Impact of Redundancy on Resilience in Distributed Optimization and Learning

Shuo Liu
Georgetown University
Washington DC, USA
sl1539@georgetown.edu

Nirupam Gupta
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
nirupam.gupta@epfl.ch

Nitin H. Vaidya
Georgetown University
Washington DC, USA
nitin.vaidya@georgetown.edu

ABSTRACT

This paper considers the problem of resilient distributed optimization and stochastic learning in a server-based architecture. The system comprises a server and multiple agents, where each agent has its own *local* cost function. The agents collaborate with the server to find a minimum of the aggregate of the local cost functions. In the context of stochastic learning, the local cost of an agent is the *loss function* computed over the data at that agent. In this paper, we consider this problem in a system wherein some of the agents may be Byzantine faulty and some of the agents may be slow (also called *stragglers*). In this setting, we investigate the conditions under which it is possible to obtain an "approximate" solution to the above problem. In particular, we introduce the notion of $(f, r; \epsilon)$-*resilience* to characterize how well the true solution is approximated in the presence of up to f Byzantine faulty agents, and up to r slow agents (or stragglers) – smaller ϵ represents a better approximation. We also introduce a measure named $(f, r; \epsilon)$-*redundancy* to characterize the *redundancy* in the cost functions of the agents. Greater redundancy allows for a better approximation when solving the problem of aggregate cost minimization.

In this paper, we constructively show (both theoretically and empirically) that $(f, r; O(\epsilon))$-*resilience* can indeed be achieved in practice, given that the local cost functions are sufficiently redundant. Our empirical evaluation considers a distributed gradient descent (DGD)-based solution; for distributed learning in the presence of Byzantine and asynchronous agents, we also evaluate a distributed stochastic gradient descent (D-SGD)-based algorithm.

CCS CONCEPTS

• **Computing methodologies** → **Distributed algorithms**.

KEYWORDS

distributed optimization, resilient optimization, fault tolerance, machine learning

ACM Reference Format:
Shuo Liu, Nirupam Gupta, and Nitin H. Vaidya. 2023. Impact of Redundancy on Resilience in Distributed Optimization and Learning. In *24th International Conference on Distributed Computing and Networking (ICDCN 2023), January 4–7, 2023, Kharagpur, India.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3571306.3571393

ICDCN 2023, January 4–7, 2023, Kharagpur, India

ACM ISBN 978-1-4503-9796-4/23/01…$15.00
https://doi.org/10.1145/3571306.3571393

1 INTRODUCTION

With the rapid growth in the computational power of modern computer systems and the scale of optimization tasks, e.g., training of deep neural networks [39], the problem of distributed optimization in a multi-agent system has gained significant attention in recent years. This paper considers the problem of *resilient* distributed optimization and stochastic learning in a server-based architecture.

The system under consideration consists of a *trusted* server and multiple agents, where each agent has its own "local" cost function. The agents collaborate with the server to find a minimum of the aggregate cost functions (i.e., the aggregate of the local cost functions) [9]. Specifically, suppose that there are n agents in the system where each agent i has a cost function $Q_i : \mathbb{R}^d \to \mathbb{R}$. The goal then is to enable the agents to compute a global minimum x^* such that

$$x^* \in \arg\min_{x \in \mathbb{R}^d} \sum_{i=1}^{n} Q_i(x). \tag{1}$$

As a simple example, consider a group of n people that want to pick a place for a meeting. Suppose that function $Q_i(x)$ represents the cost for the i-th person (agent) to travel to location x, x being its coordinates. Then, $x^* \in \arg\min_{x \in \mathbb{R}^d} \sum_{i=1}^{n} Q_i(x)$ is a location such that the travel cost of meeting at x^*, when aggregated over all the agents, is the smallest possible. In the context of training of a deep neural network, $Q_i(x)$ represents the *loss function* corresponding to the data at agent i when using machine parameter vector x. Thus, our results are relevant in the context of distributed machine learning [9]. The above multi-agent optimization problem has many other applications as well, including distributed sensing [41], and swarm robotics [42].

Such distributed multi-agent systems may encounter some challenges in practice. We consider two challenges in this paper:

- *Byzantine faulty agents*: A Byzantine faulty [28] agent may share wrong information with the server, or not send any information at all. The Byzantine fault model [28] does not impose any constraints on the behavior of a faulty agent. An agent's faulty behavior may occur either due to software/hardware failure or due to a security compromise of the agent. Prior work has shown that even a single faulty agent can compromise the entire distributed optimization process [49].

- *Slow agents (Stragglers)*: Stragglers are agents that operate much more slowly than the other agents. Synchronous algorithms that require each agent to communicate with the server in each "round" of the computation can perform poorly, since the stragglers will slow down the progress of the computation [4, 21, 30].

The related work is discussed in more detail in Section 2. Recall the travel and meeting example in the previous paragraph. A "faulty" people may deliberately send a wrong cost function so that the minimum would be his desired place; while a "straggler" may be unresponsive, delaying everyone from reaching a decision. The above two challenges have been considered independently in the past work [5, 10, 20, 23, 35, 38, 50]. This paper makes contributions that further our understanding of distributed optimization algorithms that are resilient to both the forms of adversities above.

1.1 Contributions of this paper

It is known that, under the above adverse conditions, it is not always possible to solve the problem of interest exactly. This paper addresses the interaction between these two forms of adversities. In particular, we consider a multi-agent system with up to f faulty agents and up to r stragglers (or slow agents) out of n agents. It is possible that the same agent may be faulty and slow both. Let $\mathcal{H}$ denote the set of non-faulty (or honest) agents in a given execution. In this paper, we consider the *Resilient Distributed Optimization* (RDO) problem, with the goal of approximately computing

$$\arg\min_{x\in\mathbb{R}^d} \sum_{i\in\mathcal{H}} Q_i(x) \tag{2}$$

in the presence of up to f Byzantine faulty agents and up to r slow agents. For the problem to be solvable, we assume that $n > 2f + r$. Previous research has established that Byzantine fault-tolerance problem is solvable when the non-faulty agents dominates the set of agents, or $n > 2f$ [35]; we also need enough agents to be synchronous for the problem to be solvable.

We characterize how well the true solution of (2) can be approximated, as a function of the level of "redundancy" in the cost functions. As a trivial example, if all the agents have an identical cost function, then (2) can be obtained by simply taking a majority vote on the outcome of local cost function optimization performed by each agent separately. In general, such a high degree of redundancy may not be available, and only an approximate solution of (2) may be obtainable. We introduce the notion of $(f, r; \epsilon)$-*resilience* to characterize how well the true solution is approximated in the presence of up to f Byzantine faulty agents, and up to r slow agents (or stragglers) – smaller ϵ represents a better approximation. We also introduce a measure named $(f, r; \epsilon)$-*redundancy* to characterize the *redundancy* in the cost functions of the agents. Greater redundancy allows a better approximation when solving the problem of aggregate cost minimization.

We constructively show (both theoretically and empirically) that $(f, r; O(\epsilon))$-*resilience* can indeed be achieved in practice, provided that the local cost functions are sufficiently redundant. Our empirical evaluation considers a distributed gradient descent (DGD)-based solution; in particular, for distributed learning in the presence of Byzantine and asynchronous agents, we evaluate a distributed stochastic gradient descent (D-SGD)-based algorithm.

1.2 $(f, r; \epsilon)$-*resilience* and $(f, r; \epsilon)$-*redundancy*

1.2.1 Euclidean and Hausdorff Distance. To help define the notions of $(f, r; \epsilon)$-*resilience* and $(f, r; \epsilon)$-*redundancy*, we will use Euclidean and Hausdorff distance measures. Let $\|\cdot\|$ represents the Euclidean norm. Then Euclidean distance between points x and y in $\mathbb{R}^d$ equals $\|x - y\|$. The Euclidean distance between a point x and a set Y in $\mathbb{R}^d$, denoted by $\text{dist}(x, Y)$, is defined as:

$$\text{dist}(x, Y) = \inf_{y\in Y} \text{dist}(x, y) = \inf_{y\in Y} \|x - y\|.$$

The Hausdorff distance between two sets X and Y in $\mathbb{R}^d$, denoted by $\text{dist}(X, Y)$, is defined as:

$$\text{dist}(X, Y) \triangleq \max\left\{\sup_{x\in X} \text{dist}(x, Y), \sup_{y\in Y} \text{dist}(y, X)\right\}.$$

1.2.2 Define $(f, r; \epsilon)$-resilience and $(f, r; \epsilon)$-redundancy. We now define the notion of $(f, r; \epsilon)$-*resilience* to characterize how closely (2) is approximated by a given RDO algorithm (the definition below denotes by $\widehat{x}$ the output produced by the RDO algorithm).

Definition 1 ($(f, r; \epsilon)$-resilience). *For $\epsilon \geq 0$, a distributed optimization algorithm is said to be $(f, r; \epsilon)$-resilient if its output $\widehat{x}$ satisfies*

$$\text{dist}\left(\widehat{x}, \arg\min_{x\in\mathbb{R}^d} \sum_{i\in\mathcal{H}} Q_i(x)\right) \leq \epsilon$$

for each set $\mathcal{H}$ of $n - f$ non-faulty agents, despite the presence of up to f faulty agents and up to r stragglers.

This resilience notion captures how well a given RDO algorithm performs in terms of approximating the solution of (2). Observe that the above definition considers the distance of $\widehat{x}$ (i.e., the output of the given RDO algorithm) from the true solution for every subproblem corresponding to $n - f$ non-faulty agents. Intuitively, the reason is as follows: Consider the case when the faulty agents behave badly in a manner that is *not detectable* – that is, by simply looking at the information received from the faulty agents, the trusted server cannot determine if they are faulty. Thus, it is difficult to know how many agents are faulty. In particular, it is possible that exactly $n - f$ agents are non-faulty, with the rest being faulty. Therefore, intuitively, we want $\widehat{x}$ to be within ϵ of the true solution that minimizes the aggregate cost over any $n - f$ non-faulty agents.

In this paper, we show how the *redundancy* in agents' cost functions can be utilized to obtain $(f, r; \epsilon)$-resilience. Formally, we define the property as $(f, r; \epsilon)$-*redundancy*, stated below.

Definition 2 ($(f, r; \epsilon)$-redundancy). *For $\epsilon \geq 0$, the agents' local cost functions are said to satisfy the $(f, r; \epsilon)$-redundancy property if and only if for every pair of subsets of agents $S, \widehat{S} \subsetneq \{1, ..., n\}$, where $|S| = n - f$, $\left|\widehat{S}\right| \geq n - r - 2f$ and $\widehat{S} \subsetneq S$,*

$$\text{dist}\left(\arg\min_{x} \sum_{i\in S} Q_i(x), \arg\min_{x} \sum_{i\in\widehat{S}} Q_i(x)\right) \leq \epsilon.$$

The $(f, r; \epsilon)$-redundancy condition also characterizes the trade-off between parameters f, r and ϵ. In particular, for any set of local cost functions (i.e., $Q_i(x)$'s), and for any $r < n$ and $f < (n - r)/2$, *there exists* some ϵ such that the agents' cost functions satisfy the

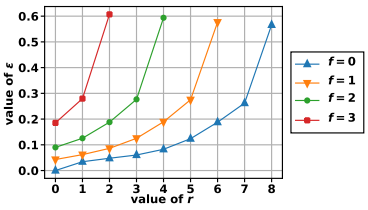

Figure 1: Trade-off between ϵ, f and r for the numerical example in Section 5. The total number of agents $n = 10$.

Table 1: Problem space of solving RDO using resilience in cost functions.

Deterministic / Stochastic	Synchronous	Asynchronous
Fault-free	-	Sec. 3.2.1 / Sec. 4.1.2
With Byzantine agents	[19, 35] / Sec. 4.1.1	Sec. 3.1 / Sec. 4

$(f, r; \epsilon)$-redundancy property for that ϵ value. For a given set of local cost functions,

- With a fixed value of f, larger r results in larger ϵ.
- With a fixed value of r, larger f results larger ϵ.

Thus, the notion of $(f, r; \epsilon)$-redundancy is a *description* of the redundancy in the cost functions. Note that for a given set of functions, we always consider the smallest ϵ for which redundancy condition holds. We will discuss a numerical example of distributed linear regression in Section 5. For this example, Figure 1 shows the relationship between ϵ, f and r. Each curve in Figure 1 corresponds to a fixed value of f. The horizontal axis varies parameter r. The corresponding value of ϵ, plotted on the vertical axis, illustrates the trade-off discussed above.

In the context of solving RDO problems with resilience in cost functions defined above, there are several special cases that have been solved in previous work. The current status of the problem space is shown in Table 1. There are three dimensions: deterministic vs. stochastic, synchronous vs. asynchronous, and fault-free vs. Byzantine agents in presence. We provide analysis from the redundancy-in-cost-function perspective for all remaining problems, as indicated in Table 1, showing that redundancy alone can be utilized to solve RDO problems.

1.3 Paper Outline

The related work is discussed in more detail in Section 2. In Section 3, we propose a gradient-based algorithm framework that can be used to solve the RDO problems in practice, and analyze its resilience under various settings, while utilizing the redundancy property of the local cost functions. In Section 4, we adapt the proposed framework to solve resilient distributed *stochastic optimization problems*, which has relevance in distributed machine learning, and present convergence results for the stochastic algorithm. To validate our theoretical findings, we present empirical results in Section 5.

The proofs of theoretical results in this paper are provided in our report [36] on arXiv.

2 RELATED WORK

Byzantine fault-tolerant optimization. Byzantine agents make it difficult to achieve the goal of distributed optimization (1) [5, 48]. Various methods have been proposed to solve Byzantine fault-tolerant optimization or learning problems [34], including robust gradient aggregation [5, 11], gradient coding [10], and other methods [18, 54, 57].

In general, instead of solving the problem exactly, an approximate solution may be found. In some cases, however, exact solution is feasible. For instance, when every agent has the same (identical) cost function, it is easy to see that an exact solution may be obtained via majority vote on the output of the agents, so long as only a minority of the agents are faulty. Stochastic versions of distributed optimization algorithms are also proposed to exploit such forms of *redundancy* in the context of distributed machine learning (e.g, [5, 11]). [19] obtained conditions under which it is possible to tolerate f Byzantine agents in a *synchronous* system and yet obtain accurate solution for the optimization problem. [35] obtained a condition under which approximately correct outcome can be obtained in a *synchronous system.*

The contribution of this paper is to generalize the results in [19, 35] to a system that allows some asynchrony in the form of stragglers – allowing for stragglers makes the proposed approach more useful in practice; we also discuss deterministic and stochastic methods at the same time, closing all remaining gaps in the context of solving resilient distributed optimization problem through redundancy in cost functions.

Asynchronous optimization. For fault-free systems (i.e., in the absence of failures), there is past work addressing distributed optimization under asynchrony. Prior work shows that distributed optimization problems can be solved using *stale* gradients with a constant delay (e.g., [29]) or bounded delay (e.g., [1, 16]). Methods such as Hogwild! allow lock-free updates in shared memory [38]. Other works use variance reduction and incremental aggregation methods to improve the convergence rate [15, 22, 43, 45].

Coding has been used to mitigate the effect of stragglers or failures [23, 24, 32, 56]. Tandon et al. [50] proposed a framework using maximum-distance separable coding across gradients to tolerate failures and stragglers. Similarly, Halbawi et al. [20] adopted coding to construct a coding scheme with a time-efficient online decoder. Karakus et al. [25] proposed an encoding distributed optimization framework with deterministic convergence guarantee. Other replication- or repetition-based techniques involve either task-rescheduling or assigning the same tasks to multiple nodes [3, 17, 44, 52, 55]. These previous methods rely on algorithm-created redundancy of data or gradients to achieve robustness, while $(f, r; \epsilon)$-redundancy can be a property of the cost functions themselves, allowing us to exploit such redundancy without extra effort.

3 ALGORITHMIC FRAMEWORK FOR RESILIENT DISTRIBUTED OPTIMIZATION PROBLEMS

From now on, we use $[n]$ as a shorthand for the set $\{1, ..., n\}$. In this section, we study the resilience of a class of distributed gradient descent (DGD)-based algorithms. In the algorithms considered here,

a server maintains an estimate for the optimum. In each iteration of the algorithm, the server sends its estimate to the agents, and the agents send the gradients of their local cost functions at that estimate to the server. The server uses these gradients to update the estimate.

Specifically, Algorithm 1 is a framework for several instances of the RDO algorithms presented later in this paper. Different instances of the algorithm differ in the manner in which the iterative update (3) is performed at the server – different gradient aggregation rules are used in different instances of the RDO algorithm.

Algorithm 1: Resilient distributed gradient descent under $(f, r; \epsilon)$-redundancy

Input: n, f, r, ϵ. A convex compact set $\mathcal{W}$. Each agent i has its cost function $Q_i(x)$.

The initial estimate of the optimum, $x^0 \in \mathcal{W}$, is chosen by the server. The new estimate x^{t+1} is computed in iteration $t \geq 0$ as follows:

Step 1: The server requests each agent for the gradient of its local cost function at the current estimate x^t. Each agent j is expected to send to the server the gradient (or stochastic gradient) g_j^t with timestamp t.

Step 2: The server waits until it receives $n - r$ gradients with the timestamp of t. Suppose $S^t \subseteq \{1, ..., n\}$ is the set of agents whose gradients are received by the server at step t where $\left|S^t\right| = n - r$. The server updates its estimate to

$$x^{t+1} = \left[x^t - \eta_t \, \text{GradAgg}\left(g_j^t | \, j \in S^t; n, f, r\right)\right]_{\mathcal{W}} \tag{3}$$

where $\eta_t \geq 0$ is the step-size for each iteration t, and $[\,\cdot\,]_{\mathcal{W}}$ denotes a projection onto $\mathcal{W}$.

A *gradient aggregation rule* (GAR)

$$\text{GradAgg}(\cdot; n, f, r) : \mathbb{R}^{d \times (n-r)} \to \mathbb{R}^d$$

is a function that takes $n - r$ vectors in $\mathbb{R}^d$ (gradients) and outputs a vector in $\mathbb{R}^d$ for the update, with the knowledge that there are up to f Byzantine faulty agents and up to r stragglers out of the n agents in the system. As an example, the GAR used in synchronous distributed optimization with no faulty agents is *averaging*, i.e., $\text{GradAgg}(g_j^t | j \in [n]; n, 0, 0) = (1/n) \sum_{j=1}^n g_j^t$. The fact that only $n-r$ gradients are in use reflects the asynchrony of the optimization process caused by stragglers, while the GAR counters the effect of Byzantine agents and stragglers, together achieving resilience.

The compact set $\mathcal{W}$ is assumed to contain the true solution to the problem. We will explain the details when introducing our convergence analyses.

3.1 Convergence of resilient distributed gradient descent

Recall the problem setting that in a n-agent system, there are up to f Byzantine agents and up to r stragglers. With the existence of Byzantine agents, the goal is to solve (2) (cf. Section 1). First, we need to introduce some standard assumptions that are necessary for our analysis. Suppose $\mathcal{H} \subseteq [n]$ is any subset of non-faulty agents with $|\mathcal{H}| = n - f$.

Assumption 1. *For each (non-faulty) agent i, the function $Q_i(x)$ is μ-Lipschitz smooth, i.e., $\forall x, x' \in \mathbb{R}^d$,*

$$\left\|\nabla Q_i(x) - \nabla Q_i(x')\right\| \leq \mu \left\|x - x'\right\|. \tag{4}$$

Assumption 2. *For any set $S \subseteq \mathcal{H}$, we define the average cost function to be $Q_S(x) = \frac{1}{|S|} \sum_{j \in S} Q_j(x)$. We assume that $Q_S(x)$ is γ-strongly convex for any S subject to $|S| \geq n - 2f$, i.e., $\forall x, x' \in \mathbb{R}^d$,*

$$\left\langle \nabla Q_S(x) - \nabla Q_S(x'), \, x - x' \right\rangle \geq \gamma \left\|x - x'\right\|^2. \tag{5}$$

We also need to assume that a solution to the problem exists, i.e., the problem is non-trivial. Specifically, for each subset of non-faulty agents S with $|S| \geq n - f$, we assume that there exists a point $x_S \in \arg\min_{x \in \mathbb{R}^d} \sum_{j \in S} Q_j(x)$ such that $x_S \in \mathcal{W}$. By Assumption 2, there exists a unique minimum point $x_{\mathcal{H}} \in \mathcal{W}$ that minimize the aggregate cost functions of agents in $\mathcal{H}$, i.e.,

$$\{x_{\mathcal{H}}\} = \mathcal{W} \cap \arg\min_{x \in \mathbb{R}^d} \textstyle\sum_{j \in \mathcal{H}} Q_j(x). \tag{6}$$

Recall (3) in Step 2 of Algorithm 1. The GAR used for RDO problems with up to f Byzantine agents and up to r stragglers is

$$\text{GradAgg}\left(g_j^t | j \in S^t; n, f, r\right) = \text{GradFilter}\left(g_j^t | j \in S^t; n - r, f\right) \tag{7}$$

for every iteration t, where GradFilter is a *robust GAR*, or *gradient filter*, that enables us fault-tolerant capability [14, 26, 35]. Specifically, a gradient filter $\text{GradFilter}(\cdot; m, f) : \mathbb{R}^{d \times m} \to \mathbb{R}^d$ is a function that takes m vectors of d-dimension and outputs a d-dimension vector given that there are up to f Byzantine agents, where $m > f \geq 0$. Each agent j sends a vector

$$g_j^t = \begin{cases} \nabla Q_j(x^t), & \text{if the agent is non-faulty,} \\ \text{arbitrary vector,} & \text{if the agent is faulty} \end{cases} \tag{8}$$

to the server at iteration t.

Following the above GAR in (7), the server receives the first $n - r$ vectors from the agents in the set S^t, and sends the vectors through a gradient filter. Now we present an asymptotic convergence property of Algorithm 1.

Theorem 1. *Let $\mathcal{H}$ be any set of $n - f$ non-faulty agents in the system. Let $x_{\mathcal{H}} = \arg\min_{x \in \mathbb{R}^d} \sum_{j \in \mathcal{H}} Q_j(x)$. Suppose that Assumptions 1 and 2 hold true, and the cost functions of all agents satisfy $(f, r; \epsilon)$-redundancy. Assume that η_t satisfy $\sum_{t=0}^{\infty} \eta_t = \infty$ and $\sum_{t=0}^{\infty} \eta_t^2 < \infty$. Suppose that $\left\|\text{GradAgg}\left(g_j^t | j \in S^t; n, f, r\right)\right\| < \infty$ for all t. The proposed algorithm with aggregation rule (7) satisfies the following:*

For the point $x_{\mathcal{H}} \in \mathcal{W}$, if there exists a real-valued constant $\text{D}^* \in \left[0, \max_{x \in \mathcal{W}} \left\|x - x_{\mathcal{H}}\right\|\right)$ and $\xi > 0$ such that for each iteration t,

$$\phi_t \triangleq \left\langle x^t - x_{\mathcal{H}}, \, \text{GradAgg}\left(g_j^t | j \in S^t; n, f, r\right)\right\rangle \geq \xi \quad \text{when } \left\|x^t - x_{\mathcal{H}}\right\| \geq \text{D}^*, \tag{9}$$

then $\lim_{t \to \infty} \left\|x^t - x_{\mathcal{H}}\right\| \leq \text{D}^*$.

Intuitively, so long as the gradient filter in use satisfies the desired properties in Theorem 1, our algorithm can tolerate up to f Byzantine faulty agents and up to r stragglers in distributed optimization: the estimate x^t will eventually be within D^* distance OF $x_{\mathcal{H}}$, i.e., the algorithm is $(f, r; \text{D}^*)$-resilient.

Gradient filters that satisfy condition (9) in Theorem 1 include CGE [35] and coordinate-wise trimmed mean [35, 57]. As an example, for the CGE gradient filter[1], it can be shown that

$$\begin{aligned} \phi_t \geq \alpha(n-r)\gamma \mathrm{D}^* \left(\mathrm{D}^* - \mathrm{D}\right) > 0 \text{ when } \left\|x^t - x_{\mathcal{H}}\right\| \geq \mathrm{D}^*, \\ \text{for all } \mathrm{D}^* > \mathrm{D} \triangleq \frac{4\mu(f+r)\epsilon}{\alpha\gamma}, \\ \text{where } \alpha = \frac{n-f}{n-r} - \frac{2\mu}{\gamma} \cdot \frac{f+r}{n-r} > 0, \end{aligned} \tag{11}$$

and therefore, $\lim_{t\to\infty} \left\|x^t - x_{\mathcal{H}}\right\| \leq \mathrm{D}^*$. Intuitively, by applying CGE, the output of our new algorithm can converge to a ϵ-dependent region centered by the true minimum point $x_{\mathcal{H}}$ of aggregate cost functions of non-faulty agents, with up to f Byzantine agents and up to r stragglers, i.e., Algorithm 1 with CGE is $(f, r; O(\epsilon))$-resilient. For other valid gradient filters, the parameters in (11) may vary.

3.2 Special cases of RDO problems

There are several meaningful special cases of RDO problems. When $f = r = 0$, RDO degenerates to the synchronous distributed optimization problem with no faulty agents in the system, which can be solved by several existing methods including DGD [9, 37, 46, 51]. When $r = 0$, RDO becomes the Byzantine fault-tolerant distributed optimization problem we discussed in Section 2. $(f, 0; \epsilon)$-redundancy is necessary to achieving $(f, 0; \epsilon)$-resilience [35].

3.2.1 Asynchronous distributed optimization. When $f = 0$, RDO becomes the asynchronous distributed optimization problem with no faulty agents. As discussed in Section 2, prior research has not considered exploiting the existing redundancy in cost functions. When $f = 0$, we have $\mathcal{H} = [n]$. The goal (2) of distributed optimization becomes (1). Recall (3) in **Step 2** of Algorithm 1. We define the GAR for asynchronous optimization to be

$$\mathrm{GradAgg}\left(g_j^t | j \in S^t; n, f, r\right) = \sum_{j \in S^t} g_j^t \tag{12}$$

for every iteration t. That is, the algorithm updates the current estimates using the sum of the first $n - r$ gradients it receives. Note that $g_j^t = \nabla Q_j(x^t)$ is the full gradient. For this problem, it can be shown for Theorem 1 that if $(0, r; \epsilon)$-redundancy is satisfied,

$$\begin{aligned} \phi_t \geq \alpha n \gamma \mathrm{D}^* (\mathrm{D}^* - \mathrm{D}) > 0 \text{ when } \left\|x^t - x_{\mathcal{H}}\right\| \geq \mathrm{D}^*, \\ \text{for all } \mathrm{D}^* > \mathrm{D} = \frac{2r\mu}{\alpha\gamma}\epsilon, \\ \text{where } \alpha = 1 - \frac{r}{n} \cdot \frac{\mu}{\gamma} > 0, \end{aligned} \tag{13}$$

and therefore, $\lim_{t\to\infty} \left\|x^t - x_{\mathcal{H}}\right\| \leq \mathrm{D}^*$. By substituting (11) with $f = 0$, we see that (13) is a tighter bound. Also note that there could be other GARs for the asynchronous problem to achieve better efficiency or further reduce computational overhead.

[1]The definition of CGE is the following: in each iteration t, sort m vectors g_j^t's as $\left\|g_{i_1}^t\right\| \leq \left\|g_{i_2}^t\right\| \leq \ldots \leq \left\|g_{i_m}^t\right\|$, then we have CGE gradient filter:

$$\mathrm{GradFilter}_{\mathrm{CGE}}(g_j^t | j \in S^t; m, f) = \sum_{l=1}^{m-f} g_{i_l}^t. \tag{10}$$

For asynchronous distributed optimization problem, we can also utilize *stale gradients*, i.e., gradients from previous iterations to further reduce waiting time. We define the following two sets: during iteration t, $S^{t;k}$ denotes the set of agents from whom the server *receives* gradients computed based on x^k, and $T^{t;k}$ denotes the set of agents from whom the most recent gradient that the server *has received* is computed using x^k. $T^{t;k}$ can be defined inductively: (i) $T^{t;t} = S^{t;t}$, and (ii) $T^{t;t-i} = S^{t;t-i} \setminus \bigcup_{k=0}^{i-1} T^{t;t-k}$, $\forall i \geq 1$. Note that the definition implies $T^{t;t-i} \cap T^{t;t-j} = \emptyset$ for any $i \neq j$. Let us further define $T^t = \bigcup_{i=0}^{\tau} T^{t;t-i}$, where $\tau \geq 0$ is a predefined *straggler parameter*. To use stale gradients, we define the GAR in the iterative update (3) to be

$$\mathrm{GradAgg}\left(g_j^t | j \in S^t; n, f, r\right) = \sum_{i=0}^{\tau} \sum_{j \in T^{t;t-i}} g_j^{t-i}. \tag{14}$$

With the above GAR, suppose that there exists a $\tau \geq 0$ such that $\left|T^t\right| \geq n - r$ for all t, and the step sizes satisfy $\eta_t \geq \eta_{t+1}$ for all t, in addition to conditions in Theorem 1, then Theorem 1 holds with the same parameters in (13). Intuitively, this method allows gradients of at most τ-iteration stale. This result indicates that the asymptotic resilience of the algorithm would not be affected by stale gradients. It can be expected that with less waiting time involved, the algorithm will be more efficient. Still, the convergence rate (number of iterations needed to converge) will be affected by τ when stale gradients are introduced.

4 RESILIENT STOCHASTIC DISTRIBUTED OPTIMIZATION

As mentioned briefly in Section 1, stochastic optimization is useful when the computation of full gradients is too expensive [6–8, 47], which is commonly used in various scenarios including machine learning [47]. So it is worthwhile for us to also examine the potential of utilizing $(f, r; \epsilon)$-redundancy for solving resilient stochastic distributed optimization problems.

Consider a distributed stochastic optimization problem on a d dimensional real-valued space $\mathbb{R}^d$. Each agent i has a *data generation distribution* $\mathcal{D}_i$ over $\mathbb{R}^m$. Each data point $z \in \mathbb{R}^m$ is a real-valued vector that incurs a *loss* defined by a *loss function* $\ell : (x; z) \mapsto \mathbb{R}$. The *expected loss function* for each agent i can be defined as

$$Q_i(x) = \mathbb{E}_{z \sim \mathcal{D}_i} \ell(x; z), \quad \text{for all } x \in \mathbb{R}^d. \tag{15}$$

With this problem formulation above, the gradient-based Algorithm 1 proposed in Section 3 can also be adapted to a stochastic version for solving various problems including resilient distributed machine learning (RDML).

Naturally, the goal of resilient distributed stochastic optimization is also (2), the same as RDO. For machine learning problems, the machine learning model Π can be parameterized as a d-dimensional vector $x \in \mathbb{R}^d$, which is the optimization target vector.

We first briefly revisit the computation of stochastic gradients in a distributed optimization system. To compute a stochastic gradient in iteration t, a (non-faulty) agent i samples k i.i.d. data points $z_{i_1}^t, \ldots, z_{i_k}^t$ from its distribution $\mathcal{D}_i$ and computes

$$g_i^t = \frac{1}{k} \sum_{i=1}^{k} \nabla \ell(x^t, z_{i_j}^t), \tag{16}$$

where k is referred to as the *batch size*. A faulty agent i sends an arbitrary vector g_i^t.

We will also use the following notations in this section. Suppose faulty agents (if any) in the system are *fixed* during a certain execution. For each non-faulty agent i, let $z_i^t = \left\{z_{i_1}^t, ..., z_{i_k}^t\right\}$ denote the collection of k i.i.d. data points sampled by agent i at iteration t. For each agent i and iteration t, we define a random variable

$$\zeta_i^t = \begin{cases} z_i^t, & \text{agent } i \text{ is non-faulty,} \\ g_i^t, & \text{agent } i \text{ is faulty.} \end{cases} \tag{17}$$

Recall that g_i^t can be an arbitrary d-dimensional random variable for each Byzantine faulty agent i. For each iteration t, let $\zeta^t = \left\{\zeta_i^t,\ i = 1, ..., n\right\}$, and let $\mathbb{E}_t$ denote the expectation with respect to the collective random variables $\zeta^0, ..., \zeta^t$, given the initial estimate x^0. Specifically,

$$\mathbb{E}_t(\cdot) = \mathbb{E}_{\zeta^0,...,\zeta^t}(\cdot),\ \forall t \geq 0. \tag{18}$$

Similar to Section 3, we make standard Assumptions 1 and 2. We also need an extra assumption to bound the variance of stochastic gradients from all non-faulty agents.

Assumption 3. *For each non-faulty agent i, assume that the variance of g_i^t is bounded. Specifically, there exists a finite real value σ such that for each non-faulty agent i,*

$$\mathbb{E}_{\zeta_i^t} \left\| g_i^t - \mathbb{E}_{\zeta_i^t}\left(g_i^t\right) \right\|^2 \leq \sigma^2. \tag{19}$$

We make no assumption over the behavior of Byzantine agents.

Suppose $\mathcal{H} \subseteq [n]$ is a subset of non-faulty agents with $|\mathcal{H}| = n - f$, and a solution $x_{\mathcal{H}}$ exists in $\mathcal{W}$. The following theorem shows the general results for Resilient distributed stochastic optimization problems. Note that here Algorithm 1 uses stochastic gradients instead of full gradients.

Theorem 2. *Consider Algorithm 1 with stochastic gradients, and the GAR in use is the CGE gradient filter* (10)*. Suppose Assumptions 1, 2, and 3 hold true, the expected cost functions of non-faulty agents satisfy $(f, r; \epsilon)$-redundancy, a resilience margin $\alpha > 0$, and the step size in* (3)*, $\eta_t = \eta > 0$ for all t. Let M denote an error-related margin. There exists an $\overline{\eta}$ such that, for $\eta < \overline{\eta}$, the following holds true:*

- *The value of a convergence rate parameter ρ satisfies $0 \leq \rho < 1$, and*
- *Given the initial estimate x^0 arbitrarily chosen from $\mathbb{R}^d$, for all $t \geq 0$,*

$$\mathbb{E}_t \left\| x^{t+1} - x_{\mathcal{H}} \right\|^2 \leq \rho^{t+1} \left\| x^0 - x_{\mathcal{H}} \right\|^2 + \frac{1 - \rho^{t+1}}{1 - \rho} \mathsf{M}. \tag{20}$$

Let $\Gamma = \max_{x \in \mathcal{W}} \left\| x - x_{\mathcal{H}} \right\|$, we have the following parameters when $f \geq 0$ and $r \geq 0$ for Theorem 2:

$$\begin{aligned} \alpha &= \frac{n-f}{n-r} - \frac{f+r}{n-r} \cdot \frac{2\mu}{\gamma}, \\ \overline{\eta} &= \frac{2(n-r)\gamma\alpha}{(n-r-f)^2\mu^2}, \\ \rho &= 1 - 2(n-f)\eta\gamma + 4(f+r)\eta\mu + (n-r-f)^2\eta^2\mu^2, \\ \mathsf{M} &= 4(n-r)\eta\mu\epsilon \left(2(f+r) + (n-r-f)^2\eta\mu\right)\Gamma \\ &\quad + 4(n-r)^2(n-r-f)^2\eta^2\mu^2\epsilon^2 \\ &\quad + 2(f+r)\eta\sigma\Gamma + (n-r-f)^2\eta^2\sigma^2. \end{aligned} \tag{21}$$

Note that we also need $n \geq 2f + 3r/2$ to guarantee that $\rho \geq 0$.

4.1 Special cases of resilient distributed stochastic optimization

Similar to what we have in Section 3, the stochastic version of resilient optimization also has several meaningful special cases. When $f = r = 0$, the problem degenerates to the synchronous distributed stochastic optimization with no faulty agents in the system. The problem has also been solved by various methods without redundancy, including D-SGD with convexity assumptions which is commonly used in machine learning [33].

4.1.1 Stochastic Byzantine optimization. When $r = 0$, the problem becomes the Byzantine fault-tolerant problem. CGE gradient filter (10) degenerates to averaging (12). Theorem 2 holds with $(f, 0; \epsilon)$-redundancy and the same parameters as in (21), substituting with $r = 0$.

4.1.2 Stochastic asynchronous optimization. When $f = 0$, our problem degenerates to the asynchronous problem. Without faulty agents, the goal becomes (1). We also have $\mathcal{H} = [n]$. Note that when $f = 0$, the gradient filter CGE degenerates to the GAR (12), i.e., plain summation of received stochastic gradients. Theorem 2 holds with $(0, r; \epsilon)$-redundancy and the following parameters:

$$\begin{aligned} \alpha &= 1 - \frac{r}{n} \cdot \frac{\mu}{\gamma}, \qquad \overline{\eta} = \frac{2n\gamma\alpha}{(n-r)^2\mu^2}, \\ \rho &= 1 - 2(n\gamma - r\mu)\eta + (n-r)^2\eta^2\mu^2, \\ \mathsf{M} &= 4n\eta\mu\epsilon \left(r + (n-r)^2\eta\mu\right)\Gamma \\ &\quad + 4n^2(n-r)^2\eta^2\mu^2\epsilon^2 + (n-r)^2\eta^2\sigma^2. \end{aligned} \tag{22}$$

Note that the requirement of $n \geq 2f + 3r/2$ is not needed here. We can obtain the result for this case by substituting $f = 0$ in (21) as well. However, this bound M in (22) is better than (21) with $f = 0$.

4.2 Discussion

The results provided above indicate that with $(f, r; \epsilon)$-redundancy, there exist algorithms to approximate the true solution to (1) or (2) with D-SGD, where *linear convergence* is achievable, and the error range of that approximation is, in expectation, proportional to ϵ and σ. Specifically, in (20) when $t \to \infty$,

$$\lim_{t \to \infty} \mathbb{E}_t \left\| x^{t+1} - x_{\mathcal{H}} \right\|^2 \leq \frac{1}{1 - \rho} \mathsf{M}, \tag{23}$$

where M changes monotonically as ϵ and σ.

Note that Gupta et al. [18] showed a special case of the Byzantine fault-tolerant problem with redundancy, where all agents have the same data distribution. Our results can be applied to a broader range of problems, including heterogeneous problems like *federated learning* [27].

5 EMPIRICAL STUDIES

In this section, we empirically show the effectiveness of our scheme in Algorithm 1 on some numerical examples and benchmark distributed machine learning tasks.

5.1 Numerical examples

We present a group of simulation results applying our algorithm framework to the problems of *distributed linear regression*. Specifically, we study behavior of Algorithm 1 according to our theoretical results in Section 3, including the special cases.

Consider a synchronous server-based system, where $n = 10$ and $d = 2$. Each agent $i \in \{1, ..., n\}$ has a data point represented by a triplet (A_i, B_i, N_i) where A_i is a d-dimensional row vector with $d = 2$, $B_i \in \mathbb{R}$ is a response, and $N_i \in \mathbb{R}$ is a noise value. Specifically, for all $i \in \{1, ..., n\}$, $n = 10$, $B_i = A_i x^* + N_i$ where $x^* = (1, 1)^T$. The collective data is represented by a triplet of matrices (A, B, N) where the i-th row of A, B, and N are equal to A_i, B_i and N_i, respectively. Each agent i has a quadratic cost function defined to be $Q_i(x) = (B_i - A_i x)^2, \forall x \in \mathbb{R}^2$.

In this group of experiments, we choose $f = 0, 1, 2$ and $r = 0, 1, 2$, with various combinations. In each execution, we always designate the first f agents to be Byzantine faulty (labeled 1 through f). Stragglers are actual stragglers in each iteration whose gradients are the last r ones received by the server. It can be verified first that the agents' cost functions satisfy $(f, r; \epsilon)$-redundancy property. The values of ϵ with different f and r are presented in Figure 1. We can see that larger the values of f and r, larger the value of ϵ, and less accurate the algorithm can we expect. We can also compute the values of the coefficients accordingly: $\mu = 2$; $\gamma = 0.788, 0.588, 0.439$ when $f = 0, 1, 2$, respectively.

We use the following parameters implementing the algorithm. In update rule (3), we use step size $\eta_t = 1.5/(t + 1)$ for iteration $t = 0, 1, ...$, which satisfies $\sum_{t=0}^{\infty} \eta_t = \infty$ and $\sum_{t=0}^{\infty} \eta_t^2 < \infty$. We assume the convex compact $\mathcal{W} \subset \mathbb{R}^d$ to be a 2-dimensional hypercube $[-1000, 1000]^2$. Note that $x_{\mathcal{H}} \in \mathcal{W}$. In all simulations presented, the initial estimate $x^0 = (0, 0)^T$. In every execution, we observe that the iterative estimates produced by the algorithm practically converge after 400 iterations. Thus, to measure the approximate result outputted by the algorithm, we use $x_{\text{out}} = x^{500}$.

Two types of faulty behaviors are simulated: (i) *gradient-reverse*: each faulty agent *reverses* its true gradient. Suppose the correct gradient of a faulty agent i at iteration t is s_i^t, the agent i sends the incorrect gradient $g_i^t = -s_i^t$ to the server. (ii) *random*: the faulty agent sends a randomly chosen vector in $\mathbb{R}^d$. In our experiments, the faulty agent in each iteration chooses i.i.d. Gaussian random vector with mean 0 and a isotropic covariance matrix with standard deviation of 200.

The outputs for with different values of f and r are presented in Table 2 alongside with plain gradient descent output ($f = r = 0$) for comparison. Note that $\text{dist}\left(x_{\mathcal{H}}, x_{\text{out}}\right) = \left\|x_{\mathcal{H}} - x_{\text{out}}\right\|$. In all executions, $\text{dist}\left(x_{\mathcal{H}}, x_{\text{out}}\right) < \text{D}^*$ as indicated by Theorem 1. The values of D^* are listed below.

	$r = 0$	$r = 1$	$r = 2$
$f = 0$	0	0.207	0.385
$f = 1$	0.369	0.670	0.957
$f = 2$	1.251	1.748	2.467

We also plot the processes of Algorithm 1 solving the numerical examples of some of these experiments, namely the cases when $f = 1$ and 2, in Figure 2, with details of the first 150 iterations of each executions. The process when $f = r = 0$, i.e., the synchronous fault-free case is also presented for comparison purpose. Theses plots show that in order to be resilient against Byzantine agents and stragglers, the convergence speed is slightly slowed down and there exists a gap between the algorithm's output and the true solution.

5.2 Machine learning experiments

Now we examine some more complex optimization problems: resilient distributed machine learning with deep neural networks. Here, we use stochastic version of our Algorithm 1, validating theoretical results in Section 4. It is worth noting that even though the actual values of redundancy parameter ϵ are difficult to compute, through the following results we can still see that the said redundancy property exists in real-world scenarios and it supports our algorithm to achieve its applicability.

We simulate a server-based distributed learning system using multiple threads, one for the server, the rest for agents, with inter-thread communications handled by message passing interface. The simulator is built in Python with PyTorch [40] and MPI4py [13], and deployed on a virtual machine with 14 vCPUs and 16 GB memory.

The experiments are conducted on three benchmark image-classification datasets:

- MNIST [6] of monochrome handwritten digits,
- Fashion-MNIST [53] of grayscale images of clothes, and
- KMNIST [12] of monochrome Japanese Hiragana characters.

Each of the above datasets comprises of 60,000 training and 10,000 testing data points in 10 non-overlapping classes. For each dataset, we train a benchmark neural network LeNet [31] with 431,080 learnable parameters. Data points are divided among agents such that each agent gets 2 out of 10 classes, and each class appears in 4 agents; $\mathcal{D}_i$ for each agent i is unique.

In each of our experiments, we simulate a distributed system of $n = 20$ agents with $f = 3$ and different values of $r = 0, 1, 3, 5, 10$. Note that when $r = 0$, the problem becomes synchronous Byzantine fault-tolerant learning. In each execution, we always designate the first f agents to be Byzantine faulty. Stragglers are agents in each iteration whose gradients are the last r ones received by the server. We also compare these results with the fault-free synchronous learning ($f = r = 0$). We choose batch size $b = 128$ for D-SGD, and fixed step size $\eta = 0.01$. Performance of algorithms is measured by model accuracy at each step. We also document the cumulative communication time of each setting. Communication time of iteration t is the time from the server's sending out of x^t to its receiving of $n - r$ gradients. Experiments of each setting are run 4 times with different random seeds[2], and the averaged performance is reported. The results are shown in Figure 3. We show the first 1,000 iterations as there is a clear trend of converging by the end of 1,000 iterations for both tasks in all four settings.

[2]Randomnesses exist in drawing of data points and stragglers in each iteration of each execution.

Table 2: Outputs of Algorithm 1 when solving distributed linear regression in Section 5.1. GAR in use: summing $n - r$ gradients when $f = 0$, and CGE gradient filter when $f = 1, 2$.

		$r = 0$	$r = 1$	$r = 2$	$x_{\mathcal{H}}$
$f = 0$	x_{out}	$(1.0117, 0.9883)^T$	$(1.0152, 0.9891)^T$	$(1.0311, 0.9872)^T$	$(1.0117, 0.9883)^T$
	dist $(x_{\mathcal{H}}, x_{\text{out}})$	~ 0	3.66×10^{-3}	1.95×10^{-2}	
$f = 1$ *grad-rev*	x_{out}	$(1.0460, 0.9883)^T$	$(1.0363, 0.9994)^T$	$(1.0346, 0.9934)^T$	$(1.0460, 0.9883)^T$
	dist $(x_{\mathcal{H}}, x_{\text{out}})$	1.48×10^{-5}	1.47×10^{-2}	1.24×10^{-2}	
$f = 1$ *random*	x_{out}	$(1.0459, 0.9883)^T$	$(1.0466, 0.9846)^T$	$(1.0403, 0.9923)^T$	
	dist $(x_{\mathcal{H}}, x_{\text{out}})$	5.42×10^{-5}	3.71×10^{-3}	6.93×10^{-3}	
$f = 2$ *grad-rev*	x_{out}	$(1.0067, 0.9621)^T$	$(1.0138, 0.9789)^T$	$(0.9891, 1.0090)^T$	$(1.0445, 0.9876)^T$
	dist $(x_{\mathcal{H}}, x_{\text{out}})$	4.56×10^{-2}	3.19×10^{-2}	5.94×10^{-2}	
$f = 2$ *random*	x_{out}	$(1.0444, 0.9876)^T$	$(1.0376, 0.9961)^T$	$(1.0296, 1.0030)^T$	
	dist $(x_{\mathcal{H}}, x_{\text{out}})$	6.00×10^{-5}	1.10×10^{-2}	2.14×10^{-2}	

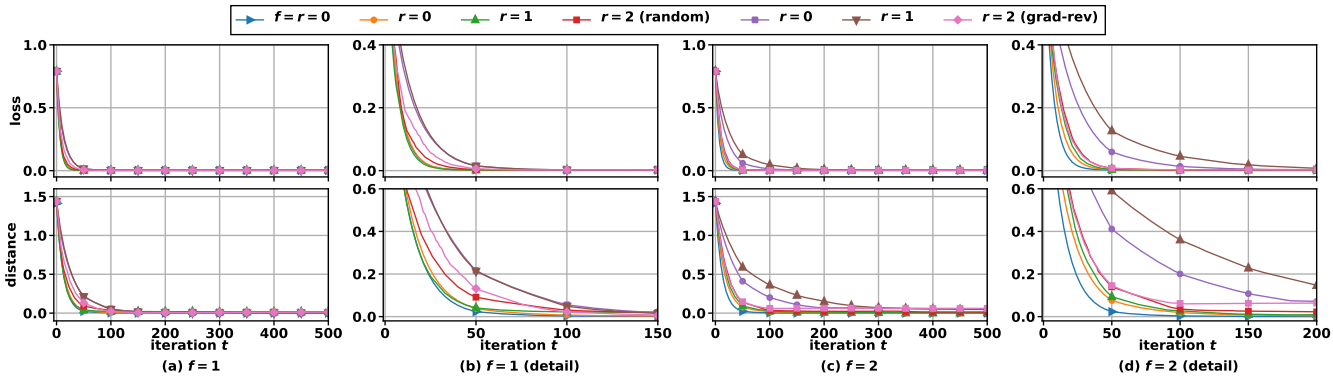

Figure 2: Changes in aggregate cost $\sum_{i \in \mathcal{H}} Q_i(x^t)$, and dist $(x^t, x_{\mathcal{H}})$, versus the number of iterations with Algorithm 1 solving the numerical example.

For Byzantine agents, we evaluate two types of faults: *reverse-gradient* where faulty agents reverse the direction of its true gradients, and *label-flipping* where faulty agents label data points of class i as class $9 - i$.

As is shown in the first row of Figure 3, the learned model reaches comparable accuracy to the one learned by synchronous algorithm at the same iteration; there is a gap between them and the fault-free case, echoing the error bound M. In the second row of Figure 3, we see that by dropping out r stragglers, the communication time is gradually reduced when increasing the value of r.

5.3 Algorithm 1 for practical use

It is worth noting that f and r as parameters of Algorithm 1 bear different meanings as f and r in the $(f, r; \epsilon)$-redundancy property. In the redundancy property, f and r together with ϵ describe how redundant the group of cost functions is. For the algorithm, f and r are hyperparameters set by users and indicate the numbers of faulty agents and stragglers we *intend* to tolerate; these two values are not necessarily equal to the actual numbers in the system, which we may never know. Still, it is possible to estimate the value of ϵ, and based on the theorems in Section 3 and 4, to estimate how close the output can be expected to be to the true solution.

Like other hyper-parameters in optimization problems, there is no golden standard in choosing the values of f and r. It would be better that f and r are close to the real numbers of faulty agents and stragglers in the system, which can be estimated by, for example, previous logs and statistics of the distributed system in use, to see how often are there failure of agents or how long of waiting can be qualified as stragglers.

Another interesting observation is that the algorithm does not require the Byzantine agents or stragglers to be "*fixed*". In practice, it might be more common that the stragglers change from time to time during the training process, while malicious agents remains the same. Nonetheless, the gradient-based nature of our algorithm allows both of them to be changing; as long as the total number of them are bounded by f and r, the algorithm remains valid.

6 SUMMARY

We studied the impact of $(f, r; \epsilon)$-redundancy in cost functions on resilient distributed optimization and machine learning. Specifically, we presented an algorithm for resilient distributed optimization and learning, and analyzed its convergence when agents' cost functions have $(f, r; \epsilon)$-redundancy – a generic characterization of redundancy in cost functions. We examined the resilient optimization and learning problem space. We showed that, under $(f, r; \epsilon)$-redundancy, Algorithm 1 with DGD achieves $(f, r; O(\epsilon))$-resilience for optimization, and Algorithm 1 with D-SGD can solve resilient distributed stochastic optimization problems with error margins proportional to ϵ. We presented empirical results showing efficacy of Algorithm 1 solving resilient distributed linear regression

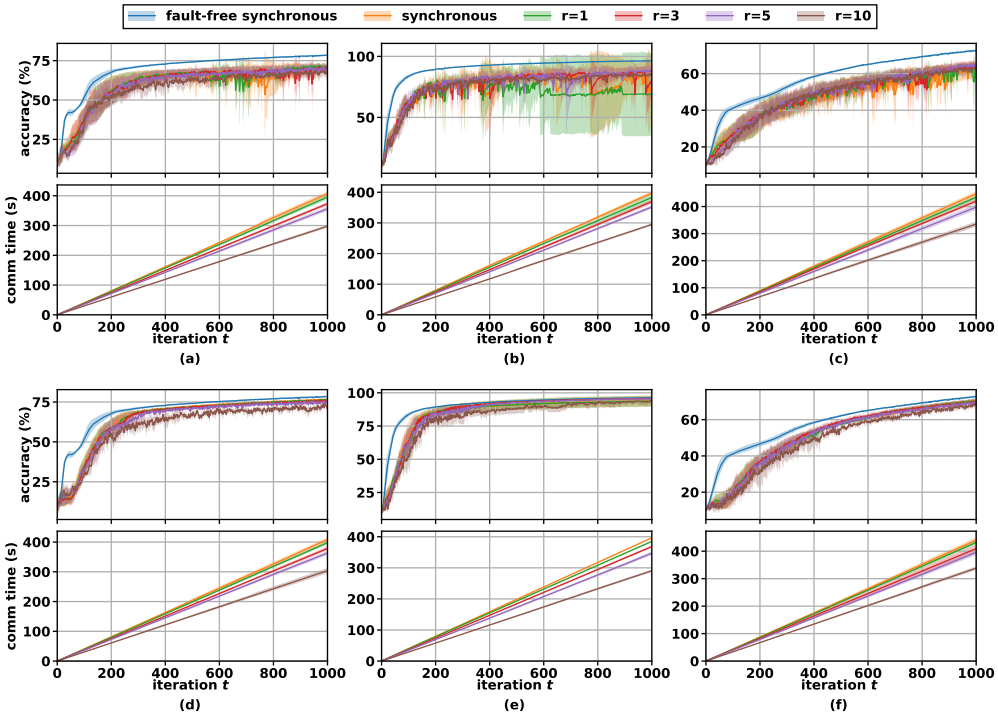

Figure 3: Results for resilient distributed learning with $n = 20$ and $f = 3$ using Algorithm 1 with D-SGD and CGE gradient filter. Datasets: (a)(d) Fashion-MNIST, (b)(e) MNIST, (c)(f) KMNIST. Byzantine faults: (a-c) *reverse-gradient*, (d-f) *label-flipping*. Solid line for average; shade for standard deviation. Communication time for fault-free synchronous cases is ommitted.

problems and resilient distributed learning tasks. Possible future work includes showing necessity of $(f, r; \epsilon)$-redundancy in solving RDO problems, analyzing the impact of redundancy in decentralized network, verifying our findings on larger, more complicated tasks, and so on.

Discussion on limitations Note that our results in this paper are proved under strongly-convex assumptions. One may argue that such assumptions are too strong to be realistic. However, previous research has pointed out that although not a global property, cost functions of many machine learning problems are strongly-convex in the neighborhood of local minimizers [8]. Also, there is a research showing that the results on non-convex cost functions can be derived from those on strongly-convex cost functions [2], and therefore our results can be applied to a broader range of real-world problems. Our empirical results showing efficacy of our algorithm also concur with this argument.

It is also worth noting that the approximation bounds in optimization problems are linearly associated with the number of agents n, and the error margins in learning problems are related to Γ, the size of $\mathcal{W}$. These bounds can be loose when n or Γ is large. We do note that in practice $\mathcal{W}$ can be arbitrary, for example, a neighborhood of local minimizers mentioned above, making Γ acceptably small. The value of ϵ can also be small in practice, as indicated by results in Section 5.

ACKNOWLEDGMENTS

Research reported in this paper was supported in part by the Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196, and by a Fritz fellowship from the Georgetown University. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government.

REFERENCES

[1] Alekh Agarwal and John C Duchi. 2012. Distributed delayed stochastic optimization. In *51st IEEE Conference on Decision and Control*. IEEE, 5451–5452.
[2] Zeyuan Allen-Zhu and Elad Hazan. 2016. Optimal black-box reductions between optimization objectives. *arXiv preprint arXiv:1603.05642* (2016).
[3] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. 2013. Effective straggler mitigation: Attack of the clones. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*. 185–198.
[4] Mahmoud Assran, Arda Aytekin, Hamid Reza Feyzmahdavian, Mikael Johansson, and Michael G Rabbat. 2020. Advances in Asynchronous Parallel and Distributed Optimization. *Proc. IEEE* 108, 11 (2020), 2013–2031.
[5] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 118–128.
[6] Léon Bottou. 1998. Online learning and stochastic approximations. *On-line learning in neural networks* 17, 9 (1998), 142.
[7] Léon Bottou and Olivier Bousquet. 2008. The Tradeoffs of Large Scale Learning. In *Advances in Neural Information Processing Systems 20 (NIPS 2007)*. NIPS Foundation, 161–168.
[8] Léon Bottou, Frank E Curtis, and Jorge Nocedal. 2018. Optimization methods for large-scale machine learning. *Siam Review* 60, 2 (2018), 223–311.

[9] Stephen Boyd, Neal Parikh, and Eric Chu. 2011. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc.

[10] Lingjiao Chen, Hongyi Wang, Zachary Charles, and Dimitris Papailiopoulos. 2018. Draco: Byzantine-resilient distributed training via redundant gradients. In *International Conference on Machine Learning*. PMLR, 903–912.

[11] Yudong Chen, Lili Su, and Jiaming Xu. 2017. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1, 2 (2017), 1–25.

[12] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. 2018. Deep Learning for Classical Japanese Literature. *arXiv preprint arXiv:1812.01718* (2018).

[13] Lisandro D Dalcin, Rodrigo R Paz, Pablo A Kler, and Alejandro Cosimo. 2011. Parallel distributed computing using Python. *Advances in Water Resources* 34, 9 (2011), 1124–1139.

[14] Georgios Damaskinos, El Mahdi El Mhamdi, Rachid Guerraoui, Arsany Hany Abdelmessih Guirguis, and Sébastien Louis Alexandre Rouault. 2019. Aggregathor: Byzantine machine learning via robust gradient aggregation. In *The Conference on Systems and Machine Learning (SysML), 2019*.

[15] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. 2014. SAGA: A fast incremental gradient method with support for non-strongly convex composite objectives. *arXiv preprint arXiv:1407.0202* (2014).

[16] Hamid Reza Feyzmahdavian, Arda Aytekin, and Mikael Johansson. 2016. An asynchronous mini-batch algorithm for regularized stochastic optimization. *IEEE Trans. Automat. Control* 61, 12 (2016), 3740–3754.

[17] Kristen Gardner, Samuel Zbarsky, Sherwin Doroudi, Mor Harchol-Balter, and Esa Hyytia. 2015. Reducing latency via redundant requests: Exact analysis. *ACM SIGMETRICS Performance Evaluation Review* 43, 1 (2015), 347–360.

[18] Nirupam Gupta, Shuo Liu, and Nitin Vaidya. 2021. Byzantine Fault-Tolerant Distributed Machine Learning with Norm-Based Comparative Gradient Elimination. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 175–181.

[19] Nirupam Gupta and Nitin H Vaidya. 2020. Resilience in collaborative optimization: redundant and independent cost functions. *arXiv preprint arXiv:2003.09675* (2020).

[20] Wael Halbawi, Navid Azizan, Fariborz Salehi, and Babak Hassibi. 2018. Improving distributed gradient descent using reed-solomon codes. In *2018 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2027–2031.

[21] Robert Hannah and Wotao Yin. 2017. More iterations per second, same quality–why asynchronous algorithms may drastically outperform traditional ones. *arXiv preprint arXiv:1708.05136* (2017).

[22] Rie Johnson and Tong Zhang. 2013. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems* 26 (2013), 315–323.

[23] Can Karakus, Yifan Sun, and Suhas Diggavi. 2017. Encoded distributed optimization. In *2017 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2890–2894.

[24] Can Karakus, Yifan Sun, Suhas Diggavi, and Wotao Yin. 2017. Straggler mitigation in distributed optimization through data encoding. *Advances in Neural Information Processing Systems* 30 (2017), 5434–5442.

[25] Can Karakus, Yifan Sun, Suhas Diggavi, and Wotao Yin. 2019. Redundancy techniques for straggler mitigation in distributed optimization and learning. *The Journal of Machine Learning Research* 20, 1 (2019), 2619–2665.

[26] Sai Praneeth Karimireddy, Lie He, and Martin Jaggi. 2021. Learning from history for byzantine robust optimization. In *International Conference on Machine Learning*. PMLR, 5311–5319.

[27] Jakub Konečnỳ, Brendan McMahan, and Daniel Ramage. 2015. Federated optimization: Distributed optimization beyond the datacenter. *arXiv preprint arXiv:1511.03575* (2015).

[28] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* 4, 3 (1982), 382–401.

[29] John Langford, Alexander Smola, and Martin Zinkevich. 2009. Slow learners are fast. *arXiv preprint arXiv:0911.0491* (2009).

[30] Rémi Leblond. 2018. *Asynchronous optimization for machine learning*. Ph. D. Dissertation. PSL Research University.

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[32] Kangwook Lee, Maximilian Lam, Ramtin Pedarsani, Dimitris Papailiopoulos, and Kannan Ramchandran. 2017. Speeding up distributed machine learning using codes. *IEEE Transactions on Information Theory* 64, 3 (2017), 1514–1529.

[33] Mu Li, David G Andersen, Alexander J Smola, and Kai Yu. 2014. Communication efficient distributed machine learning with the parameter server. *Advances in Neural Information Processing Systems* 27 (2014), 19–27.

[34] Shuo Liu. 2021. A Survey on Fault-tolerance in Distributed Optimization and Machine Learning. *arXiv preprint arXiv:2106.08545* (2021).

[35] Shuo Liu, Nirupam Gupta, and Nitin H. Vaidya. 2021. Approximate Byzantine Fault-Tolerance in Distributed Optimization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing* (Virtual Event, Italy) *(PODC'21)*. Association for Computing Machinery, New York, NY, USA, 379–389.

[36] Shuo Liu, Nirupam Gupta, and Nitin H. Vaidya. 2022. Impact of Redundancy on Resilience in Distributed Optimization and Learning. *arXiv preprint arXiv:2211.08622* (2022).

[37] Angelia Nedic and Asuman Ozdaglar. 2009. Distributed subgradient methods for multi-agent optimization. *IEEE Trans. Automat. Control* 54, 1 (2009), 48–61.

[38] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J Wright. 2011. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *arXiv preprint arXiv:1106.5730* (2011).

[39] Daniel W Otter, Julian R Medina, and Jugal K Kalita. 2020. A survey of the usages of deep learning for natural language processing. *IEEE Transactions on Neural Networks and Learning Systems* 32, 2 (2020), 604–624.

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703* (2019).

[41] Michael Rabbat and Robert Nowak. 2004. Distributed optimization in sensor networks. In *Proceedings of the 3rd international symposium on Information processing in sensor networks*. 20–27.

[42] Robin L Raffard, Claire J Tomlin, and Stephen P Boyd. 2004. Distributed optimization for cooperative agents: Application to formation flight. In *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, Vol. 3. IEEE, 2453–2459.

[43] Nicolas Le Roux, Mark Schmidt, and Francis Bach. 2012. A stochastic gradient method with an exponential convergence rate for finite training sets. *arXiv preprint arXiv:1202.6258* (2012).

[44] Nihar B Shah, Kangwook Lee, and Kannan Ramchandran. 2015. When do redundant requests reduce latency? *IEEE Transactions on Communications* 64, 2 (2015), 715–722.

[45] Shai Shalev-Shwartz and Tong Zhang. 2013. Accelerated mini-batch stochastic dual coordinate ascent. *arXiv preprint arXiv:1305.2581* (2013).

[46] Wei Shi, Qing Ling, Gang Wu, and Wotao Yin. 2015. Extra: An exact first-order algorithm for decentralized consensus optimization. *SIAM Journal on Optimization* 25, 2 (2015), 944–966.

[47] S. Sra, S. Nowozin, and S.J. Wright. 2012. *Optimization for Machine Learning*. MIT Press.

[48] Lili Su and Nitin Vaidya. 2015. Byzantine multi-agent optimization: Part I. *arXiv preprint arXiv:1506.04681* (2015).

[49] Lili Su and Nitin H Vaidya. 2016. Fault-tolerant multi-agent optimization: optimal iterative distributed algorithms. In *Proceedings of the 2016 ACM symposium on principles of distributed computing*. 425–434.

[50] Rashish Tandon, Qi Lei, Alexandros G Dimakis, and Nikos Karampatziakis. 2017. Gradient coding: Avoiding stragglers in distributed learning. In *International Conference on Machine Learning*. PMLR, 3368–3376.

[51] Damiano Varagnolo, Filippo Zanella, Angelo Cenedese, Gianluigi Pillonetto, and Luca Schenato. 2015. Newton-Raphson consensus for distributed convex optimization. *IEEE Trans. Automat. Control* 61, 4 (2015), 994–1009.

[52] Da Wang, Gauri Joshi, and Gregory Wornell. 2015. Using straggler replication to reduce latency in large-scale parallel computing. *ACM SIGMETRICS Performance Evaluation Review* 43, 3 (2015), 7–11.

[53] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747* (2017).

[54] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. 2018. Zeno: Byzantine-suspicious stochastic gradient descent. *arXiv preprint arXiv:1805.10032* 24 (2018).

[55] Neeraja J Yadwadkar, Bharath Hariharan, Joseph E Gonzalez, and Randy Katz. 2016. Multi-task learning for straggler avoiding predictive job scheduling. *The Journal of Machine Learning Research* 17, 1 (2016), 3692–3728.

[56] Yaoqing Yang, Pulkit Grover, and Soummya Kar. 2017. Coded distributed computing for inverse problems. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 709–719.

[57] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. 2018. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*. PMLR, 5650–5659.