Treeplication: An Erasure Code that is Almost as Painless as Replication

Michael Gandelman and **Yuval Cassuto**, Viterbi Dept. of Electrical Engineering, Technion

michaelgandelman@gmail.com, ycassuto@ee.technion.ac.il

***Abstract*—This paper presents a new erasure code called Treeplication which features the benefits of both coding and replication. A Treeplication code for k data fragments is defined on a binary tree with $2k-1$ vertices, along with a distribution for selecting code fragments from the tree layers. The tree structure allows to optimize the recoverability of random subsets of code fragments, while at the same time behaving similarly to replication in recovering individual data fragments. The significant performance advantages over both replication and existing erasure codes motivate the use of Treeplication in decentralized distributed storage systems.**

I. Introduction

Replication of data in distributed storage systems is a very common technique employed in order to boost many aspects of system performance. In commercial distributed storage systems, most famously [1], replication is implemented to improve the survivability, availability and accessibility of the data, and to give applications increased flexibility and robustness. Moreover, the field of *distributed computing* has been particularly successful in emulating desired features of local storage in replicated distributed storage [2]. Too bad is the steep storage cost associated with replication. A more efficient form of redundancy that can provide performance features at a much lower cost is *erasure coding*. In erasure coding, instead of storing exact copies of data units, we divide them to k fragments and store the data fragments alongside $n-k$ redundant code fragments. For survivability, erasure codes have clear advantage over replication, because they can tolerate r node failures at a cost of r additional fragments per k data fragments. For tolerating the same r failures, replication needs to add r copies of the full k-fragment data unit. Unfortunately, it is significantly harder to attain performance features other than survivability by employing erasure codes. Furthermore, distributed computing algorithms on erasure-coded storage are lagging behind, often due to inherent hardness [3], [4].

In this paper we propose and develop an erasure coding scheme we call *Treeplication*, which preserves many of the good properties of replication while delivering performance at a lower storage cost. In a Treeplication code, data fragments are encoded into code fragments that are organized over a binary *tree* (hence the name's prefix). The tree structure allows localized erasure correction that improves the communication efficiency of the code. In a way, this scheme goes the opposite direction to most recent works on erasure coding: instead of taking an erasure code and make it more "access friendly", as in regenerating codes [5] or codes with locality [6], we take the replication scheme and gracefully make it more "storage-cost friendly". The problem that Treeplication solves is storing moderately large data units in a *decentralized distributed storage system*, where access to data units is done by forming ad-hoc subsets of nodes, each accessing one fragment of the data unit. Replication does not excel in this setup: replicating the data fragments suffers from the *coupon collector problem* [7] when trying to find a random node subset that has all fragments. Erasure codes are great at providing subsets that collectively can reconstruct the data unit, but access to individual fragments usually requires centralized decoding and/or excessive amounts of communication between nodes. Treeplication offers a middle-ground solution that gets the benefits of coding in data recoverability while enjoying access performance similar to replication.

Suppose we want to store data units such that k nodes can each access a $1/k$ part fragment of the data unit. This is a common scenario in *map-reduce* distributed computing [8], where large data units are processed in parallel by multiple machines. It is not enough to store a single copy of a fragment because the node storing it may be down or busy. Addressing this problem with replication seeks to increase the likelihood that an *available set of nodes* will have all k data fragments stored in it. Here we propose an alternative to replication that stores coded fragments from a highly structured code. The structure of the code helps gain the good features of both erasure coding and replication. In the first part of the paper (Sections III, IV) we show the advantage of Treeplication over replication in terms of data recoverability: in order to achieve the same probability to get a decodable random set of available fragments, replication requires to store over 60% more fragments than Treeplication. Our main theoretical contributions in this part are analysis of the decoding probability and an efficient algorithm to find the fragment-selection distribution that yields optimal decoding probability. In the second part (Section V), we show the advantage over erasure coding with MDS codes: the communication required for decentralized recovery of the data fragments is more than an order of magnitude lower. Treeplication is advantageous over regenerating codes in that low-communication recovery is achieved with minimal recovery sets of k nodes, while it is known [5] that regenerating codes become trivial with size-k recovery sets. There are several other coding structures in the literature that may be useful for the

This work was supported in part by the Israel Science Foundation and by the US-Israel Binational Science Foundation

target problem of random-subset decentralized storage, but the special recursive structure of Treeplication offers the advantage of analyzing decodability and designing selection distributions efficiently. The structure of Treeplication is inspired by the similar structure of the fountain code proposed in [9], but, among several key differences, our codes are designed for optimal performance in fixed values of k, while the results in the prior work are asymptotic.

II. Treeplication Coding

A distributed storage system stores *data units* across storage nodes. Each data unit is broken to k *data fragments*, each of size D. Throughout the paper we assume that $k = 2^s$, for some integer s. To encode the data unit in the storage system, we use a *binary tree* of depth s. The tree has 2^s leaf vertices representing the data fragments, and a total of $2^{s+1}-1$ vertices. Note that including the root there are $d \triangleq s+1$ *layers* in the tree. The layers are numbered from bottom to root, thus for $i \in \{1,\dots,d\}$, layer i has 2^{d-i} vertices. In the sequel we use T_d to denote this tree, and T_ℓ refers to one of T_d's subtrees with layers $\{1,\dots,\ell\}$. The interpretation of the tree is that each vertex represents a *code fragment*: starting from level $i=2$ each code fragment is the bit-wise exclusive-or (XOR) of its two children, and the leaves at level $i=1$ are the "pure" data fragments also called *systematic code fragments*. An example of this tree representation is given in Fig. 1 for the case $k=8$. A subset of the vertices of T_ℓ is said to be *decodable*

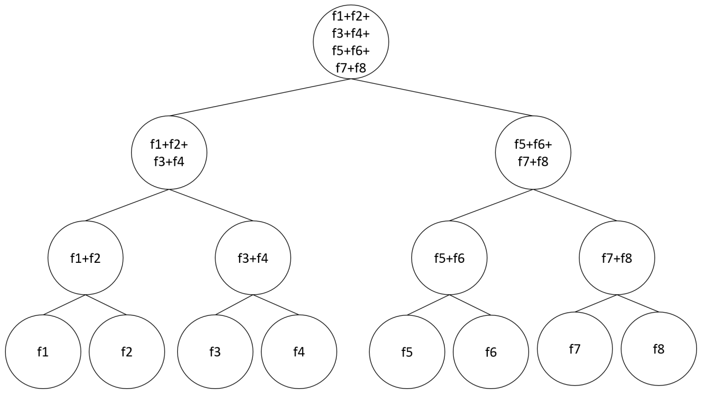


Fig. 1: Tree representation of treeplication coding for $k=8$ data fragments ($s=3$,$d=4$).

if the corresponding code fragments are sufficient to recover the $k' \triangleq 2^{\ell-1}$ data fragments. Clearly, a decodable subset must have at least k' vertices, and also all subsets of size greater than $2k'-3$ are decodable. Between k' and $2k'-3$, decodability depends on the particular subset available for decoding.(Viewed as an erasure code with block length $2k'-1$, T_ℓ can correct any single erasure, and many combinations of between 2 and $k'-1$ erasures, but not more than $k'-1$ erasures.) The following lemmas further characterize decodable subsets.

Lemma 1: If a subset of T_ℓ is decodable, then at least one immediate subtree: $T_{\ell-1}$ (left) or $T'_{\ell-1}$ (right) is decodable with only vertices from its subtree.

Proof: Since the subtrees $T_{\ell-1}$,$T'_{\ell-1}$ are disjoint in their XOR arguments, having both non-decodable internally would mean that two additional code fragments are needed. This is a contradiction because there is only the root as a potential extra code fragment. ■

III. Decodability with Uniform Selection

Treeplication is intended for use in a fully distributed storage system where nodes decide in a decentralized way which fragments to store. In the simplest model for decentralized fragment selection, n code fragments are each chosen uniformly and independently (with replacement, so multiplicity is possible) from the $2k-1$ tree vertices. We now derive the probability of obtaining a decodable subset once such a selection is performed. We first count the number of decodable subsets given $k+j$ *unique* vertices, for $j=0,\dots,k-1$; subsequently, we count n-combinations mapping to decodable $k+j$ unique vertices.

For a tree with d layers, define $D_{d,j}$ to be the number of decodable subsets with $2^{d-1}+j$ unique vertices. Note that $D_{d,j}=0$ if $j<0$ or $j>2^{d-1}-1$. We partition the decodable subsets to $D_{d,j}=t_{d,j}+r_{d,j}$, where $t_{d,j}$ is the number of subsets that include the root vertex *and* are non-decodable without it, while $r_{d,j}$ is the number of all other decodable subsets.

Lemma 2: For a tree with d layers and a subset with $2^{d-1}+j$ unique vertices, the following recursive relation applies to $r_{d,j}$

$$r_{d,j} = \sum_{l=0}^{j} D_{d-1,l}D_{d-1,j-l} + \sum_{l=0}^{j} D_{d-1,l}D_{d-1,j-l-1}. \quad (1)$$

Proof: By the definition of $r_{d,j}$, the counted decodable subsets either do not have the root vertex or they are decodable without it. In either case the immediate subtrees of the root must both be decodable themselves. The first and second terms in (1), respectively, count these decodable subsets without and with the root vertex in them. ■

The more interesting is the term $t_{d,j}$, which we count next.

Lemma 3: For a tree with d layers and a subset with $2^{d-1}+j$ unique vertices, the following recursive relation applies to $t_{d,j}$

$$t_{d,j} = 2\sum_{l=0}^{j} D_{d-1,l}t_{d-1,j-l}, \ d>1, j\geq 0; \ t_{1,0}=1. \quad (2)$$

Proof: When $d=1$ the tree is just the root vertex, and the root forms a decodable subset that is non-decodable without it (trivially); hence $t_{1,0}=1$. By Lemma 1, one immediate subtree of T_d must be decodable internally, and by definition of $t_{d,j}$ the other subtree must *not* be decodable internally. The former gives the term $D_{d-1,l}$ in (2) and the latter gives $t_{d-1,j-l}$. To understand why the latter is correct, observe that every decodable subset of T_{d-1} that contains its root can be mapped to a decodable subset of T_d by replacing the root of T_{d-1} by the root of T_d, assuming that the other subtree T'_{d-1} is decodable internally. Also, with this root swap it is clear that T_{d-1} is non-decodable without its root if and only if T_d is non-decodable without its root. Finally, the factor 2 in (2) counts the two options to choose the internally-decodable subtree among the left and right subtrees. ■

Once we have an efficient way to count decodable subsets, it is simple to derive the probability to get a decodable subset under independent uniform selection of each of the n vertices.

Theorem 4: For a tree with d layers and n vertices each chosen uniformly and independently from the $2^d - 1$ vertices of T_d, the probability to get a decodable subset is

$$P_d = \sum_{j=0}^{n-2^{d-1}} \frac{D_{d,j} S(n, 2^{d-1}+j)(2^{d-1}+j)!}{(2^d-1)^n}, \tag{3}$$

where $S(a,b)$ is the number of ways to partition a set of a objects into b nonempty subsets (also known as the Stirling number of the second kind).

Proof: Each decodable subset counted by a $D_{d,j}$ can be chosen in $S(n, 2^{d-1}+j)(2^{d-1}+j)!$ different ways by the uniform selection. The probability is obtained by normalizing by the total number of choices, decodable or not. ■

We compare the Treeplication scheme under uniform independent selection to standard (uncoded) replication with the same selection policy. For the same input block size k, in replication each choice is one of k data fragments, while in Treeplication it is one of $2k-1$ code fragments. The comparison results can be seen in the two middle columns of Table I below. The results show the advantage of Treeplication: to get to the same decoding-success[1] probability of 0.9, replication needs between 15%-30% higher n than Treeplication, which means a higher storage cost for the same availability performance.

IV. Non-uniform Selection

The optimal choice of tree vertices may not be the uniform selection assumed in the previous section. Thus, to improve decodability we now extend the analysis to non-uniform selection. In the non-uniform setup we have $n = \sum_{i=1}^{d} n_i$, where n_i code fragments are chosen (with replacement) from layer i of the tree. Within each layer the selection is as before: each of the n_i code fragments is chosen uniformly and independently from the 2^{d-i} vertices of layer i. In our analysis we map each n_i to $p_i = 1 - (1 - \frac{1}{2^{d-i}})^{n_i}$, where p_i is the probability that a certain vertex in layer i is selected to the subset at least once (same for all vertices in the layer). Note that p_i is monotonically increasing with n_i, and $p_i = 0$ when $n_i = 0$. It will be simpler for us to assume that a vertex in layer i is included in the subset (at least once) with probability p_i, independently of the other vertices in the layer, although this assumption is not consistent with the specified discrete parameters $\{n_i\}_{i=1}^{d}$. This assumption is a reasonable approximation when n_i is of the same order as 2^{d-i}, as required to get decodability with high probability [2]. The following theorem gives an expression for the decoding probability with non-uniform selection.

Theorem 5: For a tree with d layers T_d whose vertices are chosen with probability p_i in layer i, the probability to get a decodable subset is

$$Q_d = Q_{d-1}^2 + 2^{d-1} p_d \prod_{i=1}^{d-1} [(1-p_i)Q_i],\ d > 1;\ Q_1 = p_1. \tag{4}$$

Proof: When $d = 1$ the tree is just the root vertex, and the tree is decodable when the root vertex is chosen to the subset, happening with probability p_1. For $i = 1, \dots, d-1$ denote by A_i the probability that T_i is decodable if and only if its root vertex is provided to the subset externally. Then $Q_d = Q_{d-1}^2 + 2Q_{d-1}p_d A_{d-1}$, because, similar to Lemmas 2,3, the subset is decodable if both its subtrees are decodable, or if one subtree is decodable, the root is present, and the other subtree is decodable if and only if its root is provided externally. The "only if" is required to not count in the second term probabilities already included in the term Q_{d-1}^2; the "if" part guarantees that the other subtree is decodable when the parent root is present and the other subtree is decodable. A_{d-1} can be calculated with the recursive expression $A_i = 2(1-p_i)Q_{i-1}A_{i-1}$, and the initial value $A_1 = 1 - p_1$. Expanding this expression to A_{d-1} and substituting in the previous equation gives (4). ■

By calculating efficiently Q_d for every *selection distribution* $\{n_i\}_{i=1}^{d}$, Theorem 5 is a useful tool to design non-uniform Treeplication allocations that, for any given n, maximize Q_d among all $\{n_i\}_{i=1}^{d} : \sum_{i=1}^{d} n_i = n$. To find the optimal Q_d efficiently, we first prove some properties of optimal selection distributions that significantly reduce the search complexity.

Lemma 6: Every optimal selection distribution satisfies $p_i \le p_{i-1}$, $\forall i \in [2,d]$.

Proof: Assume that $p_1, \dots, p_{i-2}$ are set, and by contradiction that $p_i > p_{i-1}$. From (4) we have

$$Q_i = Q_{i-1}^2 + 2^{i-1} p_i \prod_{j=1}^{i-1} [(1-p_j)Q_j], \tag{5}$$

$$Q_{i-1} = Q_{i-2}^2 + 2^{i-2} p_{i-1} \prod_{j=1}^{i-2} [(1-p_j)Q_j]. \tag{6}$$

Denote $a := Q_{i-2}^2$ and $b := 2^{i-2} \prod_{j=1}^{i-2} [(1-p_j)Q_j]$. By substituting (6) and a,b into (5), we get

$$Q_i = (a + bp_{i-1})(a + bp_{i-1} + 2bp_i - 2bp_{i-1}p_i). \tag{7}$$

Since a,b are independent of p_i and p_{i-1}, we can see that exchanging between p_i and p_{i-1} in (7) results in an increase in Q_i because

$$\begin{aligned}(a + bp_{i-1})&(a + bp_{i-1} + 2bp_i - 2bp_{i-1}p_i) \\ &< (a + bp_i)(a + bp_i + 2bp_{i-1} - 2bp_{i-1}p_i)\end{aligned} \tag{8}$$

for any $0 \le p_{i-1} < p_i \le 1$. This is a contradiction. ■

The monotonicity of p_i in i implies the following lemma.

Lemma 7: Every optimal selection distribution satisfies $n_{i-1} \ge 2n_i$, $\forall i \in [2,d]$.

Proof: From Lemma 6 we have $1-p_i \ge 1-p_{i-1}$, and from monotonicity of the log function $\log(1-p_i) \ge \log(1-p_{i-1})$. Thus substituting the definition of p_i,p_{i-1} gives that

$$\frac{n_{i-1}}{n_i} \ge \frac{\log\left(1 - \frac{1}{2^{d-i}}\right)}{\log\left(1 - \frac{1}{2^{d-i+1}}\right)} \ge 2. \tag{9}$$

[1]In replication, decoding success is when every data fragment is selected at least once.

[2]For verification we compared the i.i.d model with uniform distribution to the true uniform results of Section III, and got almost the same results.

■

With Lemma 7 we can prove the following useful property of optimal selection distributions.

Proposition 8: Every optimal selection distribution satisfies $n_i \geq \sum_{j=i+1}^{d} n_j$, $\forall i \in [1, d-1]$.

Proof: By induction starting from $i = d-1$. True for base case $i = d-1$ because $n_{d-1} \geq 2n_d \geq n_d = \sum_{j=d}^{d} n_j$, where the first inequality is from Lemma 7. Assume true for i, then showing for $i-1$

$$n_{i-1} \geq 2n_i \geq n_i + \sum_{j=i+1}^{d} n_j = \sum_{j=i}^{d} n_j,$$

where the first inequality is from Lemma 7 and the second from the induction hypothesis. ■

Proposition 8 is the basis to Algorithm 1 that finds the optimal selection distribution based on searching the small subset of distributions that satisfy the above optimality conditions. In the algorithm we denote by $\{n_i\}_{i=1}^{\ell}$ a selection distribution $n_1, \ldots, n_\ell, 0, \ldots, 0$, where the last $d - \ell$ elements of the distribution are 0. For any selection distribution S we denote by $Q_d(S)$ the result of (4) with p_i corresponding to the n_i of S. In the algorithm, $Q^\star$ holds the maximum decoding probability among all selection distributions explored so far.

Algorithm 1 Find optimal non-uniform selection distribution

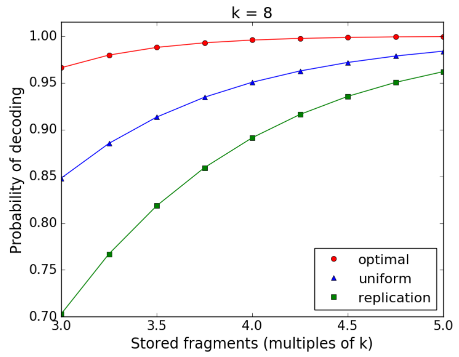
```
1:  function SEARCH($n, d$)
2:      $Q^\star := 0$
3:      Distribute($1, n, \{\}$)
4:      return $Q^\star$
5:  end function
6:  function DISTRIBUTE($j, B, \{n_i\}_{i=1}^{j-1}$)
7:      if $B == 0 \;||\; j > d$ then
8:          if $Q_d(\{n_i\}_{i=1}^{j-1}) > Q^\star$ then
9:              $Q^\star := Q_d(\{n_i\}_{i=1}^{j-1})$
10:         end if
11:         return
12:     end if
13:     for $b \in [0, \lfloor B/2 \rfloor]$ do
14:         $n_j := B - b$
15:         Distribute($j+1, b, \{n_i\}_{i=1}^{j}$)
16:     end for
17: end function
```

Thanks to the factor $1/2$ in the for loop of Algorithm 1, its running time is significantly reduced compared to trivial search. While trivial search needs to explore all compositions of n into up to d sets, only *non-squashing* partitions [10] of n are explored by Algorithm 1. For example when $d = 6$,$n = 128$, trivial search requires 275584033 steps while Algorithm 1 only 25509.

It turns out that optimal Treeplication codes greatly outperform both replication and uniform Treeplication. The right column of Table I shows that compared to optimal Treeplication, the storage cost of replication is higher by 60% or more for $k \in \{8, 16, 32\}$. This is close to quadruple the advantage of uniform Treeplication. Further results comparing the three schemes are given in Fig. 2 where the decoding probability is plotted as a function of n for $k = 8, 16$.

TABLE I: Replication vs. Treeplication (uniform and non-uniform): minimum number of stored fragments n required for decoding probability of 0.9.

k	Replication	Treeplication (uniform)	Treeplication (non-uniform)
2	5	4	3
4	13	10	8
8	33	26	20
16	79	66	49
32	181	157	113



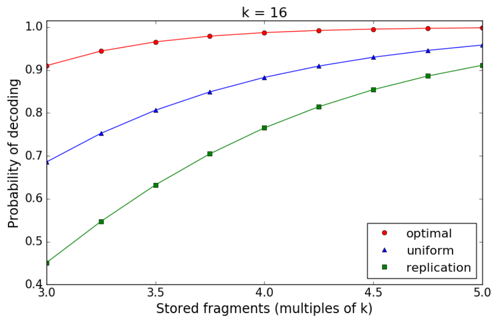


Fig. 2: Probability of decoding vs. num coded fragments n.

V. Fragment Recovery and Communication Cost

In the successful case of having a decodable subset of T_d, the distributed storage system needs to have the k data fragments recovered by the nodes storing the decodable subset. For the recovery we also assume a decentralized process, where each data fragment is recovered by one node storing a code fragment, using code fragments from other nodes if necessary. Data fragments that appear as leaves (systematic code fragments) in the subset are trivially recovered locally; the remaining data fragments are recovered by non-leaf vertices that receive code fragments (both systematic and not) of other vertices to recover the data fragment. The amount of communication required for this recovery should be minimal.

This section presents Algorithm 2, which finds the minimal-communication recovery and counts the number of fragment transmissions. At the start of the algorithm we have a decodable subset (with k or more fragments) mapped to a tree. Each fragment (tree vertex) is stored by a node in the system, and all nodes know the vertexes in the subset. Algorithm 2 is run by each of these nodes, to determine which data fragment (leaf vertex) to recover and which fragments (tree vertices) to request from the other nodes. Before presenting the algorithm, we prove properties regarding node selection for minimal-communication recovery. In the sequel, a *present*/*missing* vertex is a vertex in/not-in the decodable subset.

Lemma 9: If a subset is decodable, then 1) there is no path of missing vertices between a leaf to the root (where both ends are also missing), and 2) no vertex (present or missing) has two missing-vertex paths leading to two leaves.

Proof: The existence of a missing-vertex path from leaf to root contradicts decodability because no present code fragment depends on the leaf. Two missing-vertex paths that meet at the same vertex x imply that both subtrees directly under x are non-decodable (by condition 1 above), thus violating Lemma 1. ■

Proposition 10: Suppose present vertex x recovers leaf vertex y if and only if there is a missing-vertex path between x and y. Then in a decodable subset, each leaf is recovered by a single unique vertex, which is the lowest possible present tree vertex capable of recovering y.

Proof: From condition 1 of Lemma 9, each leaf must have a path ending at a present vertex; from condition 2 there cannot be more than one missing-vertex path that end at the same vertex. That proves that every leaf will be recovered by a unique vertex. x is the lowest present vertex in the tree that can recover y, because it is at the end of a missing-vertex path from y, making it the lowest vertex whose code fragment has y as argument. ■

Building on Proposition 10, Algorithm 2 now finds the vertices recovering the data fragments. Each of them is the lowest possible in the tree able to recover the corresponding data fragment (hence requires the least amount of communication). A vertex evaluated to "false" in line 9 is not participating in the recovery procedure (this happens when the decodable subset has more than k vertices). The variable sum holds the aggregate number of code fragments communicated to the nodes recovering the data fragments. The explicit identities of the communicated code fragments can be extracted from the identities of the vertices reached in line 6. If Algorithm 2 terminates without recovering all data fragments, from Proposition 10 we know that the subset is not decodable.

To evaluate the performance of Treeplication with respect to recovery communication cost, Table II shows the empirical average number of code fragments communicated per instance of a decodable subset. Treeplication is implemented using the optimal (non-uniform) selection parameters $\{n_i\}_{i=1}^{d}$ found by Algorithm 1, and uniform sampling of n_i vertices in layer i. The minimal-communication recovery per simulation instance is found using Algorithm 2. We compare the communication cost to similar decentralized recovery using systematic MDS codes with block length $2k-1$ (identical to the vertex count of Treeplication) also simulated as a uniform and independent selection (of n fragments from the $2k-1$ code symbols). For both schemes we used the same $n = 3k$, which is a common replication factor in pure replication settings such as the default in [1]. We can see that Treeplication is very economical in communication, requiring very small numbers of fragments per instance. When using MDS codes, recovery of non-systematic fragments requires a heavy load of $k-1$ fragments per recovering node, which results in an order of magnitude or more higher communication cost per instance.

Algorithm 2 Fragment recovery

```
1:  $\mathrm{sum} := 0$
2:  for each present vertex $x$ do
3:      $\mathrm{sum}_x := 0$
4:      for each path downwards from $x$ do
5:          traverse path until a present vertex or a missing leaf reached
6:          if a vertex that is present reached then $\mathrm{sum}_x$++
7:          end if
8:      end for
9:      if missing leaf was reached in a downward path then $\mathrm{sum} = \mathrm{sum} + \mathrm{sum}_x$    // $x$ is recovering leaf
10:     end if
11: end for
12: return $\mathrm{sum}$
```

TABLE II: Treeplication vs. MDS: communication cost.

k	# fragments Treeplication	# fragments MDS
4	0.35	1.82
8	1.18	10.64
16	2.88	49.62
32	6.552	213.10

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *SOSP*, pp. 29–43, 2003.

[2] H. Attiya, A. Bar-Noy, and Danny Dolev, "Sharing memory robustly in message-passing systems," *Journal of the ACM (JACM)*, 42(1), pp. 124–142, 1995.

[3] A. Spiegelman, Y. Cassuto, G. Chockler, and I. Keidar, "Space bounds for reliable storage: Fundamental limits of coding," *Proceedings of the 2016 ACM symposium on principles of distributed computing*, PODC 16, ACM, 2016.

[4] V. Cadambe, Z. Wang, and N. Lynch, "Information-theoretic lower bounds on the storage cost of shared memory emulation," *Proceedings of the 2016 ACM symposium on principles of distributed computing*, PODC 16, ACM, 2016.

[5] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, Sep. 2010.

[6] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols, *IEEE Trans. Inf. Theory*, vol. 58, no. 11, pp. 6925–6934, Nov. 2012.

[7] M. Mitzenmacher and E. Upfal, *Probability and Computing*. London, Cambridge University Press, 2005.

[8] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107-113, Jan. 2008.

[9] J. Edmonds and M. Luby, "Erasure Codes with a Hierarchical Bundle Structure," *IEEE Trans. Inf. Theory*, early access.

[10] N. J. A. Sloane, and J. A. Sellers, "On non-squashing partitions," *Discrete Mathematics*, vol. 294, no. 3, pp. 259–274, May 2005.

[11] M. Ye and A. Barg, "Explicit constructions of high-rate MDS array codes with optimal repair bandwidth," *IEEE Trans. Inf. Theory*, vol. 63, no. 4, pp. 2001-2014, April 2017