Self-Duplicating Random Walks for Resilient Decentralized Learning on Graphs

Maximilian Egger*, Ghadir Ayache†, Rawad Bitar*, Antonia Wachter-Zeh* and Salim El Rouayheb‡
*Technical University of Munich, Munich, Germany {maximilian.egger, rawad.bitar, antonia.wachter-zeh}@tum.de
†Etsy Inc, New York, USA, gayache@etsy.com
‡Rutgers University, New Brunswick, USA, salim.elrouayheb@rutgers.edu

GLOBECOM 2024 - 2024 IEEE Global Communications Conference | 979-8-3503-5125-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/GLOBECOM52923.2024.10901339

***Abstract*—Consider the setting of multiple random walks (RWs) on a graph executing a certain computational task. For instance, in decentralized learning via RWs, a model is updated at each iteration based on the local data of the visited node and then passed to a randomly chosen neighbor. RWs can fail due to node or link failures. The goal is to maintain a desired number of RWs to ensure failure resilience. Achieving this is challenging due to the lack of a central entity to track which RWs have failed to replace them with new ones by forking (duplicating) surviving ones. Without duplications, the number of RWs will eventually go to zero, causing a catastrophic failure of the system. We propose a decentralized algorithm called DECAFORK that can maintain the number of RWs in the graph around a desired value even in the presence of arbitrary RW failures. Nodes continuously estimate the number of surviving RWs by estimating their return time distribution and fork the RWs when failures are likely to happen. We present extensive numerical simulations that show the performance of DECAFORK regarding fast detection and reaction to failures. We further present theoretical guarantees on the performance of this algorithm.**

I. INTRODUCTION

Decentralized settings consist of a collection of users who cooperate to accomplish a given task. Users are modeled by vertices on a graph. Each pair of users that can communicate are connected by an edge. Since there is no central node coordinating the cooperation, the system functions as follows. A virtual token decides which user can run computations. The user holding the token runs local computation, updates the task, and passes the task and the token to one of its neighbors chosen randomly. This repeats until the task is declared accomplished. An example is decentralized learning [1]–[4]. The computational task is training a neural network on the union of the users' data. The user holding the token runs local iterations, updates the model, and passes it to a neighbor until some convergence criterion is met. When moving, the token draws a *random walk* (RW) over the graph.

Despite the various applications of RWs on graphs, e.g., autonomous vehicles and network crawlers, our main motivation for studying RWs stems from their application in decentralized learning. RW-based learning algorithms are proposed as a communication-efficient alternative to Gossip algorithms [5]–[11] that requires every node to run local computations and broadcast its updated model to all its neighbors. RW-based learning algorithms are key in distributed communication [12], [13], notably for decentralized model updates in distributed learning [1], [2], [14]. Research on decentralized algorithms with RWs has primarily addressed common anomalies like delays [15]. A trade-off between the advantages of RWs and Gossip-approaches is studied recently in [16].

Despite recent progress on RW-based learning algorithms, the challenging task of guaranteeing resilience to failures is, to the extent of our knowledge, not yet studied in the literature. It is enough for the node, or communication link, holding the token (RW) to fail to cause a catastrophic failure, losing all progress made thus far. Guaranteeing resilience against catastrophic failures is, hence, paramount to decentralized settings using RWs.

The main challenge in guaranteeing resilience to failures is the absence of a central entity orchestrating the process. A naive solution is to run multiple RWs in parallel. However, since no central entity is tracking the RWs, after certain failures occur, all RWs may fail, leading to a catastrophic failure. Hence, a decentralized mechanism allowing the nodes to dynamically and independently adjust the number of RWs in the system is needed. This motivates us to pose the question of how to devise decentralized algorithms that can maintain at any time a desired level of redundancy (multiple RWs running in parallel) in the system, by quickly reacting to failures and dynamically creating RWs, all this without making any a priori assumptions about the statistics of failures.

The designed algorithms must satisfy the following rules:
Rule 1: No central entity can observe and communicate with all the nodes in the graph. Moreover, nodes can only communicate with their neighbors.
Rule 2: RWs cannot directly communicate with each other[1].
Rule 3: A RW can be forked or terminated by the currently visited node. When forked, an independent duplicate copy of the RW is created.

A straightforward solution here can be to let each node independently fork (create a duplicate copy of) an RW after a prescribed time T to replace any possible failure that may have occurred. While such an algorithm satisfies our three rules above, it has the following undesired drawback that we want to avoid: either the network is flooded with an ever increasing number of RWs (for small T) or all the RWs eventually fail

This project has received funding from the German Research Foundation (DFG) under Grant Agreement Nos. BI 2492/1-1 and WA 3907/7-1. The work of S. El Rouayheb was supported in part by the Army Research Lab (ARL) under Grant W911NF-21-2-0272, and in part by the National Science Foundation (NSF) under Grant CNS2148182.

[1]I.e., the currently visited nodes by the RWs cannot directly communicate with each other outside the graph. Otherwise, the problem can be easily solved by letting the RWs regularly ping each other to indicate that they are still alive.

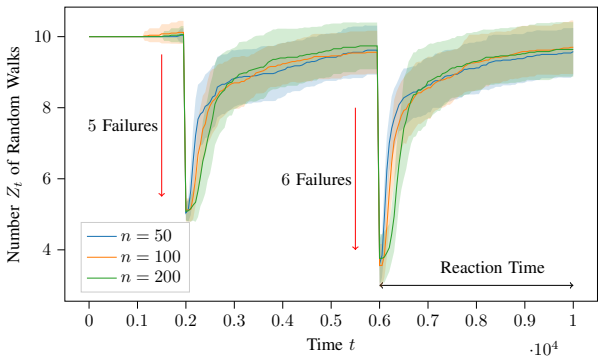

Fig. 1: Performance of DECAFORK in maintaining the number of random walks (RWs) Z_t around a desired value $Z_0 = 10$. The graph is a random 8-degree regular graph with numbers of nodes $n \in \{50, 100, 200\}$. RWs can fail arbitrarily and simultaneously. We induce two failure events at $t = 2000$ and $t = 6000$. DECAFORK immediately reacts to failures and starts creating new RWs by forking surviving ones until Z_t stabilizes around Z_0. Standard deviations over 50 simulation runs are depicted by shaded areas.

(for large T). That is because the failures are arbitrary with no assumption on their statistics that could help with designing T. Our objective is to design a decentralized algorithm that avoids these two extreme cases[2] and guarantees over time a constant number of RWs in the graph (see Fig. 1). While decentralized learning is the main motivation of this work, we abstract this aspect and focus on the robustness to failures. Learning aspects, such as the convergence speedup from multiple RWs, are deferred to future work.

Contributions: In this paper, we propose DECAFORK, a novel decentralized (randomized) algorithm forking RWs, capable of detecting RW failures and replacing them through forking. DECAFORK satisfies Rules 1-3 and does not assume any specific failure model. The number of RWs in the graph is maintained around a desired value. The main intuition is that nodes continuously estimate the current number of surviving RWs by estimating their return time distribution, i.e., the time it takes for an RW to return to a certain node after having left it, and forking the RWs when failures are likely to have happened. An important figure of merit is the reaction time representing the time the algorithm takes to bring the system back to the desired number of RWs after failures occur (see Fig. 1). We prove a bound on the reaction time of DECAFORK, derive guarantees that our algorithm will not flood the network, and study the tension between these objectives. Our numerical results confirm our theoretical findings. We defer the reader to the appendix for all proofs and additional experiments.

II. SYSTEM MODEL

Graph: We consider a system of n nodes collaborating in a decentralized fashion to run a certain compute task. Each node possesses local data and can communicate only with a subset of the other nodes. We model this system by an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n vertices $\mathcal{V} = \{1, \ldots, n\}$ representing the nodes and the set of edges (links) $\mathcal{E}$ comprising of all pairs of nodes (i, j), (j, i) that can share information. We will assume that graph $\mathcal{G}$ is connected[3] making the resulting RWs irreducible [17]. The degree $\deg(i)$ of a node $i \in \mathcal{V}$ is defined as the number of its neighbors, i.e., $\deg(i) = |\{(i, j) \in \mathcal{E}\}|$.

Random Walks: We are interested in tasks that can be accomplished by an RW carrying a token message on the graph. At discrete time steps, only the node holding the token (the currently visited node) may run some computation and update the message. Then, it passes the updated token message to one of its neighbors chosen randomly according to a fixed transition matrix P. We consider simple random walks, where the node to whom a node sends the token is selected uniformly at random from all neighbors. This is repeated until certain stopping criteria are met. We abuse nomenclature and refer to the token as the Random Walk RW. Since we consider multiple RWs, each RW, indexed by k, is distinguishable by the nodes they visit through a unique identifier.

Failures of Random Walks: Random walks on the graph may fail arbitrarily. Potential reasons include the nodes or links being temporarily down leading to the loss of the token when being passed to the next node. Similarly, a random walk can fail due to queue or buffer overflows, or the node currently holding the token may face processing issues, making it impossible to pass on the token. We want an algorithm that can dynamically adapt to any number of failures occurring over time. Therefore, we do not make any assumptions on the probabilistic model of the failures. In particular, multiple RWs can also fail simultaneously.

Forking: To avoid catastrophic failures (i.e., losing all the RWs), an RW is allowed to be duplicated in the following way. After updating the RW, the currently visited node will make a duplicated copy of the RW that walks on the graph according to the transition matrix P independently of the original one. The number of random walks Z_t at time t is a random variable, whose realization is subsequently referred to as Z_t.

Definition 1 (Forking an RW)**.** *A* forking event *occurs when a node decides to create an* identical copy *of the RW it holds and propagates both RWs in the network independently.*

The objective is to design a fully decentralized algorithm that abides by Rules 1-3 and maintains Z_t around a desired target Z_0, i.e., $\Pr(|\mathsf{Z}_t - Z_0| < x) \approx 1$ for a small $x \geq 0$, even after failures occur. The challenge is to achieve a trade-off between frequently forking RWs and flooding the network, or seldomly forking RWs and risking a catastrophic failure. Hence, the forking strategy must ensure that: the redundancy does not significantly exceed Z_0; and at least one RW maintains activity after a failure event. This is accomplished by rapid detection and reaction times to failure events.

For the optimal functioning of our algorithm and the validity of our theoretical results, all Z_0 random walks should have been active for long enough to have visited each node at least once before the first failure of an RW. Apart from this assumption, our algorithm does not rely on any assumptions about the graph structure. For a tractable theoretical analysis, we will later introduce some required assumptions in Assumption 1.

[2]Evidently, if all the RWs fail at the same time, this would be a catastrophic event that no algorithm can recover from.

[3]If the graph is not connected, our study can be applied separately to each connected component.

III. DECAFORK

We introduce DECAFORK, a decentralized algorithm that forks RWs probabilistically to avoid catastrophic failures without flooding the network with RWs. DECAFORK works as follows. The network starts with a target number Z_0 of RWs[4]. At every time instance, when a node receives an RW, it estimates the number of *active* RWs in the network. If this number is too low, the node forks the visiting RW with probability $p = 1/Z_0$ so that, on average, at most one RW is forked at a given time.

To estimate the number of RWs in the system, we rely on an RW's hitting time and return time. The *hitting time* $H_{i,j}(k)$ of an RW k at node i is the random variable that denotes the first time the RW gets to node i starting from node j. Let $v_t^{(k)}$ be the node visited by RW k at time t, we can define the hitting time as $H_{i,j}(k) = \min\{t \in \mathbf{N}, v_t^{(k)} = i \,|\, v_0^{(k)} = j, v_1^{(k)}, \ldots, v_{t-1}^{(k)} \neq i\}$. Furthermore, the random variable $R_i(k) = H_{i,i}(k)$ describes the first return time of a random walk to node i after leaving node i.

We assume that RWs walk on the graph independently. Hence, they have independent and identically distributed hitting and return times. Therefore, we focus on one random variable R_i and drop the dependency on k. We do the same for $H_{i,j}$. To have a refined estimate for R_i and its distribution, every empirically observed return time for every RW will contribute to the estimate of the random variable R_i. Hence, the algorithm requires an initialization phase without RW failures so that the Z_0 initial RWs have circulated the graph sufficiently for the nodes to have reasonable estimates of the return time distribution, at least until each RW visited each node at least once.

To measure the empirical distribution of the return time R_i, each node i keeps track of the time it has last seen RW k, denoted by the random variable $L_{i,k}(t)$. This variable is created as $L_{i,k}(t) = t_1$ at time t_1 when RW k first hits node i. Then, at time t, $L_{i,k}(t)$ is updated as $L_{i,k}(t) = t$ if and only if RW k visits node i at time t. When an RW k visits at time t, before updating $L_{i,k}(t)$, each node i measures a sample of R_i by computing $t - L_{i,k}(t)$ to build an empirical distribution of R_i. Let $\hat{F}_{R_i}(t)$ be the established empirical cumulative distribution function (CDF) of the return time of an RW at node i on the graph $\mathcal{G}$. We call *survival function* the distribution $f(t - L_{i,k}(t)) := 1 - \hat{F}_{R_i}(t - L_{i,k}(t))$ denoting the estimated probability of RWs returning after time t, i.e., $\hat{\Pr}(R_i > t - L_{i,k}(t))$.

Since $L_{i,k}(t)$ is a random variable, the function $f(t - L_{i,k}(t))$ representing the survival probability is also a random variable. Using R_i, each node i maintains an estimate of Z_t.

Forking strategy: In DECAFORK, only nodes visited by an RW can fork the visiting RW[5]. Let node i be visited by RW k at time t. Let $\mathcal{L}_i(t)$ be the set of indices of RWs that have visited node i until time t. To estimate Z_t, node i computes

$$\hat{\theta}_i(t) = 1/2 + \sum_{\ell \in \mathcal{L}_i(t) \setminus \{k\}} f(t - L_{i,\ell}(t)).$$

As we show in the sequel, the value of $\hat{\theta}_i(t)$ serves as an estimate of $Z_t/2$. For a predetermined parameter $\varepsilon > 0$, if $\hat{\theta}_i(t) < \varepsilon$, the node declares that $Z_t < Z_0$. To avoid flooding the network, i.e., avoiding that all nodes fork simultaneously, node i forks[6] RW k with probability $p = 1/Z_0$. To distinguish the random walks, each RW, even the forked one, is given its own unique identifier.[7]

DECAFORK: Executed when RW k visits node i at time t

Require: ε, Z_0, k, t, $\mathcal{L}_i(t)$, $\forall \ell \in \mathcal{L}_i(t) : L_{i,\ell}(t)$,
  if $k \in \mathcal{L}_i(t)$ **then**
    Add $t - L_{i,k}(t)$ as sample for the distribution of R_i
    Update $L_{i,k}(t) \leftarrow t$
  else Create $L_{i,k}(t) = t$
    $\mathcal{L}_i(t) \leftarrow \mathcal{L}_i(t) \cup \{k\}$
  end if
  Create $\hat{\theta}_i(t) \triangleq \frac{1}{2}$, an estimate of the number of RWs
  for $\ell \in \mathcal{L}_i(t) \setminus \{k\}$ **do**
    Calculate the survival probability $f(t - L_{i,\ell}(t))$
    of the return time R_i
    Update $\hat{\theta}_i(t) \leftarrow \hat{\theta}_i(t) + f(t - L_{i,\ell}(t))$
  end for
  if $\hat{\theta}_i(t) < \varepsilon$ **then**
    Fork RW k with probability $p = \frac{1}{Z_0}$
  end if

The difficulty lies in designing the parameters ε and p, which should facilitate both *(i)* early detection of failures and consequently forking of RWs; and *(ii)* avoiding forking when the number of walks is Z_0 or above.

DECAFORK is summarized in the algorithm above, and its performance on a random regular graph is depicted in Fig. 1. For the 50 numerical simulations, we generated 8-degree regular graphs, for different numbers of nodes $n \in \{50, 100, 200\}$ and $Z_0 = 10$ desired RWs on the graph. At time $t = 2000$ and $t = 6000$, we impose failure events that deterministically result in the failure of 5 and 6 RWs, respectively. The value of $\varepsilon \in \{1.85, 2, 2.1\}$ is well-tuned for the respective number n. We observe the desired behavior of forking RWs after the failure events. Even though several RWs fail, the recovery process, on average, does not lead to an undesired increase beyond Z_0. Without forking, the second perturbation would lead to a catastrophic failure. Intuitively, the smaller the graph, measured by n, the faster the reaction time to failure events. This is because $\hat{F}_{R_i}(t)$ is confined to a smaller support.

Fig. 2 depicts the performance of DECAFORK for different values of ε and $n = 100$ nodes through numerical experiments. Different choices for ε illustrate the trade-off between reaction

[4]This can be instantiated by one node creating Z_0 RWs in the beginning.

[5]If multiple RWs visit a node, the node chooses one of them and follows the detailed steps.

[6]Forked RWs behave immediately like active ones leaving the forking node.

[7]Such an identifier can be the index of the original random walk, i.e., $k \in [Z_0]$, at the beginning. When a node i forks a random walk at time T_c, it appends its own identifier and the time T_c of forking.

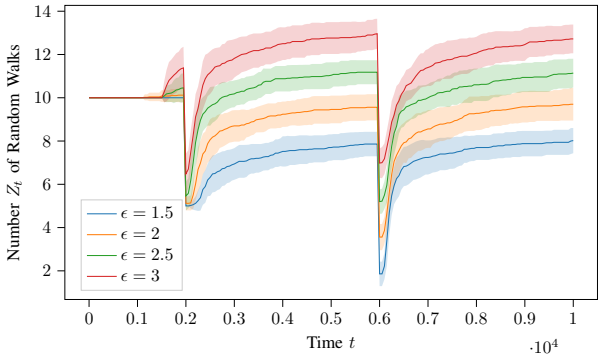

Fig. 2: Average simulation results for 50 runs on a 8-degree random regular graph with $n = 100$, and two perturbation events at $t = 2000$ and $t = 6000$. Shaded areas depict standard deviations. Different choices for ε show the trade-off between reaction time and undesired forks beyond $Z_0 = 10$.

time and undesired forks beyond $Z_0 = 10$, i.e., objectives *(i)* and *(ii)*, respectively.

Note that DECAFORK can be run on any connected graph and does not make any assumption on the distribution of the return time R_i, which is estimated on the fly. We verified this fact through numerical simulations[8] on other families of graphs, such as Erdos Rényi and Power Law graphs.

We have also tested simpler forking policies where a node compares the time since a random walk visited last to a threshold above which a new RW is forked probabilistically. The challenge in such algorithms is that the distribution of the return time depends on the graph and on the location of the nodes within the graph, making finding a good threshold for each node a challenging task. Our proposed approach in DECAFORK is less sensitive to the actual return time distribution, thus simplifying the parameter selection and improving the stability of the approach to different graph topologies. In all experiments, DECAFORK significantly outperformed alternative forking decision rules. Other algorithms we investigated either led to a massive blow-up of Z_t, hence to a network overload, or were unable to cope with arbitrary failure events.

Remark 1. *To speed up the initialization phase and the algorithm's precision, the empirical distribution $f(t-L_{i,k}(t))$ can be replaced with the analytical survival function. Such results are known, e.g., for random regular graphs [18]. In this case, it is sufficient that each RW has visited each node at least once ahead of the first failure.*

IV. TRADING REACTION TIME VS. UNDESIRED FORKING

DECAFORK exhibits good performance regarding the reaction to failure events with an appropriate choice of ε. We analytically analyze the behavior of the algorithm in the sequel. We first investigate the average of the estimation $f(t-L_{i,k}(t))$ in the case of a stable number of active random walks K on the graph, and then show the tension between the reaction time to failures and the probability of reaching $\mathrm{Z}_t > Z_0$ for any $t > 0$ immediately after the start of the algorithm.

[8]The additional simulations and the proofs required in the next Section are omitted for brevity. They will be provided in an extended version.

For our analysis, we require knowledge of the full distribution of the return time R_i and the first hitting time $H_{i,j}$ of RWs. The literature on such distributions is rare, with some exceptions. In fact, our assumptions[9] are justified by recent results on the distribution of first return and hitting times of RWs on random regular graphs [18], [19]. It is shown that both R_i and $H_{i,j}$ exhibit a behavior similar to a geometric distribution. For R_i, retroceeding trajectories, i.e., those that return to node i the same way they left, affect the distribution for small realizations of R_i. This relies on combinatoric arguments that we will neglect for tractability purposes. The dominating part of the distribution stems from non-retroceding trajectories, which provably yield an exponential behavior [18]. This is underlined by our experiments, where random regular graphs exhibit a distribution that can be well-approximated by a properly parameterized geometric distribution.

For the theoretical analysis, we study a continuous relaxation thereof, which serves as a proxy for the actually discrete random variables due to the nature of the discrete time steps. With this assumption, we can precisely describe the distribution of the estimation $\hat{\theta}_i(t)$. We choose an exponential distribution since it facilitates a rigorous and tractable analysis of the performance of the algorithm while being the continuous analog to the geometric distribution. However, it is worth noting that our analysis does not rely on the memoryless property of the exponential distribution and can hence be generalized to different types of distributions, whose CDF is invertible, following the same methodology.

Assumption 1. *Based on the discussion above, we make the following assumptions for our theoretical analysis:*

- *The return time of RWs are independently and identically distributed according to $R_i \sim \exp(\lambda_r)$.*[10]
- *The first hitting time $H_{i,j}$ for random nodes i and j of a forked RW k is distributed according to $H_{i,j} \sim \exp(\lambda_a)$.*

Knowing the analytical survival function, for the following analysis we replace $f(t-L_{i,k}(t)) = 1 - \hat{F}_{R_i}(t-L_{i,k}(t))$ by the exact distribution, i.e., $1 - F_{R_i}(t-L_{i,k}(t))$.

A. The Average of the Estimator $\hat{\theta}_i(t)$

We first verify how the estimator $\hat{\theta}_i(t)$ resembles the actual number of active random walks Z_t at time t in the absence of terminations and forks, i.e., when Z_t is deterministic.

Proposition 1. *For a constant number Z_t of RWs active for an infinitely long time, under Assumption 1 and replacing the empirical distribution of R_i by its analytical counterpart, the estimator $\hat{\theta}_i(t)$ satisfies $2\mathrm{E}[\hat{\theta}_i(t)] = Z_t$.*

Proof. For any RW ℓ and any t, let $r_i := t - L_{i,\ell}(t)$ be the time passed since RW ℓ was last seen at node i. Hence, we have $f(t - L_{i,\ell}(t)) = \Pr(R_i > t - L_{i,\ell}(t)) = \Pr(R_i > r_i)$. Since the RWs are independent, we can assume that a node i evaluates the survival function of RW ℓ at a random point t in

[9]Note that with minor adaptation, our analysis can similarly be carried out for other types of return and first-hitting time distributions.

[10]This assumption can be relaxed to capture different distributions for different nodes.

time. This point in time is when a random walk $k \neq \ell$ visits node i, which is random and independent of ℓ. Hence, the observed r_i is a random sample itself following the distribution of R_i. Consider now K RWs that have been active for an infinitely long time such that each has visited each node $i \in [n]$ at least once with probability 1. The expectation of the estimation $f(r_i)$ over the randomness of $r_i \sim R_i$ at time t for a single active random walk is $\mathrm{E}[f(r_i)] = \frac{1}{2}$. This follows from the probability integral transform [20], which states that the CDF of any random variable with invertible CDF evaluated at the random variable itself is uniformly distributed on $\mathcal{U}(0,1)$. By a symmetry argument, this equivalently holds for the survival function. Thus, its mean is $\frac{1}{2}$. In DECAFORK, this fact motivates the addition of $1/2$ to $\hat{\theta}_i(t)$ for the visiting RW k, which is known to be active. Similarly, for the remaining $K-1$ RWs, we have $\sum_{\ell \in \mathcal{L}_i(t) \setminus k} \mathrm{E}[f(r_i)] = (K-1)/2$. □

Note that Proposition 1 holds under Assumption 1 and for any continuous return time distribution with invertible CDF. Proposition 1 justifies the use of the offset $\frac{1}{2}$ in DECAFORK. Hence, we investigate the error made by the continuous approximation of the return time distributions using the PMF and CDF of the discrete geometric distribution supported on $r_i \in \{1, 2, \ldots, \infty\}$ with parameter q, probability mass function $\Pr(R_i = r_i) = (1-q)^{r_i - 1} q$ and survival function $f(r_i) = (1-q)^{r_i}$. For an active random walk k, the expected value of $\mathrm{E}[f(r_i)]$ for R_i distributed according to a geometric distribution with parameter q reads as

$$\mathrm{E}[f(r_i)] = \sum_{r_i=1}^{\infty} \Pr(R_i = r_i) f(r_i) = \sum_{r_i=1}^{\infty} (1-q)^{2r_i - 1} q = \frac{1-q}{2-q}.$$

Hence, for small values of q, such as those expected for random regular graphs [18], the expectation is close to 0.5 (as for the analytical counterpart), but a non-zero error remains. While this did not affect the algorithm's performance negatively in our experiments, the constant offset $\frac{1}{2}$ can be replaced by the actual expectation using the empirical distribution of R_i established at all the nodes.

B. On the Distribution of the Estimator $\hat{\theta}_i(t)$

We study the exact probability of forking based on $\hat{\theta}_i(t)$ to provide worst-case guarantees on the reaction time to failure events and to bound the undesired increase of the number of RWs in the graph beyond Z_0. To that end, instead of relying on concentration bounds of $\hat{\theta}_i(t)$ around its average, we derive its probability distribution in the sequel.

Proposition 2. *Under Assumption 1 and assuming that $F_{R_i}(t)$ is continuous and invertible, for K active random walks in the system, the estimation $\hat{\theta}_i(t)$ is a random variable that can be described by the CDF $F_{\Sigma_{K-1}}(\sigma)$, where*

$$F_{\Sigma_{K-1}}(\sigma) = \frac{1}{(K-1)!} \sum_{\tau=0}^{\lfloor \sigma \rfloor} (-1)^{\tau} \binom{K-1}{\tau} (\sigma - \tau)^{K-1}.$$

In fact, $F_{\Sigma_{K-1}}(\sigma)$ is known as the Irwin-Hall distribution and represents the sum of $K-1$ uniform distributions $\mathcal{U}(0,1)$.

The parameter ε in DECAFORK can be conveniently chosen based on this result. Let $F_{\Sigma_{Z_0-1}}(\varepsilon - \frac{1}{2})$ be the probability of estimating at most ε active RWs assuming that Z_0 RWs are active. Intuitively, this reflects the likelihood of the assumption of Z_0 active RWs being accurate. We chose the value of ε such that the probability of forking with Z_0 active RWs is negligible. Let $\delta' := F_{\Sigma_{Z_0-1}}(\varepsilon - \frac{1}{2})$. According to DECAFORK, with Z_0 active RWs, a node forks with probability $p_{\text{fork}} = p \cdot \delta'$. If $p = 1$, a node deterministically forks once it encounters a large deviation from the expected value of $\hat{\theta}_i(t)$, exhibiting a fast reaction to failures. The probability of forking vanishes for more than Z_0 active RWs, which is a desirable property.

The parameter p is chosen to avoid flooding the network. In cases where a failure happens and K random walks remain active, at each time step, K nodes will simultaneously be able to realize the failures and decide to fork a new random walk. Scaling the probability of forking at each node by $p = 1/Z_0$ avoids having too many forks. However, this scaling increases the reaction time to failure events. In preparation of the following analysis, we consider a single failure event that occurs at time T_f, leading to the failure of B RWs.

Proposition 3. *Under Assumption 1 and assuming that $F_{R_i}(t)$ is continuous and invertible, for B random walks indexed by $\mathcal{B}$ terminated at time T_f, the part of $\hat{\theta}_i(t)$ corresponding to $\mathcal{B} \subset \mathcal{L}_i(t)$ can be described by the CDF $F_{\Sigma_B}(\sigma e^{\lambda_r (t - T_f)})$.*

C. A Bound on the Reaction Time to Failure Events

We derive worst-case guarantees on the reaction time as a response to the failure of B RWs at time T_f. In Section IV-D, we show that improving the reaction time increases the likelihood of having more than Z_0 active RWs. We assume that K' RWs have been active for long enough to visit all nodes at least once. Let B RWs fail at time T_f due to a failure event. After the failure, $K = K' - B$ RWs remain active.

We bound the time $T_B^{R'}$ spent until at least $R' \leq B$ RWs are forked with a certain probability. The main ingredient is to bound the time T_{B-R} elapsed until at least one node forks an RW after B RWs failed and R forks took place, with probability $1 - \delta_{B-R}$, for some $0 \leq R < B$ and $0 < \delta_{B-R} < 1$. This result is given in Theorem 1.

Theorem 1. *Consider the setting explained above and the event where B RWs fail and R forks happen afterward. For any choice of $0 < \varepsilon' < \varepsilon - \frac{1}{2}$ and $T > 0$, let the quantity $\delta_{B-R}(T)$ be bounded by*

$$\delta_{B-R}(T) \leq \prod_{t=T_f}^{T} \left[1 - p F_{\Sigma_{K+R-1}}(\varepsilon') F_{\Sigma_{B-R}}\left(\frac{\varepsilon - \varepsilon' - \frac{1}{2}}{\exp(-\lambda_r (t - T_f))} \right) \right].$$

For a desired $\delta_{B-R} > 0$, the time T_{B-R} elapsed until at least one fork occurs with probability at least $1 - \delta_{B-R}$, is bounded by the smallest T satisfying $1 - \delta_{B-R}(T) \geq 1 - \delta_{B-R}$.

The ε' can be chosen to minimize T_{B-R}. Applying Theorem 1 for $R \in \{0, \ldots, R'-1\}$, with $\delta_\Sigma := \sum_{R=1}^{R'-1} \delta_{B-R}$, we can write $\Pr\left(T_B^{R'} \leq \sum_{R=0}^{R'-1} T_{B-R} \right) \geq 1 - \delta_\Sigma$. The parameter δ_Σ can be split into the δ_{B-R}'s to minimize $T_B^{R'}$.

An implication of Theorem 1 is that the time to fork increases with the number of forked RWs.

D. The Number of Random Walks is Finite

To bound the maximum number of RWs in the system, we bound the maximum number of forks that occur when using DECAFORK for a duration T without any failure. Assume a time t at which $Z_t = \nu$. The probability of forking is bounded by $p_\nu \leq p_\nu^+ := \nu \cdot p \cdot F_{\Sigma_{\nu-1}}(\varepsilon - \frac{1}{2})$, where the factor ν results from at most ν distinct nodes being visited by an RW. For $\varepsilon < 1$, the forking probability simplifies to $p_\nu \leq p_\nu^+ = \frac{\nu p(\varepsilon-\frac{1}{2})^{\nu-1}}{(\nu-1)!}$. For $\nu+1$, we have $p_{\nu+1} \leq \frac{(\nu+1)p(\varepsilon-\frac{1}{2})^{\nu}}{\nu!} = p_\nu^+ \frac{(\nu+1)(\varepsilon-\frac{1}{2})}{\nu^2}$. Hence, for any RW that gets forked in a system of ν random walks, subsequent forking probabilities decrease by a factor of $\frac{(\nu+1)(\varepsilon-\frac{1}{2})}{\nu^2}$ in the long run. However, the forking probability decreases only when all nodes are aware of all active RWs in the system. This intuition is the basis of the proof of the following theorem, which bounds the probability of Z_t exceeding the number $z > Z_0$ in a graph operating for a duration T without failures.

Theorem 2. *For $i < z$, let $T_{\nu,1} = \frac{1}{\lambda_a}\log(\frac{\lambda_a n}{p_\nu^+})$. After time T, the probability of having more than $z > Z_0$ walks in the network is, for some $m \leq z$, bounded by*

$$\delta \leq p_m^+ T_{m,2} + \sum_{i=Z_0}^{m-1} n e^{-\lambda_a T_{\nu,1}} + T_{\nu,1} p_\nu^+.$$

The statement holds for m being the largest integer (smaller than z) so that $\sum_{i=Z_0}^{m-1} T_{\nu,1} < T$. The time $T_{m,2}$ must then be chosen as $T_{m,2} = T - \sum_{i=Z_0}^{m-1} T_{\nu,1}$.

Theorem 2 can be inverted to state for any confidence $\delta > 0$, the probability $\Pr(\mathsf{Z}_t < z) \geq 1 - \delta$ as long as the algorithm runs for a time T bounded as in Corollary 1.

Corollary 1. *With probability at most δ, the time T until the number of RWs grows larger than z is bounded by*

$$T \geq T_{m,2} + \sum_{i=Z_0}^{m-1} T_{\nu,1},$$

where $T_{\nu,1}$ is as above and m is the largest integer such that $\delta < \delta_\Sigma := \sum_{i=Z_0}^{m-1} n e^{-\lambda_a T_{\nu,1}} + T_{\nu,1} p_\nu^+$, and $T_{m,2} = \frac{\delta - \delta_\Sigma}{p_m^+}$.

The trade-off between reaction time and the likelihood of increasing beyond z RWs after the start of DECAFORK is controlled by the choice of ε and is implicit in Theorems 1 and 2. The smaller ε, the larger the times $T_{\nu,1}$ in Theorem 2, which reflect fewer undesired forks at a given time. Conversely, smaller values for ε lead to smaller values of the CDFs in Theorem 1, hence to a slower decrease of the product and, thus, a larger delay to fork. This trade-off aligns with numerical experiments for different values of ε, which we depict in Fig. 2. The larger ε, the larger the average number of RWs in the system, but the faster the reaction time. Choosing even smaller values for ε will likely lead to failures of the system after the second perturbation at time $t = 6000$.

V. CONCLUSION

We introduced DECAFORK, a novel decentralized and dynamic algorithm that adapts to failures of RWs on a graph. When detecting failures of RWs, DECAFORK allows nodes to fork new RWs probabilistically, thus maintaining a desired value of RWs and avoiding catastrophic failures. Through simulations and theoretical analysis, we showed that a trade-off exists between the competing objectives of quickly forking RWs and the undesired event of forking random walks in the absence of failures. Our findings open up many interesting research directions that include but are not limited to analyzing the algorithmic properties in relation to the number of nodes for general graphs and studying the effect of allowing nodes to terminate RWs and designing corresponding algorithms.

REFERENCES

[1] B. Johansson, M. Rabi, and M. Johansson, "A randomized incremental subgradient method for distributed optimization in networked systems," *SIAM J. Optim.*, vol. 20, pp. 1157–1170, 2009.

[2] A. Nedic and A. Ozdaglar, "Distributed subgradient methods for multi-agent optimization," *IEEE Transactions on Automatic Control*, vol. 54, pp. 48–61, 2009.

[3] T. Sun, Y. Sun, and W. Yin, "On markov chain gradient descent," *Advances in neural information processing systems*, vol. 31, 2018.

[4] G. Ayache and S. E. Rouayheb, "Random walk gradient descent for decentralized learning on graphs," *IEEE International Parallel and Distributed Processing Symposium Workshops*, pp. 926–931, 2019.

[5] S. P. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Gossip algorithms: design, analysis and applications," *IEEE Computer and Communications Societies.*, vol. 3, pp. 1653–1664 vol. 3, 2005.

[6] D. Shah, "Gossip algorithms," *Found. Trends Netw.*, vol. 3, pp. 1–125, 2009.

[7] J. Lu, C. Y. Tang, P. R. Regier, and T. D. Bow, "Gossip algorithms for convex consensus optimization over networks," *IEEE Transactions on Automatic Control*, vol. 56, pp. 2917–2923, 2010.

[8] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE transactions on information theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

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

[10] A. Koloskova, S. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *International Conference on Machine Learning*, 2019, pp. 3478–3487.

[11] J. C. Duchi, A. Agarwal, and M. J. Wainwright, "Dual averaging for distributed optimization: Convergence analysis and network scaling," *IEEE Transactions on Automatic control*, vol. 57, no. 3, pp. 592–606, 2011.

[12] Z. Avarikioti, M. Bastankhah, M. A. Maddah-Ali, K. Pietrzak, J. Svoboda, and M. Yeo, "Route discovery in private payment channel networks," Cryptology ePrint Archive, Paper 2021/1539, 2021.

[13] A. D. Sarma, A. R. Molla, and G. Pandurangan, "Efficient random walk sampling in distributed networks," *Journal of Parallel and Distributed Computing*, vol. 77, pp. 84–94, 2015.

[14] G. Ayache and S. E. Rouayheb, "Private weighted random walk stochastic gradient descent," *IEEE Journal on Selected Areas in Information Theory*, vol. 2, no. 1, pp. 452–463, 2021.

[15] M. R. Glasgow and M. Wootters, "Asynchronous distributed optimization with stochastic delays," in *International Conference on Artificial Intelligence and Statistics*, vol. 151, 28–30 Mar 2022, pp. 9247–9279.

[16] P. Gholami and H. Seferoglu, "Digest: Fast and communication efficient decentralized learning with local updates," *IEEE Transactions on Machine Learning in Communications and Networking*, 2024.

[17] D. A. Levin and Y. Peres, *Markov chains and mixing times*. American Mathematical Soc., 2017, vol. 107.

[18] I. Tishby, O. Biham, and E. Katzav, "Analytical results for the distribution of first return times of random walks on random regular graphs," *Journal of Physics A: Mathematical and Theoretical*, vol. 54, no. 32, p. 325001, 2021.

[19] ——, "Analytical results for the distribution of first-passage times of random walks on random regular graphs," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2022, no. 11, p. 113403, 2022.

[20] F. N. David and N. L. Johnson, "The probability integral transformation when parameters are estimated from the sample," *Biometrika*, vol. 35, no. 1/2, pp. 182–190, 1948.