Identifying Reliable Machines for Distributed Matrix-Vector Multiplication

Sarthak Jain, Martina Cardone, Soheil Mohajer
University of Minnesota, Minneapolis, MN 55455, USA, Email: {jain0122, mcardone, soheil}@umn.edu

***Abstract*—This paper considers a distributed computing framework, where the task of T matrix-vector products is distributed among n worker machines. External adversaries have access to a subset $\mathcal{L}$ (the cardinality of which is $|\mathcal{L}|$) of these machines, and can maliciously perturb the result of each of their computations with probability α. To correctly recover each matrix-vector product, the master has to identify a set (of a fixed cardinality) of 'unattacked' worker machines. Towards this end, this work proposes four schemes that aim at performing such an identification. These schemes are analyzed and compared under different regimes of $(|\mathcal{L}|, \alpha)$ for the two cases when $|\mathcal{L}|$ is (1) known or (2) unknown at the master.**

I. Introduction

Matrix-vector multiplication is one of the most common operations employed on large datasets and it is the driving machinery behind most of the popular deep learning architectures [1]. In this work, we consider a distributed computing setting for performing a batch of T matrix-vector multiplications, where a subset of the worker machines can be attacked by adversaries. We assume that these attackers are non-communicating/non-colluding[1]. This assumption is reasonable when the worker machines are geographically distributed. For example, in online crowd-sourcing [3], [4], out of a potentially large pool of internet users, the results from some of the users might not be reliable. For such a distributed computing setting, [5] derived a condition under which a matrix-vector product can be correctly recovered by the master with a very high probability. The condition is $n - |\mathcal{L}| > k$, where n is the total number of worker machines, $\mathcal{L}$ is the subset of worker machines that the adversaries have access to, and k is the factor by which matrices are split across the machines. In this work, we extend this algorithm for the case when $T > 1$ matrix-vector multiplications need to be computed. In particular, during the t-th matrix-vector multiplication for $t \in \{1, 2, \ldots, T\} := [T]$, each worker machine in the set $\mathcal{L}$ is attacked with probability α, i.e., even though the adversaries have access to all the machines in $\mathcal{L}$, they can decide to attack (i.e., perturb the result of their computation) only some of them. During each t-th matrix-vector multiplication, the master requires results from $k+1$ unattacked worker machines for decoding purposes. We propose four schemes that can be used by the master to find such a set of $k+1$ machines: (1) Not-Identify-Restart (NIR), (2) Not-Identify-Continue (NIC), (3) Identify-Restart (IR) and (4) Identify-Continue (IC), and analyze/compare them under different regimes of $(|\mathcal{L}|, \alpha)$. We consider two cases depending on whether $|\mathcal{L}|$ is known or not at the master and compare the schemes for both cases using Monte-Carlo simulations. Furthermore, We also theoretically analyze the IR scheme when $|\mathcal{L}|$ is known at the master.

Related Work. The algorithm proposed in [5] (for distributed matrix-vector multiplication with non-colluding attackers) employs a coding scheme for distributing the sub-tasks among worker machines. Coding has been extensively used in distributed computing settings to: (i) deal with stragglers or slow worker machines [6]–[12]; (ii) reduce communication cost and bandwidth usage [8], [13], [14]; (iii) ensure data privacy from worker machines [2], [12], [14], [15]; and (iv) ensure resiliency against collaborative attackers [2], [12], [15], [16]. Identifying attacked worker machines is an important aspect of distributed computing, especially when a large number of matrix-vector multiplications have to be computed. If identified, the attacked machines can indeed be removed from the system for future computations. The authors in [5] proposed a strategy based on group testing [17]–[19] for the identification of the attacked machines. However, as we will show, since we consider the case where multiple matrix-vector products need to be computed and not all the machines in $\mathcal{L}$ may be attacked during a particular matrix-vector multiplication, identifying and removing all the attacked machines in every time-slot might not be efficient for the run-time of the algorithm.

Paper Organization. Section II illustrates the framework proposed in [5] and formulates the problem. Section III describes our proposed schemes for identifying a set of unattacked worker machines. Section IV and Section V analyze and compare the performance of these schemes for the two cases when $|\mathcal{L}|$ is known and unknown, respectively, at the master.

II. Problem Statement

We consider the problem of distributed matrix-vector product computation[2]. The master node has to compute T matrix-vector products $\mathbf{B} \cdot \mathbf{x}_t$ for $t \in [T]$, where $\mathbf{B} \in \mathbb{F}_q^{r \times c}$ and $\mathbf{x}_t \in \mathbb{F}_q^{c \times 1}$. The duration of time needed to compute $\mathbf{B} \cdot \mathbf{x}_t$ is called *time-slot* t. In order to improve the run-time of these matrix-vector products, the task is divided among n worker machines, which form the set $\mathcal{N} = [n]$. In particular, the $\mathbf{x}_t$'s are first stored in the n machines, and each machine is required to compute matrix-vector products for smaller matrices with

This research was supported in part by the U.S. National Science Foundation under Grants CCF-1907785 and CCF-1849757.

[1]This is different from the setting considered in [2] where the attackers can collaborate with each other.

[2]Operations take place over a finite field of dimension q.

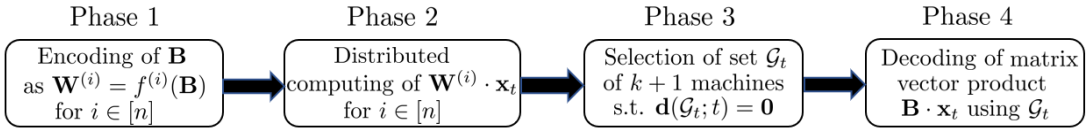

Fig. 1: Four main phases in distributed matrix-vector product computation.

$s = r/k$ rows (assuming k divides r) and then send the result back to the master. To this end, the master first divides the matrix $\mathbf{B}$ into smaller sub-matrices by using a set of encoding functions $f^{(i)} : \mathbb{F}_q^{r\times c} \to \mathbb{F}_q^{s\times c}, i \in [n]$. The master then distributes these smaller matrices among the n machines, and the ith worker machine receives

$$\mathbf{W}^{(i)} = f^{(i)}(\mathbf{B}). \tag{1}$$

For each $t \in [T]$, the ith worker machine then computes the matrix-vector product $\mathbf{a}_t^{(i)} = \mathbf{W}^{(i)} \cdot \mathbf{x}_t$, and sends the result back to the master [8]. The master is able to recover the original matrix-product from the output of any group $\mathcal{U} \subseteq \mathcal{N}$ of k machines. Let $\mathcal{P}(n,k)$ denote the collection of all k-subsets of $[n]$. Then, there exists a decoding function $g : \mathcal{P}(n,k) \times \left(\mathbb{F}_q^{s\times 1}\right)^k \to \mathbb{F}_q^{r\times 1}$ that satisfies $g\left(\mathcal{U}, \left\{\mathbf{a}_t^{(i)} : i \in \mathcal{U}\right\}\right) = \mathbf{B} \cdot \mathbf{x}_t$.

A subset $\mathcal{L} \subseteq \mathcal{N}$ of fixed but unknown *unreliable* machines, with $|\mathcal{L}| = L$, is accessed by a set of non-colluding/non-communicating attackers, that may change the result of the local computation at the corresponding machine with probability α in each time-slot $t \in [T]$. Hence, if $i \in \mathcal{L}$, then

$$\mathbf{a}_t^{(i)} = \begin{cases} \mathbf{W}^{(i)} \cdot \mathbf{x}_t & \text{with probability } 1-\alpha, \\ \mathbf{Z}_t^{(i)} & \text{with probability } \alpha, \end{cases}$$

where $\mathbf{Z}_t^{(i)}$ is a random noise, independent of $\mathbf{Z}_t^{(j)}$ for $j \neq i$, since the attackers are non-colluding. We let $\mathcal{A}_t \subseteq \mathcal{L}$ be the set of unreliable machines that are attacked at time slot $t \in [T]$.

The coding scheme proposed in [5] guarantees (with very high probability) a correct recovery of each matrix-vector multiplication whenever $n - k > L$. In particular, in the presence of unreliable machines, the master needs to first identify a subset of k unattacked machines in each time slot, and then apply the corresponding decoding function on their output to retrieve the desired product. To this end, as pointed out in [5], the master can perform a parity-check on the output of a set of $k+1$ machines to check if there is an attacked machine in that set or not. More precisely, there exists a parity-check function $\mathbf{d} : \mathcal{P}(n,k+1) \times \mathbb{F}_q^{s(k+1)\times 1} \to \mathbb{F}_q^{s\times 1}$, such that for each group $\mathcal{U} \subseteq \mathcal{N}$ with $|\mathcal{U}| = k+1$ we have $\mathbf{d}\left(\mathcal{U}, \left\{\mathbf{a}_t^{(i)} : i \in \mathcal{U}\right\}\right) = \mathbf{0}$ if and only if[3] $\mathcal{U} \cap \mathcal{A}_t = \varnothing$. Once an attacked machine is identified, the master can mark it as unreliable and never use it in the future. With this, we can denote by $\mathcal{N}_t$ the set of machines which are not marked as unreliable by time-slot $t \in [T]$. Hence, we have $\mathcal{N}_t \subseteq \mathcal{N}_{t-1} \subseteq \cdots \subseteq \mathcal{N}_1 = [n]$. It is worth noting that $\mathbf{d}\left(\mathcal{U}, \left\{\mathbf{a}_t^{(i)} : i \in \mathcal{U}\right\}\right)$ depends on the set $\mathcal{U}$, as well as on the behavior of each individual machine in $\mathcal{U}$ at time-slot t. For the sake of simplicity, we denote the parity-check function by $\mathbf{d}(\mathcal{U}; t)$ in the rest of the paper.

[3]This parity check is accurate when $q \gg 1$, which is our assumption in this work. For finite q, a probability of error proportional to $\frac{1}{q}$ has to be accounted for in the analysis.

Remark 1. *From a time-complexity point of view, the algorithm above can be divided into the four major phases shown in Fig. 1. (i) In Phase* 1*, the total time complexity of computing* $\mathbf{W}^{(i)}$ *in* (1) *for all* $i \in [n]$ *is* $O(nrc)$*. However, this computation is done only once, and we assume that* T *is large enough to almost nullify this one-time overhead cost. (ii) Phase* 2 *is carried out in parallel among the* n *machines and costs* $O(rc/k)$*. (iii) In Phase* 3*, each parity-check costs* $O(r)$ *[5]. The focus of this work is on finding algorithms to reduce the number of parity-checks. Although this cost might be lower than the cost of Phase* 2*, the parity-checks have to be performed at the master (and not at the worker machines). This can cause a bottleneck at the master if not efficiently handled, depending on the master's computing capabilities. (iv) Finally, the decoding step in Phase* 4 *costs* $O(rk)$ *computations.*

III. Unattacked Machines Identification Schemes

In Phase 3 of time-slot $t \in [T]$, the master aims to find a set $\mathcal{G}_t \subseteq \mathcal{N}_t$ of $k+1$ unattacked worker machines, i.e., for which $\mathbf{d}(\mathcal{G}_t; t) = \mathbf{0}$. Towards this end, the master could use one of the following two strategies.

- **Restart:** In time-slot $t \in [T-1]$, the master randomly samples (with replacement) sets of size $k+1$ from the set $\mathcal{N}_t$ of machines, until it finds an unattacked set $\mathcal{G}_t$ with $\mathbf{d}(\mathcal{G}_t; t) = \mathbf{0}$.
- **Continue:** In time-slot $t \geq 2$, the master first uses the unattacked group of the previous time-slot (i.e., $\mathcal{G}_{t-1}$), and checks $\mathbf{d}(\mathcal{G}_{t-1}; t)$. If $\mathbf{d}(\mathcal{G}_{t-1}; t) = \mathbf{0}$, then it *continues* with this group and sets $\mathcal{G}_t = \mathcal{G}_{t-1}$. Otherwise, it starts randomly sampling sets of size $k+1$ again to find an unattacked set.

Once the master has found an unattacked set $\mathcal{G}_t$, it has the option of identifying the attacked machines and marking them as unreliable for future time-slots. Assume that the master tries $\mathcal{U}_{t,1}, \mathcal{U}_{t,2}, \ldots, \mathcal{U}_{t,m}$ for which it gets $\mathbf{d}(\mathcal{U}_{t,i}; t) \neq \mathbf{0}$ for $i \in [m]$, until finding $\mathcal{U}_{t,m+1}$ satisfying $\mathbf{d}(\mathcal{U}_{t,m+1}; t) = \mathbf{0}$. Then, it could choose one of the two following strategies.

- **Identify:** The master identifies all the attacked machines in the set $\mathcal{U}_t^m \setminus \mathcal{U}_{t,m+1}$ and removes them from future time-slots, where $\mathcal{U}_t^m \triangleq \bigcup_{j=1}^m \mathcal{U}_{t,j}$. In order to check if a machine $i \in \mathcal{U}_t^m \setminus \mathcal{U}_{t,m+1}$ is attacked or not, the master computes $\mathbf{d}(\mathcal{U}; t)$ for $\mathcal{U} = \{i\} \cup \mathcal{U}'$, where $\mathcal{U}'$ is an arbitrary subset of $\mathcal{U}_{t,m+1}$ of size k. Since the k machines in the set $\mathcal{U}'$ are unattacked, then $\mathbf{d}(\mathcal{U}; t) = \mathbf{0}$ if and only if i is unattacked.
- **Not-Identify:** The master never tries to identify or remove the attacked worker machines in any time-slot.

Based on these options, we study four schemes, namely, Not-Identify-Restart (NIR), Not-Identify-Continue (NIC), Identify-Restart (IR), and Identify-Continue (IC). These schemes are analyzed for the cases of known and unknown L at the master, which are significantly different. To see this, assume L is known at the master and all L unreliable machines are identified by time-slot t. Then, it can safely use any set of the remaining machines for future time-slots, with no further parity-checks. However, this is not possible if L is unknown.

Remark 2. *In our schemes, we verify a set of $k+1$ machines at a time, which costs $O(r)$. Differently, in [15], one machine is verified at a time, which costs $O(c+r/k)$ and hence, it takes at least $O(ck+r)$ computations to find k unattacked machines. This becomes very expensive when c is either comparable to r or greater than r. Moreover, when L is small, our schemes will require very few parity-checks, whereas the scheme in [15] will still need at least k parity-checks. Also, when L is unknown at the master, once our schemes* IR, NIC *and* IC *converge, the master needs to perform exactly one parity-check in all the future time-slots. Differently, with the scheme in [15], the master will need at least k parity-checks in every time-slot.*

IV. L KNOWN AT THE MASTER

Let $\mathsf{F}_{\text{sch}}(n, L, \alpha, T)$ be the expected value of the total number of parity-check evaluations (see Phase 3 in Fig. 1) in T time-slots for scheme $\text{sch} \in \{\text{NIR}, \text{NIC}, \text{IR}, \text{IC}\}$. Moreover, we define $\tilde{\mathsf{F}}_{\text{sch}}(n, L, \alpha) = \lim_{T\to\infty} \mathsf{F}_{\text{sch}}(n, L, \alpha, T)$.

Remark 3. *For every scheme* $\text{sch} \in \{\text{NIR}, \text{NIC}, \text{IC}\}$*, we have $\tilde{\mathsf{F}}_{\text{sch}}(n, L, \alpha) \to \infty$ for $0 < \alpha < 1$. This is clear for* NIR *and* NIC *since the master does not identify the unreliable machines and hence, it has to evaluate one parity-check function per time-slot. This leads to infinitely many parity-checks as $T \to \infty$. For* IC*, there is a chance that the master identifies all the L unreliable machines up to some time-slot t and hence, it can trust all the remaining machines in future time-slots without further checking. However, assume that the master randomly selects a group of reliable machines $\mathcal{G}$ at time t_0 before identifying all the L unreliable ones. Then, it will continue with $\mathcal{G}$ for which it will need to evaluate $\mathbf{d}(\mathcal{G}; t)$ for all $t \geq t_0$, which leads to infinitely many parity-checks.*

IR is the only scheme for which $\tilde{\mathsf{F}}_{\text{IR}}(n, L, \alpha)$ converges to a finite value. We next analyze this scheme theoretically. In IR, the master repeatedly keeps sampling sets of size $k+1$ from $\mathcal{N}_t$ until it finds a set $\mathcal{G}_t$ of $k+1$ machines satisfying $\mathbf{d}(\mathcal{G}_t; t) = \mathbf{0}$. Let m be the number of sets sampled before such a set is found, that is, $\mathbf{d}(\mathcal{U}_{t,j}; t) \neq \mathbf{0}$ for all $j \in [m]$ and $\mathcal{G}_t = \mathcal{U}_{t,m+1}$. The master already knows that the $k+1$ machines in $\mathcal{G}_t$ are unattacked at time slot t. To check if a given machine in $\mathcal{U}_t^{m+1} \setminus \mathcal{G}_t$ is attacked or not, the master combines it with any k (out of the $k+1$) machines of the unattacked set $\mathcal{G}_t$ and computes the parity of this set[4]. Then, the given machine is attacked if and only if the parity is not $\mathbf{0}$. Hence, the total number of parity checks in time-slot t is $m + 1 + |\mathcal{U}_t^{m+1}| - (k+1)$. The following theorem (proof in Appendix A) and corollary provide recursive closed-form expressions for $\mathsf{F}_{\text{IR}}(n, L, \alpha, T)$ and $\tilde{\mathsf{F}}_{\text{IR}}(n, L, \alpha)$.

Theorem 1. *Assume that L is known at the master, and consider* IR*. Then, $\mathsf{F}_{\text{IR}}(n, L, \alpha, T)$ is given in* (2)*, at the top of the next page, where: (i) $p_{n,\ell} = \binom{n-\ell}{k+1}/\binom{n}{k+1}$; (ii) $p_{n,L,\alpha} = \sum_{\ell=0}^{L} \binom{L}{\ell} \alpha^{\ell}(1-\alpha)^{L-\ell} p_{n,\ell}$; and (iii) $H(n, \ell, k)$ and $J(\ell, b, n, k)$ are defined in* (9) *and* (11)*, respectively.*

Corollary 1. *As T grows unboundedly, $\mathsf{F}_{\text{IR}}(n, L, \alpha, T)$ converges to $\tilde{\mathsf{F}}_{IR}(n, L, \alpha)$, which is defined recursively in* (3) *at the top of the next page, where*

$$E(n, \ell, k, \alpha) = H(n, \ell, k) + \frac{1}{1 - p_{n,L,\alpha}} - (k+1) + \frac{1}{p_{n,\ell}}.$$

Remark 4. *In Fig. 2, we compare our schemes for finite T. Each scheme is run for $T = 100$ time-slots for parameters $(n, k, \alpha) = (24, 3, 0.5)$, where the average (over 5000 random executions) numbers of parity-checks are plotted versus L. Fig. 2 shows that, even for finite (but large enough) values of T,* IR *outperforms the others. However, as L increases to $n-k-1$,* IR *and* IC *perform similarly. Fig. 3 compares* (3) *against Monte Carlo numerical results from* 5000 *executions of* IR *for $n = 10$, $k = 3$, $L = 4$, and $T = 100$. The 5th and 95th percentiles of the number of parity-check evaluations are plotted for different values of α. From Fig. 3, we observe that $\tilde{\mathsf{F}}_{\text{IR}}(n, L, \alpha)$ in* (3)*, which is the exact expression for $T \to \infty$, well approximates $\mathsf{F}_{\text{IR}}(n, L, \alpha, T)$ for finite values of T.*

V. L UNKNOWN AT THE MASTER

When the master does not know L, it needs to perform at least one parity-check for each $t \in [T]$. This implies that the expected total number of parity-checks in T time-slots, denoted as $\mathsf{G}_{\text{sch}}(n, L, \alpha, T)$, never converges as $T \to \infty$. However, as we argue next, $\delta\mathsf{G}_{\text{sch}}(n, L, \alpha, T) = \mathsf{G}_{\text{sch}}(n, L, \alpha, T) - T$ converges as $T \to \infty$ for $\text{sch} \in \{\text{NIC}, \text{IR}, \text{IC}\}$.

- NIC: There always exists a finite time-slot t in which the master samples a set $\mathcal{G}_t$ with $\mathcal{G}_t \cap \mathcal{L} = \varnothing$. After this, in each time-slot, the master continues with $\mathcal{G}_t$ forever. However, since the master does not know that $\mathcal{G}_t$ is reliable, it has to check the parity for $\mathcal{G}_t$ for each time-slot after t. Therefore, $\delta\mathsf{G}_{\text{NIC}}(n, L, \alpha, T)$ converges to a finite value as $T \to \infty$.
- IR: There always exists a finite time-slot t by the end of which the master will identify and remove all the machines in $\mathcal{L}$. After this, in each time-slot, the master performs only one parity-check as any sampled set will be reliable. Thus, $\delta\mathsf{G}_{\text{IR}}(n, L, \alpha, T)$ converges to a finite value as $T \to \infty$.
- IC: A finite time-slot t always exists in which the master either finds a set $\mathcal{G}_t$ such that $\mathcal{G}_t \cap \mathcal{L} = \varnothing$, or identifies (and removes) all the machines in $\mathcal{L}$ by the end of it. In both cases, the master will continue with a group of reliable machines, and has to perform only one parity-check in each slot after t. Hence, as $T \to \infty$, $\delta\mathsf{G}_{\text{IC}}(n, L, \alpha, T)$ converges to a finite value.

[4]One can propose more efficient schemes, such as group-testing based approaches to identify the attacked machines [5]. However, in this paper, we consider a simple and naive scheme for the sake of brevity.

$$\mathsf{F}_{\text{IR}}\left(n,L,\alpha,T\right)=\sum_{\ell=1}^{L}\binom{L}{\ell}\alpha^{\ell}(1-\alpha)^{L-\ell}\Bigg[\sum_{t=1}^{T}\left(p_{n,L,\alpha}^{t-1}(1-p_{n,\ell})\left(H\left(n,\ell,k\right)+t-(k+1)+\frac{1}{p_{n,\ell}}\right)\right)$$
$$+\sum_{b=1}^{\ell}p_{n,\ell}J(\ell,b,n,k)\left(\sum_{t=1}^{T}p_{n,L,\alpha}^{t-1}\mathsf{F}_{\text{IR}}\left(n-b,L-b,\alpha,T-t\right)\right)\Bigg], \tag{2}$$

$$\tilde{\mathsf{F}}_{\text{IR}}\left(n,L,\alpha\right)=\sum_{\ell=1}^{L}\binom{L}{\ell}\alpha^{\ell}(1-\alpha)^{L-\ell}\frac{(1-p_{n,\ell})}{(1-p_{n,L,\alpha})}\left(E\left(n,\ell,k,\alpha\right)+\sum_{b=1}^{\ell}\tilde{\mathsf{F}}_{\text{IR}}\left(n-b,L-b,\alpha\right)\frac{p_{n,\ell}J(\ell,b,n,k)}{1-p_{n,\ell}}\right). \tag{3}$$

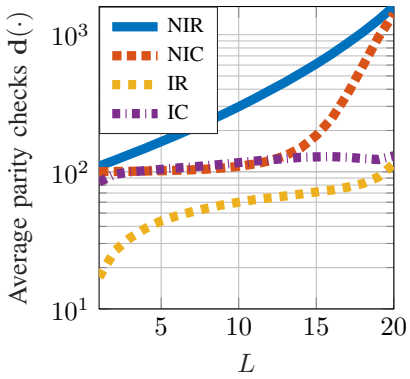

Fig. 2: $n=24$, $k=3$, $\alpha=0.5$, $T=100$, L known.

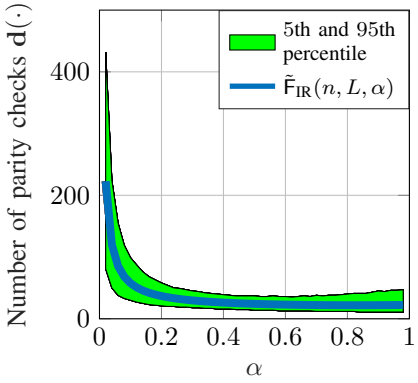

Fig. 3: $n=10$, $k=3$, $L=4$ known at the master.

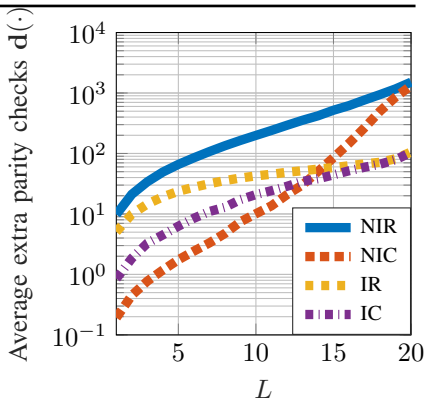

Fig. 4: $n=24$, $k=3$, $\alpha=0.5$, $T=100$, L unknown.

$$\mathsf{F}_{\text{IR}}\left(n,L,\alpha,T\right)=\sum_{t=1}^{T}\sum_{\ell=1}^{L}\mathbb{P}\big(T_B=t,A_t=\ell\big)\Bigg\{\sum_{m=1}^{\infty}\mathbb{P}\left(M_t=m\,\big|\,T_B=t,A_t=\ell\right)\Bigg[\sum_{c}\mathbb{P}\left(C_t=c\,\big|\,M_t=m,A_t=\ell,T_B=t\right)$$
$$\times\left(t+m+(c-k-1)+\sum_{b=1}^{\ell}\mathbb{P}\left(B_t=b\,\big|\,M_t=m,A_t=\ell,C_t=c\right)\mathsf{F}_{\text{IR}}\left(n-b,L-b,\alpha,T-t\right)\right)\Bigg]\Bigg\}. \tag{4}$$

In Fig. 4, we compare the average (over 1000 executions of the algorithm) of $\delta\mathsf{G}_{\text{sch}}\left(n,L,\alpha,T\right)$ in $T=100$ time-slots, for the four schemes as a function of L, for a system with parameters $(n,k,\alpha)=(24,3,0.5)$. We observe that IC performs the best when L is close to $n-k-1$, whereas NIC performs the best for small values of L. Note that when L is small, it is very likely that the master selects a reliable group in an early time-slot $t\ll T$, and continues with that, rather than "wasting" parity-check evaluations to identify the unreliable machines in IR or IC schemes. As L grows, however, the chance of selecting a reliable group decreases and hence, identifying and removing unreliable machines would be a better strategy. Moreover, from Fig. 4 we note that IR (which was the best scheme when L was known) is generally no better than IC for the case of unknown L. This is due to the infeasibility of identifying and removing "all" the unreliable machines. Finally, we observe that NIR has the worst performance, since it keeps sampling a random set without removing the unreliable machines and hence, has a positive probability of getting a non-zero parity (which leads to additional parity-checks) for every time-slot.

Appendix A

We start by defining the following random variables.

1) T_B: Time-slot in which the first set with an attacked machine is sampled. If $T_B=t$, then $\mathbf{d}(\mathcal{U}_\tau;\tau)=\mathbf{0}$ for all $\tau\in[t-1]$, and $\mathbf{d}(\mathcal{U}_t;t)\neq\mathbf{0}$.
2) A_t: Number of attacked machines in time-slot t. Hence, for $\ell\in[L]$, we have $\mathbb{P}(A_t=\ell)=\binom{L}{\ell}\alpha^{\ell}(1-\alpha)^{L-\ell}$.
3) M_t: Number of sets sampled in time-slot t before sampling an unattacked set. $\mathcal{U}_{t,1},\ldots,\mathcal{U}_{t,M_t+1}$ are all the sampled sets at time-slot t for which $\mathbf{d}\left(\mathcal{U}_{t,j};t\right)\neq\mathbf{0}$ for all $j\in[M_t]$, and $\mathbf{d}\left(\mathcal{U}_{t,M_t+1};t\right)=\mathbf{0}$ and hence, $\mathcal{G}_t=\mathcal{U}_{t,M_t+1}$.
4) C_t (respectively, B_t): Number of distinct (respectively, attacked) machines in the set $\bigcup_{j=1}^{M_t+1}\mathcal{U}_{t,j}$. Note that $C_t\leq|\mathcal{N}_t|\leq|\mathcal{N}|=n$, and $B_t\leq A_t$.

The recursive relation for $\mathsf{F}_{\text{IR}}\left(n,L,\alpha,T\right)$ is given in terms of the above random variables in (4), at the top of this page. There, we first condition the quantity of interest on the values of T_B and A_t. When $T_B=t$, the first sets selected in time-slots $\tau\in[t-1]$ are unattacked and lead to $\mathbf{d}=\mathbf{0}$; hence, we have a total of $t-1$ parity-checks before time-slot t. At time t with ℓ attacked machines, we may check m sets until we select an unattacked set, leading to $m+1$ additional checks. Next, we identify the attacked machines in different sets in time-slot t. To this end, we have to evaluate one parity-check for each machine in $\mathcal{U}_t^{m+1}\backslash\mathcal{U}_{t,m+1}$. This leads to $\left|\mathcal{U}_t^{m+1}\right|-|\mathcal{U}_{t,m+1}|=$

$c-k-1$ parity-checks, since we know that the $k+1$ machines in $\mathcal{U}_{t,m+1}$ are unattacked. This allows to identify and remove b unreliable machines from the pool. We will then continue with $n-b$ machines, out of which $L-b$ are unreliable for the remaining $T-t$ time-slots.

We now compute the various probabilities in (4). To this end, we define $p_{n,\ell} \triangleq \binom{n-\ell}{k+1}/\binom{n}{k+1}$ to be the probability of sampling an unattacked set from n machines when there are ℓ attacked machines. We then have,

$$\begin{aligned}&\mathbb{P}\left(T_B=t, A_t=\ell\right)=\mathbb{P}\left(A_t=\ell\right)\mathbb{P}\left(T_B=t \;\middle|\; A_t=\ell\right)\\&=\mathbb{P}(A_t=\ell)\left(\prod_{\tau=1}^{t-1}\mathbb{P}(\mathbf{d}\left(\mathcal{U}_{\tau,1};\tau\right)=\mathbf{0})\right)\mathbb{P}(\mathbf{d}\left(\mathcal{U}_{t,1};t\right)\neq\mathbf{0}|A_t=\ell)\,.\end{aligned}$$

We note that $\mathbb{P}(\mathbf{d}\left(\mathcal{U}_{t,1};\tau\right)=\mathbf{0}|A_\tau=j)=p_{n,j}$, and thus,

$$\begin{aligned}\mathbb{P}(\mathbf{d}\left(\mathcal{U}_{\tau,1};\tau\right)=\mathbf{0})&=\sum_{j=0}^{L}\mathbb{P}(A_\tau=j)\,\mathbb{P}(\mathbf{d}\left(\mathcal{U}_{\tau,1};\tau\right)=\mathbf{0}|A_\tau=j)\\&=\sum_{j=0}^{L}\mathbb{P}(A_\tau=j)\,p_{n,j}\triangleq p_{n,L,\alpha},\end{aligned}$$

which leads to evaluating $\mathbb{P}\left(T_B=t, A_t=\ell\right)$. We also have

$$\begin{aligned}&\mathbb{P}(M_t=m \;\middle|\; T_B=t, A_t=\ell)\\&=\frac{\mathbb{P}\left(\mathbf{d}\left(\mathcal{U}_{t,m+1};t\right)=\mathbf{0}|A_t=\ell\right)\prod_{j=1}^{m}\mathbb{P}\left(\mathbf{d}\left(\mathcal{U}_{t,j};t\right)\neq\mathbf{0}|A_t=\ell\right)}{\mathbb{P}\left(\mathbf{d}\left(\mathcal{U}_{t,1};t\right)\neq\mathbf{0}|A_t=\ell\right)}\\&=\left(1-p_{n,\ell}\right)^{m-1}p_{n,\ell}.\end{aligned}\tag{5}$$

This implies $\sum_{m=1}^{\infty}m\mathbb{P}\left(M_t=m\middle|T_B=t, A_t=\ell\right)=1/p_{n,\ell}$. Next, note that given $M_t=m$ and $A_t=\ell$, the size of $\bigcup_{j=1}^{m+1}\mathcal{U}_{t,j}$ is independent of $T_B=t$. Thus,

$$\begin{aligned}&\sum_c c\mathbb{P}\left(C_t=c\middle|M_t=m, A_t=\ell, T_B=t\right)\\&=\sum_c c\mathbb{P}\left(C_t=c\middle|M_t=m, A_t=\ell\right)=\mathbb{E}\left[C_t\middle|M_t=m, A_t=\ell\right]\\&=\sum_{v\in(\mathcal{N}\setminus\mathcal{A}_t)\cup\mathcal{A}_t}\mathbb{P}\left(\mathcal{I}_v=1\middle|M_t=m, A_t=\ell\right),\end{aligned}\tag{6}$$

where $\mathcal{I}_v=1$ if $v\in\bigcup_{j=1}^{m+1}\mathcal{U}_{t,j}$ and $\mathcal{I}_v=0$ otherwise, and $\mathcal{A}_t$ with $|\mathcal{A}_t|=A_t$ is the set of attacked machines at t. For $v\in\mathcal{N}\setminus\mathcal{A}_t$, we can evaluate $\mathbb{P}\left(\mathcal{I}_v=0 \;\middle|\; M_t=m, A_t=\ell\right)$ as

$$\begin{aligned}&\left(\prod_{j=1}^{m}\mathbb{P}\Big(v\notin\mathcal{U}_{t,j}\Big|\mathbf{d}\left(\mathcal{U}_{t,j};t\right)\neq\mathbf{0}, A_t=\ell\Big)\right)\\&\qquad\times\mathbb{P}\left(v\notin\mathcal{U}_{t,m+1} \;\middle|\; \mathbf{d}\left(\mathcal{U}_{t,m+1};t\right)=\mathbf{0}, A_t=\ell\right)\\&=\left(\frac{\binom{n-1}{k+1}-\binom{n-1-\ell}{k+1}}{\binom{n}{k+1}-\binom{n-\ell}{k+1}}\right)^m\frac{\binom{n-\ell-1}{k+1}}{\binom{n-\ell}{k+1}}\triangleq s_1^m s_0,\end{aligned}\tag{7}$$

where $\binom{n-\ell}{k+1}$ and $\binom{n-\ell-1}{k+1}$ are the numbers of ways to get an unattacked set, and to get an unattacked set without machine v, respectively. Thus, $\binom{n}{k+1}-\binom{n-\ell}{k+1}$ and $\binom{n-1}{k+1}-\binom{n-\ell-1}{k+1}$ are the numbers of ways to get an attacked set, and to get an attacked set without machine v, respectively. For $v\in\mathcal{A}_t$, $\mathbb{P}\left(\mathcal{I}_v=0 \;\middle|\; M_t=m, A_t=\ell\right)$ is given by

$$\begin{aligned}&\left(\prod_{j=1}^{m}\mathbb{P}\Big(v\notin\mathcal{U}_{t,j} \;\Big|\; \mathbf{d}\left(\mathcal{U}_{t,j};t\right)\neq\mathbf{0}, A_t=\ell\Big)\right)\\&\qquad\times\mathbb{P}\left(v\notin\mathcal{U}_{t,m+1} \;\middle|\; \mathbf{d}\left(\mathcal{U}_{t,m+1};t\right)=\mathbf{0}, A_t=\ell\right)\\&=\left(\frac{\binom{n}{k+1}-\binom{n-\ell}{k+1}-\binom{n-1}{k}}{\binom{n}{k+1}-\binom{n-\ell}{k+1}}\right)^m\triangleq s_2^m,\end{aligned}\tag{8}$$

where the second probability term is 1 because an attacked machine $v\in\mathcal{A}_t$ can not be in an unattacked set $\mathcal{U}_{t,m+1}$. Moreover, $\binom{n}{k+1}-\binom{n-\ell}{k+1}$ and $\binom{n}{k+1}-\binom{n-\ell}{k+1}-\binom{n-1}{k}$ are the numbers of ways to get an attacked set, and to get an attacked set without machine $v\in\mathcal{A}_t$, respectively. Using (7) and (8), the right-hand side of (6) becomes $(n-\ell)\left(1-s_1^m s_0\right)+\ell\left(1-s_2^m\right)$. Moreover, from (5) and (6) we have

$$\begin{aligned}&\sum_{m=1}^{\infty}\sum_c c\mathbb{P}(M_t=m, C_t=c|T_B=t, A_t=\ell)\\&=\sum_{m=1}^{\infty}\left(1-p_{n,\ell}\right)^{m-1}p_{n,\ell}\,\mathbb{E}\left[C_t\middle|M_t=m, \mathcal{A}_t=\ell\right]\\&=n-\frac{s_0 s_1(n-\ell)p_{n,\ell}}{1-s_1(1-p_{n,\ell})}-\frac{s_2\;\ell\;p_{n,\ell}}{1-s_2(1-p_{n,\ell})}\triangleq H\left(n,\ell,k\right).\end{aligned}\tag{9}$$

Finally, for $\mathcal{U}_t^m=\bigcup_{j=1}^{m}\mathcal{U}_{t,j}$, we have

$$\begin{aligned}&\sum_c\mathbb{P}\left(C_t=c, B_t=b \;\middle|\; M_t=m, A_t=\ell\right)\\&=\mathbb{P}\big(B_t=b \;\big|\; M_t=m, A_t=\ell\big)\\&=\mathbb{P}\Big(\big|\mathcal{U}_t^m\cap\mathcal{A}_t\big|=b\Big|\{\mathcal{U}_{t,j}\cap\mathcal{A}_t\neq\varnothing, j\in[m]\}, A_t=\ell\Big)\\&=\sum_{\mathcal{B}\subseteq\mathcal{A}_t,|\mathcal{B}|=b}\mathbb{P}\Big(\mathcal{U}_t^m\cap\mathcal{A}_t=\mathcal{B}\Big|\{\mathcal{U}_{t,j}\cap\mathcal{A}_t\neq\varnothing, j\in[m]\}, A_t=\ell\Big)\\&\overset{(a)}{=}\sum_{\substack{\mathcal{B}\subseteq\mathcal{A}_t\\|\mathcal{B}|=b}}\sum_{\mathcal{S}\subseteq\mathcal{B}}(-1)^{b-|\mathcal{S}|}\prod_{j=1}^{m}\mathbb{P}\left(\mathcal{U}_{t,j}\cap\mathcal{A}_t\subseteq\mathcal{S}|\mathcal{U}_{t,j}\cap\mathcal{A}_t\neq\varnothing, A_t=\ell\right)\\&\overset{(b)}{=}\binom{\ell}{b}\sum_{i=1}^{b}\binom{b}{i}(-1)^{b-i}\left(\frac{\binom{n-\ell+i}{k+1}-\binom{n-\ell}{k+1}}{\binom{n}{k+1}-\binom{n-\ell}{k+1}}\right)^m,\end{aligned}\tag{10}$$

where in (a) we used the inclusion-exclusion principle, and in (b) $\binom{n-\ell+i}{k+1}-\binom{n-\ell}{k+1}$ is the number of ways in which the attacked set $\mathcal{U}_{t,j}$ for $j\in[m]$ can have at most i specific attacked machines. Furthermore, it is not difficult to see that,

$$\begin{aligned}&\sum_{m=1}^{\infty}\left(1-p_{n,\ell}\right)^m\mathbb{P}\big(B_t=b|M_t=m, T_B=t\big)\\&=\binom{\ell}{b}\sum_{i=1}^{b}\binom{b}{i}(-1)^{b-i}\left(\frac{\binom{n-\ell+i}{k+1}-\binom{n-\ell}{k+1}}{\binom{n}{k+1}-\binom{n-\ell+i}{k+1}+\binom{n-\ell}{k+1}}\right)\\&\triangleq J(\ell,b,n,k).\end{aligned}\tag{11}$$

Now that we have computed the various probability terms of (4), after substituting them and simplifying we get the recursive expression for $\mathsf{F}_{\mathrm{IR}}\left(n,L,\alpha,T\right)$ given in (2), which reduces to (3) when $T\to\infty$.

REFERENCES

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[2] Q. Yu, N. Raviv, J. So, and A. S. Avestimehr, "Lagrange coded computing: Optimal design for resiliency, security and privacy," *arXiv:1806.00939*, 2018.

[3] "IBM World Community Grid," https://www.worldcommunitygrid.org.

[4] "Folding@Home," https://foldingathome.org.

[5] A. Solanki, M. Cardone, and S. Mohajer, "Non-colluding attacks identification in distributed computing," in *2019 IEEE Information Theory Workshop (ITW)*, 2019, pp. 1–5.

[6] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, "Gradient coding: Avoiding stragglers in distributed learning," in *Proceedings of the 34th International Conference on Machine Learning (ICML)*, vol. 70, 2017, pp. 3368–3376.

[7] Q. Yu, M. A. Maddah-Ali, and S. Avestimehr, "Polynomial codes: an optimal design for high-dimensional coded matrix multiplication," in *Advances in Neural Inf. Processing Systems*, 2017, pp. 4406–4416.

[8] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, 2018.

[9] A. B. Das and A. Ramamoorthy, "A unified treatment of partial stragglers and sparse matrices in coded matrix computation," 2021. [Online]. Available: https://arxiv.org/abs/2109.12070

[10] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Straggler mitigation in distributed matrix multiplication: Fundamental limits and optimal coding," in *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018, pp. 2022–2026.

[11] A. Behrouzi-Far and E. Soljanin, "Efficient replication for straggler mitigation in distributed computing," 2020. [Online]. Available: https://arxiv.org/abs/2006.02318

[12] C. Hofmeister, R. Bitar, M. Xhemrishi, and A. Wachter-Zeh, "Secure private and adaptive matrix multiplication beyond the singleton bound," 2021. [Online]. Available: https://arxiv.org/abs/2108.05742

[13] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, "A fundamental tradeoff between computation and communication in distributed computing," *IEEE Trans. Inf. Theory*, vol. 64, no. 1, pp. 109–128, 2018.

[14] S. Dutta, V. Cadambe, and P. Grover, ""short-dot": Computing large linear transforms distributedly using coded short dot products," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16, 2016, pp. 2100–2108.

[15] T. Tang, R. E. Ali, H. Hashemi, T. Gangwani, S. Avestimehr, and M. Annavaram, "Adaptive verifiable coded computing: Towards fast, secure and private distributed machine learning," 2022. [Online]. Available: https://arxiv.org/abs/2107.12958

[16] S. Sahraei and A. S. Avestimehr, "Interpol: Information theoretically verifiable polynomial evaluation," 2019. [Online]. Available: https://arxiv.org/abs/1901.03379

[17] R. Dorfman, "The detection of defective members of large populations," *Ann. Math. Statist.*, vol. 14, no. 4, pp. 436–440, 12 1943.

[18] D.-Z. Du and H. FK, *Combinatorial Group Testing And Its Applications*, 2000, vol. 2nd ed. World Scientific Publishing Company.

[19] M. Cheraghchi, A. Hormati, A. Karbasi, and M. Vetterli, "Group testing with probabilistic tests: Theory, design and application," *IEEE Trans. on Inf. Theory*, vol. 57, no. 10, pp. 7057–7067, 2011.