Fault-Tolerant Distributed Directories

Fault-Tolerant Distributed Directories

Judith Beestermöller ✉
ETH Zurich, Switzerland

Costas Busch ✉
Augusta University, GA, USA

Roger Wattenhofer ✉
ETH Zurich, Switzerland

Abstract

Many fundamental distributed computing problems require coordinated access to a shared resource. A distributed directory is an overlay data structure on an asynchronous graph G that helps to access a shared token t. The directory supports three basic operations: *publish*, to initialize the directory, *lookup*, to read the contents of the token, and *move*, to get exclusive update access to the token. There are known directory schemes that achieve message complexity within polylog factors of the optimal cost with respect to the number of nodes n and the diameter D of G. Motivated by fault-tolerant distributed computing implementations, we consider the impact of edge failures on distributed directories. We give a distributed directory overlay data structure that can tolerate edge failures without disrupting the directory operations. The directory can be repaired concurrently while it processes directory operations. We analyze the impact of the faults on the amortized cost of the three directory operations compared to the optimal cost. We show that f edges failures increase the amortized competitive ratio of the operations by at most factor f. We also analyze the message complexity to repair the overlay structure, in terms of the number of messages that are sent and the maximum distance a message traverses. For an edge failure, the repair mechanism uses messages of size $\mathcal{O}(\log n)$ that traverse distance at most D', the graph diameter after the fault. To our knowledge, this is the first asymptotic analysis of a fault-tolerant distributed directory.

2012 ACM Subject Classification Theory of computation → Distributed algorithms; Theory of computation → Shared memory algorithms; Software and its engineering → Software fault tolerance

Keywords and phrases distributed directory, sparse partition, fault tolerance, message complexity, path dilation

Digital Object Identifier 10.4230/LIPIcs.SAND.2024.5

Related Version *Full Version*: https://arxiv.org/abs/2207.09940

Funding Costas Busch is supported by grant NSF-CNS-2131538

1 Introduction

Many distributed computing applications require finding and accessing a shared token, where the token represents some shared resource. At all times, only the token owner has exclusive access to the token which grants the owner the ability to modify the content that the token represents. Distributed directories enable other nodes to find the token to read its contents or to get exclusive access. Distributed directories have applications in shared memory and sensor networks. Distributed transactional memory systems use distributed directories to atomically access shared memory objects and execute transactions at the network nodes [10, 21]. Sensor networks use distributed directories to track moving objects [1, 23].

We study distributed directories that facilitate access to a shared token t on an asynchronous weighted graph $G = (V, E, w)$. The directory supports three operations: (i) *publish*, which initializes the directory and announces the initial owner; (ii) *lookup*, which allows a node to read the contents of the t; (iii) *move*, which moves t to a new owner for exclusive access. These operations may be issued and processed concurrently by the nodes in G.

Several directory schemes based on an overlay data structure on graph G have been developed, for which the amortized total distance traversed by the messages in a lookup or move operation is close to optimal, namely within some poly-log factor to the number of nodes n and the diameter of the graph [10, 19, 21, 22]. However, these directory protocols are not fault tolerant. If an edge failure occurs, they are not able to maintain a directory structure that can support publish, lookup, and move operations.

In reality, a directory is implemented on a distributed network, and it is typical to have unreliable networks with link failures between nodes. As processing nodes need to continue to operate correctly during or after the occurrence of failures, designing fault-tolerant distributed algorithms is important. We provide a directory protocol that tolerates edge failures with provable correctness and performance guarantees.

We consider the impact of $f \geq 1$ edge failures. We assume that the edge failures do not disconnect G, as otherwise, the token becomes unreachable in G making the directory unusable. Nevertheless, edge failures may happen at arbitrary moments and concurrently. Given the initial partition hierarchy, our protocol is fully distributed and handles the failures without disrupting concurrent directory operations. We analyze the message complexity of repairing the directory and provide performance bounds for the amortized cost of the operations related to the number of failed edges f.

1.1 Contributions

We present a distributed directory that can handle edge failures without disrupting concurrent directory operations. Our directory is inspired by the Spiral directory protocol [21]. Spiral uses a sparse cover decomposition hierarchy of G that allows clusters at the same level to overlap. Instead, we use a sparse partition hierarchy $\mathcal{P}$ of G that does not allow clusters at the same level to overlap. As we will show, sparse partitions have improved asymptotic performance in the directory operations, and are affected less by edge failures. We consider two kinds of sparse partitions: *weak*, where the diameter of a cluster is with respect to all nodes in G, and *strong*, where the diameter of a cluster is calculated within the cluster. Weak partitions are available for more kinds of graphs than strong partitions [7, 11].

To evaluate the performance of the directory and the repair operations, we analyze their communication cost. For the basic directory operations (publish, move, lookup) that send messages sequentially, the communication cost is the sum of the distances traversed by the sequential messages for each operation. For the repair mechanisms, we often send several messages in parallel, here the communication cost consists of the total number of messages and the maximum distance that any one of these messages traverses.

Table 1 shows the communication costs of directory operations before/after f edge failures:

- **Publish:** A publish operation costs $\mathcal{O}(D \cdot \log n)$. After $f \geq 1$ failures the publish operation costs $\mathcal{O}(D' \cdot \log n)$, where D is the diameter of G before the edge failures and D' is the diameter after the edge failures. Note here that we do not compare with the optimal, as there is really no specific way that optimizes this step.
- **Lookup:** For lookup, the message cost of our algorithm is an $\mathcal{O}(\log^3 n)$ approximation of the optimal cost (compared to the shortest path to the token). This is a $\log n$ factor improvement over Spiral [21]. With f edge failures, the approximation becomes $\mathcal{O}(f \cdot \log^3 n)$ for weak partitions, and $\mathcal{O}(f \cdot \log^2 n + \log^3 n)$ for strong partitions.
- **Move:** For move, the amortized cost of a sequence of move operations is an $\mathcal{O}(\log D \cdot \log^2 n)$ approximation of the optimal. With f edge failures, the approximation factor becomes $\mathcal{O}(f \cdot \log D' \cdot \log^2 n)$ for weak and $\mathcal{O}((f + \log n) \log D' \cdot \log n)$ for strong partitions.

Table 1 Cost of operations for general/special graphs and weak/strong diameter partitions; publish cost is absolute; lookup and move costs are approximation factors compared to the optimal cost; failures are $f \geq 1$; D' is the diameter of G after the f failures; special graphs include constant doubling dimension, constant pathwidth (weak and strong partitions) and also fixed minor-free, chordal (weak partitions only) for which sparse partition schemes with $\sigma, I \in \mathcal{O}(1)$ are known [7].

Graph	Partition	Partition Parameters	Failures	Publish	Lookup	Move
general	any	$(\mathcal{O}(\log n), \mathcal{O}(\log n))$	none	$\mathcal{O}(D \cdot \log n)$	$\mathcal{O}(\log^3 n)$	$\mathcal{O}(\log D \cdot \log^2 n)$
general	weak	$(\mathcal{O}(\log n), \mathcal{O}(\log n))$	f	$\mathcal{O}(D' \cdot \log n)$	$\mathcal{O}(f \cdot \log^3 n)$	$\mathcal{O}(f \cdot \log D' \cdot \log^2 n)$
general	strong	$(\mathcal{O}(\log n), \mathcal{O}(\log n))$	f	$\mathcal{O}(D' \cdot \log n)$	$\mathcal{O}(f \cdot \log^2 n + \log^3 n)$	$\mathcal{O}((f + \log n) \log D' \cdot \log n)$
special	any	$(\mathcal{O}(1), \mathcal{O}(1))$	none	$\mathcal{O}(D)$	$\mathcal{O}(1)$	$\mathcal{O}(\log D)$
special	any	$(\mathcal{O}(1), \mathcal{O}(1))$	f	$\mathcal{O}(D')$	$\mathcal{O}(f)$	$\mathcal{O}(f \cdot \log D')$

Table 2 Cost of repair mechanism for general graphs and strong/weak partitions; σ is a sparse partition parameter, generally of order $\mathcal{O}(\log n)$; ρ is the locality parameter usually a constant; a cluster at level i has diameter at most $\sigma\rho^i$ independent of the number of failures; the hierarchy consists of $\log_\rho D$ levels; D denotes the diameter of G before the edge failure, D' denotes the diameter of G after the edge failure; n denotes the number of nodes, m denotes the number of edges.

Operation	Partition	Size of Message	Number of Messages	Maximum Distance Traversed by Individual Message
Initialize Shortest Path Tree Update	any	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(D)$
Update Shortest Path tree (per tree)	any	$\mathcal{O}(\log n)$	$\mathcal{O}(m)$	$\mathcal{O}(D')$
Splitting a cluster (per level i cluster)	strong	$\mathcal{O}(\log n)$	1	$\sigma\rho^i$
	weak	$\mathcal{O}(\log n)$	2	$\sigma\rho^i$
Informing nodes within cluster of split (per level i cluster)	strong	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\sigma\rho^i$
	weak	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$2\sigma\rho^i$
Informing neighborhood of leader change (per cluster)	any	$\mathcal{O}(\log n)$	$\mathcal{O}(n^2)$	ρ^i
Update Directory Path (per level)	any	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(D')$
Updating the Special Parent Information of level i cluster	strong	$\mathcal{O}(\log n)$	2	$\sigma\rho^{i'}$
	weak	$\mathcal{O}(\log n)$	2	$2\sigma\rho^{i'}$

For special kinds of graphs [7], we get better bounds which are $\mathcal{O}(1)$ (for f failures $\mathcal{O}(f)$) approximation for lookup, and $\mathcal{O}(\log D)$ (resp. $\mathcal{O}(f \cdot \log D')$) approximation for move, while the cost of the publish operation is simply $\mathcal{O}(D)$ (resp. $\mathcal{O}(D')$).

We also analyze the repair communication costs (see Table 2). To maintain the sparse partition, we store a spanning tree within each cluster of $\mathcal{P}$. If an edge fails in the spanning tree of a cluster X, we split X into two. This requires one message in a strong and two messages in a weak sparse partition of size $\mathcal{O}(\log n)$, traversing a distance of at most $\sigma\rho^i$ (the cluster diameter) in a strong partition and at most $2\sigma\rho^i$ in a weak partition, where σ and ρ are parameters defining the sparse partition hierarchy (cf. Subsection 1.2). There are additional steps, such as informing all nodes within X of the split, requiring $\mathcal{O}(n)$ messages of size $\mathcal{O}(\log n)$ traversing similar distance. More details are given below.

1.2 Techniques

Each level of the hierarchy $\mathcal{P}$ is a partition of $V(G)$ into clusters obtained from a (σ, I)-sparse partition scheme. At level i, each cluster has a diameter of at most $\sigma\rho^i$ (where ρ is a constant), and the ρ^i-neighborhood of a node intersects with at most I of these. The highest level of $\mathcal{P}$ consists of a single cluster, while on the lowest level, each node of G forms its own cluster. We pick a leader in each cluster X of $\mathcal{P}$. The leader of the highest level is called the "root".

The distributed directory maintains a directory path ϕ from the root to the current owner of the token t (see Figure 2). On each level of $\mathcal{P}$, exactly one leader node belongs to ϕ and has pointers to the directory path nodes at the level above and below, forming a double linked list. All operations are executed through message passing. The directory path is initialized by the first owner of t through a publish operation. A lookup or move operation, issued by a node v, searches for ϕ by checking the level i leaders of the nodes in the ρ^i-neighborhood of v, for all increasing levels i. When the operation discovers the directory path, it follows ϕ toward the token t. A move operation changes the directory path toward the new owner while it searches for ϕ.

To search for the directory path, node v needs to know the leaders of the nodes in its ρ^i-neighborhood for $0 \leq i \leq h$. To avoid double computation, every node pre-computes this information. Edge failures can increase the distance between some nodes, thereby affecting the precomputed ρ^i-neighborhoods. To update the preprocessed information, every node v maintains a shortest path tree $T(v)$. When an edge on $T(v)$ fails we use King's fully dynamic algorithm for maintaining shortest path trees [13] to update it. To initialize the update of the shortest path trees the endpoints of the failed edge inform the nodes whose shortest path tree are affected. This requires at most $\mathcal{O}(n)$ messages of size $\mathcal{O}(\log n)$ that traverse a distance at most $\mathcal{O}(D)$, where D is the diameter of G before the edge failure.

To improve the performance of lookup operations, a leader node $l(X)$ added to ϕ at level i informs its level i' leader $l_{i'}(l(X))$ for $i' = i + \log_\rho(c'\sigma)$ for an appropriately chosen constant c'. We call $l_{i'}(l(X))$ the special parent of $l(X)$. When a lookup operation finds the special parent of a node on the directory path, it traverses the directory path from there.

Upon an edge failure, both endpoints detect the failure immediately. In response to the failure, we update $\mathcal{P}$ to maintain the directory's performance. For each cluster X, we store a spanning tree $T(X)$. When an edge e on $T(X)$ fails, we split X into X_1 and X_2. X_1 has the same leader as X, and X_2 has a node incident to e as its leader. (If in a weak diameter partition e is outside X, $l(X_2)$ is selected appropriately in X_2.) This mechanism ensures that the diameter of any level i cluster of $\mathcal{P}$ is at most $2\sigma\rho^i$ regardless of the number of failures. The sparse partition scheme ensures that in a strong partition, at most one cluster splits per level of $\mathcal{P}$, and in a weak sparse partition, at most I clusters split. All these processes are initialized by the two endpoints of the failed edge.

When a cluster X with a leader on the directory path splits, we update the directory path to include the leader of the node that added $l(X)$ to ϕ. This ensures that lookup and move operations find ϕ at a level proportional to the distance between the token owner and the node that issued the operation. To update the directory path, the leader nodes of $l(X_1)$ and $l(X_2)$ need to communicate with each other and with the leader nodes on the directory path at the level below and above X. The whole process requires $\mathcal{O}(1)$ messages of size $\mathcal{O}(\log n)$, traversing a distance of at most $\mathcal{O}(D')$, where D' is the diameter of G after the edge failure.

We update the special parent information and notify nodes in the ρ^i-neighborhood of nodes in X_2 about the leader change as we update $\mathcal{P}$ and ϕ. To update special parents for a level i cluster X, two messages of size $\mathcal{O}(\log n)$ are required, traversing a distance of at most $\sigma\rho^{i'}$ in a strong partition and $2\sigma\rho^{i'}$ in a weak partition, where i' is the level of the special parents. To notify the ρ^i-neighborhood of nodes in X_2, $\mathcal{O}(n)$ messages of size $\mathcal{O}(\log n)$ are required, traversing a distance of ρ^i.

In unweighted graphs, a failure can at most double the diameter of G [3]. However, as illustrated in Figure 1, the diameter increase in a weighted graph may not be bounded. To accommodate such changes, we ensure that the number of layers in $\mathcal{P}$ equals $\log_\rho D'$, where D' represents the current graph diameter. When adding layers to the partition hierarchy, we extend the directory path accordingly.

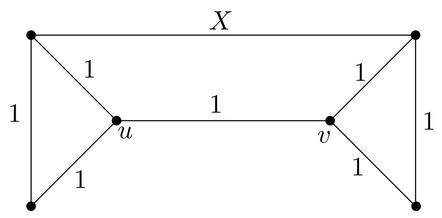

Figure 1 An example of a graph showing that we cannot bound the stretch in the diameter of the graph. Assuming $X > 1$ the initial graph has diameter 3. If edge $\{u, v\}$ fails, the diameter becomes $2 + X$. Without further assumptions on X this cannot be bounded as a constant multiple of 3.

1.3 Related Work

An alternative way to implement a distributed directory is to use a spanning tree T on G. The edges of T are directed toward the owner node of the token. If node u requests the token, then the move request redirects the edges of the tree toward u (edge reversal). The benefit of the tree is that it can easily handle distributed requests since concurrent move operations are ordered when they intersect on the tree. Several protocols have been proposed based on trees: Arrow [4,9,14,20], Relay [24], Ivy [15], Arvy [12]. The approximation factor of the operations is $\mathcal{O}(\log D_T)$, with respect to the diameter D_T of T. However, by using a tree the performance of the lookup and move operations may be sub-optimal with respect to G, as T may not accurately represent the distances in G. Considering the distance stretch s of the tree the approximation becomes $\mathcal{O}(s \log D_T)$, and s can be as large as the graph G diameter D. Nevertheless, considering an appropriate overlay tree that preserves on average the pairwise node distances of G [6], it is possible to get close to optimal performance on the average case for a set of random source operation requests [8,18]. Our approach, on the other hand, has guaranteed performance for arbitrary sources of requests (not just random).

Another work [5] considers fault-tolerant routing and labeling schemes. These rely on knowing the destinations of messages. In our case, the destinations are leaders which may not be immediately known after the failures. Hence, we cannot rely on such routing schemes directly to implement the fault-tolerant directory. Another line of research related to edge failures maintains fault-tolerant sparse spanners of G that preserve the stretch (usually poly-log) of the distances in G even after edge or node failures [2,17].

Outline of the Paper

In Section 2, we give some necessary definitions and define our model. Section 3 presents the basic directory scheme without failures and Section 4 describes our failure response mechanisms and analyzes their costs. A performance analysis of the directory after f failures is given in Section 5. In Section 6, we describe the integration of these mechanisms into the protocol. We conclude in Section 7. Omitted proofs and the pseudocode appear in the appendix. (The cases of concurrent edge failures and handling transient failures are discussed in the full version of the paper.)

2 Definitions and Preliminaries

Let $d_G(u, v)$ denote the length of a shortest path between u and v in G. The r-neighborhood of a node u, denoted $N_{G,u}(r)$, is the set of nodes that are within distance r to u. The diameter of a graph is $\operatorname{diam}(G) = \max_{u,v \in V(G)} d_G(u, v)$. For a set $X \subseteq V$, let $G[X]$ denote the subgraph of G induced by X. There are two ways to measure the diameter of X: (i) *weak diameter*, $diam_G(X)$, which considers all possible shortest paths in G that may also use nodes outside X; (ii) *strong diameter*, $diam_{G[X]}(X)$, which considers only paths in X.

A partition of G is a collection of disjoint sets of nodes whose union is V. A *sparse partition* is a partition that restricts both the diameter of each cluster and the number of clusters within a specific distance. There are weak and strong sparse partitions that differ in whether the weak or strong diameter of a cluster is restricted.

A (r, σ, I)*-weak (strong) sparse partition* of G satisfies two properties:

(i) each cluster has weak (strong) diameter at most $r\sigma$, and

(ii) the r-neighborhood of each node $u \in V$ intersects at most I clusters.

A (σ, I)*-weak (strong) sparse partition scheme* is a procedure that gives a (r, σ, I)-weak (strong) partition for any $r > 0$. Jia *et al.* [11] give a $(\mathcal{O}(\log n), \mathcal{O}(\log n))$-weak sparse partition scheme for an arbitrary metric space and general graphs. Filtser [7] gives a $(\mathcal{O}(\log n), O(\log n))$-strong partition scheme for general graphs based on the clustering technique by Miller *et al.* [16]. There are $(\mathcal{O}(1), \mathcal{O}(1))$-partition schemes for special network topologies such as for low doubling-dimension and fixed minor-free graphs [7, 11].

2.1 Model

We model the distributed network as a weighted graph $G = (V, E, w)$ with positive edge weights of at least one. The weight of an edge $e = \{u, v\}$ represents the cost of sending a message over edge e. The cost of an operation is the sum of the edge weights the request traverses. The goal of a distributed directory is to minimize the total communication cost for a request in the worst case. The edge weight represents solely the cost of sending a message but does not indicate the delay or latency of an edge. In particular, for the correctness of our protocol, no message synchronization is needed.

Each node u stores a shortest path tree $T(u)$ with $root(T(u)) = u$. The shortest path trees are built such that the shortest paths are consistent, meaning the path from u to v stored in $T(u)$ is identical (reversed) to the path from v to u stored in $T(v)$.

The communication in our network is asynchronous, and messages sent along the same edge are delivered in the order they are sent. All messages have the same size $\mathcal{O}(\log n)$ and are transmitted along shortest paths.

Our directory is built on a sparse partition hierarchy $\mathcal{P}$ of $G = (V, E, w)$. Our directory works for strong and weak sparse partitions, but we show that they have different performances. We construct $\mathcal{P}$ using any of the aforementioned partition schemes. We use $\text{diam}(X)$ to denote the strong or weak diameter of cluster X, depending on the partition type. The partition hierarchy $\mathcal{P}$ comprises $h = \lceil \log_\rho D \rceil$ levels with an exponentially increasing locality parameter ρ at each level. At level i $(0 \leq i \leq h)$, we define P_i as a (r_i, σ, I)-sparse partition of G, where $r_i = \min\{D, \rho^i\}$. We define level -1 where each node of V is a cluster by itself $(r_{-1} = 0)$. At top level h, $\mathcal{P}_h$ comprises of a single cluster that covers the entirety of V.

We select a leader $l(X)$ in each cluster X of $\mathcal{P}$. At P_{-1}, each node is the leader of its own cluster, while the leader of the single cluster in P_h is called the *root*. Each node $u \in V$ belongs to exactly one cluster in each level of $\mathcal{P}$, denoted by $C_i(u)$, and its leader is $l_i(u) = l(C_i(u))$. Each node knows the leader of all the clusters to which it belongs.

To maintain the sparse partition in the presence of edge failures, we store a spanning tree $T(X)$ for each cluster X. In a weak sparse partition, $T(X)$ is the shortest path tree of $l(X)$, while in a strong sparse partition, $T(X)$ is a shortest path tree of $G[X]$ with root $l(X)$. The choice of spanning tree ensures that for any node u in X, the path on $T(X)$ connecting u and $l(X)$ is at most $diam(X)$. We denote the subtree of $T(X)$ rooted in u by $T_{\setminus u}(X)$. Every

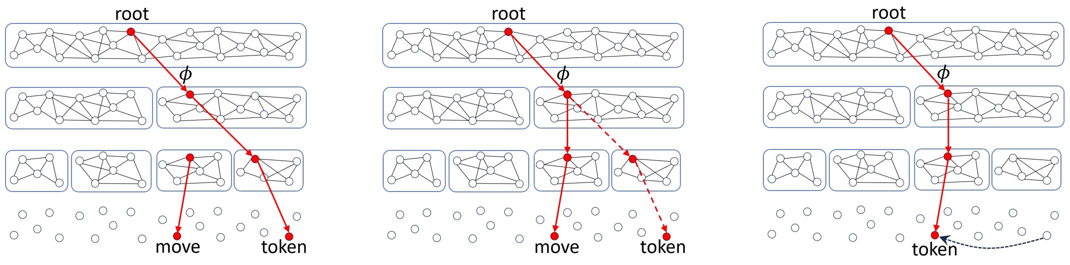

Figure 2 An example of a move operation. **Left:** a node issues a move request, which starts the bottom-up formation of a new directory path. **Middle:** the new directory path intersects the existing directory path ϕ; the part of ϕ from the intersection until the old owner will be deleted. **Right:** the token has moved to the new owner and the directory path ϕ has been revised accordingly.

node u on $T(X)$ knows $T(X)$. In a weak sparse partition, u also knows which nodes belong to X. To store this information, a node in a weak sparse partition requires $\mathcal{O}(Ihn)$ memory, and a node in a strong sparse partition requires $\mathcal{O}(hn)$ memory.

3 Directory Scheme

The directory supports three operations: *publish* to build the initial directory path, *lookup* to read the current value of the token, and *move* to request ownership of the token and update the directory path. All three operations are executed through message passing. Nodes can issue lookup and move operations at any moment and simultaneously. Lookup operations simply get a copy of the token from the latest token owner. Concurrent move operations from different nodes are ordered through the directory because there can only be one token owner node at a time. Hence, the directory acts as a distributed queue for the move requests. The queue ordering is implicit by the way the concurrent move operations intersect each other in the directory data structure. The directory ensures that the previous token owner will know which node is the next token owner. In this way, the token is passed from its previous owner to the next in the queue. There is no requirement that the requests are served in a particular order, but every move request has to be served eventually. (The pseudocode of the directory is displayed in Algorithm 1 in Appendix C.)

The token resides at an owner node at the lowest level. There is a virtual *directory path* ϕ that points to the owner (see Figure 2). The path ϕ consists of $h+2$ leader nodes, one node at every level of $\mathcal{P}$. Let ϕ_i denote the leader node of ϕ at level i, for $-1 \le i \le h$, where ϕ_{-1} is the token owner and ϕ_h is the root. For each level i, $0 \le i \le h-1$, leader ϕ_i has pointers to ϕ_{i-1} and ϕ_{i+1} which form a virtual doubly linked list.

The directory path is created via the publish operation. The initial owner u sends a `publish`-message to its leader nodes at every level of $\mathcal{P}$, namely $\phi_i = l_i(u)$. This operation creates the pointers from ϕ_i to ϕ_{i-1} (both ways) for $0 \le i \le h$. Theorem 1 measures the cost of a publish operation and the length of the initial directory path. In the next result, $cost(publish)$ is the total distance that the `publish`-message traverses.

▶ **Theorem 1.** *The cost of the publish operation and the length of the initial directory is $\mathcal{O}(D)$.*

Proof. To build the directory path, node u contacts each of its leader nodes. Therefore, $cost(publish) \le \sum_{i=0}^{h} \sigma\rho^i = \frac{\sigma\rho^{h+1}-1}{\rho-1}$. Since the nodes on the directory path u's leaders, the distance between consecutive nodes is at most $d(\phi_i,\phi_{i+1}) = d(l_i(u), l_{i+1}(u)) \le d(l_i(u), u) + d(u, l_{i+1}(u)) \le \sigma(\rho^i + \rho^{i+1})$. Summing over the entire directory path gives $length(\phi) \le \sum_{i=-1}^{h-1} \sigma(\rho_i + \rho_{i+1}) \le \frac{\sigma(\rho+1)(\rho^{h+1}-1)}{(\rho-1)\rho} = \mathcal{O}(\rho\sigma^h)$. ◀

To locate the token, the issuer of a lookup or move operation first searches for the directory path (Figure 2). Once a leader node of the directory path is found, the token owner is reached by following ϕ. To search for the directory path, a node sends messages to leader nodes near itself at increasing levels of $\mathcal{P}$: Let $P_i(v)$ be the set of clusters in P_i that intersect $N_{G,v}(\rho^i)$, where $|P_i(v)| \leq I$. Node v checks whether any of the leaders in $P_i(v)$ equals ϕ_i for $0 \leq i \leq h$. The search stops at the lowest level where a directory path leader is found (at the root in the worst case). The next Lemma bounds the cost of searching level i for ϕ.

▶ **Lemma 2.** *The sum of all distances that messages travel during the search of the directory path at level i in a lookup or move operation issued by node u is $\mathcal{O}(\rho^i \sigma I)$.*

Proof. Node u contacts every leader in $P_i(u)$. By definition, $|P_i(u)| \leq I$. Further, if X is in $P_i(u)$, then there exists a node x in X with $d(u,x) \leq \rho^i$. Therefore, $d(u, l(X)) \leq d(u,x) + d(x, l(X)) \leq \rho^i + \sigma\rho^i$. Summing over all clusters in $P_i(u)$ gives the result. ◀

A move operation by node v modifies the directory path to denote the new ownership at v. The new directory path is formed in a bottom-up way while v searches for the existing directory path (Figure 2). Node v first adds $l_{-1}(v)$ to the directory path. Let ϕ_j, $j \geq 0$, be the first node of ϕ that v discovers. For levels $0 \leq i < j$, node v searches $P_i(v)$ and adds $l_i(v)$ to the directory path when it does not find ϕ_i. At level j node v finds ϕ_j, which will remain then in the directory path but its pointer changes to ϕ'_{j-1}. The move operation then follows the old directory path toward ϕ_{-1}. While going down the move operation deletes the leaders from the old directory path, that is, $\phi_{j-1} \cdots \phi_{-1}$ are removed from ϕ. Hence, the new directory path is $\phi_h \cdots \phi_j \phi'_{j-1} \cdots \phi'_{-1}$.

Concurrent move operations create multiple partial directory paths. However, only the latest directory path includes the root node. To ensure a unique complete directory path (without splits or gaps) from the root to the owner, the upward phase of a move is atomic. Contacting ϕ_i about the directory path triggers an immediate update of its downward pointer to ϕ'_{i-1}, directing subsequent operations to the new path. As sub-paths merge, the distance between consecutive nodes on the directory path can increase, as shown by the next lemma.

▶ **Lemma 3.** *The distance between two consecutive nodes ϕ_i and ϕ_{i+1} on the directory path for $-1 \leq i < h$ is at most $d(\phi_i, \phi_{i+1}) \leq \sigma(\rho^i + \rho^{i+1}) + \rho^{i+1}$.*

Proof. Consider two consecutive directory path nodes ϕ_i and ϕ_{i+1}. There were either added by the same node v, in which case $d(\phi_i, \phi_{i+1}) \leq d(\phi_i, v) + d(v, \phi_{i+1}) \leq \sigma(\rho^i + \rho^{i+1})$, or some node v added ϕ_i to ϕ and then found ϕ_{i+1} during its search of level $i+1$. In this case, there must be a node w in v's ρ^{i+1}-neighborhood that belongs to ϕ_{i+1}'s cluster. Hence, $d(\phi_i, \phi_{i+1}) \leq d(\phi_i, v) + d(v, w) + d(w, \phi_{i+1}) \leq \sigma(\rho^i + \rho^{i+1}) + \rho^{i+1}$. ◀

The continuous modifications of the directory path imply that not all leaders on the directory path contain the current token owner w in their cluster. To ensure that a lookup operation issued by node u discovers the directory path at a level proportional to its distance to w, we use the concept of a *special parent* (as in Spiral [21]). Every leader node $l_i(x)$ that is added to ϕ informs leader node $l_{i'}(x)$, where $i' = i + \log_\rho(c'\sigma)$, for an appropriately constant c'. We call $l_{i'}(x)$ the special parent of $l_i(x)$ with respect to level i. When node u searches for the directory path, it asks the leader nodes if they are part of the directory path, or if they are the special parent of a node on the directory path. If it finds a special parent, it takes the link to the node on the directory path and continues the search from there. Like the directory path itself, special parent information is updated during a move operation. If a move removes a node from the directory path while a lookup follows a link from a special parent to that node, the lookup goes back to level i' and continues the search from there.

Using Lemmas 2 and 3, we obtain the following results for the lookup and move costs when there are no failures. (The proofs of these two results appear in the full version of the paper; here we focus on the fault analysis aspects.)

▶ **Lemma 4.** *A lookup operation finds the token with a cost that is a $O(\sigma^2 \rho I)$ factor from optimal.*

Consider a sequence of move requests $S = s_1, \ldots, s_q$, that execute in a sequential manner, so that s_i starts only after s_{i-1} completes, where $i > 0$.

▶ **Lemma 5.** *The total cost of the move operations in S is a $\mathcal{O}(h\rho\sigma(\sigma + I))$ factor from optimal.*

4 Responding to Edge Failures

In case of edge failures, our clustering may no longer satisfy the properties of a sparse partition, and some of the shortest path trees that nodes store may become disconnected. To guarantee the correctness and performance of our algorithm, we update our data structures accordingly. To accomplish this, we modify the operations described in Section 3 as follows:

1. Each node on ϕ remembers the node that added it to ϕ.
2. When w contacts $l(X)$ in a lookup or move operation to find the directory path, it includes a list of the nodes from w's ρ^i-neighborhood that it believes are part of X.
3. Node w contacts a level i leader node $l(X)$ only if $d(w, l(X)) \leq \rho^i + 2\sigma\rho^i$.

Whenever an edge e fails, our update mechanisms recompute all shortest path trees that contained edge e, split every cluster whose spanning tree contained edge e, and update the directory path accordingly. We discuss each of them below.

4.1 Updating Shortest Path Trees

Each node in our protocol stores a shortest path tree, which we update when an edge $e = \{u, v\}$ on the tree fails. We use King's fully dynamic algorithm to maintain the shortest path trees in the presence of edge failure [13]. Updating a single tree takes $\mathcal{O}(md)$ time, where m is the number of edges in $G \setminus \{e\}$, and d is the maximum distance of a node to the root of the tree. To use King's centralized algorithm in a distributed system, we let the root node compute the updated shortest path tree. This causes an additional cost because we need to inform the root node of the available edges. Namely, we need to inform the root of at most $\mathcal{O}(m)$ edges with a maximum distance of D' from the root.

The updating of the shortest path trees is initialized by the endpoints of the failed edge $e = \{u, v\}$, which can detect the failure immediately. The consistency assumption we placed on the shortest path trees implies that u and v know if any tree needs to be updated.

▶ **Observation 6.** *If edge $e = \{u, v\}$ is part of the shortest path tree of a node w, then edge e is also part of the shortest path tree of u and v.*

To initialize the updates, nodes u and v send a broadcast along the remaining parts $T(u)$ and $T(v)$. The next lemma bounds the cost of this operation.

▶ **Lemma 7.** *To initialize the update of the shortest path tree, $\mathcal{O}(n)$ messages of size $\mathcal{O}(\log n)$ are required, that travel a maximum message distance of D, where D is the diameter of G before the failure of e.*

4.2 Reclustering

When an edge e fails, we split every cluster X whose spanning tree contains e into two. For the reclustering, we distinguish between the root level and clusters below the root level. For clusters below level h, we define the two new clusters as $X_1 = X \setminus (X \cap T_{\setminus v}(X))$ and $X_2 = X \cap T_{\setminus v}(X)$. The leader node of X_1 is the same as that of X, and for X_2, either v becomes the new leader (if using strong sparse partition) or the closest node to v in X (if using weak sparse partition). The spanning tree for X_1 is $T(X) \setminus T_{\setminus v}(X)$ and for X_2 is $T_{\setminus v}(X)$ rerooted at $l(X_2)$. The following lemma bounds the diameter and the number of the generated clusters.

▶ **Lemma 8.** *Let X be a cluster at level i, $-1 \leq i < h$, and suppose at most f edges fail. Then X splits into at most $f+1$ clusters. Each new cluster X_j, generated from X, has diameter at most $diam(X_j) \leq 2\sigma\rho^j$.*

Similar to the update of the shortest path tree, the splitting of the clusters is initiated when the endpoints of the failed edge detect the failure. Recall that u and v both know which clusters have e in their spanning tree. For every cluster X that needs to split the path on $T(X)$ from $l(X)$ to one of the endpoints of e, say u, the part between u and $l(X)$ remains unaffected by the failure of e. Therefore, u can inform $l(X)$ of the failure by sending a message along $T(X)$. If node v is not in X (i.e. weak diameter sparse partition), it chooses a node w closest to it on $T(X)$ from $X \cap T_{\setminus v}(X)$ to become the new leader node and sends a message to w to inform it of the reclustering and its new leadership role.

▶ **Lemma 9.** *Splitting a level i cluster requires one message in a strong and up to two messages in a weak sparse partition. These have size $\log n$ and traverse a distance of at most $\sigma\rho^i$.*

When $l(X_1)$ is informed about the failure and $l(X_2)$ knows whether it is part of the directory tree (see Section 4.3), both broadcast the update to all nodes in their respective cluster so that these can update their knowledge of $T(X)$ and the leader for the nodes in X_2.

▶ **Lemma 10.** *To inform all nodes in X_1 and X_2 of the cluster change we need to send $\mathcal{O}(n)$ messages, each of which has size $\mathcal{O}(\log n)$. In a strong partition, the maximum distance a message traverses is $\sigma\rho^i$ and in a weak sparse partition, the maximum distance is $2\sigma\rho^i$.*

When the nodes in X_2 are informed of the cluster change, they forward this information to their ρ^i-neighborhood, so they can update their preprocessing information.

▶ **Lemma 11.** *To inform the ρ^i-neighborhood of the nodes in X_2 about the new leader requires $\mathcal{O}(n^2)$ messages of size $\mathcal{O}(\log n)$. The maximum distance traversed by any message is ρ^i.*

Initially, $\mathcal{P}$ consists of $\log_\rho D$ layers, where $D = \text{diam}(G)$. To maintain this relationship between the number of layers and the diameter, we add layers to $\mathcal{P}$ as the diameter increases.

Consider the single cluster X at level h. Before the edge failure, we have $d(r,u) \leq \sigma\rho^h$ on $T(X)$ for all nodes $u \in V$. Hence, if the failed edge e does not lie on $T(X)$, then the diameter of G is at most $2\sigma\rho^h$. In this case, we do not modify the root level.

If edge e lies on $T(X)$, then r's updated shortest path tree tells us if we need to increase the number of layers. Let V_1 and V_2 be the partition of the vertices of G defined by the connected components of $T(X) \setminus \{e\}$, and assume w.l.o.g. that $r \in V_1$. By Lemma 8, $\text{diam}(G[V_1]) \leq 2\sigma\rho^h$ and $\text{diam}(G[V_2]) \leq 2\sigma\rho^h$. Let v be the node in X_2 closest to r before the edge failure. If after the edge failure, the distance from r to v is $d(x,r) > \sigma\rho^{h+1} - 2\sigma\rho^h$, then we increase the number of layers to h', where $h' = \min_{z \in \mathbb{Z}}\{\sigma\rho^z > d(x,r)\}$ for all $x \in V$.

We split the level h the same way that we split lower level clusters whose spanning tree contains edge e. Levels $h+1,\dots,h'-1$ are simply copies of level h, and the directory path goes through the copy of the cluster that is part of the directory path at level h.

When the single cluster at level h splits, both leader nodes of the two generated clusters inform the nodes in their clusters about the additional layers. This information can be included in the usual message $l(X_1)$ and $l(X_2)$ sent to the nodes in their clusters when a failure occurs. This message will cause the directory path nodes at levels $h-(i+\log_\rho(c'\sigma))+1,\dots,h'-i+\log_\rho(c'\sigma)$ to send a message to their special parent, so that this information is also extended to the additional layers. Level h' is a single cluster that contains the entire graph. The leader node is r and the spanning tree is $T(r)$.

4.3 Updating the Directory Path and Special Parents

We need to ensure that the directory path and special parent information are maintained during cluster splitting. When a cluster X splits, we check if $l(X)$ is on the directory path. If it is not, then neither $l(X_1)$ nor $l(X_2)$ will be. If it is, then $l(X)$ can determine whether $l(X_1)$ or $l(X_2)$ becomes part of the directory path by checking if the node that added $l(X)$ to ϕ remains in X_1. Once $l(X)$ knows about $l(X_2)$'s role, it informs v (using $T(l(X))$). If v is not $l(X_2)$, then it forwards this message to $l(X_2)$. If $l(X_2)$ is to become part of the directory path, then the message contains the ids of ϕ_{i-1}, ϕ_{i+1}, and the id of the node that added $l(X)$ to the directory path. With this, $l(X_2)$ sets its pointers to ϕ_{i+1} and ϕ_{i-1} and informs them to update their pointers too. When they receive $l(X_2)$'s message ϕ_{i-1} and ϕ_{i+1} send a message to $l(X)$ to remove its outdated directory path pointers. If $l(X_2)$ is not on the directory path, then $l(X)$'s messages simply informs $l(X_2)$ that it is not part of ϕ.

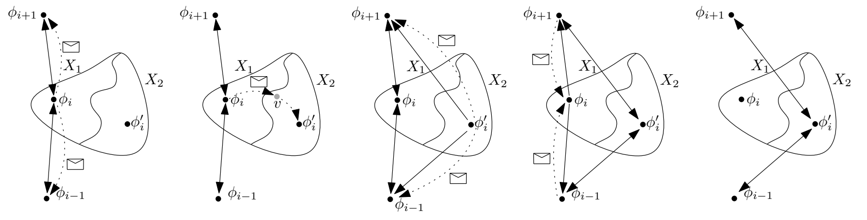

Figure 3 The steps of updating the directory path at level i: 1) Node ϕ_i sends a message to ϕ_{i-1} and ϕ_{i+1} to inform them about the update. 2) Node ϕ_i informs ϕ_i' to join the directory path. If we are using a weak sparse partition, node v acts as an intermediate in the message transfer. 3) ϕ_i' sets pointers to ϕ_{i-1} and ϕ_{i+1} and sends them a message so they update their pointers too. 4) ϕ_{i-1} and ϕ_{i+1} change their pointer to ϕ_i' and send a message to ϕ_i to remove its pointers. 5) ϕ_i removes its pointers to ϕ_{i+1} and ϕ_{i-1}.

For simplicity, we prevent consecutive nodes on the directory path to update concurrently (when the modification is due to an edge failure). Therefore, if $\phi_i = l(X)$ needs to be replaced by $l(X_2)$, then ϕ_i will first contact ϕ_{i-1} and ϕ_{i+1} to inform them of the update, before messaging $l(X_2)$. Neither of them will be able to initialize an update on their level until the update at level i is complete. In case two subsequent nodes attempt to initialize an update of the directory path simultaneously, then the node with the lower id will be allowed to update first. The process of updating the directory path is displayed in Figure 3. In the next lemma, we bound the cost of updating the directory path.

▶ **Lemma 12.** *To update the directory path we send $\mathcal{O}(1)$ messages of size $\mathcal{O}(\log n)$ which travel a distance at most $\mathcal{O}(D')$ where $D' = diam(G \setminus \{e\})$.*

At the same time that we modify the directory path, we also update the special parent information of $l(X_1)$ and $l(X_2)$. When $l(X_1)$ and $l(X_2)$ update ϕ, they also send a message to their special parent instructing them to update their pointers accordingly.

▶ **Lemma 13.** *To update the special parent information we send two messages of constant size that traverse a distance of at most $\sigma\rho^i$ in a strong sparse partition and at most $2\sigma\rho^{i'}$ in a weak sparse partition, where $i' = i + \log_\rho(c'\sigma)$.*

5 Analysis of Algorithm

We examine our protocol's performance with up to f faults. Edge failures may stretch the directory path, leading to delayed updates of special parent information. Consequently, operations that occur during or immediately after a failure may experience additional costs, referred to as *transient operations*. Once the directory path is rebuilt by a publish or move operation and the special parent information is updated, operations are considered *normal operations*. Here we analyze the cost of normal operations. (We discuss transient operations in the full version of the paper.)

We first bound the length of the directory path after failures.

▶ **Lemma 14.** *Suppose that since the last edge failure, the directory path has been rebuilt up to level i. Then the length of the directory path up to level i is at most $\mathcal{O}(\sigma\rho^i)$.*

Before we analyze the cost of lookup and move operations, we bound the number of clusters affected by an edge failure.

▶ **Lemma 15.** *A failure of edge $e = \{u, v\}$ splits at most h clusters in a strong, and Ih clusters in a weak sparse partition.*

In addition, if an edge failure occurs, we may add extra layers to the directory to accommodate the increased diameter. As explained in Section 4.2, the number of added layers is proportional to the diameter increase.

We now analyze the cost of lookup and move operations when up to f failures occur.

▶ **Theorem 16.** *Suppose we are using a strong sparse partition and that f edge failures have occurred. After updating our data structures, a lookup operation finds the token with cost that is $\mathcal{O}(\sigma^2(I + f)\rho)$ factor from optimal.*

Proof. Suppose node u issues a lookup request while the current owner is node v, where $u \neq v$, and $\rho^{i-1} \leq d_G(u, v) \leq \rho^i$. This implies the optimal cost is at least $d_G(u, v) \geq \rho^{i-1}$.

Let w be the directory path node at level i. By Lemma 14, the segment of ϕ from v to w is at most $c_1\sigma\rho^i$ for some constant c_1. Hence, $d_G(v, w) \leq c_1\sigma\rho^i$. Therefore, $d_G(u, w) \leq d_G(u, v) + d_G(v, w) \leq c_2\sigma\rho^i$, for some constant $c_2 \geq 1 + c_1$.

Let s_w be the special parent of w, which is the leader of the cluster that includes w at level $i' = i + \log_\rho(c'\sigma)$, for a constant $c' \geq c_2$. Since $\rho^{i'} = c'\sigma\rho^i$, node s_w is in the $\rho^{i'}$-neighborhood of u. Therefore, the lookup operation is guaranteed to discover s_w at level i'.

We sum the cost of the search up to level i'. On each level, node u contacts at most $I + f$ leaders by Lemma 15. When node u contacts a cluster leader node $l(X)$ at level i, then there must be a node x in X such that $d(u, x) \leq \rho^i$. Hence, the distance between u and $l(X)$ is at most $d(u, l(X)) \leq \rho^i + 2\sigma\rho^i \leq c\sigma\rho^i$ for some $c \geq 1 + 2\sigma$. Therefore,

$$\text{cost upward phase} \leq \sum_{j=0}^{i'} (I + f)c\sigma\rho^j = \mathcal{O}(\sigma^2(I + f)\rho^i).$$

The cost of the downward phase is given by Lemma 14. Thus, for strong sparse partitions, the total cost of a lookup is $\mathcal{O}(\sigma^2(I + f)\rho^i)$ which is $\mathcal{O}(\sigma^2(I + f)\rho)$ factor from optimal. ◀

▶ **Theorem 17.** *Suppose we are using a weak sparse partition and that f edge failures have occurred. After updating our data structures, a lookup operation finds the token with cost that is $\mathcal{O}(\sigma^2 f I \rho)$ factor from optimal.*

Proof. The proof is identical to Theorem 16, except that after f edge failures $P_i(u) \leq fI$, according to Lemma 15. Thus, the lookup operation visits up to fI clusters on each level. ◀

▶ **Theorem 18.** *Consider a sequence S of move requests $S = s_1, \dots, s_q$ that are all issued after the f^{th} edge failure and which are executed sequentially. The total cost of the move operations in S is a $\mathcal{O}(h'\sigma\rho((I+f)+\sigma)$ factor from optimal in a strong sparse partition (for sufficiently large S).*

Proof. Let $S_i = s_{i_1}, s_{i_2}, \dots, s_{i_z}$, $0 \leq i \leq h'$, be the sub-sequence of S that reach level i in their upward phase, where h' is the highest level of $\mathcal{P}$ after the f failures. And let u_{i_j} be the issuer of s_{i_j}. Define s_{i_0} to be the last move operation prior to S that reached level i. If no such operation exists, s_{i_0} is the initial publish operation.

Operation s_{i_j} forms a new directory path p_{i_j} that links the leaders of u_{i_j} up to level $i-1$. At level i, p_{i_j} links to ϕ_i, which is the level i leader of a node in u_{i_j}'s ρ^i-neighborhood.

We show that $d(u_{i_{j-1}}, u_{i_j}) > \rho^{i-1}$, for $j > 0$. Between $s_{i_{j-1}}$ and s_{i_j} no operation modified ϕ_{i-1}. Since s_{i_j} reaches level i, it does not discover $\phi_{i-1} = p_{i_{j-1}}$ at level $i-1$. This implies that $u_{i_{j-1}}$ is not in the $\rho^{i-1}-$neighborhood of $u_{i,j}$.

Let $C^*(S_i)$ denote the optimal cost of the operations in S_i and C^* be the optimal cost of all operations. Since the distance between any two consecutive nodes in S_i is more than ρ^{i-1}, we have that $C^*(S_i) > |S_i|\rho^{i-1}$, which implies that

$$C^*(S) \geq \max_{0 \leq i \leq h'} C^*(S_i) \geq \frac{\sum_{i=0}^{h'} C^*(S_i)}{h'+1} > \frac{\sum_{i=0}^{h'} |S_i|\rho^{i-1}}{h'+1}. \tag{1}$$

The cost of searching for the directory path up to level i is, the same for a move and a lookup operation. Hence, cost(S_i upward phase) $\leq |S_i| c\sigma^2(I+f)\rho^i$ for some constant c.

For the downward phase, we need to be more careful because s_{i_1} could be a transient operation for $0 \leq i \leq h'$, that encounters two consecutive nodes ϕ_k and ϕ_{k-1} with distance up to $d(\phi_k, \phi_{k-1}) = D'$, where D' is the diameter of G after the f edge failures. However, for each sub-sequence S_i, where $0 \leq i \leq h'$, only s_{i_1} can be a transient operation as subsequent operations will traverse the updated directory path. For normal move operations, we can bound the downward phase by the length of the upward phase. Hence, we have

$$C(S) \leq h'D' + 2\sum_{i=0}^{h'} |S_i| c\sigma^2(I+f)\rho^i. \tag{2}$$

From Equations 1 and 2, we get the competitive ratio for the move operations in S. For a strong sparse partition we have

$$\frac{C(S)}{C^*(S)} \leq \frac{(h'+1)(h'D' + 2\sum_{i=1}^{h'} |S_i| c\sigma^2(I+f)\rho^i)}{\sum_{i=0}^{h'} |S_i|\rho^{i-1}} = \mathcal{O}(h'\sigma\rho((I+f)+\sigma)),$$

where we assume the second term to be the dominating one, which holds for a sufficiently large set of move operations S (namely, $|S| = \Omega(h'^2 D')$). ◀

▶ **Theorem 19.** *The total cost of the move operations in S is a $\mathcal{O}(h'\sigma\rho(fI+\sigma))$ factor from optimal in a weak sparse partition (for sufficiently large S).*

Proof. The proof is identical to Theorem 18, except that to search a single layer for the directory path, node u needs to contact $|P_k(u_{i_j})| \leq fI$ clusters in a weak sparse partition. ◀

6 Adding Fault Tolerance to the Directory

In this Section, we explain how to integrate the fault-tolerance mechanisms into our directory. Here, we consider failures of one edge at a time. (Concurrent edge failures are discussed in the full version of the paper.) We first explain why we modified the directory operations as described at the beginning of Section 4: The first modification lets us determine if we need to update the directory path due to an edge failure. The second and third modifications ensure correctness and performance during the upward phase of a lookup or move operation.

When node w searches for the directory path its preprocessing information determines which nodes it contacts. If the leader of a node in w's ρ^i-neighborhood changes due to a cluster split, then w might contact the wrong leader if it does not get informed of the update in time. By including a list of all nodes that w believes to be part of the cluster when contacting a leader about ϕ, the leader can inform w if a node is no longer part of the cluster. In this case, w knows that it needs to wait for a cluster update to contact all leaders.

The distance to a leader $l(X)$ of a node x in w's ρ^i-neighborhood is at most $d(w, l(X)) \leq d(w,x) + d(x, l(X)) \leq \rho^i + 2\sigma\rho^i$ (by Lemma 8). If the distance between w and the node whom it believes to be x's level i leader is larger, then a cluster split must have occurred. Therefore, instead of paying a too high cost, w waits for a cluster update info by node x.

We discuss the cases of edge failures on the shortest path tree and during move operations. (The case of publish and lookup operations are covered in Appendix B.)

6.1 Edge Failure on the Shortest Path Tree

When an edge on $T(w)$ fails, w updates $T(w)$ immediately upon being informed of the failure, regardless of whether w was in the middle of a directory operation. All directory operations rely on $T(w)$: publish uses it to contact the leaders of w at the lowest cost possible, move and lookup further use on it to determine whose leaders to contact at each level.

If w is notified of the failure while performing an operation, it stops the operation, updates $T(w)$, and then resumes the operation. For a publish operation, w simply continues. For lookup and move operations, node w takes into account the updated shortest path tree: Suppose the edge failure increases the distance from x to w from d to d', where $\rho^{j-1} < d \leq \rho^j$ and $\rho^{k-1} < d' \leq \rho^k$, while w searches level i for the directory path. If $j > i$ and $k > j$, then the first time w contacts x's leader node is at level k (unless w contacts x's leader due to a different node in the cluster). If $j \leq i$ and $k > i$, then w does not contact $C_i(x)$, unless it already did so before being informed about the edge failure, or because there is a different node in w's ρ^i-neighborhood that belongs to cluster $C_i(x)$. The first time it contacts x's leader node is at level k. If $j \leq i$ and $k \leq i$, then w will contact cluster $C_i(x)$ during its level i search and at each subsequent level until it finds the directory path. The downward phase of a lookup or move operation is not affected by the edge failure on $T(w)$.

6.2 Edge Failure during Move Operation

While Searching for the Directory Path

The search phase of a move issued by w can only be affected by the edge failure if w needs to contact the leader of a cluster that splits due to the edge failure. Suppose that the level i leader of a node x in w's ρ^i-neighborhood changes due to a split of a cluster X.

There are two cases to consider depending on whether w is informed about the change before or after contacting $l(X)$. If w is informed before, then there is no issue. Otherwise, two sub-cases arise. If $l(X)$ is already aware of the failure, it informs w that it is not x's leader, causing w to wait for the cluster update message from x. If not, $l(X)$ responds to w's message as though x was still part of the cluster, and w does not need to contact x's new level i cluster since the information received from $l(X)$ is valid for w's new cluster.

The new directory path built during the search is unaffected by the failure of edge e.

While Following the Directory Path Downward

Assume the edge failure occurs while w's move follows the directory path down. If the downward phase of the move operation does not encounter any clusters that split, an update at level i completes before the move reaches level i, or if the directory path remains unchanged, then the failure does not affect this phase.

Suppose that the split of cluster X results in a modification to the directory path at level i, and $l(X)$ realizes the need for the modification before the move operation reaches ϕ_{i+1}. Two cases arise: If $l(X)$ has already initialized the modification at level i, ϕ_i sends a message to ϕ_{i+1} to inform it of the update. In this case, ϕ_{i+1} does not forward the move message until the modification is complete. If cluster X is waiting for a modification of the directory path at a level above or below i, the move operation either halts before reaching level i if the modification occurs above i. Or, if the modification occurs below i, the move operation traverses the old pointers up to the modified level and removes them, preventing $l(X)$ from initializing a directory path modification.

7 Conclusions

We presented a fault-tolerant directory scheme based on sparse partitions that tolerates edge failures. We showed that the performance of the directory is linearly affected by the number of failures f. We showed how to adjust the clusters due to failures to transform the σ and I parameters, such that σ simply doubles while I is affected by either a f factor (weak diameter clusters), or f additive term (strong diameter clusters).

There are a few open questions that remain to be studied. One is to handle partitions of G due to failures. The connected component that contains the token can still function and respond to operation requests. A related problem is examining the impact of node failures. If G has bounded-degree d a node failure corresponds to at most d edge failures, then the techniques we developed could be adapted to analyze node failures.

Another line of research related to preserving distances is building fault-tolerant sparse spanners. A sparse spanner of G is a subgraph H such that the pairwise distances on G are stretched by a small factor on H. There exist fault-tolerant sparse spanners that maintain the stretch of the distances even after edge or node failures [2, 17]. Inspired by this, a future research direction is to design failure-oblivious sparse partitions with appropriate multiple pre-selected leaders in each cluster. Such leaders would be able to handle the failures without the need for cluster restructuring.

References

1 Baruch Awerbuch and David Peleg. Concurrent online tracking of mobile users. In *Proceedings of the Conference on Communications Architecture & Protocols*, SIGCOMM '91, pages 221–233, New York, NY, USA, 1991. Association for Computing Machinery. doi:10.1145/115992.116013.

2 Greg Bodwin, Michael Dinitz, Merav Parter, and Virginia Vassilevska Williams. Optimal vertex fault tolerant spanners (for fixed stretch). In Artur Czumaj, editor, *Proceedings of the Twenty-*

Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018, pages 1884–1900. SIAM, 2018. doi:10.1137/1.9781611975031.123.

3 FRK Chung and MR Garey. Diameter bounds for altered graphs. *Journal of graph theory*, 8(4):511–534, 1984.

4 Michael J. Demmer and Maurice Herlihy. The arrow distributed directory protocol. In Shay Kutten, editor, *Distributed Computing, 12th International Symposium, DISC '98, Andros, Greece, September 24-26, 1998, Proceedings*, volume 1499 of *Lecture Notes in Computer Science*, pages 119–133. Springer, 1998. doi:10.1007/BFb0056478.

5 Michal Dory and Merav Parter. Fault-tolerant labeling and compact routing schemes. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 445–455. ACM, 2021. doi:10.1145/3465084.3467929.

6 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004. doi:10.1016/j.jcss.2004.04.011.

7 Arnold Filtser. Scattering and Sparse Partitions, and Their Applications. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 47:1–47:20, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2020.47.

8 Abdolhamid Ghodselahi and Fabian Kuhn. Dynamic analysis of the arrow distributed directory protocol in general networks. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 22:1–22:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.DISC.2017.22.

9 Maurice Herlihy, Fabian Kuhn, Srikanta Tirthapura, and Roger Wattenhofer. Dynamic analysis of the arrow distributed protocol. *Theory Comput. Syst.*, 39(6):875–901, 2006. doi:10.1007/s00224-006-1251-9.

10 Maurice Herlihy and Ye Sun. Distributed transactional memory for metric-space networks. *Distributed Comput.*, 20(3):195–208, 2007. doi:10.1007/s00446-007-0037-x.

11 Lujun Jia, Guolong Lin, Guevara Noubir, Rajmohan Rajaraman, and Ravi Sundaram. Universal approximations for tsp, steiner tree, and set cover. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, pages 386–395, New York, NY, USA, 2005. Association for Computing Machinery. doi:10.1145/1060590.1060649.

12 Pankaj Khanchandani and Roger Wattenhofer. The arvy distributed directory protocol. In Christian Scheideler and Petra Berenbrink, editors, *The 31st ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2019, Phoenix, AZ, USA, June 22-24, 2019*, pages 225–235. ACM, 2019. doi:10.1145/3323165.3323181.

13 Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 81–89. IEEE, 1999.

14 Fabian Kuhn and Roger Wattenhofer. Dynamic analysis of the arrow distributed protocol. In Phillip B. Gibbons and Micah Adler, editors, *SPAA 2004: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, June 27-30, 2004, Barcelona, Spain*, pages 294–301. ACM, 2004. doi:10.1145/1007912.1007962.

15 Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Trans. Comput. Syst.*, 7(4):321–359, 1989. doi:10.1145/75104.75105.

16 Gary L. Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*, SPAA '13, pages 196–203, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2486159.2486180.

17 Merav Parter. Nearly optimal vertex fault-tolerant spanners in optimal time: sequential, distributed, and parallel. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 1080–1092. ACM, 2022. `doi:10.1145/3519935.3520047`.

18 David Peleg and Eilon Reshef. A variant of the arrow distributed directory with low average complexity. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming*, ICALP '99, pages 615–624, Berlin, Heidelberg, 1999. Springer-Verlag.

19 Shishir Rai, Gokarna Sharma, Costas Busch, and Maurice Herlihy. Load balanced distributed directories. *Information and Computation*, 285(A), 2022. `doi:10.1016/j.ic.2021.104700`.

20 Kerry Raymond. A tree-based algorithm for distributed mutual exclusion. *ACM Transactions on Computer Systems*, 7(1):61–77, 1989. `doi:10.1145/58564.59295`.

21 Gokarna Sharma and Costas Busch. Distributed transactional memory for general networks. *Distributed computing*, 27(5):329–362, 2014.

22 Gokarna Sharma and Costas Busch. An analysis framework for distributed hierarchical directories. *Algorithmica*, 71(2):377–408, 2015. `doi:10.1007/s00453-013-9803-2`.

23 Gokarna Sharma, Hari Krishnan, Costas Busch, and Steven R. Brandt. Near-optimal location tracking using sensor networks. *International Journal of Networking and Computing*, 5(1):122–158, 2015. URL: `http://www.ijnc.org/index.php/ijnc/article/view/100`.

24 Bo Zhang and Binoy Ravindran. Dynamic analysis of the relay cache-coherence protocol for distributed transactional memory. In *24th IEEE International Symposium on Parallel and Distributed Processing, IPDPS 2010, Atlanta, Georgia, USA, 19-23 April 2010 - Conference Proceedings*, pages 1–11. IEEE, 2010. `doi:10.1109/IPDPS.2010.5470393`.

A Omitted Proofs

▶ **Observation 6.** *If edge $e = \{u, v\}$ is part of the shortest path tree of a node w, then edge e is also part of the shortest path tree of u and v.*

Proof. If this was not the case, then the shortest paths from w to u and v stored in $T(w)$ cannot be consistent with the shortest paths from v and u to w stored in $T(v)$ and $T(u)$. ◀

▶ **Lemma 7.** *To initialize the update of the shortest path tree, $\mathcal{O}(n)$ messages of size $\mathcal{O}(\log n)$ are required, that travel a maximum message distance of D, where D is the diameter of G before the failure of e.*

Proof. Broadcasting the id of the failed edge allows any node to detect if it needs to update its shortest path tree. The remainders of $T(u)$ and $T(v)$ contain at most n nodes and the maximum distance from any node to the root is at most D. ◀

▶ **Lemma 8.** *Let X be a cluster at level i, $-1 \leq i < h$, and suppose at most f edges fail. Then X splits into at most $f + 1$ clusters. Each new cluster X_j, generated from X, has diameter at most $diam(X_j) \leq 2\sigma\rho^j$.*

Proof. The connected components of $T(X) \setminus F$, where F is the set of failed edges, form the final clusters. Since $T(X) \cap F \subseteq F$, we remove at most f edges from $T(X)$, which means $T(X) \setminus F$ has at most $f + 1$ connected components.

Let X_j be a cluster generated through the splitting of the initial cluster X. By construction, the maximal distance of any node on $T(X)$ to $l(X)$ is at most $\sigma\rho^j$ (with respect to $T(X)$). Let u_j be the node in X_j that was closest to $l(X)$ on $T(X)$ and a and b be any two nodes in X_j. Then $d(a, b) \leq d(a, u_j) + d(u_j, b) \leq 2\sigma\rho^j$, because u_j must have been on the path from a, respectively b to $l(X)$ on $T(X)$. ◀

▶ **Lemma 9.** *Splitting a level i cluster requires one message in a strong and up to two messages in a weak sparse partition. These have size $\log n$ and traverse a distance of at most $\sigma\rho^i$.*

Proof. In any partition, node u sends a message to $l(X)$ to inform it about the failure. As $l(X)$ knows $T(X)$ it suffices to send u's id. If in a weak sparse partition node v is not in X_2, it selects a node w closest to it in X_2 to become $l(X_2)$. To inform w of its new leadership role, v sends a message with e's id along $T(X)$ to w, so w can update its knowledge on $T(X)$. ◀

▶ **Lemma 10.** *To inform all nodes in X_1 and X_2 of the cluster change we need to send $\mathcal{O}(n)$ messages, each of which has size $\mathcal{O}(\log n)$. In a strong partition, the maximum distance a message traverses is $\sigma\rho^i$ and in a weak sparse partition, the maximum distance is $2\sigma\rho^i$.*

Proof. Node $l(X_1)$ broadcasts the id of u and $l(X_2)$ broadcasts its own id and the id of v. This suffices to inform each node of its leader and to update $T(X)$. As $|X_1 \cup X_2| = \mathcal{O}(n)$, we send at most $\mathcal{O}(n)$ messages. Our mechanisms ensure that in a strong sparse partition, a node's distance to its leader is at most the distance it had to $l(X)$ before the failure, which is $\sigma\rho^i$. In a weak sparse partition, the new diameter is at most $2\sigma\rho^i$, according to Lemma 8. ◀

▶ **Lemma 11.** *To inform the ρ^i-neighborhood of the nodes in X_2 about the new leader requires $\mathcal{O}(n^2)$ messages of size $\mathcal{O}(\log n)$. The maximum distance traversed by any message is ρ^i.*

Proof. In our algorithm, each node in X_2 sends the id of the new leader to every node in its ρ^i-neighborhood using its shortest path tree. In the worst case, $|X_2| = \mathcal{O}(n)$, and the ρ^i-neighborhood of every node in X_2 has size $\mathcal{O}(n)$. ◀

▶ **Lemma 12.** *To update the directory path we send $\mathcal{O}(1)$ messages of size $\mathcal{O}(\log n)$ which travel a distance at most $\mathcal{O}(D')$ where $D' = diam(G \setminus \{e\})$.*

Proof. One message is sent from $l(X_1)$ to v and forwarded to $l(X_2)$ to inform $l(X_2)$ whether it is part of the directory path. The distance from $l(X)$ to v is at most D' and the distance between v and $l(X_2)$ can be bounded by the diameter of X_2, that is $d(l(X_2), v) \leq 2\sigma\rho^i$.

When $l(X_2)$ is part of the directory path, then $l(X)$ contacts ϕ_{i+1} and ϕ_{i-1} about the upcoming update. When $l(X_2)$ receives $l(X_1)$'s message, it also sends a message to ϕ_{i+1} and ϕ_{i-1}. When they receive this message, they again a message to $l(X)$. None of these messages need to travel further than D', because all messages are sent along shortest path trees. ◀

▶ **Lemma 13.** *To update the special parent information we send two messages of constant size that traverse a distance of at most $\sigma\rho^i$ in a strong sparse partition and at most $2\sigma\rho^{i'}$ in a weak sparse partition, where $i' = i + \log_\rho(c'\sigma)$.*

Proof. Nodes $l(X_1)$ and $l(X_2)$ both send a message to their respective special parent. Both $l(X)$ and $l_{i'}(l(X))$ are in $C_{i'}(l(X))$ and both $l(X_2)$ and $l_{i'}(l(X_2))$ are in $C_{i'}(l(X_2))$, thus these messages can be sent along the spanning trees of $C_{i'}(l(X))$ and $C_{i'}(l(X_2))$. In a strong sparse partition, the initial spanning tress of the clusters guarantee that the distance from any node to the leader along the spanning tree is at most $\sigma\rho^{i'}$. This property is maintained even if clusters split. In a weak sparse partition, Lemma 8 tells us that the diameter of the spanning tree of any cluster is at most $2\sigma\rho^i$. ◀

▶ **Lemma 14.** *Suppose that since the last edge failure, the directory path has been rebuilt up to level i. Then the length of the directory path up to level i is at most $\mathcal{O}(\sigma\rho^i)$.*

Proof. Suppose the last time ϕ was modified at level i was by node u and the last time ϕ was modified at level $i+1$ was by node w. Then the modification at level i must have been due to a move operation issued by u which found the directory path level $i+1$. We thus know that $d(\phi_i, \phi_{i+1}) \leq d(\phi_i, u) + d(u, w) + d(w, \phi_{i+1})$. Because u found ϕ at level $i+1$, we have $d(u, w) \leq \rho^{i+1}$. After the repair of the data structure, the diameter of a cluster X at level i is at most $\text{diam}(X) \leq 2\sigma\rho^i$. Thus, we obtain

$$\text{length}(\phi_1, \dots, \phi_i) = \sum_{j=-1}^{i-1} d(\phi_j, \phi_{j+1}) \leq \sum_{j=-1}^{i-1} 2\sigma(\rho^i + \rho^{i+1}) + \rho^{i+1} = \mathcal{O}(\sigma\rho^i). \qquad ◀$$

▶ **Lemma 15.** *A failure of edge $e = \{u, v\}$ splits at most h clusters in a strong, and Ih clusters in a weak sparse partition.*

Proof. A cluster X splits if $T(X)$ contains e. In a strong sparse partition $T(X)$ contains only nodes from X. As a node belongs to only one cluster, this implies that in a strong sparse partition at most cluster per level splits. In weak sparse partition, $T(X)$ may contain nodes not in X, but a node's ρ^i-neighborhood intersects at most I clusters of $\mathcal{P}_i$. Since a cluster whose spanning tree contains e also contains u and v, at most I clusters on a level need to be reclustering due to e failing. ◀

B Adding Fault Tolerance to the Directory (Cont.)

B.1 Edge Failure during Publish Operation

Suppose w issues a publish operation and an edge failure occurs before the publish operation reaches level h. The publish operation is only affected if at some level i the leader of w changes. If w is informed of the change before adding $l(X)$ to the directory path, then w will add its new level i leader to the directory path. If the directory path has already been built to level i, then our failure mechanisms will update the directory path.

B.2 Edge Failure during Lookup Operation

While Searching for the Directory Path

The search for the directory path of the lookup operation is similar to that of the move operation, but the lookup operation also uses the information provided by special parents and it does not modify the directory path. We thus only discuss the impact of an edge failure on the special parent information: When w's lookup finds a special parent $l(X)$ of a node $l(X')$ on the directory path the lookup follows the link from $l(X)$ to $l(X')$. If cluster X' splits and the directory path updates before the lookup reaches $l(X')$, then w's lookup will go back to X and continue its search for the directory path there.

While Following the Directory Path Downward

The downward phase of a lookup is not affected by modifications to the directory path. Suppose that the directory path gets modified at level i due to the edge failure. If the directory path is updated before the lookup operation reaches level i, then the lookup follows the updated path. Otherwise, the lookup operation follows the pointers from ϕ_{i+1} to ϕ_i, from ϕ_i to ϕ_{i-1} as these are still intact.

C Pseudocode of Basic Directory Algorithm

Algorithm 1 Directory Operations Issued by Node v.

```
Graph $G$ has partition hierarchy $\mathcal{P}$ with topmost level $h = \lceil \log_\rho D \rceil$, for constant $\rho > 1$;
Directory path $\phi = \phi_{-1}, \phi_0, \ldots, \phi_h$ points toward the current owner of token $t$;

// Publish Operation
for level $i$ from 0 to $h$ do
    $\phi_i \leftarrow l_i(v)$; // $\phi_i$ is set to be the leader of $v$ at level $i$
    Add bidirectional links between $\phi_i$ and $\phi_{i-1}$;

// Lookup Operation
$i \leftarrow 0$; // start level of upward phase
while none of the leaders of clusters in $P_i(v)$ know about $\phi$ do
    i++; // upward phase to discover $\phi$
If a special parent pointer toward $\phi_{i'}$ $(i' < i)$ was discovered at level $i$, then adjust $i \leftarrow i'$;
// downward phase toward token
for level $k \leftarrow i$ down to 0 do
    Follow the downward pointer of $\phi_k$;
Return value of token $t$ from owner node $\phi_{-1}$;

// Move Operation
$\phi_{-1} \leftarrow v$; // start forming new $\phi$ toward $v$
$i \leftarrow 0$; // upward phase discovers previous $\phi$
while none of the leaders of clusters in $P_i(v)$ are $\phi_i$ do
    $\phi_i \leftarrow l_i(v)$; // form new path $\phi$
    Add bidirectional links between $\phi_i$ and $\phi_{i-1}$;
    Inform special parent $l_{i'}$ at level $i' = i + \log_\rho(c'\sigma)$ about $\phi_i$;
    $i++$;
old $\leftarrow$ level $i-1$ node pointed downwards by $\phi_i$;
Add bidirectional links between $\phi_i$ and $l_{i-1}(v)$; // adjust topmost node
Delete upward link of old and information at special parent of old;
// downward phase deletes old $\phi$
while level of old is not $-1$ do
    $w \leftarrow$ node pointed by downward link of old;
    Delete links between $w$ and old and information at special parent of $w$;
    old $\leftarrow w$;
Transfer token $t$ from old to $v$; // $v$ is new owner
```