When and By How Much Can Helper Node Selection Improve Regenerating Codes?

Imad Ahmad, Chih-Chun Wang; {ahmadi,chihw}@purdue.edu
School of Electrical and Computer Engineering, Purdue University, USA

***Abstract*—Regenerating codes (RCs) can significantly reduce the repair bandwidth of distributed storage networks. Initially, the analysis of RCs was based on the assumption that during the repair process, the newcomer does not distinguish (among all surviving nodes) which nodes to access, i.e., the newcomer is oblivious to the set of helpers being used. Such a scheme is termed the *blind repair (BR)* scheme. Nonetheless, it is intuitive in practice that the newcomer should access only those "good" helpers. This paper focuses on characterizing the effect of choosing the helper nodes in terms of the storage-bandwidth tradeoff. The results fully answer the following fundamental questions: Under what conditions does proactively choosing the helper nodes improve the storage-bandwidth tradeoff? Can this improvement be analytically quantified?**

I. Introduction

The need for storing very large amounts of data reliably is one of the major reasons that has pushed for distributed storage systems. Examples of distributed storage systems include data centers [5] and peer-to-peer systems [15]. One way to protect from data loss is by replication coding, i.e, if a disk in the network fails, it can be replaced and its data can be recovered from a replica disk. Another way is to use maximum distance separable (MDS) codes. Recently, regenerating codes (RCs) [3] have been used to further reduce the repair bandwidth of MDS codes.

One possible mode of operation is to let the *newcomer*, the node that replaces the failed node, *always* access/connect to all the remaining nodes. On the other hand, under some practical constraints we may be interested in letting the newcomer communicate with only a subset of the remaining nodes, termed the *helpers*. For example, reducing the number of helpers decreases I/O overhead during repair and thus mitigates one of the performance bottlenecks in cloud storage systems. In the original storage versus repair bandwidth analysis of RCs [3], it is assumed that the newcomer does not distinguish/choose its helpers. We term such a solution the *blind repair (BR) scheme.* Nonetheless, it is intuitive that the newcomer should access only those "good" helpers of the remaining nodes.

To illustrate this, we consider a storage network with 4 nodes numbered from 1 to 4. Suppose that we would like to protect against one node failure by replication. To that end, we first divide the file into two fragments, fragments A and B, and we store fragment A in node 1 and fragment B in node 2. Each fragment is replicated once by storing a copy of fragment A in node 3 and a copy of fragment B in node 4. If any one of the four nodes fails, then we can retrieve the entire file by accessing the intact segments A and B in the remaining three nodes. The repair process of this replication scheme is also straightforward. Say node 4 fails, the newcomer simply accesses node 2 and restores segment B. We observe that the newcomer only accesses the good helper (the one storing the lost segment) in this replication scheme. In this scheme, each node stores half of the file, and during repair, the newcomer accesses 1 helper node and communicates half of the file. For comparison, if we apply the analysis of [3], we see that if we use RCs to protect against 1 node failure, each node has to store the whole file and during repair, the newcomer accesses 1 helper and communicates the entire file. *The simplest replication code is twice more efficient than RCs in this example.*[1]

The reason why the replication code is the superior choice is that it only chooses the good helper during repair. To illustrate this, if we do not distinguish which node is the helper, we can let node 2 fail first, and let the new node 2 choose node 1 as the helper. Then we let node 3 fail and let node 1 again be the helper. Finally, we let node 4 fail and let node 1 be the helper. Since the content of all 4 nodes are now originating from node 1, each node needs to store a complete copy of the file to tolerate the case when node 1 fails. As can be seen, blind repair is the main cause of the performance loss.

The idea of choosing good helpers in RC has already been used in constructing exact-repair codes as in [4], [10], and in locally repairable codes as in [6], [9], [11] when helper selection is fixed over time. [4] also observes that choosing good helpers can outperform BR at the minimum bandwidth point. However, a complete characterization of the effect of choosing the helper nodes in RCs, including *stationary* and *dynamic* helper selection, on the storage-bandwidth tradeoff is still lacking in the literature. This motivates the following open questions: Under what condition is it beneficial to proactively choose the helper nodes? Is it possible to analytically quantify the benefits of choosing the good helpers? Specifically, the answers to the aforementioned fundamental questions were still not known.

In this work, we answer the first question by providing a necessary and sufficient condition under which optimally choosing the helpers strictly improves the storage-bandwidth

[1]One may think that this performance improvement over the blind repair (BR) scheme [3] is due to that the parameter values $(n = 4, k = 3, d = 1)$ are beyond what is originally considered for the regenerating codes (which requires $k \leq d$). In Section II-B, we will provide another example with $(n = 6, k = 4, d = 4)$, which shows that a good helper selection can strictly outperform the BR solution in [3] for $k \leq d$.

tradeoff. This new necessary and sufficient characterization of "under what circumstances helper selection improves the performance" is by far the most important contribution of this work since it provides the first rigorous benchmark/guideline when designing the next-generation smart helper selection solutions.

It is worth reemphasizing that, which helpers are "optimal" at the current time slot t depends on the history of the failure patterns and the helper choices for all the previous time slots 1 to $(t-1)$, which makes it very difficult to quantify the corresponding performance. Therefore, even though our main result fully answers the question *whether* an optimal design can outperform the blind helper selection, the question *how* to design the optimal helper selection scheme remains largely open. As an extension to our main necessary and sufficient characterization result and as part of the continuing quest of designing high-performance helper selection methods, we propose a low-complexity solution, termed the *family repair (FR) scheme*, that can harvest the benefits of (careful) helper selection without incurring any additional complexity when compared to a BR solution. We then characterize analytically the performance of the FR scheme and its extension, the family-plus repair scheme, and prove that they are optimal (as good as any helper selection one can envision) in some cases and *weakly optimal* in general, see the discussions in Sections V and VI.

Numerical computation shows that for many cases, the family-based schemes can reduce 40% to 90% of the storage and the repair bandwidth of RCs.

II. Problem Statement

We denote the total number of nodes in a storage network by n and the minimum number of nodes that are required to reconstruct the file by k. We denote by d the number of helper nodes that a newcomer can access. The n, k, and d values must satisfy

$$2 \leq n, \quad 1 \leq k \leq n, \quad \text{and} \quad 1 \leq d \leq n-1. \tag{1}$$

In all the results in this work, we assume *implicitly* that the n, k, and d values satisfy[2] (1).The overall file size is denoted by $\mathcal{M}$. The storage size for each node is α, and during repair, the newcomer requests β amount of traffic from each of the helpers. The total repair bandwidth is thus $\gamma \triangleq d\beta$. We use the notation $(\cdot)^+$ to mean $(x)^+ = \max(x, 0)$. We also define the indicator function as follows

$$1_{\{B\}} = \begin{cases} 1, & \text{if condition } B \text{ is true} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

In this work, we consider the helper selection/repair scheme in its most general form. Among all helper selection schemes, a special class, termed stationary repair schemes, is also studied. To distinguish the special class from the most general form, we use the term *dynamic repair* schemes whenever we are focusing on the most general type of helper selection schemes. In addition to studying the performance of any dynamic or stationary repair scheme, this work also proposes a new low-complexity solution, termed the family repair schemes. Detailed discussion of dynamic repair, stationary repair, and family repair schemes is provided in the following.

A. Dynamic versus Stationary Repair Schemes

In general, the helper selection at current time t can depend on the history of the failure patterns and the helper choices for all the previous time slots 1 to $t-1$. We call such a general helper selection scheme *the dynamic helper selection*. In contrast, a much simpler way of choosing the helpers, termed *stationary repair schemes*, is described as follows.

Stationary Repair: Each node index i is associated with a set of indices D_i where the size of D_i is d. Whenever node i fails, the newcomer (for node i) simply accesses those helpers j in D_i and requests β amount of data from each helper. It is called stationary since the helper choices $\{D_1, D_2, \ldots, D_n\}$ are fixed and do not evolve over time. As can be easily seen, the stationary repair scheme is a special case of (dynamic) helper selection, which incurs zero additional complexity when compared to the BR solution.

For any helper scheme A and given system parameters n, k, d, α, and β, we say that the corresponding RC with helper selection scheme A "satisfies the reliability requirement" if it is able to protect against any failure pattern/history while being able to reconstruct the original file from arbitrary k surviving nodes. We consider exclusively single failure at any given time. The setting of multiple simultaneous failed nodes [4], [18] is beyond the scope of this work.

B. Comparison to Locally Repairable Codes

Recall that RCs are distributed storage codes that minimize the repair bandwidth (given a storage constraint). In comparison, *locally repairable codes (LRC)*, recently introduced in [6], are codes that minimize the number of helpers participating in the repair of a failed node. LRCs were proposed to address the disk I/O overhead problem that the repair process can entail

[2]The following fact is proved in [3]. Suppose $k > d$. If the storage α and the repair bandwidth β of each node allow the storage network to tolerate $(n-k)$ failed nodes using *blind-repair* (BR) regenerating codes, then the same storage network with BR codes can actually tolerate $(n-d)$ failed nodes. Therefore, any regenerating code that can support the (n,k,d) value for some $k > d$ can also support (n,d,d) value. By definition, any regenerating code that can support (n,d,d) values can also support (n,k,d) for any $k > d$. It shows that the storage-bandwidth tradeoff of (n,k,d) value is identical to that of (n,d,d) value when $k > d$. This fact prompts the authors in [3] to study only the case in which $k \leq d$ and use the results of (n,d,d) as a substitute whenever we are considering the case of $k > d$. As will be seen later, the above equivalence between the (n,k,d) and the (n,d,d) cases when $k > d$ does not hold when considering non-blind helper selection. Therefore, throughout this paper, we do not assume $k \leq d$.

Also, in practice the parameter k specifies the resilience of the system and the parameter d specifies the repair cost. The choices of k and d values are completely orthogonal from a high-level design perspective. Any coupling between k and d is usually imposed by the kind of storage codes used, e.g., replication versus Reed-Soloman versus regenerating codes versus locally repairable codes. Since we are studying the most general form of helper-selection, we discard the assumption of $k \leq d$, which was originally used for the BR solution.

TABLE I
THE COMPARISON TABLE BETWEEN BLIND-REPAIR REGENERATING CODES, LOCALLY REPAIRABLE CODES, AND THE SMART-REPAIR REGENERATING CODES.

	Original RC [3], [12], [13], [17], [22]	Locally Repairable Codes [6]–[9], [11], [14]	Dynamic Helper Selection
Repair Mode	Functional/Exact Repair	Exact Repair	Functional[3] Repair
Helper Selection	Blind	Stationary (Fixed over time)	Dynamic (helper choices may depend on failure history)
(n, k, d) range	(1) Designed for $k \leq d$. (2) Can still be applied to the case of $k > d$ with reduced efficiency.	(1) Designed for $k > d$. (2) Can still be applied to the case of $k \leq d$ with reduced efficiency.	Allow for arbitrary (n, k, d) values
Contribution	Storage/repair-bandwidth tradoff for the worst possible helper selection	Storage/repair-bandwidth characterization for the specific stationary helper selection of the proposed exact-repair local code, which may/may not be optimal	First exploration of the storage/repair-bandwidth tradeoff for the optimal dynamic helper selection

on a storage network since the number of helpers participating in the repair of a failed node is proportional to the amount of disk I/O needed during repair. Subsequent development has been done on LRCs in [7]–[9], [11], [14].

In Table I, we compare the setting of the original RCs, LRCs, and RCs that allow for dynamic helper selection considered in this work. As first introduced in [3], original RCs were proposed under the functional repair scenario, i.e., nodes of the storage network are allowed to store any combination of the original packets as long as the reliability requirement is statisfied. In subsequent works [2], [12], [13], [16], [17], [19], [21], [22], RCs were considered under the exact-repair scenario in which nodes have to store the same original packets at any given time. In constrast, LRCs are almost always considered under the exact-repair scenario. However, in this work, for RCs with dynamic helper selection, we consider functional repair as the mode of repair as we aim at understanding the absolute role of helper selection in RCs. Albeit our setting is under functional repair, we are able to present an explicit construction of exact-repair codes that achieve the optimal or weakly optimal minimum bandwidth point of the functional repair [1]. For comparison existing work [4], [12] designs an exact repair scheme that achieves the minimum bandwidth regenerating (MBR) point of the "blind-functional-repair". The main difference is that our exact repair construction achieves the MBR point of the "smart-helper-functional-repair".

Table I also summarizes the differences between RCs, LRCs, and smart helper RCs in terms of the helper selection mechanisms. The orginal RCs are codes that do not perform helper selection at all, i.e., BR, while LRCs are codes that can perform stationary helper selection only. In this work, we consider the more general setting in which codes are allowed to have dynamic helper selection. Surprisingly, we are able to find a stationary helper selection scheme that is (weakly) optimal among dynamic schemes and strictly optimal for a practical range of (n, k, d) values. Another dimension in this comparison table is the (n, k, d) values that each of the three code scenarios addresses. The original RCs were designed for storage networks with large d values as they perform rather poorly when applied to small d values. LRCs, on the other hand, are designed for small d values, and for that reason, they perform poorly when d is large. In contrast, the codes we present in this work are designed for arbitrary (n, k, d) values.

The comparison above illustrates the main differences in the goals/contributions of each scenario. Namely, the orginal RCs are concerned with the storage/repair-bandwidth tradeoff for the worst possible helper selection. LRCs, however, are concerned with only data storage (ignoring repair-bandwidth) of the codes that can perform stationary helper selection and exact repair. Some recent development [7], [8] in LRCs consider using RCs in the construction of the codes therein (as local codes) in an attempt to examine the repair-bandwidth performance of LRCs. This approach, however, is not gauranteed to be optimal in terms of storage/repair-bandwidth tradeoff. In this work, we present the first exploration of the optimal storage-bandwidth tradeoff for RCs that allow dynamic helper selection for arbitrary (n, k, d) values, including both the cases of $k \gg d$ and $k \ll d$.

III. PREVIEW OF THE RESULTS

In the following, we give a brief preview of our results through examples to illustrate the main contributions of this work. Although we only present here specific examples as a preview, the main results in Section V are for general (n, k, d) values.

Result 1: For $(n, k, d) = (6, 3, 4)$, RCs with BR are absolutely optimal, i.e., there exists no RCs with dynamic helper selection that can outperform BR. This also implies that there exists no LRCs with symmetric repair-bandwidth per node that can outperform BR since LRCs with symmetric repair are special RCs with dynamic helper selection.

Result 2: For $(n, k, d) = (6, 4, 4)$, the Family-Repair (FR) RCs proposed in this paper are absolutely optimal in terms of the storage-bandwidth tradeoff among all RCs with dynamic helper selection. In Fig. 1, the storage-bandwidth tradeoff curve of the FR scheme, the optimal helper selection scheme,

is plotted against the BR scheme with file size $\mathcal{M} = 1$. In fact, for $(n, k, d) = (6, 4, 4)$, the random LRCs in [9] designed for $\gamma = \infty$ have to satisfy $\mathcal{M} \leq k\alpha = 4\alpha$, i.e., can at most perform as good as the MSR point of the BR scheme. Moreover, the LRCs utilizing MBR codes in [8] perform equally to the MBR point of the BR scheme. Both the LRC constructions in [8] and [9] are strictly suboptimal and perform worse than the proposed family-repair scheme, which is provably optimal.

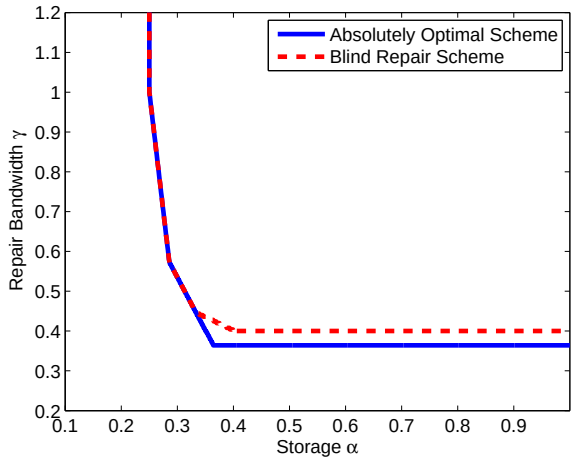

Fig. 1. Storage-bandwidth tradeoff curves of RCs with BR versus RCs with the absolutely optimal scheme (FR) for $(n, k, d) = (6, 4, 4)$ and file size $\mathcal{M} = 1$.

Result 3: For $(n, k, d) = (20, 10, 10)$, we do not know what is the absolutely optimal dynamic helper selection scheme. We, however, have that the FR scheme outperforms the BR scheme. Fig. 2 shows a tradeoff curve comparison between the FR scheme and the BR scheme.

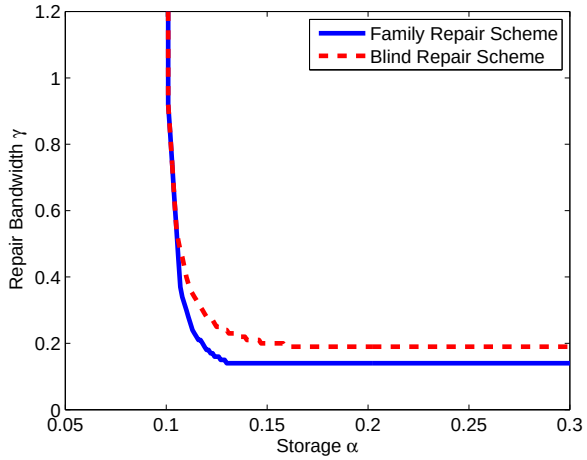

Fig. 2. Storage-bandwidth tradeoff curves of RCs with BR versus RCs with FR for $(n, k, d) = (20, 10, 10)$ and file size $\mathcal{M} = 1$.

Result 4: For $(n, d) = (60, 10)$, we do not know what is the absolutely optimal dynamic helper selection. However, in Fig. 3, we plot a k versus repair-bandwidth $\alpha = \gamma = d\beta$ curve which compares the BR scheme to the FR scheme. The curve of the MBR-LRCs in [8] is also provided in the same figure. Note that the family-plus repair scheme, as we will see in Section VI, is an extension of the FR scheme to cover the case when n is large and d is small. Examining Fig. 3, we can see that the BR scheme performs very poorly compared to the other codes when k is large. Comparing the plots of the family-plus repair scheme to the plot of the MBR-LRCs, we can see that the MBR-LRCs perform equally when k is very large but performs poorly otherwise. From this, we see that RCs with the family-plus repair scheme perform well for arbitrary (n, k, d) values as discussed in Table I.

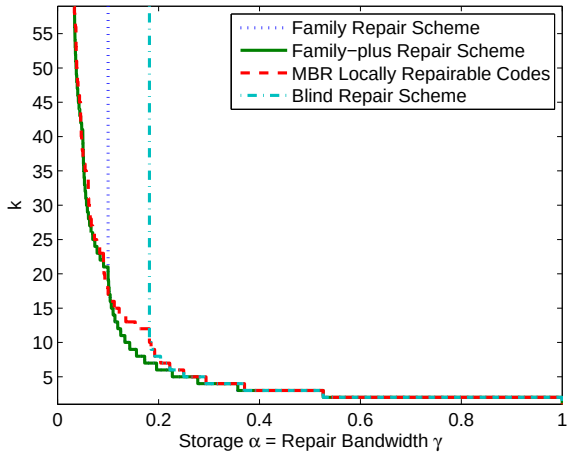

Fig. 3. The k value versus repair-bandwidth γ curve comparison at the MBR point for $(n, d) = (60, 10)$ and file size $\mathcal{M} = 1$.

IV. The Min-Cut-Based Analysis and The Proposed Family Repair Schemes

As in [3], the performance of a distributed storage system can be characterized by the concept of information flow graphs (IFG). Due to the space constraint, the detailed description of the IFG is omitted (see [3] for details).

Intuitively, each IFG reflects one unique history of the failure patterns and the helper selection choices from time 1 to $(t-1)$ [3]. For any given helper selection scheme A, since there are infinitely many different failure patterns (since we consider $t = 1$ to ∞), there are infinitely many IFGs corresponding to the same given helper selection scheme A. We denote the collection of all such IFGs by $\mathcal{G}_A(n, k, d, \alpha, \beta)$. We define $\mathcal{G}(n, k, d, \alpha, \beta) = \bigcup_{\forall A} \mathcal{G}_A(n, k, d, \alpha, \beta)$ as the union over all possible helper selection schemes A. We sometimes drop the input argument and use $\mathcal{G}_A$ and $\mathcal{G}$ as shorthands.

Given an IFG $G \in \mathcal{G}$, we use $\mathrm{DC}(G)$ to denote the collection of all $\binom{n}{k}$ *data collector nodes* in G [3]. Each data collector $t \in \mathrm{DC}(G)$ represents one unique way of choosing k out of n nodes when reconstructing the file. Given an IFG $G \in \mathcal{G}$ and a data collector $t \in \mathrm{DC}(G)$, we use $\mathrm{mincut}_G(s, t)$ to denote the *minimum cut value* [20] separating s, the root

node (source node) of G, and t. For any helper scheme A and given system parameters n, k, d, α, and β, if the following min-cut condition is satisfied

$$\min_{G\in\mathcal{G}_A}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t)\geq\mathcal{M} \tag{3}$$

where $\mathcal{M}$ is the total file size, under the given helper selection scheme A, RCs can protect against arbitrary failure patterns while still allowing file reconstruction from any k nodes.[4]

Paper [3] later proves the following:

$$\min_{G\in\mathcal{G}}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t)=\sum_{i=0}^{k-1}\min\{(d-i)^+\beta,\alpha\}. \tag{4}$$

As a result, as long as "(4) $\geq\mathcal{M}$" is true, the RCs meet the reliability requirement even for the worst possible helper selection scheme (since we take the minimum over $\mathcal{G}$). Moreover, whenever (4) $<\mathcal{M}$, there exists a bad helper selection scheme A for which the reliability requirement is no longer met. We call "(4) $\geq\mathcal{M}$", the characterization of the BR scheme.

Fix the values of n, k, and d, "(4) $\geq\mathcal{M}$" describes the storage-bandwidth tradeoff (α versus β) of the BR scheme. Two points on a storage-bandwidth tradeoff curve are of special interest: the minimum-bandwidth RC (MBR) point and the minimum-storage RC (MSR) point where the former has the smallest possible repair bandwidth (the β value) and the latter has the smallest possible storage per node (the α value). See [3] for the expressions of the MBR and MSR points (α_{MBR},γ_{MBR}) and (α_{MSR},γ_{MSR}) of the BR scheme .

It is possible mathematically that when focusing on $\mathcal{G}_A$ ($\mathcal{G}_A$ is by definition a strict subset of $\mathcal{G}$) we may have

$$\min_{G\in\mathcal{G}_A}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t)>\min_{G\in\mathcal{G}}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t). \tag{5}$$

If (5) is true, the given helper selection scheme A strictly outperforms the BR solution. Whether (or under what condition) (5) is true and how much the gap can be are the two main focuses of this work.

A. Family Repair Schemes

Now we describe the *family repair (FR) scheme*, a sub-class of the stationary repair schemes. We define a *complete family* as a group of $(n-d)$ physical nodes. Arbitrarily sort all storage nodes and denote them by 1 to n. The first $(n-d)$ nodes are grouped as the first complete family and the second $(n-d)$ nodes are grouped as the second complete family and so on. In total, there are $\left\lfloor\frac{n}{n-d}\right\rfloor$ complete families. The remaining $n \bmod (n-d)$ nodes are grouped as an *incomplete family*. The helper set D_i of any node i in a complete family contains all the nodes *not* in the same family of node i. That is, a newcomer only seeks help from *outside* its family. The intuition is that we would like each family to preserve as much information (or equivalently as diverse information) as possible so that we would like to refrain the newcomer from requesting help from its own family. For any node in the incomplete family,[5] we set the corresponding $D_i=\{1,\cdots,d\}$. For example, consider $n=8$ and $d=5$. There are 2 complete families, $\{1,2,3\}$ and $\{4,5,6\}$, and 1 incomplete family, $\{7,8\}$. Then, if node 4 fails, the corresponding newcomer will access nodes 1 to 3 and nodes 7 and 8 for repair. If node 7 (of the incomplete family) fails, the newcomer will access nodes 1 to 5 for repair.

By the above definitions, we have in total $\left\lceil\frac{n}{n-d}\right\rceil$ number of families, which are indexed from 1 to $\left\lceil\frac{n}{n-d}\right\rceil$. However, since the incomplete family has different properties from the complete family, we replace the index of the incomplete family by 0. Therefore, the family indices become from 1 to $c\triangleq\left\lfloor\frac{n}{n-d}\right\rfloor$ and then 0, where c is the index of the last complete family (if there is no incomplete family, we simply omit the index 0). Moreover, by our construction, any node of the incomplete family has $D_i=\{1,\cdots,d\}$. That is, it will request help from *all* the nodes of the first $(c-1)$ complete families, *but only from* the first $d-(n-d)(c-1)=n \bmod (n-d)$ nodes of the last complete family. Among the $(n-d)$ nodes in the last complete family, we thus need to distinguish those nodes who will be helpers for incomplete family nodes, and those who will not. Therefore, we add a negative sign to the family indices of those who will not be helpers for the incomplete family. We can now list the family indices of the n nodes as an n-dimensional *family index vector*. For the same example above where $n=8$ and $d=5$, the third node of the second complete family, node 6, will never be a helper for the incomplete family nodes, nodes 7 and 8. We thus write the family index vector of nodes 1 to n as $(1,1,1,2,2,-2,0,0)$. Mathematically, we can write the family index vector as

$$\Bigg(\overbrace{1,\cdots,1}^{n-d},\ \overbrace{2,\cdots,2}^{n-d},\cdots,\ \overbrace{c,\cdots,c}^{n \bmod (n-d)},\ \overbrace{-c,\cdots,-c}^{n-d-n \bmod (n-d)},\ \overbrace{0,\cdots,0}^{n \bmod (n-d)}\Bigg). \tag{6}$$

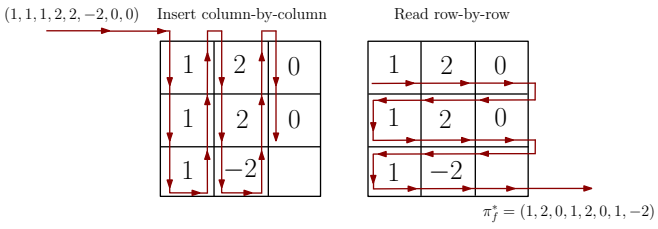

Fig. 4. The construction of the RFIP for $(n,d)=(8,5)$.

[4]We note that we need here the assumption that, whenever the cut condition of (3) is satisfied, there exists a finite field GF(q) such that the corresponding RC satisfies the reliability requirement. Rigorous proofs are needed to prove this assumption. For example, one major contribution of [22] is to prove the existence of such a finite field for the cut-value based characterization in [3]. In Proposition 1, we have proved both the cut-value-based characterization and the existence of such a finite field.

[5]All the concepts and intuitions are based on complete families. The incomplete family is used to make the scheme consistent and applicable to the case when $n \bmod (n-d)\neq 0$.

A *family index permutation* is a permutation of the above family index vector (6), which we denote by π_f. A rotating family index permutation (RFIP) π_f^* is a special family index permutation that puts the family indices of (6) in an $(n-d) \times \left\lceil \frac{n}{n-d} \right\rceil$ table column-by-column and then reads it row-by-row. Fig. 4 illustrates the construction of the RFIP for the case of $n = 8$ and $d = 5$. The input is the family index vector $(1, 1, 1, 2, 2, -2, 0, 0)$ and the output RFIP π_f^* is $(1, 2, 0, 1, 2, 0, 1, -2)$.

Due to the lack of space, we omit most of the proofs in this paper. The detailed proofs for all the propositions and their corollaries can be found in our arXiv paper [1].

V. Main Results

A. When is it beneficial to choose the good helpers?

Recall that we only consider (n, k, d) values that satisfy (1).

Proposition 1: If at least one of the following conditions is true: (i) $d = 1$, $k = 3$, and n is odd; and (ii) $k \leq \left\lceil \frac{n}{n-d} \right\rceil$, then for any dynamic helper selection scheme A, we have

$$\min_{G \in \mathcal{G}_A} \min_{t \in \mathrm{DC}(G)} \mathrm{mincut}_G(s,t) = \sum_{i=0}^{k-1} \min\{(d-i)^+\beta, \alpha\}. \quad (7)$$

That is, even the best dynamic repair scheme cannot do better than the BR solution. Conversely, for any (n, k, d) values that satisfy neither (i) nor (ii), there exists a helper selection scheme A and a pair of (α, β) values such that

$$\min_{G \in \mathcal{G}_A} \min_{t \in \mathrm{DC}(G)} \mathrm{mincut}_G(s,t) > \sum_{i=0}^{k-1} \min\{(d-i)^+\beta, \alpha\}. \quad (8)$$

Moreover, for the same (α, β) values and the same helper selection scheme A that satisfy (8), if the file size $\mathcal{M}$ also satisfies (3), then there exists a finite field GF(q) such that we can explicitly construct a RC that meets the reliability requirement.

By noticing that the left-hand sides of (7) and (8) are identical to (4), Proposition 1 thus answers the central question: Under what conditions does it help to choose the good helpers?

B. Quantifying the benefits of the Family Repair schemes

To quantify the gap in (5), we now turn our focus to the FR schemes.

Proposition 2: Consider a given FR scheme F with the corresponding IFGs denoted by $\mathcal{G}_F(n, k, d, \alpha, \beta)$. We have

$$\min_{G \in \mathcal{G}_F} \min_{t \in \mathrm{DC}(G)} \mathrm{mincut}_G(s,t) = \min_{\forall \pi_f} \sum_{i=1}^{k} \min\left\{(d - y_i(\pi_f))\,\beta, \alpha\right\}, \quad (9)$$

where π_f can be any family index permutation and $y_i(\pi_f)$ is computed as follows. If the i-th coordinate of π_f is 0, then $y_i(\pi_f)$ returns the number of j satisfying both (i) $j < i$ and (ii) the j-th coordinate > 0. If the i-th coordinate of π_f is not 0, then $y_i(\pi_f)$ returns the number of j satisfying both (i) $j < i$ and (ii) the absolute values of the j-th and the i-th coordinates of π_f are different. For example, if $\pi_f = (1, 2, -2, 1, 0, 0, 1, 2)$, then $y_6(\pi_f) = 3$ and $y_8(\pi_f) = 5$.

Remark: In general, the minimum cut of an IFG may exist in the middle of the graph. Recall that the family index permutation π_f is based on the family index vector of all "currently active nodes." Proposition 2 thus implies that we can reduce the search scope and consider only those cuts that directly separate k currently active nodes from the rest of the IFG, which allows us to explicitly compute the min-cut value.

Combining Proposition 2 and (3), we can derive the new storage-bandwidth tradeoff (α vs. β) for the FR scheme. For example, Fig. 2 plots α versus $\gamma \triangleq d\beta$ for the (n, k, d) values $(20, 10, 10)$ with file size $\mathcal{M} = 1$. As can be seen in Fig. 2, the MBR point (the smallest γ value) of the FR scheme uses only 72% of the repair bandwidth of the MBR point of the BR scheme ($\gamma_{\mathrm{MBR}} = 0.13$ vs. 0.18). It turns out that for any (n, k, d) values, the biggest improvement always happens at the MBR point. The intuition is that choosing the good helpers is most beneficial when the per-node storage α is no longer a bottleneck (thus the MBR point).

C. The MBR and MSR points of the FR scheme

Computing the right-hand side of (9) is of complexity $\mathcal{O}\left(\left(\frac{n}{n-d}\right)^k\right)$. The following proposition shows that for the most beneficial point, the MBR point, we can compute the corresponding α and β values in polynomial time.

Proposition 3: For the MBR point of (9), i.e., when α is sufficiently large, the minimizing family index permutation is the RFIP π_f^* defined in Section IV-A. That is, the α, β, and γ values of the MBR point can be computed by

$$\alpha_{\mathrm{MBR}} = \gamma_{\mathrm{MBR}} = d\beta_{\mathrm{MBR}} = \frac{d\mathcal{M}}{\sum_{i=1}^{k}(d - y_i(\pi_f^*))}. \quad (10)$$

Our best knowledge for computing the MSR point (other than directly applying the formula in Proposition 2) is the following:

Proposition 4: For arbitrary (n, k, d) values, the minimum storage of (9) is $\alpha_{\mathrm{MSR}} = \frac{\mathcal{M}}{\min(d,k)}$. If the (n, k, d) values also satisfy $d \geq k$, then the corresponding $\beta_{\mathrm{MSR}} = \frac{\mathcal{M}}{k(d-k+1)}$.

D. Is the family repair scheme optimal?

In the following, we prove that the FR scheme is indeed optimal for some (n, k, d) values. To that end, we first introduce the following corollary.

Corollary 1: For any (n, k, d) values satisfying $d \geq 2$ and $k = \left\lceil \frac{n}{n-d} \right\rceil + 1$, we consider the corresponding IFGs $\mathcal{G}_F(n, k, d, \alpha, \beta)$ generated by the family repair scheme F. We then have that

$$\min_{G \in \mathcal{G}_F} \min_{t \in \mathrm{DC}(G)} \mathrm{mincut}(s,t) = \min_{2 \leq m \leq k} C_m, \quad (11)$$

where $C_m = \sum_{i=0}^{k-1} \min\{(d-i)\beta, \alpha\} 1_{\{i \neq m-1\}} + \min\{(d - m + 2)\beta, \alpha\}$ for $2 \leq m \leq k$.

Corollary 1 is used in [1] to prove the following proposition.

Proposition 5: For the (n,k,d) values satisfying simultaneously the following three conditions (i) d is even; (ii) $n=d+2$; and (iii) $k=\frac{n}{2}+1$, we have

$$\min_{G\in\mathcal{G}_F}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t)\geq\min_{G\in\mathcal{G}_A}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}_G(s,t) \tag{12}$$

for any arbitrary dynamic helper selection scheme A.

Note that for any (n,k,d) values satisfying conditions (i) to (iii) in Proposition 5, they must also satisfy neither (i) nor (ii) in Proposition 1. As a result, by Proposition 1, there exists some helper selection scheme that strictly outperforms the BR scheme. Proposition 5 further establishes that among all those schemes strictly better than the BR scheme, the FR scheme is indeed optimal. We will show in Section VI that the FR scheme and its extension, the family-plus repair scheme, are actually also *weakly optimal* for general (n,k,d) values. The definition of weak optimality will be provided in Proposition 7.

VI. Family-plus Repair Scheme

In the FR scheme, there are $\left\lfloor\frac{n}{n-d}\right\rfloor$ complete families and 1 incomplete family (if $n \bmod (n-d)\neq 0$). For the scenario in which the n and d values are comparable, we have many complete families and the FR solution harvests almost all of the benefits of choosing good helpers, see the discussion of Proposition 5 for which $n=d+2$. However, when n is large but d is small, we have only one complete family and one incomplete family. Therefore, even though the FR scheme still substantially outperforms the BR scheme, see Fig. 3 for the case of $(n,d)=(60,10)$, the performance of the FR scheme is far from optimal due to having only 1 complete family. In this section, we propose the *family-plus repair* scheme that further improves the storage-bandwidth tradeoff when n is large but d is small.

The main idea is as follows. We first partition the n nodes into several disjoint groups of $2d$ nodes and one disjoint group of n_{remain} nodes. The first type of groups is termed the regular group while the second group is termed the remaining group. If we have to have one remaining group (when $n \bmod 2d\neq 0$), then we enforce the size of the remaining group to be as small as possible but still satisfying $n_{\text{remain}}\geq 2d+1$. For example, if $d=2$ and $n=8$, then we will have 2 regular groups and no remaining group since $n \bmod 2d=0$. If $d=2$ and $n=9$, then we choose 1 regular group $\{1,2,3,4\}$ and 1 remaining group $\{5,6,7,8,9\}$ since we need to enforce $n_{\text{remain}}\geq 2d+1$.

After the partitioning, we apply the FR scheme to the individual groups. For example, if $d=2$ and $n=8$, then we have two regular groups $\{1,2,3,4\}$ and $\{5,6,7,8\}$. Applying the FR scheme to the first group means that nodes 1 and 2 form a family and nodes 3 and 4 form another family. Whenever node 1 fails, it will access helpers from outside its family, which means that it will access nodes 3 and 4. Node 1 will never request help from any of nodes 5 to 8 as these nodes are not in the same group as node 1. Similarly, we apply the FR scheme to the second group $\{5,6,7,8\}$. All the FR operations are always performed within the same group. The above scheme is termed the *family-plus repair scheme*.

One can easily see that when $n\leq 2d$, there is only one group and the family-plus repair scheme collapses to the FR scheme. When $n>2d$, there are approximately $\frac{n}{2d}$ regular groups, each of which contains two complete families. Therefore, the construction of the family-plus repair scheme ensures that there are many complete families even for the scenario of $n\gg d$. In the following proposition, we characterize the performance of the family-plus repair scheme.

Proposition 6: Consider any given (n,k,d) values and the family-plus repair scheme F^+. Suppose we have B groups in total (including both regular and remaining groups) and each group has n_b number of nodes for $b=1$ to B. Specifically, if the b-th group is a regular group, then $n_b=2d$. If the b-th group is a regular group (when $n \bmod 2d\neq 0$), then $n_b=n-2d(B-1)$. We use $\mathcal{G}_{F^+}(n,k,d,\alpha,\beta)$ to denote the IFGs generated by the family-plus repair scheme. We have that

$$\min_{G\in\mathcal{G}_{F^+}}\min_{t\in\mathrm{DC}(G)}\mathrm{mincut}(s,t)= \min_{\mathbf{k}\in K}\sum_{b=1}^{B}\min_{H\in\mathcal{G}_F(n_b,k_b,d,\alpha,\beta)}\min_{t_b\in\mathrm{DC}(H)}\mathrm{mincut}_H(s,t), \tag{13}$$

where $\mathbf{k}$ is a B-dimensional integer-valued vector, $K=\{(k_1,k_2,\cdots,k_B):\forall b\in\{1,\cdots,B\},0\leq k_b\leq n_b,\text{and}\sum_{b=1}^{B}k_b=k\}$. Note that for any given $\mathbf{k}$, the right-hand side of (13) can be evaluated by Proposition 2.

To evaluate the right-hand side of (13), we have to try all possible choices of the $\mathbf{k}$ vectors and for each given $\mathbf{k}$, we evaluate each of the B summands by Proposition 2, which requires checking all $n_b!$ different family index permutations. On the other hand, for the MBR point of the family-plus repair scheme, we can further simplify the computation complexity following similar arguments as used in Proposition 3.

Corollary 2: The MBR point of the family-plus repair scheme is

$$\alpha_{\mathrm{MBR}}=\gamma_{\mathrm{MBR}}=d\beta_{\mathrm{MBR}} \tag{14}$$

where β_{MBR} can be computed by solving the following equation

$$\left(d^2\left\lfloor\frac{(k-n_{\text{remain}})^+}{2d}\right\rfloor+\sum_{i=0}^{q}\left(d-i+\left\lfloor\frac{i}{2}\right\rfloor\right)+\sum_{i=0}^{\min(k,2d)-1}\left(d-i+\left\lfloor\frac{i}{2}\right\rfloor\right)^+\right)\beta_{\mathrm{MBR}}=\mathcal{M},$$

$q=(k-n_{\text{remain}})^+ \bmod (2d)-1$ and $\mathcal{M}$ is the file size.

In Fig. 3, we plot the k vs. γ curves for the BR, the FR, and the family-plus repair schemes for $(n,d)=(60,10)$, Proposition 3, and Corollary 2. As can be seen, when $k=40$, the family-plus repair scheme only uses 28% of the repair bandwidth of the BR scheme (cf. the FR scheme uses 58% repair bandwidth of the BR scheme). This demonstrates the benefits of the family-plus repair scheme, which creates as

many complete families as possible by further partitioning the nodes into several disjoint groups.

We close this section by stating the weak optimality of the family-plus repair scheme for all (n, k, d) values.

Proposition 7: Consider a family-plus repair scheme denoted by F^+, and the corresponding IFGs denoted by $\mathcal{G}_{F^+}$. For any (n, k, d) values satisfying neither of the (i) and (ii) conditions in Proposition 1, there exists a pair (α, β) such that

$$\min_{G\in\mathcal{G}_{F^+}} \min_{t\in \mathrm{DC}(G)} \mathrm{mincut}_G(s,t) > \sum_{i=0}^{k-1} \min\{(d-i)^+\beta, \alpha\}. \tag{15}$$

Propositions 7 and 1 jointly show that whenever helper selection can improve the performance, so can the family-plus repair scheme, which we term the "weak optimality."

VII. Conclusion

This paper provided a necessary and sufficient condition under which optimally choosing good helpers improves the storage-bandwidth tradeoff. We also analyzed a new class of low-complexity solutions termed the *family repair scheme*, including its storage-bandwidth tradeoff, the expression of its MBR point, and its (weak) optimality. The main goal of this work is to characterize, for the first time in the literature, when and by how much dynamic helper selection improves RCs. We thus consider only the scenario of single failures in a similar way as in the original RC paper [3]. Since a practical system can easily have multiple failures, as ongoing work, we are studying the helper selection problem under the multiple failures scenario.

References

[1] I. Ahmad and C.-C. Wang, "When and by how much can helper node selection improve regenerating codes?" *[Online]. Available: arXiv:1401.4509 [cs.IT]*.

[2] V. R. Cadambe, S. A. Jafar, and H. Maleki, "Distributed data storage with minimum storage regenerating codes-exact and functional repair are asymptotically equally efficient," *[Online]. Available: arXiv:1004.4299 [cs.IT]*.

[3] 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, 2010.

[4] S. El Rouayheb and k. Ramchandran, "Fractional repetition codes for repair in distributed storage systems," in *Proc. 48th Annual Allerton Conf. on Comm., Contr., and Computing.*, Monticello, Illinois, Sept. 2010, pp. 1510–1517.

[5] S. Ghemawat, H. Gobioff, and S. T. Leung, "The google file system," in *Proc. 19th ACM Symp. on Operating Systems Principles (SOSP)*, Bolton Landing, NY, Oct. 2003, pp. 29–43.

[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, 2012.

[7] G. M. Kamath, N. Prakash, V. Lalitha, and P. V. Kumar, "Codes with local regeneration," in *IEEE Information Theory and Applications Workshop (ITA)*, San Diego, CA, Feb. 2013, pp. 1–5.

[8] G. M. Kamath, N. Silberstein, N. Prakash, A. S. Rawat, V. Lalitha, O. O. Koyluoglu, P. Kumar, and S. Vishwanath, "Explicit mbr all-symbol locality codes," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Istanbul, Turkey, Jul. 2013, pp. 504–508.

[9] D. S. Papailiopoulos and A. G. Dimakis, "Locally repairable codes," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Cambridge, MA, Jul. 2012, pp. 2771–2775.

[10] D. S. Papailiopoulos, J. Luo, A. G. Dimakis, C. Huang, and J. Li, "Simple regenerating codes: Network coding for cloud storage," in *Proc. IEEE INFOCOM*, Orlando, FL, Mar. 2012, pp. 2801–2805.

[11] N. Prakash, G. M. Kamath, V. Lalitha, and P. V. Kumar, "Optimal linear codes with a local-error-correction property," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Cambridge, MA, Jul. 2012, pp. 2776–2780.

[12] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction," *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 5227–5239, 2011.

[13] K. V. Rashmi, N. B. Shah, P. V. Kumar, and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," in *Proc. 47th Annu. Allerton Conf. Communication, Control, and Computing*, Urbana-Champaign,IL, Sep. 2009, pp. 1243–1249.

[14] A. S. Rawat, O. O. Koyluoglu, N. Silberstein, and S. Vishwanath, "Optimal locally repairable and secure codes for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 60, no. 1, pp. 212–236, 2012.

[15] S. Rhea, C. Wellis, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz, "Maintenance-free global data storage," *Internet Computing, IEEE*, vol. 5, no. 5, pp. 40–49, 2001.

[16] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, "Distributed storage codes with repair-by-transfer and nonachievability of interior points on the storage-bandwidth tradeoff," *IEEE Trans. Inf. Theory*, vol. 58, no. 3, pp. 1837–1852, 2012.

[17] ——, "Interference alignment in regenerating codes for distributed storage: Necessity and code constructions," *IEEE Trans. Inf. Theory*, vol. 58, no. 4, pp. 2134–2158, 2012.

[18] K. W. Shum and Y. Hu, "Cooperative regenerating codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 11, pp. 7229–7258, 2013.

[19] C. Suh and K. Ramchandran, "On the existence of optimal exact-repair mds codes for distributed storage," *[Online]. Available: arXiv:1004.4663 [cs.IT]*.

[20] D. B. West, *Introduction to graph theory*. Prentice Hall Upper Saddle River, NJ., 2001, vol. 2.

[21] Y. Wu, "A construction of systematic mds codes with minimum repair bandwidth," *IEEE Trans. Inf. Theory*, vol. 57, no. 6, pp. 3738–3741, 2011.

[22] Y. Wu and A. G. Dimakis, "Reducing repair traffic for erasure coding-based storage via interference alignment," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, Seoul, South Korea, Jul. 2009, pp. 2276–2280.