A Survey on Fault-tolerance in Distributed Optimization and Machine Learning

Shuo Liu
Department of Computer Science
Georgetown University
Washington DC, USA
sl1539@georgetown.edu

Abstract

The robustness of distributed optimization is an emerging field of study, motivated by various applications of distributed optimization including distributed machine learning, distributed sensing, and swarm robotics. With the rapid expansion of the scale of distributed systems, resilient distributed algorithms for optimization are needed, in order to mitigate system failures, communication issues, or even malicious attacks. This survey investigates the current state of fault-tolerance research in distributed optimization, and aims to provide an overview of the existing studies on both fault-tolerant distributed optimization theories and applicable algorithms.

Contents

1 Introduction

With rapid development in communication, sensing and learning systems, development in computation and storage capacity of computer systems, and growth in data collection, the problems in networked systems have gained significant attention [29, 39, 89, 122, 126]. A *networked system*[1] typically consists of a number of *agents*, which work collaboratively to achieve certain global objective. Many problems in networked systems can be solved in the framework of optimization [10]. The distributed nature of these problems and performance limitations of centralized strategies lead to increasing interest in distributed approaches to solve the optimization problems [10, 80].

Among all the topics in researches of distributed systems, the robustness of networked systems received recent research attention [30, 59, 71]. People have realized, in both theory and practice, that agent failures or adversarial behaviors of some agents may render non-robust distributed algorithms useless. In the context of distributed optimization, the problem of *fault-tolerance*, sometimes also *resilience* or *robustness*, becomes increasingly important. People wish to solve optimization problems distributed, while also being able to counter agent failures, communication issues, malicious attackers, etc., while utilizing computational power from various sources [97–101, 103]. This survey intends to study the current state of researches in fault-tolerant distributed optimization problems.

1.1 Distributed Optimization

Generally, an optimization problem intends to find optima (maximum or minimum point(s)) of a given objective function [37]. In the problem of distributed optimization, we consider a multi-agent system (or *network*) of n agents, and each agent i has a local cost function $Q_i(x)$, where $x \in \mathbb{R}^d$ is the optimization variable, a d-dimensional vector of real values. The objective of distributed optimization is to minimize a global objective function, which is the aggregation of cost functions of all agents:

$$\min_{x\in\mathbb{R}^d} \sum_{i=0}^{n} Q_i(x), \tag{1}$$

while the algorithm is typically in a distributed manner, i.e., with local computation by the agents and communication among the agents. In practice, an algorithm of distributed optimization can also be required to output only one of the minimum points to the global objective function when there are multiple possible solutions, i.e., output a vector x^* such that

$$x^* \in \arg\min_{i\in\mathbb{R}^d} \sum_{i=0}^{n} Q_i(x). \tag{2}$$

The problem of optimization with multiple agents has been studied since the end of last century in the context of parallel and distributed computation [5, 108, 109].

[1] In this survey, without specification, the phrase *networked system* is interchangeable to *distributed system* and *multi-agent system*, whilst "network system" stresses on the communication network, "distributed system" stresses on the distributed nature of the task, and "multi-agent system" stresses on the number of members participating in the system.

In recent years it has gained renewed interest due to its applications in various fields, including communication networks, power systems, sensor networks, and machine learning [14, 39, 79, 80, 91]. Recent reviews of distributed optimization include following surveys and books: [10, 40, 77, 80, 89, 91, 119].

1.2 Fault-tolerance in distributed optimization

The studies on fault-tolerance start from the increasing need of robustness in distributed systems. In many applications of distributed system, the reliability of such a system can be affected by component failures, e.g., malfunctioning agents, communication difficulties, or malicious attacks, e.g., some agents intend to sabotage the collective effort by sending false information [6, 12, 15, 61]; therefore, the central goal regarding robustness of distributed systems is to make the automated systems able of making correct decisions in presence of faulty data [12]. In the context of general distributed systems, the two major tools for solving the problem algorithmically are *Byzantine agreement* [28, 62] and *sensor fusion* [20, 72]. Byzantine agreement allows a distributed decision-making system tolerate a certain number of failed agents, while sensor fusion enhances the accuracy of sensors (or agents) by deploying sensors redundantly.

In the context of distributed optimization, fault-tolerance is also gaining more attention for the same reason. Correspondingly, there are also two major lines of work. One intends to allow the algorithm to produce correct output in presence of a certain number of failed agents. This includes countering faulty agents regardless of their behavior, i.e., *Byzantine fault-tolerance*, or *Byzantine resilience* [45, 101, 104], or modeling and countering certain kinds of failures or adversarial behaviors [30, 59]. The other line of work intends to counter faulty agents by assigning redundant workloads to agents [18, 85].

There are different distributed optimization models considered by the fault-tolerance problem. Some analyses take the communication topology of the network into consideration [102, 104, 125], while other researches focus on one or two major system architectures [6, 19, 45]. Some papers consider only the case in which the cost functions are scalar-valued [101, 104]. Some works are interested in the fault-tolerant problems in specific distributed optimization tasks (e.g., distributed machine learning), or specific distributed optimization methods (e.g., distributed gradient descent or distributed stochastic gradient descent) [19, 49, 68].

This survey intends to investigate the current state of researches in the fault-tolerant distributed optimization problem. The rest of this survey is organized as follows. In Section 2 we revisit some related concepts and notations in graph theory and Byzantine consensus, the two related fields. In Section 3 we discuss the existing research in Byzantine fault-tolerance problem of distributed optimization, including the problem formulation, solvability, and algorithms. We also discuss the special case of Byzantine fault-tolerant distributed machine learning. We then discuss some other fault-tolerance problems in distributed optimization, including different adversary models, and possible combination of privacy-preservation and fault-tolerance in Section 4. Finally, we summarize this survey in Section 5 and

discuss possible future work.

2 Preliminaries

In this section, we introduce some backgrounds related to fault-tolerance in distributed optimization.

2.1 Graph theory

We revisit some basics in graph theory [41] that is also used in modeling and analysis of fault-tolerant distributed optimization algorithms. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a *directed graph* with the set of nodes (in our case, agents) $\mathcal{V} = \{1, ..., n\}$ and the set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$. (i, j) denotes a directed edge from node i to node j. A graph becomes *undirected* if and only if $(i, j) \in \mathcal{E}$ always implies $(j, i) \in \mathcal{E}$. For simplicity, we assume in this survey that no directed graph contains self loop, i.e., $(i, i) \notin \mathcal{E}$ for all $i \in \mathcal{V}$, unless specified otherwise. The *in-neighbors* of node i consists of all nodes that can transmit information to node i directly, denoted by the set $\mathcal{N}_i^{\text{in}} = \{j \in \mathcal{V} | (j, i) \in \mathcal{E}\}$. In contrast, the *out-neighbors* of node i consists of all nodes that i can transmit information to directly, denoted by the set $\mathcal{N}_i^{\text{out}} = \{j \in \mathcal{V} | (i, j) \in \mathcal{E}\}$. For undirected graphs, we denote $\mathcal{N}_i = \mathcal{N}_i^{\text{in}} = \mathcal{N}_i^{\text{out}}$ as the *neighbors* of node i.

A directed *path* from node i_1 to i_k is a sequence of nodes $\{i_1, ..., i_k\}$ such that $(i_j, i_{j+1}) \in \mathcal{E}$ for $j = 1, ..., k-1$; and if such a path exists, we say i_k is reachable from i_1. An undirected graph is *connected* if for any $i, j \in \mathcal{V}$, $i \neq j$, there exists a path between i and j. A directed graph is *strongly connected* if every node is reachable from every other node. The vertex cut $S \subset \mathcal{V}$ of a connected graph of nodes that by removing the nodes in S and edges connected to them, the graph becomes unconnected. The (vertex) *connectivity* of a connected graph is the size of the graph's minimum vertex cut. A *source component* in a directed graph is a subset of nodes, in which each node has a path to every other nodes in the graph. A *connected dominating set* of an undirected connected graph is the set of nodes in which every node not in the set has a neighbor in the set, and the set of nodes with their edges forms a connected subgraph itself.

A communication network may also be modeled with time-varying feature, denoted by $\mathcal{G}(t) = (\mathcal{V}, \mathcal{E}(t))$, where the edge set can change over time. Path, connectivity, and neighborhood sets defined above can also be adapted to this model with the time stamp t.

2.2 Byzantine consensus

Although Byzantine fault-tolerant distributed optimization is proposed quite recently [97], the question of Byzantine consensus has been proposed and well studied for some time [28, 62, 63, 70, 81]. It would be useful for us to revisit the concepts of consensus and Byzantine consensus, the two basic problems in distributed computing.

In the original *consensus* problem (sometimes *agreement* problem) in a multi-agent system, each of the n agents starts with an input of either 0 or 1. By communications between agents, the goal is for all the agents to eventually decide on a value in $\{0, 1\}$ that satisfies the following conditions:

Agreement No two agents decide on different values;
Validity If all agent start with the same value, they decide on that value, and
Termination All agents eventually decide.

Intuitively, a consensus protocol allows a group of agents in a multi-agent system to agree on a single value, after certain amount of communications.

The *Byzantine consensus* problem expands the original problem that instead of 0 or 1, each agent can now hold a value $v \in V$, where V is a set of allowed values. Also, some agents in the system can be non-compliant to the prescribed algorithm and exhibit arbitrary behavior, including starting in an arbitrary state, sending arbitrary message, and making arbitrary updates to its value. These agents are called *Byzantine faulty* agents. Under this setting, the goal is to satisfy the following correctness conditions:

Agreement No two non-faulty agents decide on different values;
Validity If all non-faulty agent start with the same value $v \in V$, they decide on the value v, and
Termination All non-faulty agents eventually decide.

The major change here comparing to plain consensus is that it is only reasonable to apply correctness conditions on all non-faulty agents in Byzantine consensus. Studies on Byzantine consensus include necessity and sufficiency conditions for solving the problem, and practical algorithms under different system architectures. Readers can refer to [70] for detailed introduction on Byzantine consensus.

The consensus problem can be further extended to consensus on multi-dimensional values (i.e., vectors) [111, 112], asymptotic or approximate consensus with reasonable definition of acceptable outcome [3, 38, 63, 107], and their corresponding Byzantine version [63, 74, 112]. Results in Byzantine consensus are important in research of Byzantine distributed optimization, since Byzantine optimization can in fact be viewed as a special case of Byzantine consensus [97].

2.3 Notations

We summarize the notations frequently used in this survey in Table 1 for reference. The table includes both notations already introduced and those that will be introduced in the following sections.

3 Byzantine fault-tolerant distributed optimization

The majority of current studies on the fault-tolerant distributed optimization problem assumes the faulty agents to be *Byzantine*, i.e., there is no assumption made

Table 1: Notations used in this survey.

Notation	Meaning
$\mathbb{R}$	The set of real numbers.
$\mathbb{R}^d$	The set of d-dimensional real-valued vectors.
$\mathbb{R}^{d\times n}$	The set of real-valued matrices with d rows and n columns.
$\mathbb{Z}$	The set of integers.
$\mathcal{G}(\mathcal{V},\mathcal{E})$	The graph $\mathcal{G}$, with node (agent) set $\mathcal{V}$ and edge set $\mathcal{E}$.
$(i,j)\in\mathcal{E}$	A directed edge from node i to node j in $\mathcal{E}$.
$\mathcal{N}_i^{\text{in}}$, $\mathcal{N}_i^{\text{out}}$, $\mathcal{N}_i$	In-, out-neighbors, and neighbors of node i.
$\mathcal{G}(t)=(\mathcal{V},\mathcal{E}(t))$	The graph $\mathcal{G}$ at time stamp t.
n	The number of agents in a distributed system.
f	The upper-bound of the number of faulty agents in a distributed system.
t	The iteration number in an iterative algorithm.
$x[k]$	The k-th element of a vector x.
$Q_i(x)$	The cost function of agent i.
v_i^t	Some vector v produced by agent i at time t.
$\boldsymbol{A}$, $\boldsymbol{A}_{ij}$	A matrix $\boldsymbol{A}$ and the element of $\boldsymbol{A}$ at position (i,j).
$\mathcal{B}$	The set of faulty (adversarial) agents.
$\mathcal{H}$	The set of non-faulty (honest) agents.
$z\sim\mathcal{D}$	A random variable z drawn from distribution $\mathcal{D}$.
$\lvert\cdot\rvert$	Absolute value or cardinality of a set.
$\lVert\cdot\rVert$	Some kind of norm.
$\langle\cdot,\cdot\rangle$	Inner product of two vectors.
$\mathbb{E}$	Expectation.
$\operatorname{dist}(\cdot,\cdot)$	Hausdorff distance between two sets (including points). Also see Appendix A.
$\nabla f(x)$	Derivative of a function $f(x)$.

on the behavior of faulty agents in the system. This problem is often referred to as the *Byzantine fault-tolerant*, *Byzantine-resilient*, or *Byzantine-robust* distributed optimization problem. The advantage of formulating the problem this way is obvious: by making no behavioral assumption on faulty agents, the applicability of the results is quite broad. It is also a reasonable choice, since in practice, the trusted parts often cannot predict what faulty agents would do.

3.1 Problem formulation

Su and Vaidya [97] formally proposed the Byzantine fault-tolerant distributed optimization problem of a sum of convex cost functions with real-valued scalar input and output in a complete communication graph. Recall in Section 1.1 that we discussed the formulation of distributed optimization, where each agent i in the system of n

agents has a cost function $Q_i(x)$, and the goal is to find a point x^* such that

$$x^* \in \arg\min_x \sum_{i=0}^{n} Q_i(x). \tag{3}$$

In presence of faulty-agents, it is impractical to achieve the goal stated in (2), since it is possible that the non-faulty agents can never know any information regarding cost functions of the faulty agents.

Suppose among the n agents in the system, up to f agents may be Byzantine faulty. Let $\mathcal{V} = \{1, ..., n\}$ denote the set of all agents, $\mathcal{B}$ denote the set of faulty agents, and $\mathcal{H} = \mathcal{V} - \mathcal{B}$ be the set of non-faulty agents. The problem assumes $|B| \leq f$. Also suppose $Q_i : \mathbb{R} \to \mathbb{R}$ for all i. Su and Vaidya [97] showed that the goal of finding a minimum point for the averaged cost functions of all non-faulty agents

$$\widetilde{x} \in \arg\min_{x \in \mathbb{R}} \frac{1}{|\mathcal{H}|} \sum_{i \in \mathcal{H}} Q_i(x). \tag{4}$$

is also impossible. However, it is possible to achieve the following goal,

$$\widetilde{x} \in \arg\min_{x \in \mathbb{R}} \sum_{i \in \mathcal{H}} \alpha_i Q_i(x),$$
$$\text{such that } \ \forall i \in \mathcal{H},\ \alpha_i \geq 0,\ \text{and } \sum_{i \in \mathcal{H}} \alpha_i = 1. \tag{5}$$

i.e., find a minimum point of a *convex combination* of non-faulty agents' cost functions. Ideally, we want all $\alpha_i = \frac{1}{|\mathcal{H}|}$, effectively goal (4). Therefore, we want to maximize the number of α_i's bounded away from 0. The authors proved in [97] that such a goal can be achieved with certain restrictions when defining "bounded away from 0", and the maximum achievable number of bounded away weights (i.e., α_i's) is $|\mathcal{H}| - f$. The results are further extended to arbitrary directed networks in [98].

We can easily generalize the ideal goal (4) to real-valued multivariate cost functions: suppose for every agent i, its cost function $Q_i : \mathbb{R}^d \to \mathbb{R}$, the goal becomes

$$\widetilde{x} \in \arg\min_{x \in \mathbb{R}^d} \frac{1}{|\mathcal{H}|} \sum_{i \in \mathcal{H}} Q_i(x). \tag{6}$$

Comparing to the scalar version, this problem has wider applicability. However, due to the impossibility of its scalar counterpart, solving (6) is not generally achievable either. Note that optimization goal (6) on the *averaged* cost functions is the same as the following

$$\widetilde{x} \in \arg\min_{x \in \mathbb{R}^d} \sum_{i \in \mathcal{H}} Q_i(x), \tag{7}$$

i.e., the *aggregated* cost functions, since a positive scaling coefficient does not affect the optimum.

For clarification of the concepts, we call the set of the minimum points

$$X^* = \arg\min_x \sum_{i \in \mathcal{H}} Q_i(x) \tag{8}$$

the *true* minimum point set. Correspondingly, the point of that set $x^* \in X^*$ is called a *true minimum point*. For any subset of agents $S \subseteq \mathcal{V}$, we also use the following notations:

$$X_S = \arg\min_x \sum_{i \in S} Q_i(x) \tag{9}$$

for the minimum point set of the aggregated cost functions of agents in S, and $x_s \in X_S$ for a minimum point in X_S.

Note that without specifying the property of cost functions of agents in an arbitrary subset S, the property of the minimum point set X_S is also undecided, including but not limited to the following possible cases: (1) $|X_S|$=0, i.e., there is no minimum point; (2) $|X_S|$=1, i.e., there is only one minimum point; (3) the set X_S forms a region or shape in the Euclidean space $\mathbb{R}^d$, or (4) the set X_S consists of many unconnected parts.

3.2 Solvability: the importance of redundancy in cost-functions

Since it is impossible to solve the Byzantine fault-tolerant distributed optimization problem (6) in general, researchers eyed on reasonable extra conditions that can help solving the problem. A line of work shows the relationship between solvability and redundancy in cost functions. The cost functions of agents, although not identical, have underlying connections with each other. This kind of conditions can help achieving the optimization goal without requiring side knowledge of the agents.

Su and Vaidya [98] in their extension of their previous work [97] analyzed the scenario where the cost function $Q_i(x)$ of each agent i is formed as a convex combination of k *input functions*. Formally, a function $h : \mathbb{R} \to \mathbb{R}$ is *admissible* if $h(x)$ is convex, L-Lipschitz continuous, and its minimum point set $\arg\min h(x)$ is non-empty and compact. Given k admissible input functions $h_1(x), ..., h_k(x)$, there exists a matrix $\boldsymbol{A} \in \mathbb{R}^{k \times n}$, such that the cost function $Q_i(x)$ of each agent $i \in \mathcal{V}$ is of the form

$$Q_i(x) = \boldsymbol{A}_{1i} h_1(x) + \boldsymbol{A}_{2i} h_2(x) + ... + \boldsymbol{A}_{ki} h_k(x), \tag{10}$$

where $\boldsymbol{A}_{ji} \geq 0$ and $\sum_{j=1}^{k} \boldsymbol{A}_{ji} = 1$ for all $i \in \mathcal{V}$ and $j = 1, ..., k$. The goal is to find a solution $\widetilde{x} \in \mathbb{R}$ to the following optimization problem

$$\widetilde{x} \in \arg\min_{x \in \mathbb{R}} h(x) = \frac{1}{k} \sum_{j=1}^{k} h_j(x) \tag{11}$$

instead of (4), in presence of up to f Byzantine faulty agents. The authors considered both cases where the agents are aware or unaware of the matrix $\boldsymbol{A}$. Suppose agents are aware of $\boldsymbol{A}$, (11) can be achieved if $\boldsymbol{A}$ can correct up to f arbitrary entry-wise errors as stated in [13] and the communication graph admits *Byzantine broadcast* [11]. On the other hand, suppose agents are not aware of $\boldsymbol{A}$ beforehand,

(11) can also be solved if all input functions share at least one common minimum point, and there is a necessity condition of $n > 3f$ setting an upper bound for this special case.

Gupta and Vaidya [45, 46] studied a different type of redundancy in cost functions named $2f$-redundancy, defined as follows:

Definition 1 ($2f$-redundancy) *The cost functions of a set of non-faulty agents $\mathcal{H}$ is said to satisfy $2f$-redundancy if for every subset $S \subseteq \mathcal{H}$ of size at least $n-2f$,*

$$\arg\min_x \sum_{i \in S} Q_i(x) = \arg\min_x \sum_{i \in \mathcal{H}} Q_i(x). \tag{12}$$

Note that here $x \in \mathbb{R}^d$ for any $d \in \mathbb{Z}_{>0}$. This redundancy condition implies that the aggregated cost functions of every $n-2f$ non-faulty agents minimizes at the same set of points. It is further showed that it is only possible to achieve goal (6) in presence of up to f Byzantine faulty agents if the non-faulty cost functions satisfy $2f$-redundancy. Note that this result does not require convexity of the cost functions. The authors also pointed out that although the conditions mentioned in Definition 1 appears technical, in many practical applications, such redundancy in cost functions "occurs naturally" [45], including applications in distributed sensing and distributed learning.

Liu et al. [68] further generalized the redundancy notation to $(2f, \epsilon)$-redundancy, where ϵ is an approximate parameter, defined as follows:

Definition 2 ($(2f, \epsilon)$-redundancy) *The agents' cost functions are said to have* $(2f, \epsilon)$-redundancy *property if and only if for every pair of subsets S, $\widehat{S} \subseteq \{1, \ldots, n\}$ with $|S| = n - f$, $\left|\widehat{S}\right| \geq n - 2f$ and $\widehat{S} \subseteq S$,*

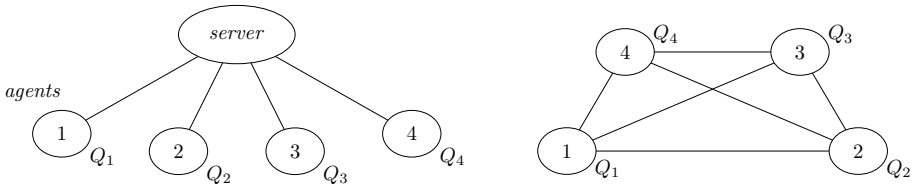
$$\operatorname{dist}\left(\arg\min_{x \in \mathbb{R}^d} \sum_{i \in S} Q_i(x),\ \arg\min_{x \in \mathbb{R}^d} \sum_{i \in \widehat{S}} Q_i(x)\right) \leq \epsilon. \tag{13}$$

where $\operatorname{dist}(\cdot, \cdot)$ is Hausdorff distance[2] between two sets in d-dimensional Euclidean space. Intuitively, this redundancy condition implies that every set of no less than $n-2f$ non-faulty agents has its minimum point set of aggregated cost functions within ϵ distance to the true minimum point set. The authors showed that $(2f, \epsilon)$-redundancy is necessary and sufficient for a deterministic algorithm to output a point close enough (also measured by ϵ) to a true minimum point. Similarly, this finding also does not require the cost functions to be convex, but only requires a compact minimum point set.

3.3 Practical solutions to Byzantine fault-tolerant distributed optimization problems

After analyzing the theoretical solvabitlity of Byzantine fault-tolerant distributed optimization problems, we further dive in to the algorithms that can be used to solve the problems in practice.

[2]Defined in Appendix A



(a) Server-based (b) Peer-to-peer

Figure 1: Illustrations of the two major network architectures in Byzantine fault-tolerant distributed optimization problems.

3.3.1 Commonly-studied system architectures

Before we start, it is useful to discuss first the different system architectures those algorithms are designed for. As a recently started research topic, researchers make various assumptions to the system. But most of those assumptions can be summarized into the following categories.

Network architectures There are two major architectures studied in the researches of Byzantine fault-tolerant distributed optimization, illustrated in Figure 1: (1) the server-based architecture (or centralized architecture) and (2) the peer-to-peer architecture (or decentralized architecture). In the *server-based* architecture, in the majority of researches, there is one server and n agents; the agents communicate with the server, but not with each other. It is usually assumed that the server is trustworthy, but up to f agents may be Byzantine faulty. There are also some works that use multiple servers or different communication models in order to achieve other specific goals. Unless specified otherwise, when referring to the server-based architecture in this survey, we are referring to the former common architecture. In the *peer-to-peer* architecture, n agents are connected with each other, and up to f of those agents may be Byzantine faulty. Note that the network is not necessarily complete, undirected, or fixed in the peer-to-peer architecture.

Problems under the two architectures can sometimes be equivalent. For example, provided that $f < \frac{n}{3}$, any algorithm for the server-based architecture can be simulated in a complete-graph peer-to-peer architecture using Byzantine broadcast primitive [70]. Therefore, some algorithms applicable to server-based architecture are also theoretically applicable to complete-graph peer-to-peer architecture.

Byzantine status We briefly mention that for the majority of the algorithms, the Byzantine status of agents may change during the execution of an algorithm, i.e, at different times, different agents may be Byzantine faulty, but the total number of agents exhibiting Byzantine behavior at any given time is bounded by f. An example in practice for the reason of this assumption is that any agent in a distributed system can experience system failure, but statistically, the total number of failed agents can be bounded by a certain number. Unless otherwise specified, the algorithms we introduced below do not require fixed Byzantine status.

However, in some researches, it is also useful to assume that Byzantine faulty

agents are fixed during an execution. This kind of assumptions are necessary for some algorithms to achieve their fault-tolerance goal. We will discuss several algorithms of such kind in the following sections.

Data distributions In a general distributed optimization problem, each agent i has its own cost function $Q_i(x)$ that is potentially different from other agents'. If in server-based network, the server does not have a cost function, nor does it have the knowledge of cost functions of agents.

In distributed learning setting, however, there might be different assumptions. The assumptions mainly fall in these categories: (1) all agents have their data samples drawn from the same data distribution $\mathcal{D}$; (2) each agent i has their data drawn from a data distribution $\mathcal{D}_i$, or (3) all agents have the same dataset (parallel setting), including the cases in server-based network, the server also has the same dataset, or the server assigns (sends) data samples to agents during the training process. We will discuss fault-tolerance in distributed learning further in Section 3.4.

3.3.2 Gradient descent with gradient filters

One major category of these algorithms is called *gradient filters* [49], or *robust gradient aggregation* [7, 19], which are designed and used mainly with (distributed) gradient descent (abbr. DGD) [78]. DGD is a popular method solving distributed optimization problems, hence it is natural to build upon it to achieve Byzantine fault-tolerance.

DGD is originally designed for peer-to-peer architecture, where each agent keeps its own local estimate x_i. During the algorithm, the agents communicate in iterations. In each iteration t, each agent i shares its local estimate x_i with other agents, then performs a consensus step and then a descent step. Specifically, the agent conducts the following update:

$$x_i^{t+1} = \sum_{j=1}^{n} w_{ij}^t x_j^t - \eta_t g_i^t, \tag{14}$$

where $x_i^t \in \mathbb{R}^d$ indicates agent i's local estimate at iteration t, w_{ij}^t is the weight of communication link on edge $(j,i) \in \mathcal{E}(t)$ at iteration t such that $\boldsymbol{W}(t) = \left(w_{ij}^t\right) \in \mathbb{R}^{n\times n}$ is doubly stochastic for all t, g_i^t is the (sub)gradient[3] of the local cost function $Q_i(x)$ at $x_i(t)$, and $\eta_t > 0$ is a diminishing step size[4]. For server-based architecture, DGD can also be adapted [67].

One simple version of server-based DGD is described in Algorithm 1. Generally speaking, in each iteration t, each agent receives the current estimate x^t from the server, computes and sends its update g_i^t back to the server; the server keeps the current estimate x^t, computes its update in iteration t using all update vector received from the agents, and updates its current estimate according to (15).

[3]DGD is originally proposed for convex but not necessarily differentiable cost functions.
[4]Defined in Appendix A

Algorithm 1 DGD for server-based architecture

1: **procedure** DGDSERVER ▷ Server executes this procedure
2: **Initialize**: arbitrarily select an initial estimate x^0
3: **Loop** until convergence. For current iteration $t = 0, 1, 2, ...$:
4: Broadcast the current estimate x^t to all agents
5: Wait until receive gradient g_i^t from each agent i
6: Update the current estimate x^t by summing the updates of all agents:

$$x^{t+1} = x^t - \eta_t \sum_{i=1}^{t} g_i^t \tag{15}$$

7: **end procedure**

8: **procedure** DGDAGENT ▷ Each agent i executes this procedure
9: **Loop** until server stops. For each iteration t:
10: Wait until receive current server estimate x^t
11: Compute (sub)gradient of local function $Q_i(x)$ at x^t:

$$g_i^t = \nabla Q_i(x^t) \tag{16}$$

12: Send g_i^t as agent i's update to the server
13: **end procedure**

Byzantine agents in the system may not follow the prescribed algorithms. Specifically, in line 11 of Algorithm 1, instead of computing and sending its gradient following (16), a Byzantine agent may choose to send arbitrary information as its update. When the server receives g_i^t's from agents in line 5, some of them may be incorrect. In fact, Blanchard et al. [6] showed that no linear combination methods for aggregating the updates (including summing and averaging, indicated by (15)) can tolerate even a single Byzantine agent. Thus comes the *gradient filters*, methods that redesign the aggregation rule. A general framework for server-based Byzantine fault-tolerant optimization with DGD, or Byzantine gradient descent (BGD), is shown in Algorithm 2.

The major difference between Algorithms 2 and DGD is (17): instead of updating the estimates by the sum of all agents updates, BGD uses an aggregation rule $\mathsf{GradFilter}(\cdot)$. Generally speaking, $\mathsf{GradFilter} : \mathbb{R}^d \times n \to \mathbb{R}^d$ is a function that takes n vectors of d-dimension, and output a vector of d-dimension. Normally, the n vectors would be those updates received from n agents at iteration t, i.e., $\left\{g_i^t\right\}_{i=1}^n$, and the output is used as the server's update at iteration t. Also note that only non-faulty agents follow the procedure BGDAGENT and send their correct gradient as its update. Byzantine agents may behave arbitrarily, including but not limited to sending its correct update, sending an arbitrary d-dimensional vector, sending other arbitrary information, or not sending anything. Note that only an arbitrary d-dimensional vector would confuse the server, since other behaviors will immediately indicate the agent deviating from the prescribed algorithm, and therefore must be faulty; if the Byzantine status of agents is fixed, the algorithm can further remove

Algorithm 2 BGD framework for server-based architecture

procedure BGDSERVER ▷ Server executes this procedure
  Initialize: arbitrarily select an initial estimate x^0
  Loop until convergence. For current iteration $t = 0, 1, 2, ...$:
    Broadcast the current estimate x^t to all agents
    Wait until receive gradient g_i^t from each agent i
    Update the current estimate x^t by summing the updates of all agents:

$$x^{t+1} = x^t - \eta_t \, \mathsf{GradFilter}\left(g_1^t, ..., g_n^t\right) \tag{17}$$

end procedure

procedure BGDAGENT ▷ Each *non-faulty* agent i executes this procedure
  Loop until server stops. For each iteration t:
    Wait until receive current server estimate x^t
    Compute (sub)gradient of local function $Q_i(x)$ at x^t:

$$g_i^t = \nabla Q_i(x^t) \tag{18}$$

    Send g_i^t as agent i's update to the server
end procedure

the ill-behaved agent in future iterations.

Suppose out of n agents, up to f can be Byzantine faulty. We introduce some gradient filters as follows. For simplicity, we denote $\boldsymbol{g}^t = \left\{g_i^t\right\}$ as the set of update vectors unless otherwise specified.

Krum Krum [6] is an angle-based (or direction-based) gradient filter, built upon a Krum score $s(i)$. Suppose among the set of vectors $\boldsymbol{g}^t$, $i \to j$ denotes the fact that g_j^t belongs to the $n - f - 2$ closest vectors to g_i^t, with distance between two vectors defined by some norm $\|\cdot\|$. The score $s(i) = \sum_{i \to j} \left\|g_i^t - g_j^t\right\|^2$, i.e., the sum of the distances between g_i^t and the $n - f - 2$ vectors closest to it. Then,

$$\mathsf{GradFilter}\left(g_1^t, \ldots, g_n^t\right) = g_{i_*}^t, \tag{19}$$

where i_* is an agent with the smallest score $s(i_*)$. Krum outputs one of the vectors submitted by the agents. The expected time complexity of computing this filter is $O(n^2 d)$. Krum is originally proposed as a distributed machine learning filter for stochastic gradient descent (SGD), but it can also be applied to general optimization problems.

Multi-KRUM Multi-Krum [6, 7] is a variant of Krum. Instead of selecting one vector, multi-Krum selects m vectors and averages them, where m is a hyperparameter. There are two versions, the first version (also known as m-Krum) iterates m times, each time it calculates the scores for all vectors in $\boldsymbol{g}^t$, selects a vector with the highest score and remove it from the set $\boldsymbol{g}^t$; the second version selects m vectors

with the m smallest scores. Obviously, the time complexity of the second version is the same as Krum, which is significantly better than that of the first version.

Coordinate-wise methods Coordinate-wise median and trimmed mean [120] process the received vectors by coordinates. Suppose $v[k]$ denotes the k-th coordinate of a vector v, the gradient filter can be written as follows

$$\begin{aligned} &\mathsf{GradFilter}\left(g_1^t, ..., g_n^t\right) = v, \text{ where} \\ &v[k] = \begin{cases} \mathsf{median}\left\{g_1^t[k], ..., g_n^t[k]\right\}, & \text{for median,} \\ \mathsf{trmean}_\beta\left\{g_1^t[k], ..., g_n^t[k]\right\}, & \text{for trimmed mean,} \end{cases} \end{aligned} \tag{20}$$

where $\mathsf{median}\{\cdot\}$ finds the median from a set of scalars, while $\mathsf{trmean}_\beta\{\cdot\}$ first drop the smallest and largest β fraction, then computes the mean of the rest of the values from a set of scalars. β is a hyperparameter, which is required to be larger than the fraction of faulty agents α, i.e., $\beta \geq \alpha = f/n$. Note that coordinate-wise median does not require a known fraction of faulty agents.

Phocas [116] is a variant of trimmed mean method. The authors also noted that for coordinate-wise methods, the number of faults can also be viewed coordinate-wisely, i.e., the update vectors from every agent can be corrupted, so long as for each coordinate $k \in \{1, ..., d\}$, the number of faulty values in the set $\{g_i^t[k]\}_{i=1}^n$ is upper-bounded by f. Another coordinate-wise method, mean around median [115], calculates each coordinate k by the mean of $n - f$ values $g_i^t[k]$'s that are closest to the median of all values. These filters are also originally proposed as a distributed learning filter for SGD.

Geometric median Geometric median also has some robustness [19, 75, 90, 115], and sometimes is used as a benchmark comparing other gradient filters. Geometric median can be computed to arbitrary precision ϵ in nearly linear time $O(nd \log^3 \frac{1}{\epsilon})$ [21]. That being said, in practice the (geometric) median-based aggregation still dominates the training time in large-scale settings [18].

Median of means Geometric median of means [19] extends the statistical estimator to vectors. Specifically, suppose b divides n, agents are divided into $k = n/b$ groups, each of size b. In each iteration, the gradient filter finds the mean vector in each group, and then uses the geometric median of those k mean vectors as its update. Formally,

$$\mathsf{GradFilter}\left(g_1^t, ..., g_n^t\right) = \mathsf{median}\left\{\frac{1}{b}\sum_{j=1}^{b} g_j^t, ..., \frac{1}{b}\sum_{j=(k-1)b+1}^{n} g_j^t\right\}. \tag{21}$$

The number of groups k is a hyperparameter, which should be chosen such that $k > 2f$. This filter is originally proposed as a distributed learning filter for SGD.

MDA Minimum-diameter averaging [32], also called *Brute* [75], is another median-based method, originally proposed as a statistical estimator in [90]. It uses the

average of the subset S of $n-f$ vectors which has the minimum diameter. Formally,

$$\begin{aligned}&\mathsf{GradFilter}\left(g_1^t,...,g_n^t\right)=\frac{1}{n-f}\sum_{i\in S}g_i^t,\ \text{where}\\&S=\arg\min_{\substack{T\subset\mathcal{V}\\|T|=n-f}}\left(\max_{i,j\in T}\left\|g_i^t-g_j^t\right\|\right).\end{aligned}\tag{22}$$

The computation complexity of MDA is $O\left(\binom{n}{f}+n^2d\right)$. This filter is originally proposed as a distributed learning filter for SGD.

Norm-based methods Norm filtering (or comparative gradient elimination, CGE) and norm-cap filtering (or comparative gradient clipping, CGC) [43, 46, 49] are a group of filters looking at the norms of the vectors. Intuitively, these filters keep the $n-f$ vectors in $\boldsymbol{g}^t$ with smallest norms, while dropping the rest vectors (elimination) or scaling the rest so that they also have the $n-f$-largest norm (clipping). Formally, suppose the vectors in $\boldsymbol{g}^t$ are sorted as follows:

$$\left\|g_{i_1}^t\right\|\leq...\leq\left\|g_{i_{n-f}}^t\right\|\leq\left\|g_{i_{n-f+1}}^t\right\|\leq...\leq\left\|g_{i_n}^t\right\|,\tag{23}$$

the gradient filters can be written as follows:

$$\mathsf{GradFilter}\left(g_1^t,...,g_n^t\right)=\begin{cases}\displaystyle\sum_{j=1}^{n-f}g_{i_j}^t, & \text{for CGE,}\\ \displaystyle\sum_{j=1}^{n-f}g_{i_j}^t+\sum_{j=n-f+1}^{n}\frac{\left\|g_{i_{n-f}}^t\right\|}{\left\|g_{i_j}\right\|}g_{i_j}^t, & \text{for CGC.}\end{cases}\tag{24}$$

The computational complexity of the two methods are both $O(n(\log n+d))$. Intuitively, even if a gradient from faulty agent is kept, its norm is bounded by a gradient from non-faulty agent, and therefore could not do much damage to the server's update. The two methods are originally proposed in the context of distributed linear regression, and analyzed also in general distributed optimization [45] and distributed learning [49].

Bulyan Bulyan [75] is a meta-aggregation rule that can be applied on another gradient filters GradFilter. Bulyan consists of two steps. In the first step, Bulyan iterates $n-2f$ times; in each iteration, it runs GradFilter on the set $\boldsymbol{g}^t$, selects a g_i^t closest to GradFilter's output, add it to a set S, and remove it from $\boldsymbol{g}^t$. After the first step, the set S contains $n-2f$ vectors. In the second step, Bulyan generates a new d-dimensional vector coordinate-wisely from the $n-2f$ vectors with a median-based method, and uses it as the update. It is designed such that it has provable resilience property even if GradFilter in use does not have such a property (e.g., geometric median).

It is worth noting that the convergence analyses of most of the gradient filters requires standard assumptions like continuity, differentiability, Lipschitz smoothness, convexity, strong convexity, including those that are proposed for distributed

learning or SGD. However, the most popular machine learning methods, including deep neural networks, are known to be generally non-convex [22, 56].

It is also worth noting that some gradient filters such as coordinate-wise trimmed mean and CGE are also analyzed under the peer-to-peer setting [47].

For reference purpose, we summarize all gradient filters listed in this section in Table 2 in Appendix B.

3.3.3 Gradient coding

One of the applications of coding theory is to enable error correction [50]. Gradient coding is originally proposed as a straggler mitigation method [105], which is used to speed up synchronous distributed first-order methods [17, 23, 88]. Several works build upon it and extend it to the adversarial setup. Generally, this type of methods work under the parallelization setting of distributed learning, where agents (and server, if any) all have the same dataset.

Draco Draco [18] is a gradient-coding-based method, in which the server assigns same data samples to (on average) r agents, agents compute multiple stochastic gradients and send the gradients to the server in a coded way. The server receives the coded messages from the agents, and decodes the message to recover the correct gradients and identifies the faulty messages. With proper-selected coding method, Draco can tolerate up to $(r-1)/2$ Byzantine agents with linear-time encoding and decoding. The assignment of same tasks to multiple agents is also known as *algorithmic redundancy*. DETOX [85] is an extension of Draco that combines algorithmic redundancy with robust aggregation, with increased speed and improved robustness.

Randomized reactive redundancy After Draco and DETOX, Gupta and Vaidya [44] proposed a similar coding-based framework, in which instead of applying coding every iteration, the server invokes the coding scheme (i.e., check for faults) with probability $q > 0$, while in other iterations, the server simply carries out DGD. By properly choosing the value of q, this framework reduces the computational overhead caused by coding to arbitrarily small. The scheme is especially effective when Byzantine agents are fixed, since once detected, the faulty agents will be removed. The authors also proposed heuristic checking by server and combination with gradient filters.

3.3.4 Other methods

There are some other methods that do not fall into the above major categories. Though those methods are more or less related to previous methods.

Zeno Zeno [117] is a gradient aggregation rule based on the fact that the server obtains certain number of data samples, designed for distributed learning with SGD. In each iteration, instead of assigning data samples to agents, the server calculates a *reliability* score of the vectors sent from the agents, based on the fact that all

the data samples are drawn from the same data distribution $\mathcal{D}$. The server then aggregate using those $n - f$ most reliable gradients to obtain its update.

One-round robust aggregation Instead of checking or filtering every iteration, Yin et al. [120] propose a Robust One-round Algorithm, in which the non-faulty agents conduct their own optimization process using their local cost functions separately, and send their final estimates to the server; Byzantine agents may send arbitrary information. The server then aggregates the estimates (including faulty ones) to compute its own final estimate. There is no communication between agents and the server the whole process except the final step. In [120] geometric median is used as the final aggregation rule, and convergence analysis is provided. The method is proposed originally for distributed learning, and suppose all agents have the same data distribution, the method can achieve comparable empirical performance to some gradient filters.

Variance reducing techniques There are certain techniques used in machine learning known to have the ability of reducing the variance of stochastic gradients. Since the stochastic gradients are generated from randomly drawn data samples, a variance-reduce method is possible to speed up the training process by "stablizing" the gradients [24, 84]. Gupta et al. [49] empirically studied that averaging historical gradients and increasing batch size helps reducing the variance of stochastic gradients, and therefore boosts the performance of fault-tolerant gradient filters. Karimireddy et al. [60] showed that using momentum [84] helps achieving provable convergence of any Byzantine robust gradient filter, while El-Mhamdi et al. [33] studied the boosting of robustness by momentum when computed at agents.

3.3.5 Studies under peer-to-peer architecture

Many methods, including the most of aforementioned gradient filters are proposed under the server-based system architecture. Although we already mentioned in the beginning of this section that we can simulate server-based algorithms under peer-to-peer architecture using Byzantine broadcast primitive, it is worth noting that there is also research specifically studies the peer-to-peer, or decentralized system architecture. Note that in peer-to-peer settings the agents do not broadcast gradients, but rather their local estimates to other agents (recall (14), peer-to-peer DGD).

Su and Vaidya [102] showed that, based-on Byzantine consensus results on peer-to-peer networks [110], a distributed optimization problem with scalar local cost functions has a Byzantine-resilient gradient descent algorithm if the network has a non-empty source component after removing all faulty agents and their edges. Sundaram and Gharesifard [104] proposed *Local Filtering (LF) Dynamics*, a protocol that allows Byzantine fault-tolerant distributed optimization in a f-local network, i.e., each non-faulty agent i has up to f faulty neighbors in $\mathcal{N}_i$, if the network satisfies certain connectivity property called (r, s)-robustness.

Gupta et al. [48] studied the $2f$-redundancy property in decentralized system. Specifically, the authors proposed *Comparative Elimination* (abbr. CE) method,

similar to gradient filters under server-based architecture, as a decentralized optimization algorithm for a fully-connected network. The method will mitigate the detrimental impact of potentially incorrect values from Byzantine faulty agents.

3.4 Byzantine fault-tolerant distributed learning

Distributed machine learning is a popular subproblem of distributed optimization. Naturally, a lot of work in Byzantine fault-tolerant distributed optimization also focuses on the case of distributed learning. The formulation of Byzantine fault-tolerant distributed learning problem is related to that of distributed optimization, but with its own characteristics.

One typical formulation of the problem can be described as follows [19, 45, 49, 75, 120]. Suppose there are n agents in the system, out of them up to f can be Byzantine faulty. The training data samples are drawn i.i.d. from some unknown data distribution $\mathcal{D}$. A machine learning model Θ has a learning parameter x in the form of d-dimensional vectors. The model Θ decides a loss function $\ell(x; z)$ for each data point $z \sim \mathcal{D}$. Let $\mathbb{E}_{z\sim\mathcal{D}}$ denote the expectation with respect to the random data sample z. The goal is to minimize with respect to x the function

$$Q(x) \triangleq \mathbb{E}_{z\sim\mathcal{D}}\left[\ell(x; z)\right], \tag{25}$$

namely the population cost (or loss) function, i.e. find a point $\widehat{x}$ such that

$$\widehat{x} \in \arg\min_{x\in\mathbb{R}^d} Q(x). \tag{26}$$

Comparing this formulation with the general fault-tolerant distributed optimization goal (6), we see that here, $Q_i(x) = Q(x)$ for every agent i, therefore, $2f$-redundancy holds trivially. However, none of the agents, including the server, knows the cost function exactly. For gradient-based algorithms, DGD should be changed to distributed stochastic gradient descent (abbr. D-SGD) [49], where instead of gradient $g_i^t = \nabla Q_i(x^t)$, the agent i draws one or some data samples z or $\boldsymbol{z}$ and computes its stochastic gradient

$$g_i^t = \begin{cases} \nabla\ell(x; z), & \text{when drawing one data sample, or} \\ \nabla\ell(x; \boldsymbol{z}) = \displaystyle\sum_{z\in\boldsymbol{z}} \ell(x; z), & \text{when drawing multiple data sample,} \end{cases} \tag{27}$$

in each iteration t.

Another more general formulation [51, 66, 68, 82] assumes each agent i has a potentially different data distribution $\mathcal{D}_i$. Then each agent i has a local cost function

$$Q_i(x) \triangleq \mathbb{E}_{z\sim\mathcal{D}_i}\left[\ell(x; z)\right]. \tag{28}$$

The goal is to find a point $\widehat{x}$ that minimizes the aggregated cost functions of non-faulty agents,

$$\widehat{x} \in \arg\min_{x\in\mathbb{R}^d} \sum_{i\in\mathcal{H}} Q(x).$$

Note this goal is the same as (7). One application of this formulation is federated learning [73], where different agents have different data samples and underlying data distributions. Gradient filters like RSA [66], RFA [82], and RGE [26] are proposed specifically under this formulation.

3.5 Resilience notations

Given the facts that the original goal of distributed optimization (3) is not achievable in presence of Byzantine faulty agents, and that there are a variety of ways to measure the resilience of a Byzantine fault-tolerant distributed optimization algorithm, here we introduce some of them.

(f, ϵ)-resilience (f, ϵ)-resilience [68] is a resilience notation for Byzantine fault-tolerant distributed optimization algorithms. A deterministic algorithm is said to be (f, ϵ)-resilient for some $\epsilon \geq 0$, if the output of the algorithm $\widehat{x}$ satisfies the following:

$$\text{dist}\left(\widehat{x},\ \arg\min_{x\in\mathbb{R}^d} \sum_{i\in\mathcal{H}} Q_i(x)\right) \leq \epsilon. \tag{29}$$

Intuitively, to be (f, ϵ)-resilient, the output of the algorithm should be within ϵ distance to the true minimum point set. Furthermore, DGD with gradient filters, i.e., Algorithm 2 can achieve (f, ϵ)-resilience if the gradient satisfies certain conditions, with several continuity and convexity assumptions; those gradient filters include coordinate-wise trimmed mean and CGE [68]. $(f, 0)$-resilience is also called *exact fault-tolerance* [45], requiring that the algorithm's output satisfies the goal (7) exactly. It is also shown that $(2f, \epsilon)$-redundancy and $2f$-redundancy are the necessary conditions to (f, ϵ)-resilience and $(f, 0)$-resilience, respectively [45, 68].

(α, f)-resilience (α, f)-resilience [6] is a notation used for measuring a gradient aggregation rule, under the same distribution setting of fault-tolerant distributed learning. Suppose a group of vectors $V_1, ..., V_n \in \mathbb{R}^d$ are drawn i.i.d. from some distribution G, with $\mathbb{E}[G] = g$. Also suppose $B_1, ..., B_f \in \mathbb{R}^d$ be a group of arbitrary vectors. An aggregation rule $\mathsf{GradFilter}$ is said to be (α, f)-Byzantine resilient for some $0 \leq \alpha < \pi/2$ if, for any $1 \leq j_1 \leq ... \leq j_f \leq n$, the output vector of the aggregation rule

$$V = \mathsf{GradFilter}(V_1, ..., \underbrace{B_1}_{j_1}, ..., \underbrace{B_f}_{j_f}, ..., V_n)$$

satisfies the following:

(i) $\langle \mathbb{E}[V],\ g\rangle \geq (1 - \sin\alpha) \cdot \|g\|^2 > 0$;

(ii) For $r = 2, 3, 4$, $\mathbb{E}\|V\|^2$ is bounded above by a linear combination of terms $\mathbb{E}\|G\|^{r_1}$,...,$\mathbb{E}\|G\|^{r_{n-1}}$ with $r_1 + ... + r_{n-1} = r$.

Intuitively, we want the output of a gradient filter to be close enough to the expected gradient, and the filter can control the effects of the discrete nature of SGD dynamics [8]. Some aforementioned gradient filters and others are known to be (α, f)-Byzantine resilient, including Krum [7], geometric median [19, 115], coordinate-wise median [115], mean near median [115], and Bulyan [75].

$(\delta_{\max}, c)$-robust aggregator $(\delta_{\max}, c)$-robust aggregator [60] is another notation of measuring aggregation rules. Similarly, consider a group of independent random vectors $V_1, ..., V_n$, such that a non-faulty subset $N \subseteq \{1, ..., n\}$ of size $|N| \geq (1-\delta)n$ satisfies that for any apriori fixed $i, j \in N$, $\mathbb{E}[V_i] = \mathbb{E}[V_j]$, and $\mathbb{E}\|V_i - V_j\|^2 \leq \rho^2$ for

some ρ. An aggregation rule GradFilter is said to be $(\delta_{\max}, c)$-robust if its output vector

$$V = \mathsf{GradFilter}\left(V_1, ..., V_n\right)$$

satisfies that for some constant c, $\mathbb{E}\left\|V - \frac{1}{|N|}\sum_{i\in N} V_i\right\|^2 \leq c\delta\rho^2$. The authors then argued that combining $(\delta_{\max}, c)$-robust aggregator and momentum SGD, an algorithm can solve distributed learning problems with non-convex smooth cost functions.

4 Other fault-tolerant distributed optimization problems

Outside the scope of our discussion in Byzantine fault-tolerance distributed optimization in Section 3, some other problems related to fault-tolerance in distributed optimization are also studied.

4.1 Alternative adversarial models

There is a handful of recent researches considering specific adversarial models other than Byzantine models, certain types of attacks against distributed optimization algorithms, or analyzing possible behavior of an adversarial agent in the system. Such research are rather ad hoc or unstructured, but still provide important view points other than the common Byzantine model.

We use the similar notations as we used for Byzantine fault-tolerant distributed optimization problems in Section 3; specifically, out of all agents $\mathcal{V}$, $\mathcal{H}$ stands for the set of honest agents, and $\mathcal{B}$ stands for the set of adversarial agents, with $|\mathcal{V}| = n$ and $|B| \leq f$.

Yin et al. [121] considers *saddle point attack* against existing Byzantine fault-tolerant distributed (non-convex) learning algorithms. Many machine learning models have non-convex cost functions. Although gradient descent or its variants are known to converge to a local minimum point with high probability [58, 64], Byzantine agents can manipulate those methods into a fake local minimum near a saddle point, i.e., saddle point attack. This is a specific type of attack that only happens when the cost function contains saddle points, at which the gradient of the cost function would also be 0, and therefore satisfies the stopping criteria of many Byzantine distributed learning algorithms. The authors proposed ByzantinePGD with *perturbation* [58] to escape saddle point during the training process.

Wu et al. [113] discussed a *data injection attack* against distributed optimization with peer-to-peer DGD update (14). Specifically, suppose the adversarial agents' goal is to steer the final estimate of all agents to a target $x \in \mathbb{R}^d$, an adversarial agent $i \in \mathcal{B}$ will not send its local estimate x_i^t to other agents, but rather x with artificial noise z_i^t, i.e. $x + z_i^t$. The adversarial agents will also try to behave as if they are converging, having $\lim_{t\to\infty} \left\|z_i^t\right\| \overset{a.s.}{=} 0$. The authors then proposed a local metric

that trustworthy agents can compute to *detect* (notice the existence of an adversary in its neighborhood) and *localize* (distinguish which neighbor is adversarial) the adversarial agents performing such kind of attack. There is a group of research focusing on the detection of adversarial agents in distributed optimization [65, 86, 114].

Prasad et al. [83] studied two specific types of statistical models: (1) arbitrary outliers in Huber's ϵ-contamination model [54], and (2) heavy-tails, i.e., the data distribution $\mathcal{D}$ has weak moment assumptions. The authors argued that such kind of statistical models are common in real-world datasets. The authors then introduced a class of robust estimators (gradient filters) with robustness guarantees for a variety of statistical models: linear regression, logistic regression, and exponential family models. Such robust estimators can be easily applied to distributed settings.

Ravi et al. [87] analyzed possible behavior of malicious agents in the system. Suppose the malicious agents intend to manipulate its objective function, such that the output using cost functions from all agents x^a will deviate from a correct output x^* by a vector ϵ, i.e. $x^a = x^* + \epsilon$. The authors derived that the magnitude of ϵ is bounded by a function of the number of faults f, and the gradients of the malicious agents. Therefore, in order to launch a substantial attack, either the value of f needs to be large, or the gradients from malicious agents need to be large, which might become giveaways of malicious agents.

Charikar et al. [16] views the fault-tolerance learning problem from a data perspective. It is assumed that the α fraction of the data is drown from an unknown data distribution $\mathcal{D}$ and the rest $(1 - \alpha)$ is not. The *list-decodable* learning returns a list of poly$(1/\alpha)$ answers and one of them is correct. The *semi-verified* learning suggests that if a small trusted dataset, also drawn from $\mathcal{D}$, is provided, it is possible to use the trusted dataset to enable accurate extraction of information from the larger dataset with untrusted data. Such findings can be applied to Byzantine fault-tolerant distributed learning problems.

Based on resilient consensus with trusted agents [2], research shows that *trusted agents* in the system can be crucial against adversarial agents [4, 34, 125]. Baras and Liu [4] presented a trust-aware consensus algorithm in peer-to-peer networks that can effectively detect Byzantine adversaries and exclude them, even in sparse networks with connectivity less than $2f + 1$. Zhao et al. [125] presented another algorithm that if trusted agents induce a connected dominating set, the algorithm outputs a point bounded by the convex minimum point set of weighted average of all non-faulty agents' cost functions.

4.2 Fault-tolerance and privacy

Privacy issue in optimization has gained increasing attention in recent years, in both non-distributed settings [1, 25, 53, 93, 95] and distributed settings [57, 69, 76, 106, 118]. In distributed settings, some research proposes encryption-based methods to prevent passive attackers from intercepting the exchanged information between agents in the network [69, 106], while others utilizes *differential privacy* [76, 118], a gold standard notion for privacy-preserving in data [31]. Informally, a differential-

private algorithm is insensitive to small differences in its input dataset.

Some recent research tries to simultaneously achieve privacy-preservation and fault-tolerance. He et al. [52] presented a Byzantine-resilient and privacy-preserving machine learning solution with a two-server protocol. Achieving *local differential privacy* [35], the data of each agent is secure against any other agents in the system, and the two honest-but-curious servers. The Byzantine resilience can be provided by any gradient filter, e.g., those we mentioned in Section 3.3.2, and the protocol achieves the same result as non-private algorithms using the same gradient filter. The authors also showed that their protocol only has negligible computation and communication overhead comparing to non-private methods.

So et al. [94] proposed Byzantine-resilient secure aggregation (BREA) framework to achieve both privacy-preservation and fault-tolerance in federated learning. Different from the previous work, BREA only has one server in the system. The secrecy among agents is achieved by a verifiable secret sharing protocol [36] ensuring that updates from an agent cannot be learnt by other agents, while fault-tolerant aggregation is managed by the server using a gradient filter such as Krum. It is worth mentioning though, BREA does not have a provable differentially-private property.

Guerraoui et al. [42] analyzed the compatibility between differentially-private noisy injection methods [76, 92] and Byzantine-resilient gradient filters for distributed learning under a one-server architecture (i.e., the server-based architecture in Figure 1). The authors showed that in order to simultaneously guarantee Byzantine-resilience and differential privacy, the agents must sample data with batch size of the order of $\sqrt{d}$, where d is the parameter size of the machine learning model. Such large batch size is often impractically large, since many state-of-the-art machine learning models have a huge number of parameters [123]. The authors further showed that, in strongly-convex cost function machine learning tasks, with differentially-private noise injection, the training error rate of a Byzantine-resilient gradient filter is of the order of $\frac{d}{b^2}$, where b is the batch size; while non-private training error rate of the same gradient filters is independent from d.

Needless to say, the last results of compatibility from [42] are rather frustrating. However, since this is an emerging research area, there are still many unexplored methods and mechanisms to be studied. As the authors of [42] pointed out, Byzantine fault-tolerant methods other than gradient filters, and variance reduction techniques [9] both still show potential based on their analysis.

5 Summary

This survey summarizes the current state of studies in the fault-tolerance problem of distributed optimization, including both Byzantine fault-tolerant distributed optimization and other fault-tolerant distributed optimization researches. For Byzantine fault-tolerance distributed optimization, current researches studied the formulation and solvability of the problem; the practical solutions to the problem, including gradient filters, gradient coding methods, and other methods; the special case of

Byzantine fault-tolerance in distributed learning; and commonly seen resilience notations characterizing Byzantine fault-tolerant distributed optimization algorithms. For other fault-tolerant distributed optimization researches, there is a group of work that proposed and studied some specific adversarial models. There is also an emerging line of work that intends to combine both robustness and privacy to distributed optimization algorithms.

5.1 Future work

Based on the findings in our survey, there are various open questions in this field. We list some possible future work as follows.

Gradient filters Although there already are a variety of gradient filters presented in this survey, many of them are either computationally extensive, or with weaker convergence property or stochastic error rate (when applying to D-SGD). It is still interesting to see if there are other gradient filters that can achieve both efficiency and correctness. One interesting idea, similar to Bulyan that applies the same filter multiple times, would be to see both theoretically and empirically effects of applying multiple different gradient filters in the same fault-tolerant algorithm, i.e, the effects of combinations. Also, efficiency could be achieved by heuristic filtering [43], instead of applying the gradient filter in every iteration.

Peer-to-peer network The majority of Byzantine fault-tolerant algorithms are built under the server-based architecture, e.g., [6, 18, 49], etc.; while studies under peer-to-peer architectures, although exist [47, 48], are rather rare, and the results are not as systematic. It would be interesting to explore further how the communication network structure is related to solvability of the fault-tolerance problem, and also practical algorithms to achieve fault-tolerance. Recall the difference of DGD under peer-to-peer and server-based architectures, the most gradient filters and other methods cannot be directly applied to peer-to-peer settings.

Asynchrony Asynchronous distributed optimization is a major branch of distributed optimization studies [55, 96, 124]. Although some previous work suggested that their results can be easily extended to asynchronous setting, the combination of asynchrony and fault-tolerance is still a topic to be explored. It would be interesting to see some directed results on asynchronous systems, e.g., the effect of achieving both goals on issues such as convergence property. For example, $(f, r; \epsilon)$-*redundancy*, an extension of $(2f, \epsilon)$-redundancy discussed in Section 3.2 is proposed in [68], which can be utilized to tackle both up to r stragglers and up to f Byzantine faulty agents at the same time.

Privacy-preservation We already discussed the current attempts on combination of privacy-preservation and fault-tolerance. Still, this is an emerging topic with real-world applications and impact.

Adversary models The majority of the fault-tolerant optimization researches focus on Byzantine fault-tolerance, both in theory and in practice. However, in many

real-world scenarios, such assumption may be too strong. A group of omnipotent faulty agents that have knowledge on the algorithm, status of other agents, or even all the data is unlikely in many cases. Instead, it is more likely that only a number of faulty agents can collaborate with each other, or can be corrupted by an adversary [27], or their adversarial behavior is limited.

Acknowledgements

This survey is supported by a Fritz Fellowship from Georgetown University.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[2] Waseem Abbas, Yevgeniy Vorobeychik, and Xenofon Koutsoukos. Resilient consensus protocol in the presence of trusted nodes. In *2014 7th International Symposium on Resilient Control Systems (ISRCS)*, pages 1–7. IEEE, 2014.

[3] Natalia Amelina, Alexander Fradkov, Yuming Jiang, and Dimitrios J Vergados. Approximate consensus in stochastic networks with application to load balancing. *IEEE Transactions on Information Theory*, 61(4):1739–1752, 2015.

[4] John S Baras and Xiangyang Liu. Trust is the cure to distributed consensus with adversaries. In *2019 27th Mediterranean Conference on Control and Automation (MED)*, pages 195–202. IEEE, 2019.

[5] Dimitri P Bertsekas and John N Tsitsiklis. Parallel and distributed computation: numerical methods. 2003.

[6] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 118–128, 2017.

[7] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Byzantine-tolerant machine learning. *arXiv preprint arXiv:1703.02757*, 2017.

[8] Léon Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142, 1998.

[9] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[10] Stephen P Boyd. Convex optimization: from embedded real-time to large-scale distributed. In *KDD*, page 1, 2011.

[11] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[12] Richard R Brooks and S Sitharama Iyengar. Robust distributed computing and sensing algorithm. *Computer*, 29(6):53–60, 1996.

[13] Emmanuel J Candes and Terence Tao. Decoding by linear programming. *IEEE transactions on information theory*, 51(12):4203–4215, 2005.

[14] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial informatics*, 9(1):427–438, 2012.

[15] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[16] Moses Charikar, Jacob Steinhardt, and Gregory Valiant. Learning from untrusted data. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 47–60, 2017.

[17] Zachary Charles, Dimitris Papailiopoulos, and Jordan Ellenberg. Approximate gradient coding via sparse random graphs. *arXiv preprint arXiv:1711.06771*, 2017.

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

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

[20] Paul Chew and Keith Marzullo. Masking failures of multidimensional sensors. Technical report, Dept. of Computer Science, Cornell University, 1991.

[21] Michael B Cohen, Yin Tat Lee, Gary Miller, Jakub Pachocki, and Aaron Sidford. Geometric median in nearly linear time. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 9–21, 2016.

[22] Ronan Collobert, Fabian Sinz, Jason Weston, and Léon Bottou. Trading convexity for scalability. In *Proceedings of the 23rd international conference on Machine learning*, pages 201–208, 2006.

[23] Andrew Cotter, Ohad Shamir, Nathan Srebro, and Karthik Sridharan. Better mini-batch algorithms via accelerated gradient methods. *arXiv preprint arXiv:1106.4574*, 2011.

[24] Ashok Cutkosky and Francesco Orabona. Momentum-based variance reduction in non-convex sgd. *arXiv preprint arXiv:1905.10018*, 2019.

[25] Georgios Damaskinos, Celestine Mendler-Dünner, Rachid Guerraoui, Nikolaos Papandreou, and Thomas Parnell. Differentially private stochastic coordinate descent. *arXiv preprint arXiv:2006.07272*, 2020.

[26] Deepesh Data and Suhas Diggavi. Byzantine-resilient sgd in high dimensions on heterogeneous data. *arXiv preprint arXiv:2005.07866*, 2020.

[27] Carole Delporte-Gallet, Hugues Fauconnier, Rachid Guerraoui, and Andreas Tielmann. The disagreement power of an adversary. *Distributed Computing*, 24(3):137–147, 2011.

[28] Danny Dolev. The byzantine generals strike again. *Journal of algorithms*, 3 (1):14–30, 1982.

[29] Xiwang Dong, Yongzhao Hua, Yan Zhou, Zhang Ren, and Yisheng Zhong. Theory and experiment on formation-containment control of multiple multirotor unmanned aerial vehicle systems. *IEEE Transactions on Automation Science and Engineering*, 16(1):229–240, 2018.

[30] John C Duchi, Alekh Agarwal, and Martin J Wainwright. Dual averaging for distributed optimization: Convergence analysis and network scaling. *IEEE Transactions on Automatic control*, 57(3):592–606, 2011.

[31] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4): 211–407, 2014.

[32] El-Mahdi El-Mhamdi, Rachid Guerraoui, Arsany Guirguis, Lê Nguyên Hoang, and Sébastien Rouault. Genuinely distributed byzantine machine learning. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 355–364, 2020.

[33] El-Mahdi El-Mhamdi, Rachid Guerraoui, and Sébastien Rouault. Distributed momentum for byzantine-resilient learning. *arXiv preprint arXiv:2003.00010*, 2020.

[34] Iyanuoluwa Emiola, Laurent Njilla, and Chinwendu Enyioha. On distributed optimization in the presence of malicious agents. *arXiv preprint arXiv:2101.09347*, 2021.

[35] Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proceedings of the twenty-second ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 211–222, 2003.

[36] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE, 1987.

[37] Christodoulos A Floudas and Panos M Pardalos. *Encyclopedia of optimization*. Springer Science & Business Media, 2008.

[38] Matthias Függer and Thomas Nowak. Fast multidimensional asymptotic and approximate consensus. *arXiv preprint arXiv:1805.04923*, 2018.

[39] Weinan Gao, Jingqin Gao, Kaan Ozbay, and Zhong-Ping Jiang. Reinforcement-learning-based cooperative adaptive cruise control of buses in the lincoln tunnel corridor with time-varying topology. *IEEE Transactions on Intelligent Transportation Systems*, 20(10):3796–3805, 2019.

[40] Pontus Giselsson and Anders Rantzer. *Large-scale and distributed optimization*, volume 2227. Springer, 2018.

[41] Chris Godsil and Gordon F Royle. *Algebraic graph theory*, volume 207. Springer Science & Business Media, 2001.

[42] Rachid Guerraoui, Nirupam Gupta, Rafaël Pinot, Sébastien Rouault, and John Stephan. Differential privacy and byzantine resilience in sgd: Do they add up? *arXiv preprint arXiv:2102.08166*, 2021.

[43] Nirupam Gupta and Nitin H Vaidya. Byzantine fault tolerant distributed linear regression. *arXiv preprint arXiv:1903.08752*, 2019.

[44] Nirupam Gupta and Nitin H Vaidya. Randomized reactive redundancy for byzantine fault-tolerance in parallelized learning. *arXiv preprint arXiv:1912.09528*, 2019.

[45] Nirupam Gupta and Nitin H Vaidya. Fault-tolerance in distributed optimization: The case of redundancy. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 365–374, 2020.

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

[47] Nirupam Gupta and Nitin H. Vaidya. Byzantine fault-tolerance in peer-to-peer distributed gradient-descent, 2021.

[48] Nirupam Gupta, Thinh T Doan, and Nitin H Vaidya. Byzantine fault-tolerance in decentralized optimization under minimal redundancy. *arXiv preprint arXiv:2009.14763*, 2020.

[49] Nirupam Gupta, Shuo Liu, and Nitin H Vaidya. Byzantine fault-tolerant distributed machine learning using stochastic gradient descent (sgd) and norm-based comparative gradient elimination (cge). *arXiv preprint arXiv:2008.04699*, 2020.

[50] Richard W Hamming. *Coding and information theory*. Prentice-Hall, Inc., 1986.

[51] Lie He, Sai Praneeth Karimireddy, and Martin Jaggi. Byzantine-robust learning on heterogeneous datasets via resampling. *arXiv preprint arXiv:2006.09365*, 2020.

[52] Lie He, Sai Praneeth Karimireddy, and Martin Jaggi. Secure byzantine-robust machine learning. *arXiv preprint arXiv:2006.04747*, 2020.

[53] Xin He, Wee Peng Tay, Lei Huang, Meng Sun, and Yi Gong. Privacy-aware sensor network via multilayer nonlinear processing. *IEEE Internet of Things Journal*, 6(6):10834–10845, 2019.

[54] Peter J Huber. A robust version of the probability ratio test. *The Annals of Mathematical Statistics*, pages 1753–1758, 1965.

[55] Franck Iutzeler, Pascal Bianchi, Philippe Ciblat, and Walid Hachem. Asynchronous distributed optimization using a randomized alternating direction method of multipliers. In *52nd IEEE conference on decision and control*, pages 3671–3676. IEEE, 2013.

[56] Prateek Jain and Purushottam Kar. Non-convex optimization for machine learning. *arXiv preprint arXiv:1712.07897*, 2017.

[57] KR Jayaram, Archit Verma, Ashish Verma, Gegi Thomas, and Colin Sutcher-Shepard. Mystiko: Cloud-mediated, private, federated gradient descent. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 201–210. IEEE, 2020.

[58] Chi Jin, Rong Ge, Praneeth Netrapalli, Sham M Kakade, and Michael I Jordan. How to escape saddle points efficiently. In *International Conference on Machine Learning*, pages 1724–1732. PMLR, 2017.

[59] Bhavya Kailkhura, Swastik Brahma, and Pramod K Varshney. Consensus based detection in the presence of data falsification attacks. *arXiv preprint arXiv:1504.03413*, 2015.

[60] Sai Praneeth Karimireddy, Lie He, and Martin Jaggi. Learning from history for byzantine robust optimization. *arXiv preprint arXiv:2012.10333*, 2020.

[61] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, pages 45–58, 2007.

[62] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3): 382–401, 1982.

[63] Heath J LeBlanc, Haotian Zhang, Xenofon Koutsoukos, and Shreyas Sundaram. Resilient asymptotic consensus in robust networks. *IEEE Journal on Selected Areas in Communications*, 31(4):766–781, 2013.

[64] Jason D Lee, Max Simchowitz, Michael I Jordan, and Benjamin Recht. Gradient descent converges to minimizers. *arXiv preprint arXiv:1602.04915*, 2016.

[65] Gangqiang Li, Sissi Xiaoxiao Wu, Shengli Zhang, and Qiang Li. Detect insider attacks using cnn in decentralized optimization. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8758–8762. IEEE, 2020.

[66] Liping Li, Wei Xu, Tianyi Chen, Georgios B Giannakis, and Qing Ling. Rsa: Byzantine-robust stochastic aggregation methods for distributed learning from heterogeneous datasets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1544–1551, 2019.

[67] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling

distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, pages 583–598, 2014.

[68] Shuo Liu, Nirupam Gupta, and Nitin H Vaidya. Approximate byzantine fault-tolerance in distributed optimization. *arXiv preprint arXiv:2101.09337*, 2021.

[69] Yang Lu and Minghui Zhu. Privacy preserving distributed optimization using homomorphic encryption. *Automatica*, 96:314–325, 2018.

[70] Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.

[71] Stefano Marano, Vincenzo Matta, and Lang Tong. Distributed detection in the presence of byzantine attacks. *IEEE Transactions on Signal Processing*, 57(1):16–29, 2008.

[72] Keith Marzullo. Tolerating failures of continuous-valued sensors. *ACM Transactions on Computer Systems (TOCS)*, 8(4):284–304, 1990.

[73] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282. PMLR, 2017.

[74] Hammurabi Mendes, Maurice Herlihy, Nitin Vaidya, and Vijay K Garg. Multidimensional agreement in byzantine systems. *Distributed Computing*, 28(6): 423–441, 2015.

[75] El Mahdi El Mhamdi, Rachid Guerraoui, and Sébastien Rouault. The hidden vulnerability of distributed learning in byzantium. *arXiv preprint arXiv:1802.07927*, 2018.

[76] Mohammad Naseri, Jamie Hayes, and Emiliano De Cristofaro. Toward robustness and privacy in federated learning: Experimenting with local and central differential privacy. *arXiv preprint arXiv:2009.03561*, 2020.

[77] Angelia Nedić and Ji Liu. Distributed optimization for control. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:77–103, 2018.

[78] Angelia Nedic and Asuman Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 54(1): 48–61, 2009.

[79] Angelia Nedić, Alex Olshevsky, and Wei Shi. Improved convergence rates for distributed resource allocation. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 172–177. IEEE, 2018.

[80] Angelia Nedich et al. Convergence rate of distributed averaging dynamics and optimization in networks. *Foundations and Trends® in Systems and Control*, 2(1):1–100, 2015.

[81] Fabio Pasqualetti, Antonio Bicchi, and Francesco Bullo. Consensus computation in unreliable networks: A system theoretic approach. *IEEE Transactions on Automatic Control*, 57(1):90–104, 2011.

[82] Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. Robust aggregation for federated learning. *arXiv preprint arXiv:1912.13445*, 2019.

[83] Adarsh Prasad, Arun Sai Suggala, Sivaraman Balakrishnan, and Pradeep Ravikumar. Robust estimation via robust gradient estimation. *arXiv preprint arXiv:1802.06485*, 2018.

[84] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[85] Shashank Rajput, Hongyi Wang, Zachary Charles, and Dimitris Papailiopoulos. Detox: A redundancy-based framework for faster and more robust gradient aggregation. *arXiv preprint arXiv:1907.12205*, 2019.

[86] Nikhil Ravi and Anna Scaglione. Detection and isolation of adversaries in decentralized optimization for non-strongly convex objectives. *IFAC-PapersOnLine*, 52(20):381–386, 2019.

[87] Nikhil Ravi, Anna Scaglione, and Angelia Nedić. A case of distributed optimization in adversarial environment. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5252–5256. IEEE, 2019.

[88] Netanel Raviv, Itzhak Tamo, Rashish Tandon, and Alexandros G Dimakis. Gradient coding from cyclic mds codes and expander graphs. *IEEE Transactions on Information Theory*, 66(12):7475–7489, 2020.

[89] Wei Ren and Yongcan Cao. *Distributed coordination of multi-agent networks: emergent problems, models, and issues*. Springer Science & Business Media, 2010.

[90] Peter J Rousseeuw. Multivariate estimation with high breakdown point. *Mathematical statistics and applications*, 8(283-297):37, 1985.

[91] Ali H Sayed. Adaptation, learning, and optimization over networks. *Foundations and Trends in Machine Learning*, 7(ARTICLE):311–801, 2014.

[92] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*, pages 1310–1321, 2015.

[93] Yasser Shoukry, Konstantinos Gatsis, Amr Alanwar, George J Pappas, Sanjit A Seshia, Mani Srivastava, and Paulo Tabuada. Privacy-aware quadratic optimization using partially homomorphic encryption. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 5053–5058. IEEE, 2016.

[94] Jinhyun So, Başak Güler, and A Salman Avestimehr. Byzantine-resilient secure federated learning. *IEEE Journal on Selected Areas in Communications*, 2020.

[95] Shuang Song, Kamalika Chaudhuri, and Anand D Sarwate. Stochastic gradient descent with differentially private updates. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 245–248. IEEE, 2013.

[96] Kunal Srivastava and Angelia Nedic. Distributed asynchronous constrained stochastic optimization. *IEEE Journal of Selected Topics in Signal Processing*, 5(4):772–790, 2011.

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

[98] Lili Su and Nitin Vaidya. Byzantine multi-agent optimization: Part II. *arXiv preprint arXiv:1507.01845*, 2015.

[99] Lili Su and Nitin Vaidya. Fault-tolerant multi-agent optimization: Part III. *arXiv preprint arXiv:1509.01864*, 2015.

[100] Lili Su and Nitin H Vaidya. Fault-tolerant distributed optimization (Part IV): Constrained optimization with arbitrary directed networks. *arXiv preprint arXiv:1511.01821*, 2015.

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

[102] Lili Su and Nitin H Vaidya. Byzantine-resilient multi-agent optimization. *IEEE Transactions on Automatic Control*, 2020.

[103] Shreyas Sundaram and Bahman Gharesifard. Consensus-based distributed optimization with malicious nodes. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 244–249. IEEE, 2015.

[104] Shreyas Sundaram and Bahman Gharesifard. Distributed optimization under adversarial nodes. *IEEE Transactions on Automatic Control*, 64(3):1063–1076, 2018.

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

[106] Fengyi Tang, Wei Wu, Jian Liu, Huimei Wang, and Ming Xian. Privacy-preserving distributed deep learning via homomorphic re-encryption. *Electronics*, 8(4):411, 2019.

[107] Lewis Tseng and Nitin Vaidya. Iterative approximate consensus in the presence of byzantine link failures. In *International Conference on Networked Systems*, pages 84–98. Springer, 2014.

[108] John Tsitsiklis, Dimitri Bertsekas, and Michael Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986.

[109] John Nikolas Tsitsiklis. Problems in decentralized decision making and computation. Technical report, Massachusetts Inst of Tech Cambridge Lab for Information and Decision Systems, 1984.

[110] Nitin Vaidya. Matrix representation of iterative approximate byzantine consensus in directed graphs. *arXiv preprint arXiv:1203.1888*, 2012.

[111] Nitin H Vaidya. Iterative byzantine vector consensus in incomplete graphs. In *International conference on distributed computing and networking*, pages 14–28. Springer, 2014.

[112] Nitin H Vaidya and Vijay K Garg. Byzantine vector consensus in complete graphs. In *Proceedings of the 2013 ACM symposium on Principles of distributed computing*, pages 65–73, 2013.

[113] Sissi Xiaoxiao Wu, Hoi-To Wai, Anna Scaglione, Angelia Nedić, and Amir Leshem. Data injection attack on decentralized optimization. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3644–3648. IEEE, 2018.

[114] Sissi Xiaoxiao Wu, Gangqiang Li, Shengli Zhang, and Xiaohui Lin. Detection of insider attacks in distributed projected subgradient algorithms. *arXiv preprint arXiv:2101.06917*, 2021.

[115] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Generalized byzantine-tolerant sgd. *arXiv preprint arXiv:1802.10116*, 2018.

[116] Cong Xie, Oluwasanmi Koyejo, and Indranil Gupta. Phocas: dimensional byzantine-resilient stochastic gradient descent. *arXiv preprint arXiv:1805.09682*, 2018.

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

[118] Liyang Xie, Inci M Baytas, Kaixiang Lin, and Jiayu Zhou. Privacy-preserving distributed multi-task learning with asynchronous updates. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1195–1204, 2017.

[119] Tao Yang, Xinlei Yi, Junfeng Wu, Ye Yuan, Di Wu, Ziyang Meng, Yiguang Hong, Hong Wang, Zongli Lin, and Karl H Johansson. A survey of distributed optimization. *Annual Reviews in Control*, 47:278–305, 2019.

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

[121] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. Defending against saddle point attack in byzantine-robust distributed learning. In *International Conference on Machine Learning*, pages 7074–7084. PMLR, 2019.

[122] Ye Yuan, Hai-Tao Zhang, Yue Wu, Tao Zhu, and Han Ding. Bayesian learning-based model-predictive vibration control for thin-walled workpiece machining processes. *IEEE/ASME transactions on mechatronics*, 22(1):509–520, 2016.

[123] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[124] Ruiliang Zhang and James Kwok. Asynchronous distributed admm for consensus optimization. In *International conference on machine learning*, pages 1701–1709. PMLR, 2014.

[125] Chengcheng Zhao, Jianping He, and Qing-Guo Wang. Resilient distributed optimization algorithm against adversarial attacks. *IEEE Transactions on Automatic Control*, 65(10):4308–4315, 2019.

[126] Minghui Zhu and Sonia Martínez. *Distributed optimization-based control of multi-agent networks in complex environments*. Springer, 2015.

A Appendix: Definition of some mathematical concepts

In this appendix, we note some definitions of the mathematical concepts mentioned without explanation in the survey for readers' reference.

A.1 Hausdorff distance

To begin with, the *distance* of two points x, y in d-dimensional Euclidean space $\mathbb{R}^d$ induced by a given norm $\|\cdot\|$ is

$$\operatorname{dist}(x,\, y) = \|x - y\|. \tag{30}$$

The distance between a point $x \in \mathbb{R}^d$ and a set $Y \subset \mathbb{R}^d$ can then be defined as

$$\operatorname{dist}(x,\, Y) = \inf_{y \in Y} \|x - y\|. \tag{31}$$

The Hausdorff distance between two set $X, Y \subset \mathbb{R}^d$ is then defined as follows:

$$\operatorname{dist}(X,\, Y) = \max\left\{ \sup_{x \in X} \operatorname{dist}(x,\, Y),\ \sup_{y \in Y} \operatorname{dist}(y,\, X) \right\}. \tag{32}$$

Note that the definition of 32 also applies to the distance between two points, or between a point and a set, if viewing a point as a set with one element.

A.2 Diminishing step size

A diminishing step size η_t in the learning process is an sequence satisfying the following conditions [8]:

$$\sum_{t=1}^{\infty} \eta_t = \infty, \text{ and } \sum_{t=1}^{\infty} \eta_t^2 < \infty. \tag{33}$$

B Appendix: Summary table

Table 2: Summary of gradient filters. Cells with "-" indicates that information is not provided in the original resource or other research to the best of our knowledge.

Gradient filter	**Type**	**Outputs an input vector**	**Time complexity at agents per iteration**	**Is (α, f)-resilient**	**Originally designed for**	**Fault-tolerance threshold**	**Comments**
Krum	Angle-based	Yes	$O(n^2 d)$	Yes	D-SGD	$f < (n-2)/2$	
m-krum		No	$O(n^2 d)$	Yes	D-SGD		
Multi-Krum		No	$O(n^2 d)$	Yes	D-SGD		
Coordinate-wise median	Coordinate-wise	No	$O(nd)$	Yes	D-SGD	See [120]	
Coordinate-wise trimmed mean		No	$O(nd)$	Yes	D-SGD	$f < n/2$	
Phocas		No	$O(nd)$	-	D-SGD	$f < n/2$	
Mean around median		No	$O(nd)$	Yes	D-SGD	$f < n/2$	
Geometric median	Median-based	No	$O\left(nd \log^3 \frac{1}{\epsilon}\right)$	-	-	-	ϵ as approx. param.
Median of means		No	$O\left(nd + fd \log^3 \frac{1}{\epsilon}\right)$	-	D-SGD	$f < n/2$	
MDA		No	$O\left(\binom{n}{f} + n^2 d\right)$	Yes	D-SGD	$f \leq (n-1)/2$	
CGC	Norm-based	No	$O\left((n+f)d + n \log n\right)$	-	linear regression	$f < n/2$	
CGE		No	$O\left(n(\log n + d)\right)$	-	DGD	$f < n/2$	
Bulyan	Meta-method	No	$O((n-2f)C + nd)$	Yes	D-SGD	$f \leq (n-3)/4$	C is the complexity of a filter