq v_i$, there exists a directed path from v_i to v_j . A directed path of length t from v_i to v_j exists if we can find a sequence of nodes $v_i \equiv v_{l_0}, v_{l_1}, \dots, v_{l_t} \equiv v_j$ such that $(v_{l_{\tau+1}}, v_{l_\tau}) \in \mathcal{E}$ for $\tau = 0, 1, \dots, t-1$. Furthermore, the diameter D of a digraph is the longest shortest path between any two nodes $v_j, v_i \in \mathcal{V}$ in the network. The subset of nodes that can directly transmit information to node v_j is called the set of in-neighbors of v_j and is represented by $\mathcal{N}_j^- = \{v_i \in \mathcal{V} \mid (v_j, v_i) \in \mathcal{E}\}$. The cardinality of \mathcal{N}_j^- is called the *in-degree* of v_j and is denoted by \mathcal{D}_j^- . The subset of nodes that can directly receive information from node v_j is called the set of out-neighbors of v_j and is represented by $\mathcal{N}_j^+ = \{v_l \in \mathcal{V} \mid (v_l, v_j) \in \mathcal{E}\}$. The cardinality of \mathcal{N}_j^+ is called the *out-degree* of v_j and is denoted by \mathcal{D}_j^+ .

III. PROBLEM FORMULATION

Let us consider a strongly connected network $\mathcal{G}_d = (\mathcal{V}, \mathcal{E})$. We focus on the scenario where nodes in a network cooperatively minimize a common additive cost function. Traditionally, each one of the $n = |\mathcal{V}|$ nodes is endowed with (and has information only for) a scalar local cost function $f_i : \mathbb{R} \mapsto \mathbb{R}$. Since, in this work, we consider the exchange of integer¹ values, the local cost function takes rational numbers as inputs. In addition, since the update of each node is also quantized, the outputs are also rational, i.e., $f_i : \mathbb{Q} \mapsto \mathbb{Q}$. Nodes aim to cooperatively solve the optimization problem, herein called P1, in finite time and terminate their operation once calculating the optimal solution. P1 is as follows:

$$\mathbf{P1} : \min_{x \in \mathbb{Q}^n} f(x_1, x_2, \dots, x_n) \equiv \sum_{i=1}^n f_i(x_i) \quad (1a)$$

$$\text{s.t. } x_i = x_j \quad \forall v_i, v_j \in \mathcal{V} \quad (1b)$$

$$\text{and nodes exchange integer values.} \quad (1c)$$

¹We assume that states are integer-valued, which captures a class of quantization effects, such as uniform quantization.

In this work, we restrict our attention to a quadratic local cost function for every node v_i of the form, cf., [7] and [31]

$$f_i(x_i) = \frac{1}{2} \alpha_i (x_i - \rho_i)^2 \quad (2)$$

where $\alpha_i \in \mathbb{Q}$ and $\rho_i \in \mathbb{Q}$ are given parameters. This cost function represents the demand at node v_i with x_i being a global optimization parameter that will determine the optimal solution at each node. Note that the choice of quadratic local cost functions allows us to calculate a closed-form expression of the optimal solution, which can be computed distributively by applying consensus algorithms. Specifically, the optimization problem (1a) can be solved in a closed form, and the optimal solution x^* is given by

$$x^* = \frac{\sum_{i=1}^n \alpha_i \rho_i}{\sum_{i=1}^n \alpha_i}. \quad (3)$$

Since $\alpha_i \in \mathbb{Q}$ and $\rho_i \in \mathbb{Q}$, then $x^* \in \mathbb{Q}$. Note that if $\alpha_i \in \mathbb{Q}$ and $\rho_i \in \mathbb{Q}$, we can use simple transformations so that we can transform $\alpha_i \in \mathbb{Z}$ and $\rho_i \in \mathbb{Z}$; for simplicity of exposition, we adopt this assumption, i.e., $\alpha_i \in \mathbb{Z}$ and $\rho_i \in \mathbb{Z}$. Furthermore, we assume that the initial values of the states, $x_i[0]$, are integers, i.e., $x_i[0] \in X_0 \subset \mathbb{Z}$ (e.g., $x_i[0]$ can be the CPU utilization percentage of a server). Furthermore, note that our proposed algorithm's calculation of the exact optimal solution depends on the quantization level. This means that the distance of the calculated and the exact solution is always less than or equal to the quantization level.

IV. QUANTIZED DISTRIBUTED SOLUTION

In this section, we propose two distributed quantized information exchange algorithms that solve the problem described in Section III. The proposed algorithms are detailed as Algorithms 1 and 2, and they calculate x^* shown in (3) in finite time for the case where the updates of every node during the algorithm's operation are synchronous and asynchronous, respectively. In order to solve the problem in a distributed way, we make the following assumptions.

Assumption 1: The communication topology is a strongly connected digraph.

Assumption 2: The diameter of the network D (or an upper bound D') is known to all nodes $v_j \in \mathcal{V}$.

Assumption 1 is a necessary condition for each node v_j to be able to calculate the optimal allocation after a finite number of time steps. Assumption 2 is necessary for the operation of our algorithm and for coordinating the min- and max-consensus algorithms, such that each node v_j is able to determine whether convergence has been achieved and thus the operation can be terminated.

A. Synchronous Quantized Distributed Solution

We now describe the main steps of Algorithm 1.

Step 1—Input and initialization: Each node v_j has two integer² values $y_j[0], z_j[0] \in \mathbb{Z}$, which represent its initial state

²Since communication is over digital channels, the initial states, if not already quantized, are quantized by the nodes. We aim to find the optimal solution for the case where the initial states are quantized (or are real and were quantized by the nodes).

(note that $y_j[0] = \alpha_j \rho_j$ and $z_j[0] = \rho_j$). Furthermore, each node v_j has knowledge of the diameter of the network D (or an upper bound D'). During initialization, each node selects a set of probabilities $\{b_{lj} \mid v_l \in \mathcal{N}_j^+ \cup \{v_j\}\}$ such that $0 < b_{lj} < 1$ and $\sum_{l=1}^n b_{lj} = 1$; see Initialization step 1 (note that $b_{lj} = 0$ for $v_l \notin \mathcal{N}_j^+ \cup \{v_j\}$). Each value b_{lj} represents the probability for node v_j to transmit toward itself or out-neighbor $v_l \in \mathcal{N}_j^+ \cup \{v_j\}$ at any given time step (independently between time steps). Furthermore, each node v_j sets its flag $_j$ equal to zero; see Initialization step 2. This flag $_j$ allows node v_j to determine whether it needs to terminate its operation, because the proposed algorithm has reached completion.

Step 2—Synchronously computing the quantized optimal solution: At each time step k , each node v_j updates its state variables $z_j^s[k]$, $y_j^s[k]$, and $q_j^s[k]$; see Iteration step 3. Then, it splits $y_j[k]$ into $z_j[k]$ equal integer pieces, with the possibility of some pieces having values greater by one; see Iteration steps 5.1 and 5.2. Then, node v_j transmits the $z_j[k] - 1$ pieces to randomly selected out-neighbors or to itself according to the probabilities b_{lj} ; see Iteration step 5.3. It receives the transmitted values from its in-neighbors and updates its variables $z_j[k+1]$ and $y_j[k+1]$; see Iteration step 6.

Step 3—Determining when to stop: Every D (or D') time steps, the two integer values M_j and m_j are updated to match the node's state; see Iteration step 1. For the subsequent D (or D') time steps, node v_j simultaneously runs a max-consensus algorithm [32] using M_j to compute the maximum state in the network, as well as a min-consensus algorithm, using m_j to determine the minimum state in the network; see Iteration step 2. Upon completion of the D (or D') time steps, if the maximum state is equal to the minimum state in the network, or their difference is just one, node v_j infers that the algorithm has successfully converged. In this case, it proceeds to calculate the optimal solution x^* as defined in (3), and subsequently concludes its operation; see Iteration step 6.

The main advantage of Algorithm 1 is its ability to leverage the structure of quadratic cost functions and deliver an optimal solution in finite time. This is facilitated by the fact that the variable z_j for each node v_j serves as the denominator, whereas the variable y_j functions as the numerator in the optimal solution described in (3). Compared to other approaches in the literature, Algorithm 1 eliminates the need for each node v_j to compute the gradient of its local objective function. This unique feature steers each node away from the typical asymptotic convergence toward the optimal solution, enabling them to converge in finite time. Furthermore, it enables the utilization of quantized values for both processing and communication. This dual advantage not only enhances efficiency but also leads to fast convergence rates, as will be demonstrated later in this article.

Theorem 1: Consider a strongly connected digraph $\mathcal{G}_d = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges and $z_j[0], y_j[0]$ for every node $v_j \in \mathcal{V}$ at time step $k = 0$. Suppose that each node $v_j \in \mathcal{V}$ follows the Initialization and Iteration steps as described in Algorithm 1. Each node v_j is able to calculate the optimal x^* shown in (3) after a finite number of time steps and terminate its operation after calculating x^* .

Proof: See Appendix A. ■

Algorithm 1: Synchronous Quantized Distributed Optimization.

Input: A strongly connected digraph $\mathcal{G}_d = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. Each node $v_j \in \mathcal{V}$ has knowledge of $D, y_j[0], z_j[0]$.

Initialization: Each node $v_j \in \mathcal{V}$ does the following:

- 1) Assigns a nonzero probability b_{lj} to each of its outgoing edges m_{lj} , where $v_l \in \mathcal{N}_j^+ \cup \{v_j\}$, as follows

$$b_{lj} = \begin{cases} \frac{1}{1+D_j^+}, & \text{if } l = j \text{ or } v_l \in \mathcal{N}_j^+, \\ 0, & \text{if } l \neq j \text{ and } v_l \notin \mathcal{N}_j^+. \end{cases}$$

- 2) Sets flag $_j = 0, y_j[0] := 2y_j[0], z_j[0] := 2z_j[0]$.

Iteration: For $k = 1, 2, \dots$, each node $v_j \in \mathcal{V}$ does:

• **while** flag $_j = 0$ **then**

- 1) **if** $k \bmod D = 1$ **then** sets $M_j = \lceil y_j[k]/z_j[k] \rceil, m_j = \lfloor y_j[k]/z_j[k] \rfloor$;
- 2) broadcasts M_j, m_j to every $v_l \in \mathcal{N}_j^+$; receives M_i, m_i from every $v_i \in \mathcal{N}_j^-$; sets $M_j = \max_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} M_i, m_j = \min_{v_i \in \mathcal{N}_j^- \cup \{v_j\}} m_i$;
- 3) sets $z_j^s[k] = z_j[k], y_j^s[k] = y_j[k]$, and $q_j^s[k] = \left\lceil \frac{y_j^s[k]}{z_j^s[k]} \right\rceil$;
- 4) sets $d_j^z = z_j[k], y_j[k+1] = 0, z_j[k+1] = 0$;
- 5) **while** $d_j^z > 1$, **then**
 - 5.1) $d_j^y = \lfloor y_j[k] / z_j[k] \rfloor$;
 - 5.2) $y_j[k] = y_j[k] - d_j^y, z_j[k] = z_j[k] - 1$, and $d_j^z = d_j^z - 1$;
 - 5.3) transmits d_j^y to randomly chosen out-neighbor $v_l \in \mathcal{N}_j^+ \cup \{v_j\}$ according to b_{lj} ;
 - 5.4) receives d_i^y from in-neighbor $v_i \in \mathcal{N}_j^-$ and sets

$$y_j[k+1] = y_j[k+1] + \sum_{j=1}^n w_{ji}[k] d_j^y \quad (4)$$

$$z_j[k+1] = z_j[k+1] + \sum_{j=1}^n w_{ji}[k] \quad (5)$$

where $w_{ji}[k] = 1$ if node v_j receives a message from $v_i \in \mathcal{N}_j^-$ at iteration k (otherwise $w_{ji}[k] = 0$);

- 6) **if** $k \bmod D = 0$ **then, if** $M_j - m_j \leq 1$ **then** sets $q_j^s[k] = m_j, x_j^* = x^* = y_j[0]/q_j^s[k]$ and flag $_j = 1$.

Output: (3) holds for every $v_j \in \mathcal{V}$.

B. Asynchronous Quantized Distributed Solution

We now focus on the case where nodes operate in an asynchronous fashion.

Assumption 3: The number of time steps required for a node v_j to process the information received from its in-neighbors is upper bounded by \mathcal{B} .

Assumption 3 is necessary for the operation of the asynchronous version of the max/min consensus algorithm. Specifically, if we have a bound on the number of time steps required for a node v_j to process the information received from its in-neighbors, we can ensure that each node v_j can still determine

Algorithm 2: Asynchronous Quantized Distributed Optimization.

Input: A strongly connected digraph $\mathcal{G}_d = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. Each node $v_j \in \mathcal{V}$ has knowledge of $D, \mathcal{B}, y_j[0], z_j[0]$.

Initialization: Same as Algorithm 1.

Iteration: For $k = 1, 2, \dots$, each node $v_j \in \mathcal{V}$, does:

• **while** $\text{flag}_j = 0$ **then**

- 1) **if** $k \bmod (DB) = 1$ **then** sets $M_j = \lceil y_j[k]/z_j[k] \rceil$, $m_j = \lfloor y_j[k]/z_j[k] \rfloor$;
- 2 – 5.3) same as Algorithm 1;
- 5.4) receives d_i^y from in-neighbor $v_i \in \mathcal{N}_j^-$ and sets

$$y_j[k+1] = y_j[k+1] + \sum_{i=1}^n \sum_{r=0}^{\mathcal{B}} w_{k-r,ji}[r] d_i^y, \quad (6)$$

$$z_j[k+1] = z_j[k+1] + \sum_{i=1}^n \sum_{r=0}^{\mathcal{B}} w_{k-r,ji}[r], \quad (7)$$

where $w_{k-r,ji}[r] = 1$ when the required processing time of node v_i is equal to r at time step $k-r$, so that node v_j receives a message from v_i at time step k (otherwise $w_{k-r,ji}[r] = 0$ and v_j receives no message at time step k from v_i);

- 6) **if** $k \bmod (DB) = 0$ **then, if** $M_j - m_j \leq 1$ **then** sets $q_j^s[k] = m_j$, $x_j^* = x^* = y_j[0]/q_j^s[k]$ and $\text{flag}_j = 1$.

Output: (8) holds for every $v_j \in \mathcal{V}$.

whether convergence has been achieved or not, even when operating asynchronously.

We now describe the main steps of Algorithm 2.

Step 1—Input and initialization: Same as Algorithm 1, but each node also has knowledge of \mathcal{B} .

Step 2—Asynchronously computing quantized optimal solution: Same as Algorithm 1, but processing requires a number of time steps upper bounded by \mathcal{B} .

Step 3—Asynchronously determining when to stop: The main idea is that each node prolongs for DB time steps in the max/min consensus operation in order for every node to participate in the max/min consensus operation (because the random processing delays are upper bounded by \mathcal{B}). Specifically, every DB (or $D'\mathcal{B}$) time steps, the integer values M_j and m_j are set equal to v_j 's state; see Iteration step 1. Then, for the subsequent DB (or $D'\mathcal{B}$) time steps, node v_j executes an asynchronous max-consensus algorithm with M_j for calculating the maximum state in the network, and an asynchronous min-consensus algorithm with m_j for calculating the minimum state in the network; see Iteration step 2. If the maximum state is equal to the minimum state in the network (or their difference is equal to one), then node v_j knows that the algorithm has converged; see Iteration step 6.

Theorem 2: Consider a strongly connected digraph $\mathcal{G}_d = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges, and initial values $z_j[0], y_j[0]$ for every node $v_j \in \mathcal{V}$ at time step $k = 0$. Suppose that each node $v_j \in \mathcal{V}$ follows the Initialization and Iteration

steps as described in Algorithm 2. Each node v_j is able to 1) calculate the optimal x^* shown in (3) after a finite number of time steps, and 2) after calculating x^* , terminate its operation.

Proof: See Appendix B. ■

V. APPLICATIONS

We now present applications of Algorithms 1 and 2 for 1) task scheduling in a data center [1], [6], and 2) global model aggregation in a federated learning system [23], [25], [26]. Finally, we compare Algorithms 1 and 2 with other algorithms in the current literature.

A. Quantized Task Scheduling

Resource management in data centers involves efficiently allocating tasks to CPU resources to meet performance goals. Resource allocation can be a challenging optimization problem due to the scale, heterogeneity, and dynamic nature of modern computer networks. Despite high-bandwidth communication, adopting quantized values is pivotal. They enhance resource efficiency, which is vital for large-scale data centers. Quantized values ensure cost-effectiveness, energy efficiency, and expedite low-latency applications, such as real-time data processing. Scalability benefits as well, reducing computational loads and potential congestion. The adaptability of quantized values to variable network conditions is crucial, ensuring robust performance. Finally, quantized values offer an ideal framework for encryption, enhancing security for sensitive data in data centers.

Overall, task scheduling aims to balance CPU utilization across server nodes by carefully deciding how to allocate tasks to CPU resources in a distributed fashion. In what follows, we describe the task modeling and optimization problem for CPU scheduling. Note that these are borrowed from [6].

Task Modeling (see [6]): A job is defined as a group of tasks, and \mathcal{J} denotes the set of all jobs to be scheduled. Each job $b_j \in \mathcal{J}, j \in \{1, \dots, |\mathcal{J}|\}$, requires ρ_j cycles to be executed. The estimated amount of resources (i.e., CPU cycles) needed for each job is assumed to be known before the optimization starts. A job task could require resources ranging from 1 to ρ_j cycles, and the total sum of resources for all tasks of the same job is equal to ρ_j cycles. The total task workload due to the jobs arriving at node v_i is denoted by l_i . The time horizon T_h is defined as the time period for which the optimization is considering the jobs to be running on the server nodes, before the next optimization decides the next allocation of resources. Hence, in this setting, the CPU capacity of each node, considered during the optimization, is computed as $\pi_i^{\max} := c_i T_h$, where c_i is the sum of all clock rate frequencies of all processing cores of node v_i given in cycles/second. The CPU availability for node v_i at optimization step m (i.e., at time mT_h) is given by $\pi_i^{\text{avail}}[m] := \pi_i^{\max} - u_i[m]$, where $u_i[m]$ is the number of unavailable/occupied cycles due to predicted or known utilization from already running tasks on the server over the time horizon T_h at step m . Note here that all the above quantities are discrete values. Thus, they can be represented by integer values.

Assumption 4: We assume that the time horizon is chosen such that the total amount of resources demanded at a specific optimization step m , denoted by $\rho[m] := \sum_{j=1}^n \rho_j[m]$, is smaller than the total capacity of the network available, given by $\pi^{\text{avail}}[m] := \sum_{i=1}^n \pi_i^{\text{avail}}[m]$, i.e., $\rho[m] \leq \pi^{\text{avail}}[m]$.

Assumption 4 indicates that there is no more demand than the available resources. This assumption is realistic, since the time horizon T_h can be chosen appropriately to fulfill the requirement. In case this assumption is violated, the solution will be that all resources are being used and some tasks will not be scheduled, due to lack of resources. Note that handling this case is out of the scope of this article.

Optimization Problem (see [6]): Server nodes require to calculate the optimal solution at every optimization step m via a distributed coordination algorithm, which relies on the exchange of quantized values and converges after a finite number of time steps. Specifically, all nodes aim to balance their CPU utilization (i.e., the same percentage of capacity) during the execution of the tasks, i.e.,

$$\begin{aligned} \frac{w_i^*[m] + u_i[m]}{\pi_i^{\text{max}}} &= \frac{w_j^*[m] + u_j[m]}{\pi_j^{\text{max}}} \\ &= \frac{\rho[m] + u_{\text{tot}}[m]}{\pi^{\text{max}}} \quad \forall v_i, v_j \in \mathcal{V} \end{aligned} \quad (8)$$

where $w_i^*[m]$ is the *optimal* task workload to be added to server node v_i at optimization step m , $\pi^{\text{max}} := \sum_{i=1}^n \pi_i^{\text{max}}$, and $u_{\text{tot}}[m] = \sum_{i=1}^n u_i[m]$. To achieve the requirement set in (8), we need the solution [according to (3)] to be [6]

$$x^* = \frac{\sum_{i=1}^n \pi_i^{\text{max}} \frac{\rho_i + u_i}{\pi_i^{\text{max}}}}{\sum_{i=1}^n \pi_i^{\text{max}}} = \frac{\rho + u_{\text{tot}}}{\pi^{\text{max}}}. \quad (9)$$

Hence, we modify (2) accordingly. Then, the cost function $f_i(z)$ in (2) is given by $f_i(z) = \frac{1}{2} \pi_i^{\text{max}} (z - \frac{\rho_i + u_i}{\pi_i^{\text{max}}})^2$. In other words, each node computes its proportion of task workload and from that it computes the task workload w_i^* to receive, i.e.,

$$w_i^* = \frac{\rho + u_{\text{tot}}}{\pi^{\text{max}}} \pi_i^{\text{max}} - u_i. \quad (10)$$

Application of Algorithms 1 and 2: During the task scheduling problem, each node v_j aims to 1) calculate the optimal required task workload w_j^* [shown in (10)] after a finite number of time steps, and 2) terminate its operation after calculating w_j^* . In order to solve the above task scheduling problem, Algorithms 1 and 2 need to be modified as follows: each node v_j needs to 1) have knowledge of $D, \mathcal{B}, l_j, u_j, \pi_j^{\text{max}} \in \mathbb{Z}$ (\mathcal{B} is required only for Algorithm 2), and 2) initialize $z_j[0] := l_j + u_j, y_j[0] = \pi_j^{\text{max}}$.

Remark 1: Note that the main difference of the operation of Algorithm 1 compared with the algorithm presented in [1] is that each server node $v_j \in \mathcal{V}$ does *not* need knowledge of an upper bound π^{upper} regarding the total capacity of the network π^{max} (i.e., $\pi^{\text{upper}} \geq \pi^{\text{max}}$, where $\pi^{\text{max}} := \sum_{j=1}^n \pi_j^{\text{max}}$). Specifically, each node does not need to multiply its initial value $y_j[0]$ with π^{upper} so that $y_j[0] > z_j[0]$, since it is already guaranteed that $y_j[0] > z_j[0]$ (which is necessary during the operation of our algorithm, so that each node v_j is able to split $y_j[k]$ into $z_j[k]$ equal integer pieces (or with maximum difference between them

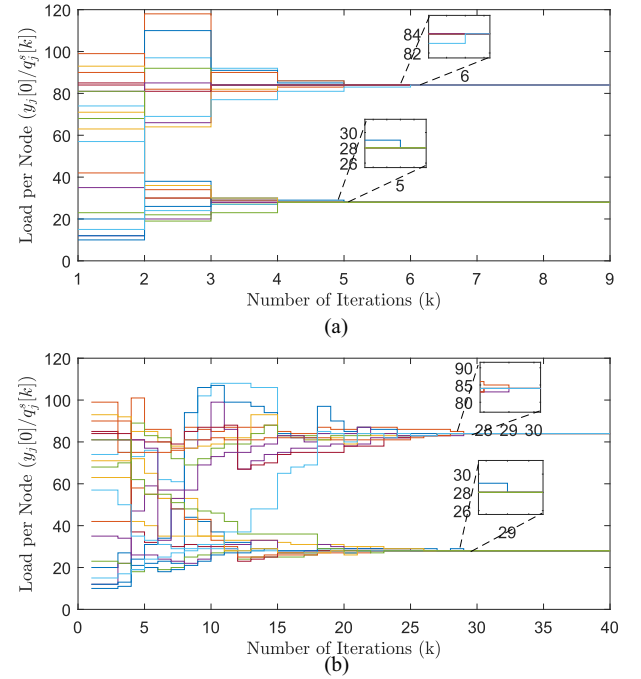


Fig. 1. Execution of (a) Algorithm 1, and (b) Algorithm 2 with $\mathcal{B} = 5$, over a random network comprised of 20 nodes having a diameter equal to 2. Note the different scale of the x-axis; the convergence of the synchronous algorithm is about five times faster than the asynchronous one.

equal to 1) at each time step $k \in \mathbb{N}$). Note that this relaxation of requirements does not affect the operation of the algorithm and its fast convergence speed, as it will be shown later.

Numerical Evaluation over a Small Network: We now present simulation results to illustrate the behavior of our proposed distributed algorithms. To foster reproducibility, the code, datasets, and experiments are made publicly available.³ We show the evolution of the nodes' states against the number of iterations during the operation of Algorithms 1 and 2. The network in this example comprises 20 nodes and was randomly generated (an edge between a pair of nodes exists with probability 0.5). This process resulted in a digraph that had a diameter equal to 2. Small digraph diameters are indicative of data-center topologies and are normally preferred due to their locality and the benefit of having few hops between each node [33]. The task workload l_j of each node v_j was generated randomly using an integer random distribution uniform in the range $[1, 100]$. The node capacities π_j^{max} in this experiment were set to either 100 or 300 for even and odd node numbers, respectively. Our simulation results are shown in Fig. 1, which depicts the load per node according to its processing capacity during the operation of Algorithms 1 and 2. Specifically, we show $q_j^s[k] = y_j[0]/q_j^s[k]$ for every node. We can see that Algorithm 1 converges monotonically within a few iterations (i.e., after only eight iterations) without being affected by value oscillations or ambiguities. Furthermore, we can see that Algorithm 2 requires more iterations due to the number

³[Online]. Available: <https://github.com/andylamp/federated-quantized-ratio-consensus>

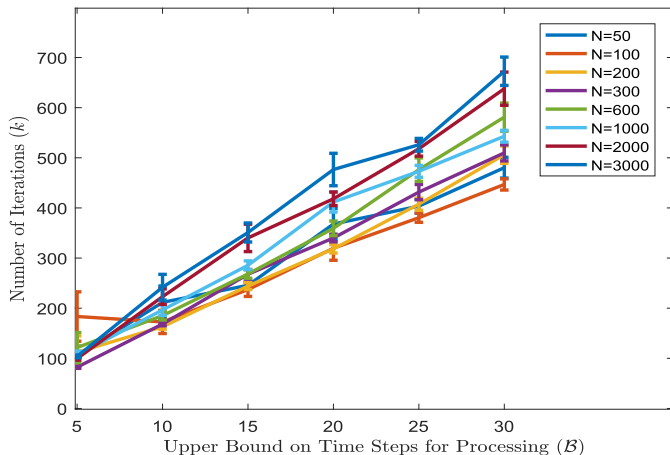


Fig. 2. Required iterations for convergence of Algorithm 2 against upper bound on required time steps for processing B over different network sizes along with their error bars, where each network size is evaluated across 3000 trials and the aggregated values were averaged out before plotting.

of time steps that each node requires to process information (which has an upper bound equal to 5 i.e., $B = 5$). We can see that Algorithm 2 requires more time steps to converge compared with Algorithm 1; this is mainly due to processing delays leading to delayed execution of the algorithm's iteration steps (instead of instant execution for the case of no processing delays) [34]. However, note here that in most cases, asynchronous algorithms are able to deal with processing delays more efficiently than synchronous ones, which perform poorly in heterogeneous environments [35], [36], [37]. This means that for the case where every node suffers from processing delays, synchronous algorithms suffer from delays while waiting for the slow processing nodes. In our case, this means that every node should wait for five time steps before executing a synchronized iteration, and Algorithm 1 would require 45 time steps for convergence. On the other hand, Algorithm 2 is able to achieve faster convergence (since it is not sensitive to issues, such as slow computing nodes) and uses computational resources more efficiently than synchronous approaches [35].

Numerical Evaluation. We now present a more quantitative analysis over a larger set of network sizes, which would be more applicable to practical deployments, such as in modern data-centers. In Fig. 2, we evaluate Algorithm 2 on networks sized from 50 nodes up to 3000 nodes. We show the number of required iterations for convergence for different network sizes and different values of the upper bound B on the number of time steps required for a node to process information from 5 to 30. The topologies are randomly generated and result in digraphs that have a diameter from 2 to 10. We evaluated each network size across 3000 trials, and the aggregated values were averaged out before plotting. It is interesting to see that for $B = 5$, Algorithm 2 required less than 250 iterations to converge for every network size. Also, for $B = 5$, we can see that, as the network size increases, the required iterations for convergence decrease. Another interesting observation is that as the value of the upper bound B on the number of time steps required for a

node to process information increases, the number of required time steps for convergence increases linearly. This means, for example, that 1) for $B = 10$, Algorithm 2 required less than 280 iterations to converge for every network size, and 2) for $B = 15$, Algorithm 2 required less than 350 iterations to converge for every network size.

B. Quantized Global Model Aggregation

In federated learning, we aim to learn the parameters of a specific statistical model from data stored on thousands (or millions) of remote devices. Most current approaches consider the existence of a central server (hosted in the cloud), which collects and aggregates the computed local models from every node in the network [38]. However, communication overhead during each iteration becomes a major bottleneck as the model size gets large and the computed models increase in dimension. Therefore, aggregating the local models in a centralized manner is not an ideal approach due to inefficient operation and practical limitations. In our setting, we consider a set of remote devices (i.e., processing units or nodes) over a network. Each node has a stored local dataset, and uses it to calculate the local parameters of the statistical model. Global model aggregation is the procedure of aggregating the set of local parameters of every node in the network, in a distributed fashion. This procedure aims to calculate the global parameters of the statistical model. Furthermore, global model aggregation needs to be performed in a communication efficient manner, since communication efficiency is a critical bottleneck in federated learning systems [24]. Thus, in our case, nodes need to transmit quantized values in order to achieve more efficient usage of communication resources.

Global and Local Models: For each node v_j , its stored dataset is denoted as r_j and its size as $|\mathcal{R}_j|$. Each node trains a local model with its own dataset, and then transmits the local model parameters (e.g., gradients) in the network for aggregation. The local model parameters of each node v_j at aggregation step m are denoted as $\mathcal{W}_j[m]$. Furthermore, the global model parameters at aggregation step m are denoted as $\mathcal{W}[m]$ and are calculated after aggregating the local model parameters $\mathcal{W}_j[m]$ of every node v_j . Note that in our case, the communication channels are bandwidth-limited. As a result, the parameters of the local model $\mathcal{W}_j[m]$ for every node v_j , and the global model $\mathcal{W}[m]$ are quantized values. In this scenario, we represent them as integer values in order to present an illustrative application of our proposed algorithms.

Optimization Problem: In a federated learning system, computing nodes require to calculate the parameters of the global model by aggregating the parameters of their local model. This is done at aggregation step m via a distributed coordination algorithm, which relies on the exchange of quantized values and converges after a finite number of time steps. Specifically, each node v_j aims to calculate the global model parameters $\mathcal{W}[m]$ at aggregation step m defined as [25], [26]

$$\mathcal{W}[m] = \frac{\sum_{j=1}^n |\mathcal{R}_j| \mathcal{W}_j[m]}{\sum_{j=1}^n |\mathcal{R}_j|}. \quad (11)$$

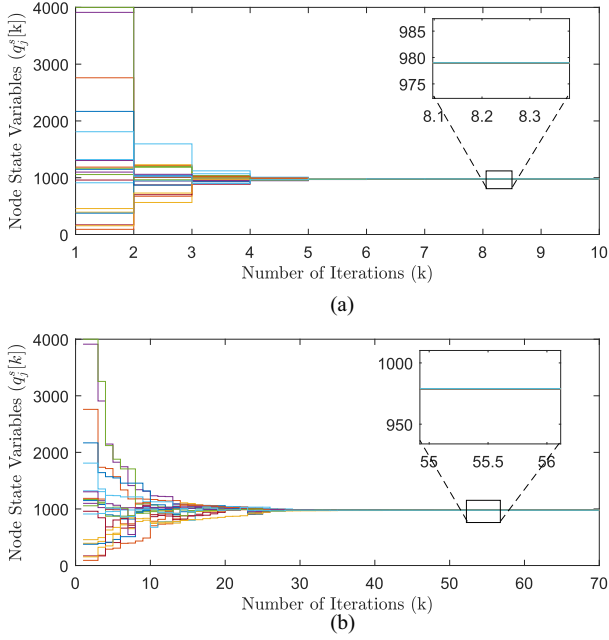


Fig. 3. Execution of (a) Algorithm 1, and (b) Algorithm 2 with $\mathcal{B} = 5$, over a random network comprised of 20 nodes having a diameter equal to 3.

For simplicity of exposition, and since we consider a single aggregation step, we drop index m . To achieve the requirement in (11), we need to modify (2) to be

$$f_j(z) = \frac{1}{2} |\mathcal{R}_j| (z - \mathcal{W}_j)^2. \quad (12)$$

This means that the closed-form solution of (3) becomes

$$x^* = \frac{\sum_{j=1}^n |\mathcal{R}_j| \mathcal{W}_j}{\sum_{j=1}^n |\mathcal{R}_j|}. \quad (13)$$

In other words, each node computes the parameters of the global model x^* , as shown in (13).

Application of Algorithms 1 and 2: During the global model aggregation problem, each node v_j aims to 1) calculate the global model parameters x^* [shown in (13)] after a finite number of time steps, and 2) terminate its operation after calculating x^* . In order to solve the global model aggregation problem, Algorithms 1 and 2 need to be modified as follows: each node v_j needs to 1) have knowledge of D , \mathcal{B} , $|\mathcal{R}_j|$, and \mathcal{W}_j (\mathcal{B} is required only for Algorithm 2), and 2) initialize $z_j[0] := |\mathcal{R}_j|$ and $y_j[0] := \mathcal{W}_j$.

Numerical Evaluation: We illustrate via simulations the behavior of our proposed distributed algorithms over a network represented by a randomly generated graph of 20 nodes (an edge between a pair of nodes was created with probability 0.5). This resulted in a digraph with a diameter equal to 3. The size $|\mathcal{R}_j|$ of the stored dataset r_j for each node was generated randomly using an integer random distribution, uniform in the range [10, 100]. The parameters of the local models \mathcal{W}_j for each node v_j were generated randomly using an integer random distribution uniform in the range [1000, 100 000]. Our simulation results are presented in Fig. 3, which depicts the local model parameters (initial state) of each node and the calculation of the global model parameters (final state of each node) in finite time. During

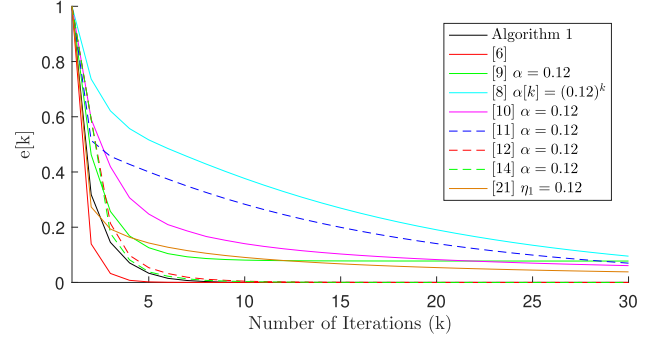


Fig. 4. Normalized error $e[k]$ [defined in (14)] for Algorithm 1, and the algorithms in [6], [8], [9], [10], [11], [12], [14], and [21], averaged over 20 randomly generated strongly connected digraphs of 20 nodes each.

Algorithm 1, we can see that each node converges monotonically after nine iterations and then terminates its operation. During Algorithm 2, we can see that convergence keeps its monotonic nature, but each node converges after 69 iterations due to the required time for information processing (which is equal to 5).

C. Comparison With Current Literature

We now compare the performance of Algorithm 1 against existing algorithms over static strongly connected directed networks of 20 nodes. Specifically, we show the normalized error $e[k]$ defined as

$$e[k] = \sqrt{\frac{\sum_{j=1}^n ((q_j[k])^{-1} - x^*)^2}{\sum_{j=1}^n ((q_j[0])^{-1} - x^*)^2}} \quad (14)$$

where x^* is defined in (3). The error $e[k]$ was evaluated and averaged across 20 trials. In Fig. 4, we can see that Algorithm 1 is among the fastest algorithms in the literature outperformed only by Grammenos et al. [6]. Our algorithm operates with quantized values, which influence the convergence rate (thus outperformed by Grammenos et al. [6]), but admits finite time convergence to the *proximity of the* optimal solution, with the distance from the exact solution depending on the quantization level. Most algorithms in the literature assume that the messages exchanged among nodes in the network are real numbers and admit asymptotic convergence within some error [6], [8], [9], [10], [11], [12], [14]. In [21], the exchanged messages are quantized but the approach still exhibits asymptotic convergence. Furthermore, an additional advantage of our algorithm is that its operation does not rely on a set of weights on the digraph links⁴ that form a double stochastic matrix, unlike the works in [8] and [10].

VI. CONCLUSION

In this article, we considered the problem of distributed optimization in large-scale networks with quadratic local cost functions. The authors introduced a fast distributed algorithm that converges in a finite number of time steps, reaching the

⁴The algorithms in [8], [10], and [21] require the underlying graph to be undirected. For this reason, in Fig. 4, for [8], [10], and [21], we make the randomly generated underlying digraphs undirected (by enforcing that if $(v_j, v_i) \in \mathcal{E}$ then also $(v_i, v_j) \in \mathcal{E}$). For the algorithms in [6], [9], [11], [12], and [14], the randomly generated underlying graph is generally directed.

proximity of the optimal solution (that depends on the quantization level). The algorithm is capable of distributed stopping and operates exclusively with quantized values through event-triggered updates. The authors also present a fully asynchronous algorithm to handle diverse processing times by the nodes. Applications include task scheduling in data centers and global model aggregation for federated learning, demonstrating fast convergence through extensive empirical evaluations. Finally, our algorithms compare favorably with existing literature.

In the future, we plan to extend the functionality of our proposed algorithms to encompass convex and nonconvex optimization problems. Furthermore, we plan to explore how our algorithms can be integrated within the context of federated learning scenarios involving local SGD or batch SGD.

ACKNOWLEDGMENT

This article includes 1) an improved version of the algorithm presented in [1], which imposes less operational requirements, 2) full proofs regarding the completion of the proposed algorithm (not provided in [1]), 3) a fully asynchronous algorithm that operates by performing max-consensus in an asynchronous fashion, 4) an application over federated learning systems, and (5) an extended analysis regarding the operation of our algorithms in large-scale networks, such as data centers.

APPENDIX A PROOF OF THEOREM 1

The authors first consider Lemma 1, *mutatis mutandis*, which is necessary for our subsequent development. Then, we consider Theorem 3 (due to space considerations, we provide a sketch of the proof, which is an adaptation of the proof of Theorem 1 in [29]). Then, we present the proof of Theorem 1.

Lemma 1 (see [39]): Consider a strongly connected digraph $\mathcal{G}_d = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. Suppose that each node v_j assigns a nonzero probability b_{lj} to each of its outgoing edges m_{lj} , where $v_l \in \mathcal{N}_j^+ \cup \{v_j\}$, as follows:

$$b_{lj} = \begin{cases} \frac{1}{1 + \mathcal{D}_j^+}, & \text{if } l = j \text{ or } v_l \in \mathcal{N}_j^+ \\ 0, & \text{if } l \neq j \text{ and } v_l \notin \mathcal{N}_j^+. \end{cases}$$

At time step $k = 0$, node v_j holds a “token” while the other nodes $v_l \in \mathcal{V} - \{v_j\}$ do not. Each node v_j transmits the “token” (if it has it, otherwise it performs no transmission) according to the nonzero probability b_{lj} it assigned to its outgoing edges m_{lj} . The probability $P_{T_i}^D$ that the token is at node v_i after D time steps (where D is the diameter of the digraph \mathcal{G}_d , for which it holds that $D \leq n - 1$) satisfies $P_{T_i}^D \geq (1 + \mathcal{D}_{\max}^+)^{-D} > 0$, where $\mathcal{D}_{\max}^+ = \max_{v_j \in \mathcal{V}} \mathcal{D}_j^+$.

Theorem 3: Consider a strongly connected digraph $\mathcal{G}_d = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. At time step $k = 0$, each node v_j knows $z_j[0], y_j[0]$. Suppose that each node $v_j \in \mathcal{V}$ follows the Initialization and Iteration steps as described in Algorithm 1. For any ε , where $0 < \varepsilon < 1$, there exists $k_0 \in \mathbb{N}$, so that with probability $(1 - \varepsilon)^{(y^{\text{init}} + n)}$, we have $(q_j^s[k] = \lfloor q^{\text{tasks}} \rfloor, k \geq k_0)$ or $(q_j^s[k] = \lceil q^{\text{tasks}} \rceil, k \geq k_0)$ for every $v_j \in \mathcal{V}$,

where

$$q^{\text{tasks}} = \frac{\sum_{j=1}^n y_j[0]}{\sum_{j=1}^n z_j[0]} \quad (15)$$

and

$$y^{\text{init}} = \sum_{\{v_j \in \mathcal{V}: y_j[0] > \lfloor q^{\text{tasks}} \rfloor\}} (y_j[0] - \lfloor q^{\text{tasks}} \rfloor) + \sum_{\{v_j \in \mathcal{V}: y_j[0] < \lfloor q^{\text{tasks}} \rfloor\}} (\lfloor q^{\text{tasks}} \rfloor - y_j[0]) \quad (16)$$

is the total initial state error.

Proof: The operation of Algorithm 1 can be interpreted as the “random walk” of $\sum_{j=1}^n z_j[0] - n$ “tokens” in a Markov chain. Specifically, at time step $k = 0$, node v_j holds $z_j[0]$ “tokens.” One token is T_j^{ins} and is stationary, whereas the other $z_j[0] - 1$ tokens are $T_j^{\text{out}, \vartheta}$, where $\vartheta = 1, 2, \dots, z_j[0] - 1$, and perform independent random walks. Each token T_j^{ins} and $T_j^{\text{out}, \vartheta}$ contains a pair of values $y_j^{\text{ins}}[k], z_j^{\text{ins}}[k], y_j^{\text{out}, \vartheta}[k]$, and $z_j^{\text{out}, \vartheta}[k]$, where $\vartheta = 1, 2, \dots, z_j[0] - 1$, respectively. Initially, we have 1) $y_j^{\text{ins}}[0] = \lceil y_j[0]/z_j[0] \rceil$, 2) $y_j^{\text{out}, \vartheta}[0] = \lceil y_j[0]/z_j[0] \rceil$ or $y_j^{\text{out}, \vartheta}[0] = \lfloor y_j[0]/z_j[0] \rfloor$, and 3) $z_j^{\text{ins}}[0] = z_j^{\text{out}, \vartheta}[0] = 1$ for $\vartheta = 1, 2, \dots, z_j[0] - 1$, such that $y_j^{\text{ins}}[0] + \sum_{\vartheta=1}^{z_j[0]-1} y_j^{\text{out}, \vartheta}[0] = y_j[0]$ and $z_j^{\text{ins}}[0] + \sum_{\vartheta=1}^{z_j[0]-1} z_j^{\text{out}, \vartheta}[0] = z_j[0]$. At each time step k , each node v_j keeps the token T_j^{ins} (i.e., it never transmits it) while it transmits the tokens $T_j^{\text{out}, \vartheta}$, where $\vartheta = 1, 2, \dots, z_j[0] - 1$, independently to out-neighbors according to the nonzero probability b_{lj} it assigned to its outgoing edges m_{lj} during the Initialization steps. If v_j receives one or more tokens $T_i^{\text{out}, \vartheta}$ from its in-neighbors v_i , the values $y_i^{\text{out}, \vartheta}[k]$ and $y_j^{\text{ins}}[k]$ become equal (or with maximum difference equal to 1); then, v_j transmits each received token $T_i^{\text{out}, \vartheta}$ to a randomly selected out-neighbor according to the nonzero probability b_{lj} it assigned to its outgoing edges m_{lj} . Note here that during the operation of Algorithm 1, we have

$$\sum_{j=1}^n \sum_{\vartheta=1}^{z_j[0]-1} y_j^{\text{out}, \vartheta}[k] + \sum_{j=1}^n y_j^{\text{ins}}[k] = \sum_{j=1}^n y_j[0] \quad \forall k \in \mathbb{Z}_+. \quad (17)$$

The main idea of this proof is that one token $T_\lambda^{\text{out}, \vartheta}$ visits a specific node v_i (for which it holds $|y_\lambda^{\text{out}, \vartheta} - y_i^{\text{ins}}| > 1$) and obtains equal values y (or with maximum difference between them equal to 1) with the token T_i^{ins} , which is kept at node v_i . Thus, we analyze the required time steps for the specific token $T_\lambda^{\text{out}, \vartheta}$ (which performs a random walk) to visit node v_i according to a probability. Note here that if each token $T_\lambda^{\text{out}, \vartheta}$ visits y^{init} times each node v_i , then every token in the network (including both the tokens performing random walk and the stationary tokens) obtains y value equal to $\lfloor q^{\text{tasks}} \rfloor$ or $\lceil q^{\text{tasks}} \rceil$.

From Lemma 1, we have that the probability $P_{T_{\text{out}}}^D$ that “the specific token $T_\lambda^{\text{out}, \vartheta}$ is at node v_i after D time steps” is

$$P_{T_{\text{out}}}^D \geq (1 + \mathcal{D}_{\max}^+)^{-D}. \quad (18)$$

This means that the probability $P_{N_{-T^{\text{out}}}}^D$ that “the specific token $T_\lambda^{\text{out},\vartheta}$ has not visited node v_i after D time steps” is

$$P_{N_{-T^{\text{out}}}}^D \leq 1 - (1 + \mathcal{D}_{\max}^+)^{-D}. \quad (19)$$

By extending this analysis, we can state that for any ε , where $0 < \varepsilon < 1$, and after τD time steps, where

$$\tau \geq \left\lceil \frac{\log \varepsilon}{\log (1 - (1 + \mathcal{D}_{\max}^+)^{-D})} \right\rceil \quad (20)$$

the probability $P_{N_{-T^{\text{out}}}}^\tau$ that “the specific token $T_\lambda^{\text{out},\vartheta}$ has not visited node v_i after τD time steps” is

$$P_{N_{-T^{\text{out}}}}^\tau \leq [P_{N_{-T^{\text{out}}}}^D]^\tau \leq \varepsilon. \quad (21)$$

This means that after τD time steps, where τ fulfills (20), the probability that “the specific token $T_\lambda^{\text{out},\vartheta}$ has visited node v_i after τD time steps” is equal to $1 - \varepsilon$.

Thus, by extending this analysis, for $k \geq (y^{\text{init}} + n)\tau D$, where y^{init} fulfills (16) and τ fulfills (20), we have $q_j^s[k] = \lfloor q^{\text{tasks}} \rfloor$ or $q_j^s[k] = \lceil q^{\text{tasks}} \rceil$ with probability $(1 - \varepsilon)^{(y^{\text{init}} + n)}$, for every $v_j \in \mathcal{V}$. ■

Proof of Theorem 1: From Theorem 3, we have that the operation of Algorithm 1 can be interpreted as the “random walk” of $\sum_{j=1}^n z_j[0] - n$ “tokens” in a Markov chain. Furthermore, we also have that n “tokens” remain stationary, one token at each node. Each of these $\sum_{j=1}^n z_j[0] - n$ tokens contains a pair of values $y^{\text{out},\vartheta}[k], z^{\text{out},\vartheta}[k]$, where $\vartheta = 1, 2, \dots, \sum_{j=1}^n z_j[0] - n$, and each of the n stationary tokens contains a pair of values $y^{\text{ins}}[k], z^{\text{ins}}[k]$. From Theorem 3, we have that after $(y^{\text{init}} + n)\tau D$ time steps, where y^{init} fulfills (16) and τ fulfills (20), the state $q_j^s[k]$ of each node v_j becomes $q_j^s[k] = \lfloor q^{\text{tasks}} \rfloor$ or $q_j^s[k] = \lceil q^{\text{tasks}} \rceil$ with probability $(1 - \varepsilon)^{(y^{\text{init}} + n)}$, where $0 < \varepsilon < 1$ and q^{tasks} fulfills (15). This means that after $(y^{\text{init}} + n)\tau D$ time steps, where y^{init} fulfills (16) and τ fulfills (20), for each of the $\sum_{j=1}^n z_j[0] - n$ tokens in the network, it holds that $y^{\text{out},\vartheta}[k] = \lfloor q^{\text{tasks}} \rfloor$ or $y^{\text{out},\vartheta}[k] = \lceil q^{\text{tasks}} \rceil$, $\vartheta = 1, 2, \dots, \sum_{j=1}^n z_j[0] - n$, while for each of the n stationary tokens in the network, it also holds that $y^{\text{ins}}[k] = \lfloor q^{\text{tasks}} \rfloor$ or $y^{\text{ins}}[k] = \lceil q^{\text{tasks}} \rceil$, with probability $(1 - \varepsilon)^{(y^{\text{init}} + n)}$, where $0 < \varepsilon < 1$. Specifically, the y value of every token in the network is equal either to $\lfloor q^{\text{tasks}} \rfloor$ or $\lceil q^{\text{tasks}} \rceil$ after $(y^{\text{init}} + n)\tau D$ time steps with probability $(1 - \varepsilon)^{(y^{\text{init}} + n)}$, where $0 < \varepsilon < 1$.

During the operation of Algorithm 1, every D timesteps, each node v_j reinitializes its voting variables M_j and m_j to be $M_j = \lfloor y_j[k]/z_j[k] \rfloor$ and $m_j = \lfloor y_j[k]/z_j[k] \rfloor$. Note here that the max-consensus algorithm (or the min-consensus algorithm) converges to the maximum value among all nodes in a finite number of steps s , where $s \leq D$ (see, e.g., [30, Th. 5.4]). Thus, after $(y^{\text{init}} + n)\tau D$ time steps, the value $q_j^s[k]$ of each node v_j is equal to $\lfloor q^{\text{tasks}} \rfloor$ or $\lceil q^{\text{tasks}} \rceil$, with probability $(1 - \varepsilon)^{(y^{\text{init}} + n)}$, where $0 < \varepsilon < 1$. This means that M_j and m_j are reinitialized to be equal to $M_j = \lfloor q^{\text{tasks}} \rfloor$ or $M_j = \lceil q^{\text{tasks}} \rceil$ and $m_j = \lfloor q^{\text{tasks}} \rfloor$ or $m_j = \lceil q^{\text{tasks}} \rceil$ after $\lceil ((y^{\text{init}} + n)\tau D/D) \rceil D$ time steps with probability $(1 - \varepsilon)^{(y^{\text{init}} + n)}$, where $0 < \varepsilon < 1$. After an additional number of D time steps, the variables M_j and m_j of each node are updated to $M_j = \lfloor q^{\text{tasks}} \rfloor$ and $m_j = \lfloor q^{\text{tasks}} \rfloor$

(since the max-consensus algorithm [32] converges after D time steps). Thus, $M_j - m_j \leq 1$ holds for every node v_j . This means that every node v_j calculates the optimal x^* [shown in (3)] and terminates its operation. As a result, we have that after $\lceil ((y^{\text{init}} + n)\tau D/D) \rceil D + D$ time steps, each node v_j calculates the optimal $x_j^* = x^* = \lfloor y_j[0]/q^{\text{tasks}} \rfloor$ with probability $(1 - \varepsilon)^{(y^{\text{init}} + n)}$, where $0 < \varepsilon < 1$. □

APPENDIX B PROOF OF THEOREM 2

We first consider Lemma 2 and Theorem 4, which are necessary for our subsequent development. Then, we present the proof of Theorem 2.

Lemma 2: Consider a strongly connected digraph $\mathcal{G}_d = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. Suppose that each node v_j assigns a nonzero probability b_{lj} to each of its outgoing edges m_{lj} , where $v_l \in \mathcal{N}_j^+ \cup \{v_j\}$, as follows:

$$b_{lj} = \begin{cases} \frac{1}{1 + \mathcal{D}_j^+}, & \text{if } l = j \text{ or } v_l \in \mathcal{N}_j^+ \\ 0, & \text{if } l \neq j \text{ and } v_l \notin \mathcal{N}_j^+. \end{cases}$$

At time step $k = 0$, node v_j holds a “token” while the other nodes $v_l \in \mathcal{V} - \{v_j\}$ do not. Each node v_j transmits the “token” (if it has it, otherwise it performs no transmission) according to the nonzero probability b_{lj} it assigned to its outgoing edges m_{lj} . Furthermore, each node v_j requires at most \mathcal{B} time steps to process the information in the received token from its in-neighbors. The integer number of time steps that each node v_j requires to process the information is a bounded discrete random variable with some distribution. Specifically, node v_j requires λ time steps, where $\lambda \in \{1, 2, \dots, \mathcal{B}\}$, for processing information with probability $\mathcal{B}_j^{(\lambda)}$, where $\sum_{\lambda=1}^{\mathcal{B}} \mathcal{B}_j^{(\lambda)} = 1$, for every v_j . The probability $P_{T_i}^{\mathcal{B}D}$ that the token is at node v_i after $\mathcal{B}D$ time steps (note that D is the diameter of the digraph \mathcal{G}_d and it holds that $D \leq n - 1$) satisfies $P_{T_i}^{\mathcal{B}D} \geq (1 + \mathcal{D}_{\max}^+)^{-D} (\mathcal{B}_{\min}^{(\mathcal{B})})^D > 0$, where $\mathcal{D}_{\max}^+ = \max_{v_j \in \mathcal{V}} \mathcal{D}_j^+$ and $\mathcal{B}_{\min}^{(\mathcal{B})} = \min_{v_j \in \mathcal{V}} \mathcal{B}_j^{(\mathcal{B})}$.

Proof: We have that the diameter D of every strongly connected digraph \mathcal{G}_d is upper bounded by $n - 1$, where $n = |\mathcal{V}|$. Specifically, from node v_j to node v_i , there exists a sequence of nodes $v_j \equiv v_{l_0}, v_{l_1}, \dots, v_{l_t} \equiv v_i$, such that $(v_{l_{\tau+1}}, v_{l_\tau}) \in \mathcal{E}$ for $\tau = 0, 1, \dots, t - 1$, where $t \leq D$. This means that the shortest path from node v_j to node v_i ($v_j \neq v_i$) has length at most D . Let us assume that the token at time step $k = 0$ is at node v_j . In one scenario, node v_j processes the information of the token for \mathcal{B} time steps with probability $(\mathcal{B}_{\min}^{(\mathcal{B})})$. Then, at time step \mathcal{B} , node v_j selects node v_{l_1} , with probability at least $(1 + \mathcal{D}_{\max}^+)^{-1}$. This means that the token will be at node v_{l_1} after \mathcal{B} time steps with probability at least $(1 + \mathcal{D}_{\max}^+)^{-1} (\mathcal{B}_{\min}^{(\mathcal{B})})$. Again, in the worst-case scenario, node v_{l_1} processes the information of the received token for \mathcal{B} time steps with probability $(\mathcal{B}_{\min}^{(\mathcal{B})})$. Then, it transmits the token to node v_{l_2} with probability at least $(1 + \mathcal{D}_{\max}^+)^{-1}$. This means that the token will be at node v_{l_2} after $2\mathcal{B}$ time steps with probability at least $(1 + \mathcal{D}_{\max}^+)^{-2} (\mathcal{B}_{\min}^{(\mathcal{B})})^2$. By repeating this analysis, we have that after $\mathcal{B}(t - 1)$ time steps, the token will be at node v_i with probability at least $P_{T_i}^{\mathcal{B}(t-1)} \geq$

$(1 + \mathcal{D}_{\max}^+)^{-(t-1)} (\mathcal{B}_{\min}^{(B)})^{(t-1)} > 0$. For the remaining $\mathcal{B}(D - t)$ time steps, we have that node v_i processes the information for \mathcal{B} time steps and then transmits it to itself. So, as a result, we have that after $\mathcal{B}D$ time steps, the token will be at node v_i with probability at least $P_{T_i}^{\mathcal{B}D} \geq (1 + \mathcal{D}_{\max}^+)^{-D} (\mathcal{B}_{\min}^{(B)})^D > 0$. ■

Theorem 4: Consider a strongly connected digraph $\mathcal{G}_d = (\mathcal{V}, \mathcal{E})$ with $n = |\mathcal{V}|$ nodes and $m = |\mathcal{E}|$ edges. At time step $k = 0$, each node v_j knows $z_j[0], y_j[0]$. Furthermore, each node v_j requires at most \mathcal{B} time steps to process the information in the received token from its in-neighbors. The number of time steps that each node v_j requires to process the information follows a random probability distribution. Specifically, node v_j requires λ time steps, where $\lambda \in \{1, 2, \dots, \mathcal{B}\}$ for processing with probability $\mathcal{B}_j^{(\lambda)}$, where $\sum_{\lambda=1}^{\mathcal{B}} \mathcal{B}_j^{(\lambda)} = 1$, for every v_j . Suppose that each node $v_j \in \mathcal{V}$ follows the Initialization and Iteration steps, as described in Algorithm 2. For any ε , where $0 < \varepsilon < 1$, there exists $k_0 \in \mathbb{N}$, so that with probability $(1 - \varepsilon)^{(y^{\text{init}} + n)}$ we have $(q_j^s[k] = \lfloor q^{\text{tasks}} \rfloor, k \geq k_0)$ or $(q_j^s[k] = \lceil q^{\text{tasks}} \rceil, k \geq k_0)$, for every $v_j \in \mathcal{V}$, where q^{tasks} and y^{init} fulfill (15) and (16).

Proof: The proof is similar to the proof of Theorem 3. For this reason, we only mention the differences in comparison to the proof of Theorem 3.

From Lemma 2, we have that the probability $P_{T_{\text{out}}}^{\mathcal{B}D}$ that “the specific token $T_{\lambda}^{\text{out}, \vartheta}$ is at node v_i after $\mathcal{B}D$ time steps” is $P_{T_{\text{out}}}^{\mathcal{B}D} \geq (1 + \mathcal{D}_{\max}^+)^{-D} (\mathcal{B}_{\min}^{(B)})^D$, where $\mathcal{D}_{\max}^+ = \max_{v_j \in \mathcal{V}} \mathcal{D}_j^+$ and $\mathcal{B}_{\min}^{(B)} = \min_{v_j \in \mathcal{V}} \mathcal{B}_j^{(B)}$. This means that the probability $P_{N_{T_{\text{out}}}}^{\mathcal{B}D}$ that “the specific token $T_{\lambda}^{\text{out}, \vartheta}$ has not visited node v_i after $\mathcal{B}D$ time steps” is $P_{N_{T_{\text{out}}}}^{\mathcal{B}D} \leq 1 - (1 + \mathcal{D}_{\max}^+)^{-D} (\mathcal{B}_{\min}^{(B)})^D$. By extending this analysis, we can state that for any $\varepsilon, 0 < \varepsilon < 1$, and after $\tau(\mathcal{B}D)$ time steps, where

$$\tau \geq \left\lceil \frac{\log \varepsilon}{\log (1 - (1 + \mathcal{D}_{\max}^+)^{-D} (\mathcal{B}_{\min}^{(B)})^D)} \right\rceil \quad (22)$$

the probability $P_{N_{T_{\text{out}}}}^{\tau}$ that “the specific token $T_{\lambda}^{\text{out}, \vartheta}$ has not visited node v_i after $\tau(\mathcal{B}D)$ time steps” is $P_{N_{T_{\text{out}}}}^{\tau} \leq [P_{N_{T_{\text{out}}}}^{\mathcal{B}D}]^{\tau} \leq \varepsilon$. This means that after $\tau(\mathcal{B}D)$ time steps, where τ fulfills (22), the probability that “the specific token $T_{\lambda}^{\text{out}, \vartheta}$ has visited node v_i after $\tau(\mathcal{B}D)$ time steps” is $1 - \varepsilon$.

Thus, by extending this analysis, for $k \geq (y^{\text{init}} + n)\tau(\mathcal{B}D)$, where y^{init} fulfills (16) and τ fulfills (22), we have $q_j^s[k] = \lfloor q^{\text{tasks}} \rfloor$ or $q_j^s[k] = \lceil q^{\text{tasks}} \rceil$ with probability $(1 - \varepsilon)^{(y^{\text{init}} + n)}$, for every $v_j \in \mathcal{V}$. ■

Proof of Theorem 2: The proof is similar to the proof of Theorem 1. For this reason, we only mention the differences in comparison to the proof of Theorem 1.

During the operation of Algorithm 2, every DB time steps, each node v_j reinitializes its voting variables M_j and m_j to be $M_j = \lfloor y_j[k]/z_j[k] \rfloor$ and $m_j = \lfloor y_j[k]/z_j[k] \rfloor$. After $(y^{\text{init}} + n)\tau(\mathcal{B}D)$ time steps, where y^{init} fulfills (16) and τ fulfills (22), the value $q_j^s[k]$ of each node v_j is equal to $\lfloor q^{\text{tasks}} \rfloor$ or $\lceil q^{\text{tasks}} \rceil$, with probability $(1 - \varepsilon)^{(y^{\text{init}} + n)}$, where $0 < \varepsilon < 1$. This

means that M_j and m_j are reinitialized to be equal to $M_j = \lfloor q^{\text{tasks}} \rfloor$ or $M_j = \lceil q^{\text{tasks}} \rceil$ and $m_j = \lfloor q^{\text{tasks}} \rfloor$ or $m_j = \lceil q^{\text{tasks}} \rceil$ after $\lceil ((y^{\text{init}} + n)\tau(\mathcal{B}D)/DB) \rceil DB$ time steps with probability $(1 - \varepsilon)^{(y^{\text{init}} + n)}$, where $0 < \varepsilon < 1$. After an additional number of DB time steps, the variables M_j and m_j of each node are updated to $M_j = \lfloor q^{\text{tasks}} \rfloor$ and $m_j = \lfloor q^{\text{tasks}} \rfloor$ (since the asynchronous max-consensus algorithm converges after $\mathcal{B}D$ time steps). Thus, the condition $M_j - m_j \leq 1$ holds for every node v_j . This means that every node v_j calculates the optimal x^* [shown in (3)] and terminates its operation. As a result, we have that after $\lceil ((y^{\text{init}} + n)\tau(\mathcal{B}D)/DB) \rceil DB + DB$ time steps, each node v_j calculates the optimal required task workload $x_j^* = x^* = \lfloor y_j[0]/q^{\text{tasks}} \rfloor$ with probability $(1 - \varepsilon)^{(y^{\text{init}} + n)}$, where $0 < \varepsilon < 1$. □

REFERENCES

- [1] A. I. Rikos, A. Grammenos, E. Kalyvianaki, C. N. Hadjicostis, T. Charalambous, and K. H. Johansson, “Optimal CPU scheduling in data centers via a finite-time distributed quantized coordination mechanism,” in *Proc. IEEE Conf. Decis. Control*, 2021, pp. 6276–6281.
- [2] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, “Learning scheduling algorithms for data processing clusters,” in *Proc. ACM Special Int. Group Data Commun.*, in ser. SIGCOMM, 2019, pp. 270–288.
- [3] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, “Quincy: Fair scheduling for distributed computing clusters,” in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Princ.*, New York, NY, USA, 2009, pp. 261–276.
- [4] A. Tumanov, T. Zhu, J. W. Park, M. A. Kozuch, M. Harchol-Balter, and G. R. Ganger, “TetriSched: Global rescheduling with adaptive plan-ahead in dynamic heterogeneous clusters,” in *Proc. Eleventh Eur. Conf. Comput. Syst.*, 2016, pp. 1–16.
- [5] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, “GRAPHENE: Packing and dependency-aware scheduling for data-parallel clusters,” in *Proc. USENIX Symp. Operating Syst. Des. Implementation*, Savannah, GA, USA, 2016, pp. 81–97.
- [6] A. Grammenos, T. Charalambous, and E. Kalyvianaki, “CPU scheduling in data centers using asynchronous finite-time distributed coordination mechanisms,” *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 4, pp. 1880–1894, Jul./Aug. 2023.
- [7] T. Yang et al., “A survey of distributed optimization,” *Annu. Rev. Control*, vol. 47, pp. 278–305, 2019.
- [8] A. Nedic and A. Ozdaglar, “Distributed subgradient methods for multi-agent optimization,” *IEEE Trans. Autom. Control*, vol. 54, no. 1, pp. 48–61, Jan. 2009.
- [9] K. I. Tsianos, S. Lawlor, and M. G. Rabbat, “Push-sum distributed dual averaging for convex optimization,” in *Proc. 51st IEEE Conf. Decis. Control*, 2012, pp. 5453–5458.
- [10] J. Xu, S. Zhu, Y. C. Soh, and L. Xie, “Convergence of asynchronous distributed gradient methods over stochastic networks,” *IEEE Trans. Autom. Control*, vol. 63, no. 2, pp. 434–448, Feb. 2018.
- [11] C. Xi, R. Xin, and U. A. Khan, “ADD-OPT: Accelerated distributed directed optimization,” *IEEE Trans. Autom. Control*, vol. 63, no. 5, pp. 1329–1339, May 2018.
- [12] R. Xin and U. A. Khan, “A linear algorithm for optimization over directed graphs with geometric convergence,” *IEEE Control Syst. Lett.*, vol. 2, no. 3, pp. 315–320, Jul. 2018.
- [13] A. Lalitha, S. Shekhar, T. Javidi, and F. Koushanfar, “Fully decentralized federated learning,” in *Proc. 3rd Workshop Bayesian Deep, NIPS Workshop*, 2018.
- [14] S. Pu, W. Shi, J. Xu, and A. Nedic, “Push–pull gradient methods for distributed optimization in networks,” *IEEE Trans. Autom. Control*, vol. 66, no. 1, pp. 1–16, Jan. 2021.
- [15] H. Liu and W. Yu, “Discrete-time algorithm for distributed unconstrained optimization problem with finite-time computations,” *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 68, no. 1, pp. 351–355, Jan. 2021.
- [16] W. Jiang and T. Charalambous, “A fast finite-time consensus based gradient method for distributed optimization over digraphs,” in *Proc. IEEE 61st Conf. Decis. Control*, 2022, pp. 6848–6854.

- [17] C.-S. Lee, N. Michelusi, and G. Scutari, "Finite rate quantized distributed optimization with geometric convergence," in *Proc. IEEE 52nd Asilomar Conf. Signals, Syst., Comput.*, 2018, pp. 1876–1880.
- [18] A. Reiszadeh, A. Mokhtari, H. Hassani, and R. Pedarsani, "An exact quantized decentralized gradient descent algorithm," *IEEE Trans. Signal Process.*, vol. 67, no. 19, pp. 4934–4947, Oct. 2019.
- [19] A. Koloskova, S. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *Proc. 36th Int. Conf. Mach. Learn.*, 2019, pp. 3478–3487.
- [20] A. Reiszadeh, A. Mokhtari, H. Hassani, A. Jadbabaie, and R. Pedarsani, "FedPAQ: A communication-efficient federated learning method with periodic averaging and quantization," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2020, pp. 2021–2031.
- [21] M. Doostmohammadian, A. Aghasi, M. Pirani, E. Nekouei, U. A. Khan, and T. Charalambous, "Fast-convergent anytime-feasible dynamics for distributed allocation of resources over switching sparse networks with quantized communication links," in *Proc. Eur. Control Conf.*, 2022, pp. 84–89.
- [22] Y. Xiong, L. Wu, K. You, and L. Xie, "Quantized distributed gradient tracking algorithm with linear convergence in directed networks," *IEEE Trans. Autom. Control*, vol. 68, no. 9, pp. 5638–5645, Sep. 2023.
- [23] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, "Federated learning: Challenges, methods, and future directions," *IEEE Signal Process. Mag.*, vol. 37, no. 3, pp. 50–60, May 2020.
- [24] K. Bonawitz et al., "Towards federated learning at scale: System design," in *Proc. Mach. Learn. Syst.*, A. Talwalkar, V. Smith, and M. Zaharia, Eds., 2019, vol. 1, pp. 374–388.
- [25] W. Y. B. Lim et al., "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surv. Tut.*, vol. 22, no. 3, pp. 2031–2063, Thirdquarter 2020.
- [26] C. Hu, J. Jiang, and Z. Wang, "Decentralized federated learning: A segmented gossip approach," 2019, *arXiv:1908.07782*.
- [27] S. T. Cady, A. D. Domínguez-García, and C. N. Hadjicostis, "Finite-time approximate consensus and its application to distributed frequency regulation in islanded AC microgrids," in *Proc. Hawaii Int. Conf. System Sci.*, 2015, pp. 2664–2670.
- [28] A. D. Domínguez-García and C. N. Hadjicostis, "Coordination and control of distributed energy resources for provision of ancillary services," in *Proc. 1st IEEE Int. Conf. Smart Grid Commun.*, 2010, pp. 537–542.
- [29] A. I. Rikos, C. N. Hadjicostis, and K. H. Johansson, "Non-oscillating quantized average consensus over dynamic directed topologies," *Automatica*, vol. 146, 2022, Art. no. 110621.
- [30] S. Giannini, D. Di Paola, A. Petitti, and A. Rizzo, "On the convergence of the max-consensus protocol with asynchronous updates," in *Proc. IEEE Conf. Decis. Control*, 2013, pp. 2605–2610.
- [31] M. Rabbat and R. Nowak, "Distributed optimization in sensor networks," in *Proc. 3rd Int. Symp. Inf. Process. Sensor Netw.*, 2004, pp. 20–27.
- [32] J. Cortés, "Distributed algorithms for reaching consensus on general functions," *Automatica*, vol. 44, pp. 726–737, Mar. 2008.
- [33] M. Besta and T. Hoefler, "SlimFly: A cost effective low-diameter network topology," in *Proc. IEEE Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2014, pp. 348–359.
- [34] H. Feyzmahdavian, T. Charalambous, and M. Johansson, "Exponential stability of homogeneous positive systems of degree one with time-varying delays," *IEEE Trans. Autom. Control*, vol. 59, no. 6, pp. 1594–1599, Jun. 2014.
- [35] B. M. Assran, A. Aytakin, H. R. Feyzmahdavian, M. Johansson, and M. G. Rabbat, "Advances in asynchronous parallel and distributed optimization," *Proc. IEEE Proc. IRE*, vol. 108, no. 11, pp. 2013–2031, Nov. 2020.
- [36] X. Lian, W. Zhang, C. Zhang, and J. Liu, "Asynchronous decentralized parallel stochastic gradient descent," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 3043–3052.
- [37] 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 *Proc. 21st Int. Conf. Artif. Intell. Statist.*, 2018, pp. 803–812.
- [38] V. Smith, S. Forte, C. Ma, M. Takac, M. I. Jordan, and M. Jaggi, "CoCoA: A general framework for communication efficient distributed optimization," *J. Mach. Learn. Res.*, vol. 18, no. 230, pp. 1–47, 2018.
- [39] A. I. Rikos and C. N. Hadjicostis, "Distributed average consensus under quantized communication via event-triggered mass splitting," in *Proc. 20th IFAC World Congr.*, 2020, pp. 3019–3024.



Apostolos I. Rikos (Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in electrical engineering from the Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus, in 2010, 2012, and 2018, respectively.

In 2018, he joined the KIOS Research and Innovation Center of Excellence, Aglandjia, Cyprus, where he was a Research Lecturer. Since February 2020, he has been a Postdoctoral Researcher with the Division of Decision and Control Systems, KTH Royal Institute of Technology, Stockholm, Sweden.



Andreas Grammenos received the B.Sc. degree from the Technical University of Crete, Kounoupidiana, Greece, in 2015, and the Ph.D. degree from the Department of Computer Science and Technology, University of Cambridge, Cambridge, U.K., in 2021.

He was a Founding Member of COVID-19 sounds, an audio-based study, which builds predictive models for COVID-19 detection using crowdsourced smartphone respiratory recordings. His research interests include the broad field of large-scale data and streaming analytics with a focus on topics around distributed, federated, and decentralized systems.



Evangelia Kalyvianaki received the B.Sc. and M.Sc. degrees in computer science from the Computer Science Department, University of Crete, Rethymno, Greece, in 2000 and 2002, respectively, and the Ph.D. degree in computer science from the Computer Laboratory, Cambridge University, Cambridge, U.K., in 2009.

She was a Lecturer with the Department of Computer Science, City University London, and a Postdoctoral Researcher with the Department of Computing, Imperial College London. She is currently a Senior Lecturer (Assistant/Associate Professor) with the Department of Computer Science and Technology, University of Cambridge, and a Turing Fellow.



Christoforos N. Hadjicostis (Fellow, IEEE) received the B.S. degree in electrical engineering, computer science and engineering, and mathematics and the M.Eng. and Ph.D. degrees in electrical engineering and computer science from the Massachusetts Institute of Technology, Cambridge, MA, USA, in 1993, 1995, and 1999, respectively.

In 1999, he joined the Faculty with the University of Illinois at Urbana-Champaign, where he was an Assistant and then an Associate Professor with the Department of Electrical and Computer Engineering, the Coordinated Science Laboratory, and the Information Trust Institute. Since 2007, he has been with the Department of Electrical and Computer Engineering, University of Cyprus, Nicosia, Cyprus, where he is currently a Professor and an Interim Director with the Deдалus Research Center on Cyber-Physical Systems, Smart Technologies, Robotics and Networks.



Themistoklis Charalambous (Senior Member, IEEE) received the B.A. and M.Eng. degrees in electrical and information sciences from Trinity College, Cambridge University, Cambridge, U.K., in 2005, and the Ph.D. degree from the Control Laboratory of the Engineering Department, Cambridge University, in 2010.

Following his Ph.D., he held postdoctoral positions with Imperial College London, Royal Institute of Technology, and with the Chalmers University of Technology. In January 2017, he got appointed as an Assistant Professor with Aalto University, where, in July 2020, he got promoted to Associate Professor. In September 2021, he joined the Department of Electrical and Computer Engineering, University of Cyprus as a tenure-track Assistant Professor and he remains associated with Aalto University as a Visiting Professor. Since April 2023, he is also a Visiting Professor at the FinEst Centre for Smart Cities.



Karl H. Johansson (Fellow, IEEE) received the M.Sc. degree in electrical engineering and the Ph.D. degree in automatic control from Lund University, Lund, Sweden, in 1992 and 1997, respectively.

He has held visiting positions with UC Berkeley, Caltech, NTU, HKUST Institute of Advanced Studies, and NTNU. He is currently a Professor with the School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Stockholm, Sweden, and the Director of

Digital Futures, Stockholm.

Dr. Johansson was on the IEEE Control Systems Society Board of Governors and the Swedish Scientific Council for Natural Sciences and Engineering Sciences. He was the recipient of several best paper awards and other distinctions from IEEE, IFAC, and ACM, and the Swedish Research Council Distinguished Professor, Wallenberg Scholar with the Knut and Alice Wallenberg Foundation, Future Research Leader Award from the Swedish Foundation for Strategic Research, the triennial IFAC Young Author Prize, and IEEE Control Systems Society Distinguished Lecturer. He is Fellow of the Royal Swedish Academy of Engineering Sciences. He is the President of the European Control Association and Member of the IFAC Council.