Distributed Online Min-Max Load Balancing with Risk-Averse Assistance

Jingrong Wang and Ben Liang
Department of Electrical and Computer Engineering, University of Toronto, Canada
Email: jr.wang@mail.utoronto.ca, liang@ece.utoronto.ca

***Abstract*—Motivated by a wide range of applications from parallel computing to distributed learning, we study distributed online load balancing among multiple workers. We aim to minimize the pointwise maximum over the workers' local cost functions. We propose a novel algorithm termed Distributed Online Load Balancing with rIsk-averse assistancE (DOLBIE), which jointly considers the worker heterogeneity and system dynamics. The workload is distributed to workers in an online manner, where the underloaded workers learn to provide an appropriate amount of assistance to the most overloaded worker for the next online round without making themselves overwhelmed. In DOLBIE, all workers participate in updating the workload simultaneously, and no computationally intensive gradient or projection calculation is required. DOLBIE can be implemented in both the master-worker and fully-distributed architectures. We analyze the worst-case performance of DOLBIE by deriving an upper bound on its dynamic regret. We further demonstrate the application of DOLBIE to online batch-size tuning in distributed machine learning. Our experimental results show that, in comparison with state-of-the-art alternatives, DOLBIE can substantially speed up the training process and reduce the workers' idle time.**

I. Introduction

Load balancing is an important component of parallel computing, especially in heterogeneous systems [1]. It helps improve service quality and maintain system stability. One typical load balancing object is the pointwise maximum over a set of local cost functions, e.g., the makespan, defined as the maximum completion time over all workers. This metric arises in a wide range of applications, e.g., Bulk Synchronous Parallel [2], data partitioning in distributed learning [3], and task offloading in edge computing [4].

Most existing load balancing algorithms are *offline* and require a priori knowledge of the state of the system, e.g., the processing power of the workers and the channel state information [5]. However, in real-world distributed computing applications, it may be impractical to collect such information due to the delayed feedback, while the system may evolve in an unpredictable fashion. Take as an example synchronous distributed training of a machine learning model. The load balancing task is to split the dataset into multiple disjoint partitions in each round of training, and assign them to the workers. The objective is to minimize the total wall-clock training time. We note that the per-round training time is not known a priori since the data samples are randomly drawn and the computation and communication capabilities of the workers may fluctuate over time, making this naturally an *online* load balancing problem.

Most online load balancing algorithms share a common understanding that the more powerful workers that incur lower cost in the previous online rounds should take up more workload so as to reduce the impact of the straggler, defined as the worker who incurs the highest cost. Recent works achieve load balancing by taking the workload inversely proportional to the historical local cost [3], and by shifting a predefined amount of work from the straggler to the most underloaded worker [6]. However, the proportional adjustment in [3] is not robust to non-linear cost functions, and the prescribed fixed increment in [6] overlooks the system heterogeneity. We further note that in these works, the workloads on the non-straggling workers are passively updated without consideration for these workers' own local cost functions, which puts them at risk of becoming worse stragglers.

Instead, a guiding design principle in this work is that the assisting non-straggling workers should stay risk-averse and judiciously share an appropriate amount of work without being overwhelmed themselves. However, there are several obstacles to realizing this vision: 1) The cost function is generally non-convex and not continuously differentiable due to the max operation. 2) Unlike in min-sum optimization [7], [8], the local cost functions are coupled in min-max load balancing optimization, impeding problem decomposition and distributed implementation. 3) There is strong need for online optimization, so as to adapt to the unpredictable fluctuations in the cost functions over time. 4) In large-scale distributed computation, an online optimization algorithm needs to be lightweight and scalable to provide timely decision making.

To address these challenges, we propose a new distributed online optimization algorithm termed Distributed Online Load Balancing with rIsk-averse assistancE (DOLBIE), which dynamically balances the workload among workers over time, even though the local cost functions of the workers can be unpredictable and arbitrarily time varying. Our objective is to minimize the accumulation over time of a global cost function, defined as the pointwise maximum over the local cost functions. DOLBIE proceeds in an online round-by-round manner. At the beginning of each round, we do not assume a priori knowledge of the system in the current round. The local cost functions become available only *after* the decision-making at the current round. The workers learn to provide an

This work has been funded in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

appropriate level of assistance to the straggler without risking becoming worse stragglers themselves, so as to reduce the global cost. Our contributions are summarized as follows:

- We formulate an online load balancing problem subject to the constraint that all workload is assigned, which has broad application in real-world systems. Our objective is to minimize the accumulation of the pointwise maximum over a set of time-varying local cost functions.
- We present DOLBIE, a novel distributed online algorithm where all workers participate in updating the workload decisions. Without a priori knowledge of future information, the workers with lower local costs learn over time to share a risk-averse amount of workloads to help the worker who incurs the highest cost in the past. We further demonstrate the flexibility of implementing DOLBIE under both the master-worker and fully-distributed architectures.
- The performance DOLBIE is analyzed in terms of the worst-case dynamic regret, which measures the time-accumulated cost difference between the decisions generated by DOLBIE and a sequence of instantaneous minimizers. It compares favorably with online gradient descent and grows sublinearly with respect to the number of decision variables.
- We demonstrate an example application of DOLBIE to online batch size tuning for distributed learning. Our experiment results show the advantage of DOLBIE over state-of-the-art algorithms, in terms of substantially reduced training time and lower computational complexity.

The rest of this paper is structured as follows. Section II presents the related work. Section III illustrates motivating examples and introduces the general formulation of online min-max load balancing problem. Section IV presents the design of DOLBIE, which can be implemented in both the master-worker architecture and the fully-distributed architecture. We analyze the dynamic regret in Section V. Experimental results are presented and discussed in Section VI. Section VII presents the conclusion.

II. Related Work

A. Offline Load Balancing

The objective of load balancing is to optimize certain system performance through even work distribution, e.g., maximizing throughput and minimizing the maximum or average delay [1], [9]. In this work, we restrict our attention to minimizing the maximum cost incurred by any worker, e.g., makespan minimization and min-max fairness. This belongs to the family of min-max optimization problems. Offline min-max optimization has been extensively studied in the literature [10]–[13]. All of these studies require a priori knowledge of the state of the system. In real-world applications, the system may evolve in an unpredictable fashion. Therefore, we seek an online solution to compute a sequence of decisions over time under uncertainty.

B. Online Load Balancing

A large family of prior studies on online load balancing focus on the scheduling, among multiple servers, of independent tasks that arrive at arbitrary times. They mostly concern task migration or re-assignment among the servers [14]–[17] (see also the surveys [18]–[20] and the references therein). In contrast, in this work we consider non-movable workload, which is important to applications with non-preemptive or non-resumable jobs, for example in machine learning [6].

Our problem formulation and solution are most closely related to those of [3] and [6]. In [3], an online algorithm was proposed to balance the workload by updating the decisions inversely proportional to the historical local cost of each worker, e.g., the local processing time. However, such proportional adjustment is not robust to non-linear cost functions. To cope with non-linear cost functions, it was proposed in [6] to assign an additional amount of work to the worker who incurred the lowest cost in the previous round, thereby reducing the workload on the worker who incurred the highest cost in the previous round. Furthermore, this method uses a prescribed fixed amount of workload increment each time. In contrast, in our work all non-straggling workers simultaneously take up additional workload, and we dynamically compute appropriate amounts of workload increment for different workers at different times.

C. Online Min-Max Optimization

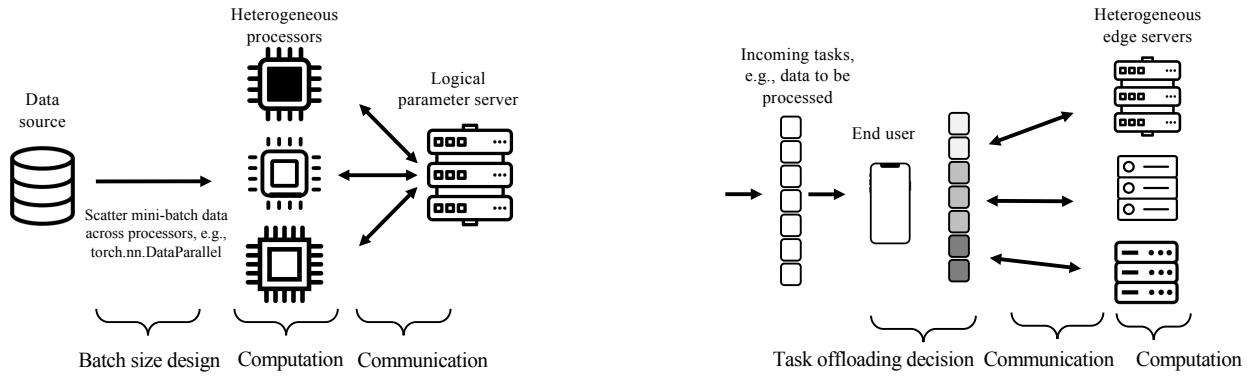
Our problem belongs to the general family of online min-max optimization. With the assumption that the local cost functions are convex, an exact penalty approach and a primal-dual Lagrangian approach were proposed to solve the min-max optimization over a time-varying topology in [21] and [22], respectively. In [23], a repeated game approach was proposed for online min-max load balancing, but it is applicable only to linear local cost functions. In the online min-max algorithm of [24], the convexity assumption was removed, but the local cost functions must be decreasing in the decision variables, which is opposite to the relation between the cost and the workload in load balancing. Therefore, none of the above solutions are applicable to our problem, where the cost functions are typically non-convex and non-decreasing.

III. Motivation and Problem Formulation

Load balancing over the pointwise maximum naturally arises in a broad range of applications. In this section, we will first discuss two example use cases and then present the general problem formulation.

A. Example 1: Batch Size Tuning in Distributed Learning

In distributed machine learning (ML), the learning task is parallelized by breaking down the data samples and assigning them to multiple workers, e.g., parallel processors or mobile devices [25]. Since the workers have heterogeneous computing capacity, they complete their tasks at different speeds, incurring wasted idle time at the synchronization barrier. Therefore, it is important to assign different sizes of data batches to



(a) Distributed learning.

(b) Task offloading.

Fig. 1: Examples of load balancing for makespan minimization.

different workers to balance their workload. This batch size tuning problem can be formulated as online load balancing to minimize the wall-clock training time [4], [26], [27].

As illustrated in Fig. 1a, a set of parallel processors $\mathcal{N} = \{1, ..., N\}$, e.g., CPUs and GPUs, cooperatively train a shared model. The parameter server is responsible for global model aggregation. The training is carried out over rounds $\mathcal{T} = \{1, ..., T\}$. Let B be the global batch size, i.e., the total number of data samples that need to be processed in each round, and $b_{i,t}$, $0 \leq b_{i,t} \leq 1$, be the portion of B assigned to processor i in round $t \in \mathcal{T}$. Let $\boldsymbol{b}_t = [b_{1,t}, ..., b_{N,t}]$.

After each processor calculates the local gradients, the parameter server will aggregate the local gradients from all processors and update the global model. Let $f_{i,t}(b_{i,t}) = f^{\mathrm{P}}_{i,t}(b_{i,t}) + f^{\mathrm{C}}_{i,t}$ denote local latency function of processor i, where $f^{\mathrm{P}}_{i,t}(b_{i,t}) = \frac{b_{i,t}B}{\gamma_{i,t}}$ is the batch processing time on processor i, $\gamma_{i,t}$ is the data processing speed, and $f^{\mathrm{C}}_{i,t} = \frac{d_{i,t}}{\varphi_{i,t}}$ is the communication time between processor i and the parameter server, $d_{i,t}$ is the transmitted ML model size and $\varphi_{i,t}$ is the data rate. The per-round training latency is dominated by the slowest processor that sends its local gradients to the parameter server, i.e., $\max_{i\in\mathcal{N}} \left\{ f^{\mathrm{P}}_{i,t}(b_{i,t}) + f^{\mathrm{C}}_{i,t} \right\}$.

We aim at minimizing the total wall-clock training time by efficiently tuning the batch size over time. The batch size tuning problem can be formulated as

$$\begin{aligned} \min_{\{\boldsymbol{b}_t\}_{t\in\mathcal{T}}} \quad & \sum_{t\in\mathcal{T}} \max_{i\in\mathcal{N}} \left\{ f^{\mathrm{P}}_{i,t}(b_{i,t}) + f^{\mathrm{C}}_{i,t} \right\}, \\ \text{s.t.} \quad & \sum_{i\in\mathcal{N}} b_{i,t} = 1, \forall t \in \mathcal{T}, \\ & b_{i,t} \geq 0, \forall i \in \mathcal{N}, \forall t \in \mathcal{T}. \end{aligned}$$

The first constraint ensures that all data samples are processed. The second constraint prevents negative values for the batch size. Note that due to the uncertain available processing power $\gamma_{i,t}$ and data rate $\varphi_{i,t}$, the processor can only observe $f_{i,t}(\cdot)$ after processing the data samples and sending its local gradient to the parameter server. This necessitates online optimization.

B. Example 2: Task Offloading in Edge Computing

Edge computing distributes computation power closer to the end users than conventional cloud computing [28]. The computation offloading problem can be formulated as the minimization of the total task completion time [4], [26], [27], [29]. The computation tasks of end users can be either executed locally or independently offloaded to nearby more powerful servers. As illustrated in Fig. 1b, a set of heterogeneous edge servers $\mathcal{N} = \{1, ..., N\}$ help process a portion of user tasks. Let $\mathcal{T} = \{1, ..., T\}$ denote the time horizon and $\boldsymbol{\lambda}_t = [\lambda_{0,t}, \lambda_{1,t}, ..., \lambda_{N,t}]$ denote the task partition scheme at time t, where $\lambda_{0,t}$ denotes the portion of the tasks for local computation and $\lambda_{i,t}$ denotes the portion of the tasks offloaded to edge server $i, \forall i \in \mathcal{N}$.

The cost function associated with local computation $f_{0,t}(\lambda_{0,t})$ is the task completion time at the user. The cost function associated with task offloading $f_{i,t}(\lambda_{i,t})$ consists of the task transmission time from the user to the edge server and the task execution time at the edge server. In each round t, the overall completion time is determined by the maximum completion time, either in local computing mode or offloading mode at server i, i.e., $\max_{i\in\{0,1,...,N\}} f_{i,t}(\lambda_{i,t})$. The objective is to minimize the total task completion time over a time horizon $\mathcal{T}$, subject to the task partition constraints. This computation offloading problem can be written as

$$\begin{aligned} \min_{\{\boldsymbol{\lambda}_t\}_{t\in\mathcal{N}}} \quad & \sum_{t\in\mathcal{T}} \max_{i\in\{0,1,...,N\}} f_{i,t}(\lambda_{i,t}) \\ \text{s.t.} \quad & \sum_{i\in\{0,1,...,N\}} \lambda_{i,t} = 1, \\ & \lambda_{i,t} \geq 0, \forall i \in \{0, 1, ..., N\}. \end{aligned}$$

Similar to the previous example, since the cost functions are unpredictable and time-varying, this is an online optimization problem.

C. General Problem Formulation

Motivated by the above examples and many other real-world applications, we formulate the general problem of online min-max load balancing as follows.

Consider a set of parallel workers $\mathcal{N} = \{1, ..., N\}$ in a system where the time is slotted into rounds $\mathcal{T} = \{1, ..., T\}$. At each round $t \in \mathcal{T}$, let $\boldsymbol{x}_t = [x_{1,t}, ..., x_{N,t}]$ denote the load balancing scheme at round t, where $x_{i,t}$ denotes the portion of workload assigned to worker i. The local cost function associated with $x_{i,t}$ is $f_{i,t}(x_{i,t})$. It is increasing, but not necessarily strictly increasing, with respect to $x_{i,t}$. Furthermore, $f_{i,t}$ may depend on other random factors such as the worker's available computation capacity or communication environment, so it varies over time. The global cost function in round t is defined as the pointwise maximum over the set of local cost functions, i.e.,

$$f_t(\boldsymbol{x}_t) = \max_{i \in \mathcal{N}} f_{i,t}(x_{i,t}).$$

Our objective is to minimize the accumulation of the sequence of global cost functions over time. Then, the load balancing problem is

$$\min_{\{\boldsymbol{x}_t\}_{t \in \mathcal{T}}} \quad \sum_{t \in \mathcal{T}} f_t(\boldsymbol{x}_t), \tag{1}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{N}} x_{i,t} = 1, \forall t \in \mathcal{T}, \tag{2}$$

$$x_{i,t} \geq 0, \forall i \in \mathcal{N}, \forall t \in \mathcal{T}. \tag{3}$$

Constraint (2) ensures that all workloads are executed. Constraint (3) prevents negative workload assignment.

Thus, the decision variables are a sequence of load balancing schemes over time. As illustrated in the above motivating examples, in many real-world often the local cost functions are unpredictable and time-varying. In particular, they are revealed only after decision making in each round. This calls for an online optimization algorithm. Moreover, directly solving this problem at a central controller might incur large communication overhead, as the controller would require the knowledge of all components in the local cost functions. This could also raise privacy concerns for the workers. Therefore, we are interested in developing a distributed online algorithm to solve problem (1).

IV. Distributed Online Load Balancing with Risk-Averse Assistance

In this section, we present the design of DOLBIE to solve the online min-max load balancing problem in (1). In DOLBIE, the workers simultaneously learn to track the unknown local cost functions, to make judicious decisions on load sharing with the straggler without becoming worse stragglers themselves. Furthermore, they update the local decision variables by only sharing the local cost values instead of the local cost functions.

A. Important Quantity $\boldsymbol{x}'_t$

In round t, given the decision $x_{i,t}$, let us label the local cost at worker i by $l_{i,t} = f_{i,t}(x_{i,t})$. Then the global cost is $l_t = \max_{i \in \mathcal{N}} l_{i,t}$. We call the worker with the highest cost the *straggler* in this round, with random tie breaking if necessary, i.e.,

$$s_t = \arg\max_{i \in \mathcal{N}} l_{i,t}.$$

Intuitively, if a small amount of workload from the straggler is moved to the non-stragglers, we can reduce l_t, but we must take care not to change the workload by too much, since otherwise a non-straggler may suffer a new cost great than that of the original straggler, thus incurring a worse global cost.

Let $\tilde{x}_{i,t}$ denote the maximum value such that $f_{i,t}(\tilde{x}_{i,t}) \leq l_t$, i.e.,

$$\tilde{x}_{i,t} = \max\{\tilde{x} | f_{i,t}(\tilde{x}) \leq l_t\}.$$

Since $f_{i,t}(\cdot)$ is increasing, $\tilde{x}_{i,t}$ can be found efficiently with function inverse or bisection search [30]. Since $l_t \geq f_{i,t}(x_{i,t})$, we have $\tilde{x}_{i,t} \geq x_{i,t}$. Since a feasible amount of workloads is restricted by the total amount of workloads $x_{i,t} \leq 1$, we refine the maximum acceptable workload by truncating it to

$$x'_{i,t} = \min\left\{\tilde{x}_{i,t}, 1\right\}. \tag{4}$$

The non-negative quantity $(x'_{i,t} - x_{i,t})$ represents the maximum additional workload that worker i could have shared with the straggler without making itself the worse straggler. Furthermore, since $l_t = f_{s_t,t}(x_{s_t,t})$, the straggler does not need to acquire additional workload, i.e., $x'_{s_t,t} = x_{s_t,t}$. We will use the quantity $x'_{i,t}$ extensively in the design of DOLBIE.

If the workers update with $x_{i,t+1} = x'_{i,t}$ for the next round t, any increase in the workloads of the non-stragglers involves a corresponding decrease in that of the straggler. Therefore, the maximum total workloads that can be reduced from the straggler is $\sum_{i \in \mathcal{N}}(x_{i,t+1} - x_{i,t})$. However, if we use this maximum, i.e., setting $x_{i,t+1} = x'_{i,t}, \forall t$, the decision variable on the straggler could become infeasible , i.e., $x_{s_t,t+1} = x_{s_t,t} - \sum_{i \in \mathcal{N}}(x_{i,t+1} - x_{i,t}) < 0$. Furthermore, such aggressive behavior could make the non-stragglers easily become a worse straggler in the next round. In the next section, we will address this issue in the design of DOLBIE.

B. DOLBIE Algorithm Design

DOLBIE can work in both the master-worker architecture and the fully-distributed architecture.

*1) **DOLBIE in Master-Worker Architecture**:* Here we first illustrate the master-worker version where either an external controller or an elected worker acts also as the master.

To address the feasibility issues discussed in Section IV-A, we update the decision variable by taking a step $\alpha_t \in [0, 1]$ from $x_{i,t}$ towards the potential maximum workload $x'_{i,t}, \forall i$. However, this step size α_t needs to be carefully coordinated among the workers.

As shown in Algorithm 1, at the beginning of round t, each worker proceeds with the workloads $x_{i,t}, \forall i$ (line 1). Then

Algorithm 1: DOLBIE (master-worker version)

Input: Number of rounds T.
Initialization: Arbitrary partition $\boldsymbol{x}_1$ and step size α_1.
for Round $t = 1$ to T **do**
  Worker $i = 1, 2, \ldots, N$ **runs in parallel:**
1 Proceed with $x_{i,t}$; ▷ **Play action**
2 Suffer cost $l_{i,t} = f_{i,t}(x_{i,t})$; ▷ **Reveal local cost**
3 Observe local cost function $f_{i,t}(\cdot)$;
4 Send $l_{i,t}$ to master; ▷ **Share local cost**
5 Receive l_t, α_t, and $\mathbf{1}_{\{i \neq s_t\}}$ from master;
  if $\mathbf{1}_{\{i \neq s_t\}}$ **then**
6 Update $x_{i,t+1}$ with ▷ **Risk-averse assistance**

$$x_{i,t+1} = x_{i,t} - \alpha_t(x_{i,t} - x'_{i,t})$$

7 Send $x_{i,t+1}$ to master;
  else
8 Receive $x_{i,t+1}$ from master;
  end
  Master runs:
9 Receive local cost $l_{i,t}$ from all workers $i \in \mathcal{N}$;
10 Observe the global cost $l_t = \max_{i \in \mathcal{N}} l_{i,t}$;
11 Identify the straggler $s_t = \arg\max_{i \in \mathcal{N}} l_{i,t}$. If multiple stragglers exist, select the worker that ranks higher in the worker list;
12 Send l_t, α_t, and $\mathbf{1}_{\{i \neq s_t\}}$ to all workers $i \in \mathcal{N}$;
13 Receive $x_{i,t+1}$ from all workers $i \in \mathcal{N}$;
14 Update $x_{s_t,t+1} = 1 - \sum_{i \neq s_t} x_{i,t+1}$;
15 Send $x_{s_t,t+1}$ to worker s_t;
16 Update α_{t+1} such that ▷ **Retain feasibility**

$$\alpha_{t+1} \leq \min\left\{\alpha_t, \frac{x_{s_t,t+1}}{N-2+x_{s_t,t+1}}\right\};$$

end

Algorithm 2: DOLBIE (fully-distributed version)

Input: Number of rounds T and worker list with size N.
Initialization: Arbitrary partition $\boldsymbol{x}_1$ and step size $\bar{\boldsymbol{\alpha}}_1$.
for Round $t = 1$ to T **do**
  Worker $i = 1, 2, \ldots, N$ **runs in parallel:**
1 Proceed with $x_{i,t}$; ▷ **Play action**
2 Suffer cost $l_{i,t} = f_{i,t}(x_{i,t})$; ▷ **Reveal local cost**
3 Observe local cost function $f_{i,t}(\cdot)$;
4 Send $l_{i,t}$ and $\bar{\alpha}_{i,t}$ to (and receive $l_{j,t}$ and $\bar{\alpha}_{j,t}$ from) worker $j, \forall j \in \mathcal{N}\backslash\{i\}$; ▷ **Share local cost**
5 Observe the global cost $l_t = \max_{j \in \mathcal{N}} l_{j,t}$;
6 Consensus on the step size $\alpha_t = \min_{j \in \mathcal{N}}\{\bar{\alpha}_{j,t}\}$;
7 Identify the straggler $s_t = \arg\max_{j \in \mathcal{N}} l_{j,t}$. If multiple stragglers exist, select the worker that ranks higher in the worker list;
  if $i \neq s_t$ **then**
8 Update $x_{i,t+1}$ with ▷ **Risk-averse assistance**

$$x_{i,t+1} = x_{i,t} - \alpha_t(x_{i,t} - x'_{i,t})$$

9 Send $x_{i,t+1}$ to the straggler s_t;
10 Update $\bar{\alpha}_{i,t+1} = \bar{\alpha}_{i,t}$;
  else
11 Receive $x_{j,t+1}$ from other workers $j, \forall j \in \mathcal{N}\backslash\{i\}$;
12 Update $x_{i,t+1} = 1 - \sum_{j \neq i} x_{j,t+1}$;
13 Update $\bar{\alpha}_{i,t+1}$ such that ▷ **Retain feasibility**

$$\bar{\alpha}_{i,t+1} \leq \min\left\{\bar{\alpha}_{i,t}, \frac{x_{i,t+1}}{N-2+x_{i,t+1}}\right\};$$

  end
end

the workers observe the resultant local cost $l_{i,t} = f_{i,t}(x_{i,t})$ assiciated with $x_{i,t}$ (line 2), and become aware of its local cost function $f_{i,t}(\cdot)$ (line 3). If the master cannot naturally obtain the global cost, each worker sends the local cost $l_{i,t}$ to it (line 4). After receiving the local cost information, the master obtains the global cost l_t and identifies the straggler s_t (lines 9-11). If multiple stragglers exist, the master will randomly choose one of the stragglers. Then, the master sends the global cost l_t and step size α_t to all workers, as well as a signal $\mathbf{1}_{\{i \neq s_t\}}$ indicating whether worker i is the straggler, where $\mathbf{1}_{\{\cdot\}}$ is the indicator function (line 12).

After receiving the information sent by the master (lines 5), the workers compute the maximum acceptable workload $x'_{i,t}$ that they could have handled, as defined in (4). If worker i is not the straggler, based on the step size α_t, it updates its decision by moving towards $x'_{i,t}$ (line 6):

$$x_{i,t+1} = x_{i,t} - \alpha_t(x_{i,t} - x'_{i,t}), \forall i \in \mathcal{N}. \tag{5}$$

To inform the straggler its new workload, the non-straggling works sent its new local decisions $x_{i,t+1}$ to the master (line 7). The master then aggregates the local decisions $x_{i,t+1}$, update and inform the straggler its updated decision variable (lines 8, 13-15), i.e.,

$$x_{s_t,t+1} = 1 - \sum_{i \neq s_t} x_{i,t+1} = x_{s_t,t} - \alpha_t \sum_{i \neq s_t}(x'_{i,t} - x_{i,t}), \tag{6}$$

With the update rules in (5) and (6), the constraint $\sum_{i \in \mathcal{N}} x_{i,t+1} = 1$ is naturally guaranteed in round $t+1, \forall t$. We emphasize here that the proposed online workload updating is different from task mitigation where the tasks are transmitted among workers. Instead, here the value $x_{i,t+1}$ instructs worker i on the amount of workload to be executed in the *next round*.

To retain feasibility with respect to constraint (3), the key is to properly coordinate the workers with step size α_t. This constraint indicates that the total workloads additionally assigned to the non-stragglers should not exceed the existing workloads of the straggler. We select a step size such that (line 16)

$$\alpha_{t+1} \leq \min\left\{\alpha_t, \frac{x_{s_t,t+1}}{N-2+x_{s_t,t+1}}\right\}. \tag{7}$$

The min operation enforces a diminishing step size, which provides useful properties for our convergence analysis, which will be discussed in Section V. The rationale behind the second term in (7) is to limit the amount of work additionally assigned

to the non-stragglers. This update of the step size provides the following two benefits: first, the non-stragglers avoid behaving aggressively and reduce the risk of becoming a worse straggler in the next round; second, constraint (3) can be naturally guaranteed. To see this, for any t, since $x'_{i,t+1} \leq 1, \forall i$, in (4), we have

$$\begin{aligned}\alpha_t &\leq \frac{x_{s_t,t}}{N-2+x_{s_t,t}} = \frac{x_{s_t,t}}{N-1-(1-x_{s_t,t})} \\ &\leq \frac{x_{s_t,t}}{\sum_{i\neq s_t} x'_{i,t} - (1-x_{s_t,t})} = \frac{x_{s_t,t}}{\sum_{i\neq s_t} x'_{i,t} - \sum_{i\neq s_t} x_{i,t}} \\ &= \frac{x_{s_t,t}}{\sum_{i\neq s_t}(x'_{i,t} - x_{i,t})}.\end{aligned}$$

In (6), we must have $x_{s_t,t+1} \geq 0$. For $i \neq s_t$, since $x'_{i,t} \geq x_{i,t}$ and $\alpha_t \in [0,1]$, we always have $x_{i,t+1} \geq x_{i,t} \geq 0$. Therefore, constraint (3) is satistied for any i and t.

The initial step size α_1 can be set similarly to (7). Since $\frac{x}{a+x}$ is increasing with x for any non-negative a, we can initialize the step size as $\alpha_1 = \frac{\min_i x_{i,1}}{N-2+\min_i x_{i,1}}, \forall i$.

*2) **DOLBIE in Fully-Distributed Architecture**:* The fully-distributed architecture helps to avoid a single point of failure or overwhelming one worker. A naive extension of the master-work version of DOLBIE to the fully-distributed version would be to make every worker i obtain the same knowledge as the master in each round t, i.e., the values of $l_{i,t}$, l_t, s_t, and $x_{j,t+1}, \forall j \in \mathcal{N}\backslash\{i\}$. However, this can lead to large communication overhead, and the workers may not want to share all private information with each other. Therefore, here we propose a communication-efficient fully-distributed version of DOLBIE with limited information exchange. Each worker shares its decision variable $x_{i,t+1}$ with only the straggler in the current round. By design, a non-straggler $i \in \mathcal{N}\backslash\{s_t\}$ will not know the decision variables $x_{j,t+1}$ of the other workers $j \in \mathcal{N}\backslash\{i\}$.

As shown in Algorithm 2, to retain feasibility on $\boldsymbol{x}_t$, we allow each worker i to maintain a local step size $\bar{\alpha}_{i,t} \in [0,1]$ in round t. Let $\bar{\boldsymbol{\alpha}}_t = [\bar{\alpha}_{1,t}, ..., \bar{\alpha}_{i,t}, ..., \bar{\alpha}_{N,t}]$. After observing the local cost $l_{i,t}$, each worker shares the cost value $l_{i,t}$ and the local step size $\bar{\alpha}_{i,t}$ with all other workers. Then, all workers obtain the global cost l_t and can independently identify the straggler s_t and compute the step size α_t (lines 4-7). If multiple stragglers exist, the worker identity numbers can be used for tie breaking. Based on the step size α_t, each worker updates its decisions by moving towards $x'_{i,t}$ (line 10). Correspondingly, the workload of the straggler is adjusted by computing the remaining workload (lines 9, 11, and 12). Note that the constraint $\sum_{i\in\mathcal{N}} x_{i,t+1} = 1$ is naturally guaranteed in round $t+1, \forall t$.

To retain feasibility with respect to constraint (3), the step size is updated by the straggler via selecting a value such that (line 13)

$$\bar{\alpha}_{s_t,t+1} \leq \min\left\{\bar{\alpha}_{s_t,t}, \frac{x_{s_t,t+1}}{N-2+x_{s_t,t+1}}\right\}. \tag{8}$$

Therefore, for any t, since $x'_{i,t+1} \leq 1, \forall i$, similar to the master-worker case, we have

$$\alpha_t = \min_{j\in\mathcal{N}}\{\bar{\alpha}_{j,t}\} \leq \bar{\alpha}_{s_t,t} \leq \frac{x_{s_t,t}}{\sum_{i\neq s_t}(x'_{i,t} - x_{i,t})},$$

and constraint (3) is satisfied for any i and t.

We remark here that both the master-worker and fully-distributed versions of DOLBIE enjoy some beneficial features by design:

1) **No gradient calculation:** In general, the gradients or subgradients of complex functions are difficult to calculate. DOLBIE enjoys substantially reduced computation complexity with no need for gradient calculation.
2) **No projection calculation:** The projection operation is expensive even for simple feasible sets [31]. For DOLBIE, the feasibility of the decision variables is naturally guaranteed by design in each round. Therefore, DOLBIE eliminates the need for projection.
3) **Distributed implementation:** DOLBIE facilitates distributed implementation in both the master-worker and fully-distributed versions. Much of the DOLBIE algorithm can be run in parallel. Moreover, each worker only needs to keep its local variable $x_{i,t}$ rather than a full copy of all decision variables $\boldsymbol{x}_t$.
4) **Privacy protection:** DOLBIE addresses the privacy concern by only communicating the workload decisions $x_{i,t}$ and the cost value $l_{i,t}$ instead of the full information contained in the local cost functions $f_{i,t}(\cdot)$.

C. Computation and Communication Complexity

In the master-worker version, in each round t, each worker i sends the local cost $l_{i,t}$, the updated decision $x_{i,t+1}$ to the master, and receives l_t, α_t, and $\mathbf{1}_{\{i\neq s_t\}}$ from the master, each of which is a scalar value. In each round t, each worker updates the value $x_{i,t+1}$. Therefore, the total per-round communication and computation complexity over all workers are both $O(N)$, achieving the same communication complexity as [3] and [6]. When compared with the projection-based online algorithms, e.g., projected online gradient methods with at least $O(N^2 \log N)$ computation complexity [31], [32], DOLBIE enjoys substantially reduced computation overhead.

In the fully-distributed version, in each round t, each worker i broadcasts its local cost $l_{i,t}$ and the local step size $\alpha_{i,t}$, and sends its decision $x_{i,t+1}$ only to the straggler s_t. Therefore, the total per-round communication complexity over all workers is $O(N^2)$. Although this is substantially increased when compared with the master-worker architecture, the fully-distributed version enjoys a higher level of privacy protection. In each round t, each worker updates its decision $x_{i,t+1}$ and the local step size $\bar{\alpha}_{i,t}$ locally, each of which is a scalar value. Therefore, the total per-round computation complexity over all workers remains $O(N)$,

V. Dynamic Regret Analysis

In this section, we analyze the theoretical performance guarantee provided by DOLBIE, by deriving an upper bound on its dynamic regret.

We consider a version of the dynamic regret over the time horizon T defined as [33]

$$\mathrm{Reg}_T^{\mathrm{d}} = \sum_{t\in\mathcal{T}} f_t(\boldsymbol{x}_t) - \sum_{t\in\mathcal{T}} f_t(\boldsymbol{x}_t^*),$$

where $\boldsymbol{x}_t^* \in \arg\min_{\boldsymbol{x}\in\mathcal{F}} f_t(\boldsymbol{x})$ is the instantaneous minimizer in round t and $\mathcal{F}$ is the feasible set of problem (1). The dynamic regret typically involves some regularity measures of the system dynamics, e.g., the path-length of the instantaneous minimizers, which is defined as $P_T = \sum_{t=2}^{T} ||\boldsymbol{x}_{t-1}^* - \boldsymbol{x}_t^*||_2$.

To proceed with our analysis, we require the following assumption, which is common in the online optimization literature:

Assumption 1 (Lipschitz)**.** *We assume $f_{i,t}(\cdot)$ is Lipschitz continuous with constant L, i.e., $|f_{i,t}(x_1) - f_{i,t}(x_2)| \le L|x_1 - x_2|, \forall x_1, x_2, i$ and t.*

Theorem 1. *Consider the online min-max load balancing problem defined in (1) with Assumption 1. The dynamic regret Reg_T^d for the sequence of decisions $\boldsymbol{x}_t$ generated by DOLBIE is upper bounded by $\mathrm{Reg}_T^d \le \sqrt{TL^2(\frac{1}{\alpha_T} + \frac{P_T}{\alpha_T} + \sum_{t=1}^{T}\frac{\frac{N-1}{2}+N\alpha_t}{2})}$, where P_T is the path length of the dynamic minimizers.*

Proof. For notation simplicity, let $G_{i,t}$ denote the maximum acceptable workload on worker i in round t defined in the design of DOLBIE, i.e.,

$$G_{i,t} = \begin{cases} x_{i,t} - x'_{i,t}, & \text{if } i \neq s_t, \\ -\sum_{j\neq s_t}(x_{j,t} - x'_{j,t}), & \text{o.w.} \end{cases}$$

Let $\boldsymbol{G}_t = [G_{1,t}, ..., G_{N,t}]$. The updating rule in (5)–(6) can be rewritten as

$$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t - \alpha_t \boldsymbol{G}_t. \tag{9}$$

We emphasize that $\boldsymbol{G}_t$ is not the gradient but the designed degree of risk-averse assistance in DOLBIE. We first show that DOLBIE implies the following lemmas:

Lemma 1. *For any round t, let $\boldsymbol{x}_t^* = [x_{1,t}^*, ..., x_{N,t}^*]$ denote the instataneous minimizer of problem (1). Any feasible solution $\boldsymbol{x}_t = [x_{1,t}, ..., x_{N,t}] \in \mathcal{F}$ has the following properties:*
i) $x_{s_t,t} \ge x_{s_t,t}^*$;
ii) $x'_{i,t} \ge x_{i,t}, \forall i \in \mathcal{N}$;
iii) $x'_{i,t} \ge x_{i,t}^*, \forall i \in \mathcal{N}$;
iv) $\sum_{i\neq s_t}(x_{i,t} - x'_{i,t})(x_{i,t} - x_{i,t}^*) \ge -1/4$,
where $s_t = \arg\max_{i\in\mathcal{N}} f_{i,t}(x_{i,t})$ denotes the worker with the highest cost when applying the solution $\boldsymbol{x}_t$ and $x'_{i,t}$ is defined the same as in (4).

***Proof Sketch of Lemma* 1** For any round t and any feasible solution $\boldsymbol{x}_t$,

$$f_{s_t,t}(x_{s_t,t}) = f_t(\boldsymbol{x}_t) \ge f_t(\boldsymbol{x}_t^*) = \max_{j\in\mathcal{N}} f_{j,t}(x_{j,t}^*) \ge f_{s_t,t}(x_{s_t,t}^*).$$

Since $f_{s_t,t}(\cdot)$ is a non-decreasing function, we have i).

Since $x_{i,t}^*, x_{i,t} \le 1$ for any i and t, if $x'_{i,t} = 1$, we directly have ii) and iii). If $x'_{i,t} = \tilde{x}_{i,t}$, since $f_t(\boldsymbol{x}_t) \ge f_{i,t}(x_{i,t})$ and $f_t(\boldsymbol{x}_t) \ge f_t(\boldsymbol{x}_t^*)$, we have

$$f_{i,t}(x'_{i,t}) \ge f_{i,t}(x_{i,t}), \text{ and } f_{i,t}(x'_{i,t}) \ge f_{i,t}(x_{i,t}^*).$$

Since $f_{i,t}(\cdot)$ is a non-decreasing function, we have ii) and iii).

To prove iv), we can derive its left-hand side as follows:

$$\begin{aligned} &\sum_{i\neq s_t}(x_{i,t} - x'_{i,t})(x_{i,t} - x_{i,t}^*) \\ &= \sum_{i\neq s_t}\left\{\left(x_{i,t} - \frac{x'_{i,t} + x_{i,t}^*}{2}\right)^2 - \left(\frac{x'_{i,t} - x_{i,t}^*}{2}\right)^2\right\} \\ &\ge -\frac{\sum_{i\neq s_t}(x'_{i,t} - x_{i,t}^*)^2}{4} \ge -\frac{N-1}{4}. \end{aligned}$$

The last step holds since $0 \le x'_{i,t}, x_{i,t}^* \le 1, \forall i \in \mathcal{N}$. □

Lemma 2. *Any instantaneous feasible solution $\boldsymbol{x}_t \in \mathcal{F}$ to problem (1) with Assumption 1 satisfies*

$$\left[\frac{f_t(\boldsymbol{x}_t) - f_t(\boldsymbol{x}_t^*)}{L}\right]^2 \le \frac{N-1}{4} + \boldsymbol{G}_t^{\mathsf{T}}(\boldsymbol{x}_t - \boldsymbol{x}_t^*) \tag{10}$$

***Proof of Lemma* 2** From iv) in Lemma 1, we have

$$\begin{aligned} \boldsymbol{G}_t^{\mathsf{T}}(\boldsymbol{x}_t - \boldsymbol{x}_t^*) &= \sum_{i\neq s_t}(x_{i,t} - x'_{i,t})(x_{i,t} - x_{i,t}^*) \\ &\quad - (x_{s_t,t} - x_{s_t,t}^*)\sum_{i\neq s_t}(x_{i,t} - x'_{i,t}) \\ &\ge -\frac{N-1}{4} + (x_{s_t,t} - x_{s_t,t}^*)\sum_{i\neq s_t}(x'_{i,t} - x_{i,t}) \end{aligned}$$

From iii) in Lemma 1, we have $x'_{i,t} \ge x_{i,t}^*$. Since $\sum_{i\in\mathcal{N}} x_{i,t} = \sum_{i\in\mathcal{N}} x_{i,t} = 1$, we have

$$\begin{aligned} &\sum_{i\neq s_t}(x'_{i,t} - x_{i,t}) \ge \sum_{i\neq s_t}(x_{i,t}^* - x_{i,t}) \\ &= (1 - x_{s_t,t}^*) - (1 - x_{s_t,t}) = x_{s_t,t} - x_{s_t,t}^*. \end{aligned}$$

Therefore, we have

$$\boldsymbol{G}_t^{\mathsf{T}}(\boldsymbol{x}_t - \boldsymbol{x}_t^*) \ge -\frac{N-1}{4} + \left(x_{s_t,t}^* - x_{s_t,t}\right)^2.$$

Then, with Assumption 1, we have

$$\begin{aligned} \left(x_{s_t,t}^* - x_{s_t,t}\right)^2 &\ge \left[\frac{f_{s_t,t}(x_{s_t,t}) - f_{s_t,t}(x_{s_t,t}^*)}{L}\right]^2 \\ &\overset{(a)}{\ge} \left[\frac{f_{s_t,t}(x_{s_t,t}) - f_{s_t^*,t}(x_{s_t^*,t}^*)}{L}\right]^2 \ge \left[\frac{f_t(\boldsymbol{x}_t) - f_t(\boldsymbol{x}_t^*)}{L}\right]^2, \end{aligned}$$

where $s_t^* = \arg\max_{i\in\mathcal{N}} f_{i,t}(x_{i,t}^*)$ denotes the worker who is the straggler in round t with the instantaneous optimal solution $\boldsymbol{x}_t^*$. Inequality (a) holds since $f_{s_t,t}(x_{s_t,t}^*) \le f_{s_t^*,t}(x_{s_t^*,t}^*)$. Therefore, we further have

$$\frac{N-1}{4} + \boldsymbol{G}_t^{\mathsf{T}}(\boldsymbol{x}_t - \boldsymbol{x}_t^*) \ge \left[\frac{f_t(\boldsymbol{x}_t) - f_t(\boldsymbol{x}_t^*)}{L}\right]^2. \quad \square$$

We are now ready to proceed with the proof of Theorem 1. With the updating rule of DOLBIE in (9), we have

$$\begin{aligned} &||\boldsymbol{x}_{t+1} - \boldsymbol{x}_t^*||^2 = ||\boldsymbol{x}_t - \boldsymbol{x}_t^* - \alpha_t \boldsymbol{G}_t||^2 \\ &= ||\boldsymbol{x}_t - \boldsymbol{x}_t^*||^2 + {\alpha_t}^2||\boldsymbol{G}_t||^2 - 2\alpha_t \boldsymbol{G}_t^{\mathsf{T}}(\boldsymbol{x}_t - \boldsymbol{x}_t^*). \end{aligned} \tag{11}$$

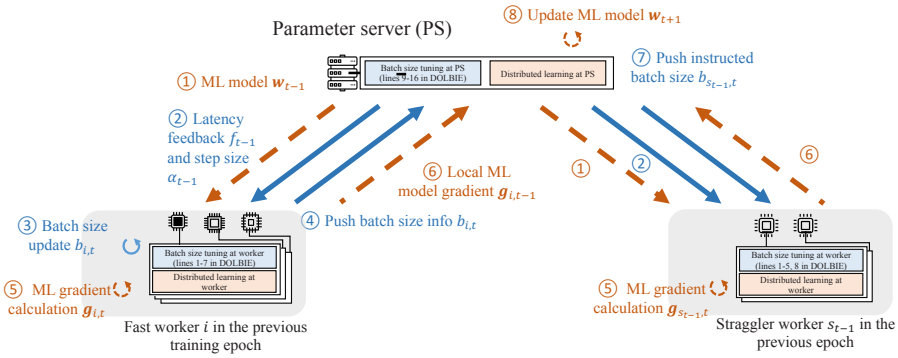


Fig. 2: DOLBIE for batch size tuning in distributed learning.

By summing up the left-hand side of (10) over t, we have

$$\frac{1}{L^2}\sum_{t=1}^{T}(f_t(\boldsymbol{x}_t)-f_t(\boldsymbol{x}_t^*))^2 \le \sum_{t=1}^{T}\left\{\boldsymbol{G}_t^{\mathsf{T}}(\boldsymbol{x}_t-\boldsymbol{x}_t^*)+\frac{N-1}{4}\right\}$$
$$\overset{(a)}{\le} \sum_{t=1}^{T}\frac{1}{2\alpha_t}(||\boldsymbol{x}_t-\boldsymbol{x}_t^*||^2-||\boldsymbol{x}_{t+1}-\boldsymbol{x}_t^*||^2)+\sum_{t=1}^{T}\frac{\frac{N-1}{2}+N\alpha_t}{2}$$
$$\overset{(b)}{=} \frac{||\boldsymbol{x}_1||^2}{2\alpha_1}-\frac{||\boldsymbol{x}_{T+1}||^2}{2\alpha_T}+\sum_{t=2}^{T}(\frac{1}{2\alpha_t}-\frac{1}{2\alpha_{t-1}})||\boldsymbol{x}_t||^2-\frac{\boldsymbol{x}_1^{\mathsf{T}}\boldsymbol{x}_1^*}{\alpha_1}$$
$$+\frac{\boldsymbol{x}_t^{\mathsf{T}}\boldsymbol{x}_{t+1}}{\alpha_T}+\sum_{t=2}^{T}(\frac{\boldsymbol{x}_t^{\mathsf{T}}\boldsymbol{x}_{t-1}^*}{\alpha_{t-1}}-\frac{\boldsymbol{x}_t^{\mathsf{T}}\boldsymbol{x}_t^*}{\alpha_t})+\sum_{t=1}^{T}\frac{\frac{N-1}{2}+N\alpha_t}{2}$$
$$\overset{(c)}{\le} \frac{||\boldsymbol{x}_1||^2}{2\alpha_1}+\frac{\boldsymbol{x}_t^{\mathsf{T}}\boldsymbol{x}_{t+1}}{2\alpha_T}+\frac{1}{\alpha_T}\sum_{t=2}^{T}(\boldsymbol{x}_{t-1}^*-\boldsymbol{x}_t^*)^{\mathsf{T}}\boldsymbol{x}_t$$
$$+\sum_{t=1}^{T}\frac{\frac{N-1}{2}+N\alpha_t}{2} \overset{(d)}{\le} \frac{1}{\alpha_T}+\frac{P_T}{\alpha_T}+\sum_{t=1}^{T}\frac{\frac{N-1}{2}+N\alpha_t}{2},$$

where (a) is due to (11), (b) holds since $||\boldsymbol{x}_t-\boldsymbol{x}_t^*||^2-||\boldsymbol{x}_{t+1}-\boldsymbol{x}_t^*||^2=||\boldsymbol{x}_t||^2-||\boldsymbol{x}_{t+1}||^2+2(\boldsymbol{x}_{t+1}-\boldsymbol{x}_t)^{\mathsf{T}}\boldsymbol{x}_t^*$, and $||\boldsymbol{G}_t||^2\le N$, (c) holds due to diminishing step sizes $\alpha_{t-1}\ge\alpha_t, \forall t$ defined in (8), and (d) holds since $||\boldsymbol{x}_t||^2\le 1$ and

$$\sum_{t=2}^{T}(\boldsymbol{x}_{t-1}^*-\boldsymbol{x}_t^*)^{\mathsf{T}}\boldsymbol{x}_t \le \sum_{t=2}^{T}||\boldsymbol{x}_{t-1}^*-\boldsymbol{x}_t^*||\cdot||\boldsymbol{x}_t||.$$

As the inequality $\frac{1}{T}\sum_t x_t \le \sqrt{\frac{1}{T}\sum_t x_t^2}$ holds for any x_t, we further have

$$\text{Reg}_T^{\text{d}}=\sum_{t=1}^{T}f_t(\boldsymbol{x}_t)-f_t(\boldsymbol{x}_t^*) \le \sqrt{T\sum_{t=1}^{T}(f_t(\boldsymbol{x}_t)-f_t(\boldsymbol{x}_t^*))^2}$$
$$\le \sqrt{TL^2(\frac{1}{\alpha_T}+\frac{P_T}{\alpha_T}+\sum_{t=1}^{T}\frac{\frac{N-1}{2}+N\alpha_t}{2})}. \qquad \square$$

Since we do not make prior assumptions on how system varies over time, the dynamic regret scales linearly in the worst case. Our upper bound compares favorably with online gradient descent. It scales the same as in Theorem 2 of [33], while it does not require any assumption on the convexity of the cost function. Moreover, the dynamic regret bound grows sublinearly with respect to the number of workers, i.e., the number of decision variables, making DOLBIE an attractive alternative to existing solutions.

VI. Experimental Performance Evaluation

As an example for experimental performance evaluation, we apply DOLBIE to the problem of batch size tuning in distributed ML as defined in Section III-A. We conduct extensive experiments to compare the performance of DOLBIE with that of several state-of-the-art alternatives.

A. Integration of DOLBIE and Distributed ML

As illustrated in Fig. 2, both the parameter server and workers are responsible for batch size tunning as well as learning over the designed batch of data samples. The round index t of DOLBIE is the same as the ML training rounds, each of which consists of a batch size tuning phrase of running DOLBIE and a prescribed ML learning phrase. In the batch size tuning phrase, the parameter server has access to the training latency in the previous round f_{t-1} and the workers locally determine the batch size simultaneously for the current round. The parameter server only makes decisions for the straggler who is the slowest in the previous round.

At the end of the previous training round $t-1$, the parameter server becomes aware of the training latency $f_{t-1}(\boldsymbol{b}_{t-1})$. After each worker calculates the local gradients of the ML model, the processing speed $\gamma_{i,t-1}$ is observed by the worker. After each worker sends its local gradient to the parameter server, the communication time $f_{i,t-1}^{\text{C}}$ is observed by the worker. Therefore, at the beginning of round t, the previous training latency $f_{t-1}(\boldsymbol{b}_{t-1})$ becomes available to all workers and the local information $f_{i,t-1}(\cdot)$ becomes available to the corresponding worker i. Then, each worker independently updates the workload by first calculating the maximum workload it can handle without making itself the straggler in (4), i.e.,

$$b'_{i,t-1}=\min\left\{1,\frac{\left(f_{t-1}-f_{i,t-1}^{\text{C}}\right)\gamma_{i,t-1}}{B}\right\}.$$

The non-stragglers update their decision variables by processing a suitably increased amount of data samples, i.e.,

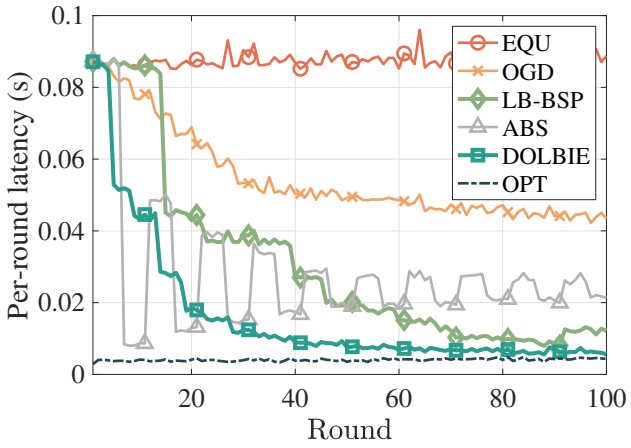


Fig. 3: Per-round latency of one realization.

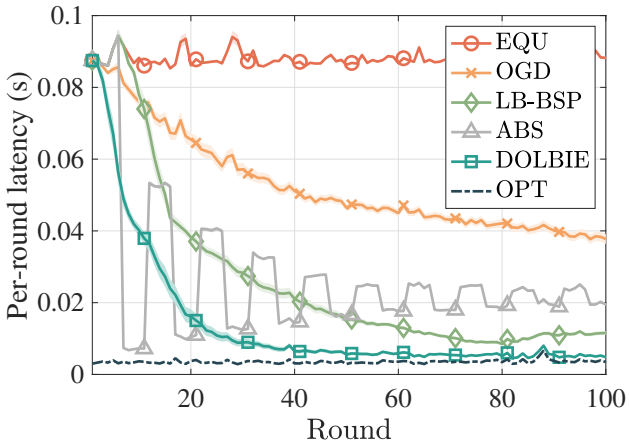


Fig. 4: Per-round latency with 95% CI.

$b_{i,t} = b_{i,t-1} - \alpha_{t-1}(b_{i,t-1} - b'_{i,t-1})$. The remaining workload that has not been claimed by the non-stragglers is assigned to the straggler, i.e., $b_{s_{t-1},t} = 1 - \sum_{i \neq s_{t-1}} b_{i,t}$. The parameter server then updates the step size α_t to retain feasibility for the next round, i.e., $\alpha_t \leq \min\left\{\alpha_{t-1}, \frac{b_{s_t,t}}{N-2+b_{s_t,t}}\right\}$. After this batch size tuning phrase, each worker i calculates the local gradients of the ML model by processing a batch of data samples of size $b_{i,t}B$. After aggregating the local gradients from all workers, the parameter server updates the global ML model. Afterward, the current training round ends and all workers enter the next training round.

B. Experimental Results

We perform experiments using the actual measured computation time and data transfer time of training various ML models over various processors. We consider a distributed learning scenario where $N = 30$ heterogeneous workers cooperatively train ML models in parallel. Each worker is equipped with one of the following processors uniformly at random: NVIDIA Tesla V100 Pascal, NVIDIA Tesla P100 Volta, NVIDIA T4 Turing, Intel Xeon Gold 6238 Cascade Lake @ 2.10GHz, and Intel E5-2683 v4 Broadwell @ 2.1GHz. We train LeNet5 [34], ResNet18 [35], and VGG16 [36] on the CIFAR-10 dataset [37]. Our learning system is implemented with the distributed package (i.e., torch.distributed) in PyTorch.

We train the three ML models with the cross-entropy loss and the stochastic gradient descent (SGD) optimizer. The learning rate is set to 0.1. We set a global batch size $B = 256$ as the required number of data samples to be processed in each training round.

We compare the performance of DOLBIE with that of the following benchmarks:

- **Equal assignment (EQU)** : Each worker processes data of batch size $\frac{B}{N}$ in each round. This is frequently assumed in the analysis of distributed training.
- **Online Gradient Descent (OGD)** [38]: The batch size is updated with gradient descent, i.e., $\boldsymbol{x}_{t+1} \leftarrow \pi_{\mathcal{F}}\left(\boldsymbol{x}_t - \beta \tilde{\mathbf{g}}_t\right)$, where β is the learning rate, $\tilde{\mathbf{g}}_t$ is the gradient or a subgradient of $f_t(\boldsymbol{x}_t)$ when the gradient does not exist, and $\pi_{\mathcal{F}}(\cdot)$ denotes the Euclidean projection onto the feasible set $\mathcal{F}$ of problem (1), which is implemented using the method in [39].
- **Adaptive Batch Size (ABS)** [3]: The batch size is updated proportionally to the training time over the previous P training rounds on each worker, where P is a predefined tuning period.
- **Load-Balanced Bulk Synchronous Parallel (LB-BSP)** [6]: If the fastest worker in the previous round preceded the straggler for consecutive D rounds, the workload of the straggler in the previous training round is reduced by Δ. The same amount of work Δ is additionally assigned to the fastest worker.
- **Dynamic Optimum (OPT)**: We assume a priori knowledge of all system variables, and we solve the instantaneous optimization problem in each round. This is also the comparator in the definition of dynamic regret. We note that OPT cannot be implemented in reality due to the lack of future information.

All algorithms are initialized with batch size B/N. The step size β of OGD and the initial step size α_1 of DOLBIE are both set to 0.001. We set $\Delta = 5$ and $P = D = 5$, which are the same as in [3] and [6]. Our experiments are run over the actual processing speed and the parameter transfer time among processors in each round.

Fig. 3 shows one realization of the per-round latency when training ResNet18 on CIFAR-10. Figs. 4 and 5 show the performance of various algorithms, with 95% confidence intervals (CI), over 100 realizations of processor sampling. We observe that EQU incurs the worst latency and ABS shows a radical fluctuation. The performance of ABS and LB-BSP is affected by the design of the window sizes P and D, resulting in a step-down decrease in latency. The other three methods, OGD, LB-BSP, and DOLBIE, adjust the workload partition by identifying the straggler. However, OGD and LB-BSP require many more rounds than DOLBIE. Because the update in OGD and LB-BSP occurs only at the fastest and slowest workers

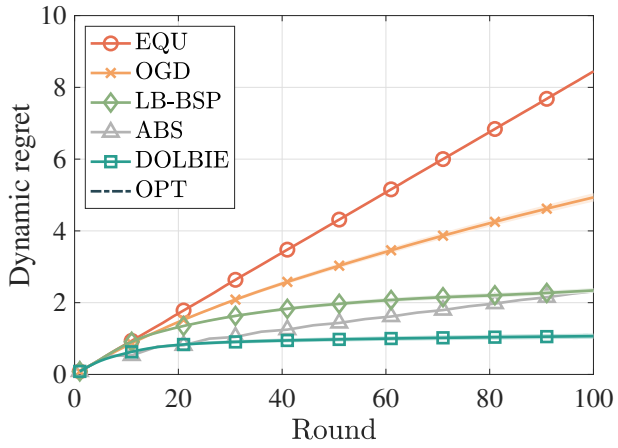


Fig. 5: Dynamic regret with 95% CI.

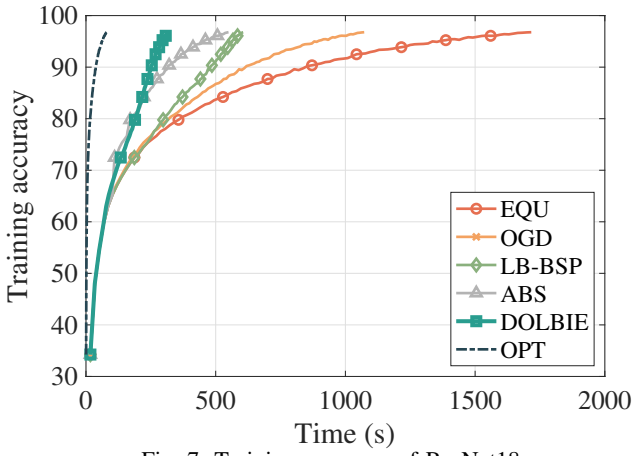


Fig. 7: Training accuracy of ResNet18.

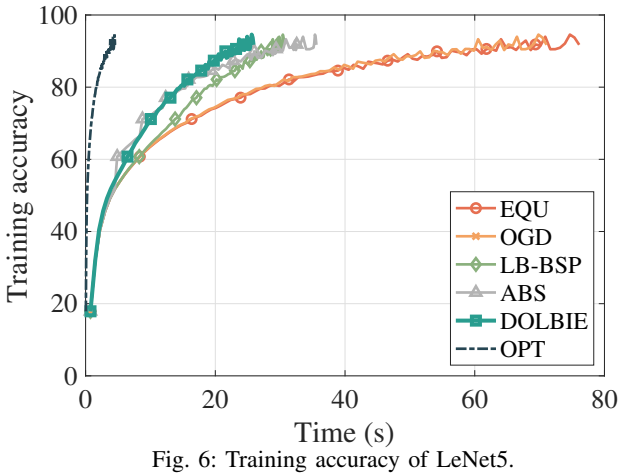


Fig. 6: Training accuracy of LeNet5.

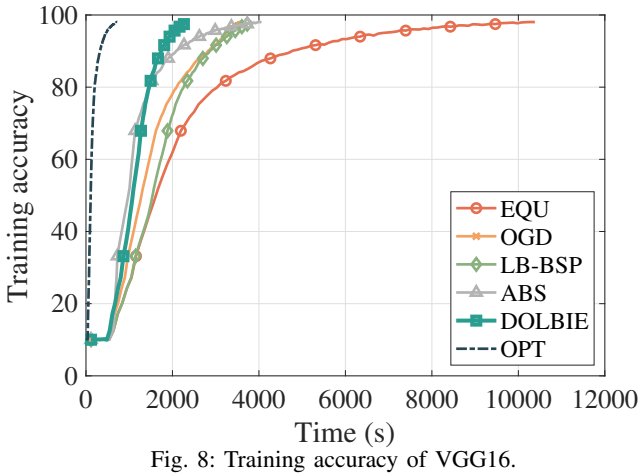


Fig. 8: Training accuracy of VGG16.

and the amount of the update in LB-BSP does not account for system heterogeneity and dynamics. In contrast, in DOLBIE, all non-straggling workers participate in assisting the straggler such that the global latency can be reduced sharply. As shown in Fig. 3, by round 40, DOLBIE has reduced the per-round latency by 89.6%, 82.2%, 67.4%, and 47.6%, respectively, when compared with EQU, OGD, LB-BSP, and ABS.

Figs. 6–8 show the training accuracy of LeNet5, ResNet18, and VGG16 versus the wall-clock time for 100 epochs, respectively. When training ResNet18 for 95% training accuracy, DOLBIE speeds up the training time by 78.1%, 67.4%, 46.9%, and 34.1%, respectively, when compared with EQU, OGD, LB-BSP, and ABS. The performance advantages of DOLBIE become more substantial as we go from LeNet5 to ResNet18 and then VGG16, i.e., as both the computational cost and the number of model parameters increase, which inevitably amplifies the system heterogeneity. With DOLBIE, each worker dynamically adjusts its workload based on the historical latency differences with the straggler workers. For 95% training accuracy, the performance advantage of DOLBIE over LB-BSP increases from 27.6% to 83.2%, when the ML task is changed from LeNet5 to VGG16.

We further investigate the performance of the individual workers under various load balancing algorithms and elaborate on the actions taken by these algorithms. Each subfigure in Figs. 9 and 10 shows the per-round training time and the corresponding batch size assigned to the workers, with the most powerful GPUs in green, Cascade Lake in orange and the straggler Broadwell in red. Overall, all load balancing algorithms reduce the global latency by assigning more workload to the non-stragglers and reducing the stragglers' workload. In EQU, each worker processes the same amount of data samples in each round and thus the per-round training latency is dominated by the same type of slow processors. In OPT, all workers incur similar latency at the optimum. The radical fluctuation of ABS is affected by its update window P and the proportional scaling in ABS overlooks the system heterogeneity that is independent of the batch size, i.e., the communication component in the training latency. OGD, LB-BSP, and DOLBIE all tune the batch size to achieve similar latency among the workers and track the system dynamics, but the lines representing different workers converge much more

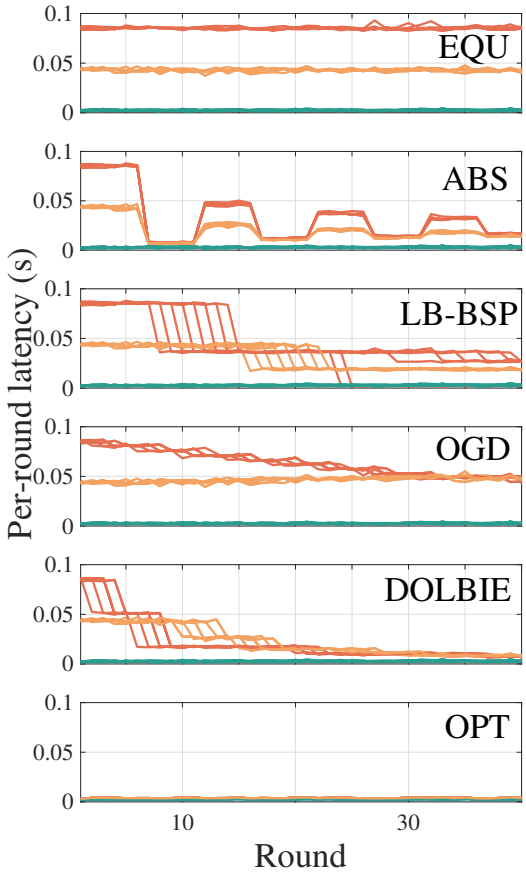


Fig. 9: Latency per worker per round.

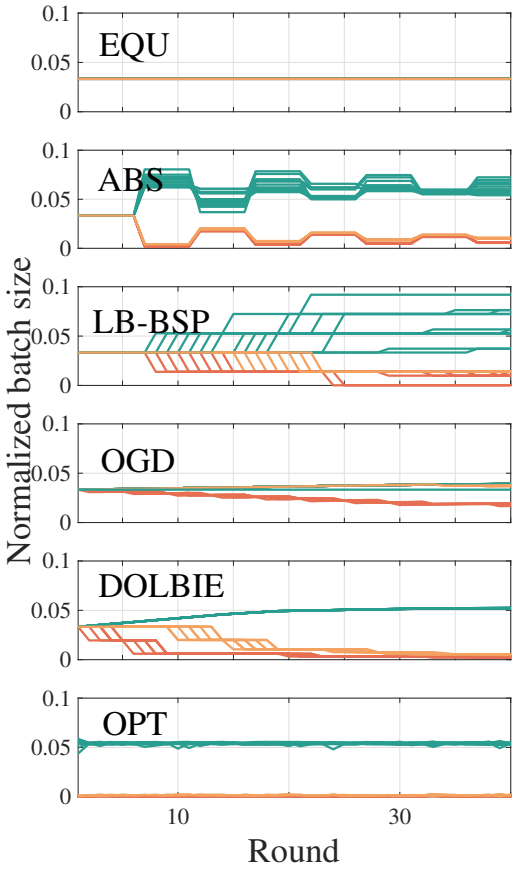


Fig. 10: Batch size per worker per round.

quickly in DOLBIE. The non-stragglers in OGD and LB-BSP are passively updated without consideration for these workers' own local cost functions, while DOLBIE jointly considers the processing and communication heterogeneity as well as the system dynamics based on historical feedback.

Fig. 11 shows the utilization averaged over all workers in 100 rounds. Besides the quality of solutions, another important criterion of online algorithms is the speed of decision-making. The upper figure decomposes the training latency into computation time, communication time, and waiting time. The lower figure shows the statistics of the overhead introduced by the load balancing algorithms. Each data point is obtained from 100 realizations. As expected, OPT provides the optimal solution where the training latency of the workers reaches equilibrium, and hence all workers are always busy with negligible waiting time. However, the algorithm run time of OPT and OGD ranks high since they either solve an instantaneous optimization problem or require gradient and projection calculation. In comparison, DOLBIE improves worker utilization leading to a significantly reduced waiting time. With DOLBIE, the average idle time among the workers in 100 epochs is reduced by 84.6%, 71.1%, 67.2%, and 42.8%, respectively when compared with EQU, OGD, LB-BSP, and ABS. Overall, DOLBIE is efficient by jointly considering system heterogeneity and system dynamics. It is also light-weighted due to its gradient- and projection-free distributed computing.

VII. Conclusion

We have proposed a new distributed online algorithm termed DOLBIE to solve the problem of dynamic load balancing in a multi-worker system with coupling linear constraints. DOLBIE has high computation efficiency without gradient or projection calculation. We have analyzed the dynamic regret of DOLBIE. When DOBIE is applied to online batch size tuning in distributed ML, our experimental results show that it outperforms state-of-the-art OGD, ABS, and LB-BSP algorithms in terms of significantly reduced training time.

References

[1] Y. Jiang, "A survey of task allocation and load balancing in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 585–599, Feb. 2016.

[2] A. V. Gerbessiotis and L. G. Valiant, "Direct bulk-synchronous parallel algorithms," *Journal of Parallel and Distributed Computing*, vol. 22, no. 2, pp. 251–267, 1994.

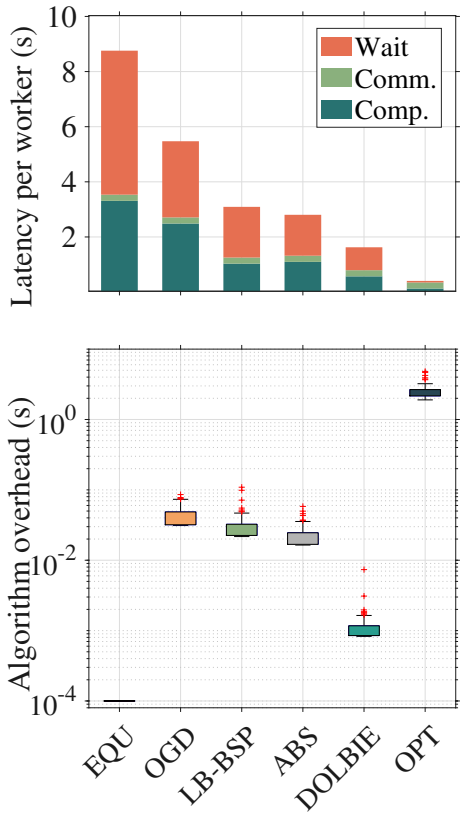


Fig. 11: Avarage time spent per worker.

[3] Q. Su, M. Wang, D. Zheng, and Z. Zhang, "Adaptive load balancing for parallel GNN training," in *Proc. MLSys Workshop on Graph Neural Networks and Systems (GNNSys)*, 2021.

[4] L. Zhang, R. Chai, T. Yang, and Q. Chen, "Min-max worst-case design for computation offloading in multi-user MEC system," in *Proc. IEEE INFOCOM Workshops*, 2020, pp. 1075–1080.

[5] P. Kumar and R. Kumar, "Issues and challenges of load balancing techniques in cloud computing: A survey," *ACM Computing Surveys*, vol. 51, no. 6, pp. 1–35, Feb. 2019.

[6] C. Chen, Q. Weng, W. Wang, B. Li, and B. Li, "Accelerating distributed learning in non-dedicated environments," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 515–531, Jan.–Mar. 2023.

[7] J. Zhang, S. Ge, T.-H. Chang, and Z.-Q. Luo, "Decentralized non-convex learning with linearly coupled constraints: Algorithm designs and application to vertical learning problem," *IEEE Transactions on Signal Processing*, vol. 70, pp. 3312–3327, 2022.

[8] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, no. 1, pp. 48–61, Jan. 2009.

[9] K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, and L. G. Gervasio, "New challenges in dynamic load balancing," *Applied Numerical Mathematics*, vol. 52, no. 2-3, pp. 133–152, 2005.

[10] I. Notarnicola, M. Franceschelli, and G. Notarstefano, "A duality-based approach for distributed Min-Max optimization," *IEEE Transactions on Automatic Control*, vol. 64, no. 6, pp. 2559–2566, Dec. 2019.

[11] S. Liu, S. Lu, X. Chen, Y. Feng, K. Xu, A. Al-Dujaili, M. Hong, and U.-M. OReilly, "Min-max optimization without gradients: Convergence and applications to black-box evasion and poisoning attacks," in *Proc. ICML*, 2020.

[12] S. Wu, X. Peng, and G. Tian, "Decentralized max-min resource allocation for monotonic utility functions," in *Proc. INFOCOM Workshops*, 2021.

[13] H. Rafique, M. Liu, Q. Lin, and T. Yang, "Weakly-convex–concave min–max optimization: provable algorithms and applications in machine learning," *Optimization Methods and Software*, vol. 37, no. 3, pp. 1087–1121, 2022.

[14] R. D. Blumofe and C. E. Leiserson, "Scheduling multithreaded computations by work stealing," *J. ACM*, vol. 46, no. 5, pp. 720–748, Sep. 1999.

[15] J. Dinan, D. B. Larkins, P. Sadayappan, S. Krishnamoorthy, and J. Nieplocha, "Scalable work stealing," in *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2009.

[16] U. A. Acar, A. Charguéraud, and M. Rainey, "Scheduling parallel programs by work stealing with private deques," in *Proc. ACM SIGPLAN Notices*, 2013, pp. 219–228.

[17] B. Acun and L. V. Kale, "Mitigating processor variation through dynamic load balancing," in *Proc. IEEE IPDPS Workshops*, 2016, pp. 1073–1076.

[18] Y. Azar, "On-line load balancing," in *Online Algorithms*. Springer, 1998.

[19] E. J. Ghomi, A. M. Rahmani, and N. N. Qader, "Load-balancing algorithms in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 88, pp. 50–71, 2017.

[20] J. Yang and Q. He, "Scheduling parallel computations by work stealing: A survey," *International Journal of Parallel Programming*, vol. 46, pp. 173–197, 2018.

[21] K. Srivastava, A. Nedić, and D. Stipanović, "Distributed Bregman-distance algorithms for min-max optimization," in *Agent-Based Optimization*. Springer, 2013.

[22] ——, "Distributed min-max optimization in networks," in *Proc. ICDSP*, 2011.

[23] Y. Liu, K. Hatano, and E. Takimoto, "Improved algorithms for online load balancing," in *Proc. International Conference on Current Trends in Theory and Practice of Informatics*, 2021, pp. 203–217.

[24] J. Wang and B. Liang, "Gradient and projection free distributed online min-max resource optimization," in *Proc. Learning for Dynamics and Control Conference*, 2022, pp. 391–403.

[25] M. Li, D. G. Andersen, A. J. Smola, and K. Yu, "Communication efficient distributed machine learning with the parameter server," in *Proc. Advances in Neural Information Processing Systems*, 2014.

[26] F. Chiti, R. Fantacci, and B. Picano, "A matching theory framework for tasks offloading in fog computing for IoT systems," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 5089–5096, Dec. 2018.

[27] Y. Wu, J. Wu, L. Chen, J. Yan, and Y. Han, "Load balance guaranteed vehicle-to-vehicle computation offloading for min-max fairness in VANETs," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 11 994–12 013, Aug. 2021.

[28] M. H. Kashani and E. Mahdipour, "Load balancing algorithms in fog computing: A systematic review," *IEEE Transactions on Services Computing*, vol. 16, no. 2, pp. 1505–1521, Mar.–Apr. 2023.

[29] J. Du, L. Zhao, J. Feng, and X. Chu, "Computation offloading and resource allocation in mixed fog/cloud computing systems with min-max fairness guarantee," *IEEE Transactions on Communications*, vol. 66, no. 4, pp. 1594–1608, Apr. 2018.

[30] E. Hansen and M. Patrick, "A family of root finding methods," *Numerische mathematik*, vol. 27, no. 3, pp. 257–269, 1976.

[31] J. Liu and J. Ye, "Efficient Euclidean projections in linear time," in *Proc. ICML*, 2009.

[32] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[33] M. Zinkevich, "Online convex programming and generalized infinitesimal gradient ascent," in *Proc. ICML*, 2003.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE CVPR*, 2016, pp. 770–778.

[36] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *Proc. ICLR*, 2015.

[37] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.

[38] E. Bampis, D. Christou, B. Escoffier, and N. K. Thang, "Online-learning for min-max discrete problems," in *Proc. PGMO*, 2020.

[39] M. Blondel, A. Fujino, and N. Ueda, "Large-scale multiclass support vector machine training via Euclidean projection onto the simplex," in *Proc. ICPR*, 2014.