Explicit Codes Uniformly Reducing Repair Bandwidth in Distributed Storage

Nihar B. Shah[†], K. V. Rashmi[†], P. Vijay Kumar[†], Kannan Ramchandran[#]

[†] Dept. of ECE, Indian Institute Of Science, Bangalore, India.
Email: {rashmikv, nihar, vijay}@ece.iisc.ernet.in
[#] Dept. of EECS, University of California, Berkeley, USA.
Email: kannanr@eecs.berkeley.edu

***Abstract*—A distributed storage setting is considered where a file of size B is to be stored across n storage nodes. A data collector should be able to *reconstruct* the entire data by downloading the symbols stored in any k nodes. When a node fails, it is replaced by a new node by downloading data from some of the existing nodes. The amount of download is termed as *repair bandwidth*. One way to implement such a system is to store one fragment of an (n, k) MDS code in each node, in which case the repair bandwidth is B. Since repair of a failed node consumes network bandwidth, codes reducing repair bandwidth are of great interest.**

Most of the recent work in this area focuses on reducing the repair bandwidth of a set of k nodes which store the data in uncoded form, while the reduction in the repair bandwidth of the remaining nodes is only marginal. In this paper, we present an explicit code which reduces the repair bandwidth for all the nodes to approximately $B/2$. To the best of our knowledge, this is the first explicit code which reduces the repair bandwidth of all the nodes for all feasible values of the system parameters.

I. Introduction

Consider a distributed storage system with n storage nodes. A file of size B units of data (symbols) is to be stored across these n nodes with the property that a data collector (DC) should be able to obtain the entire file by downloading data from any k nodes. This is termed as *reconstruction* property.

Let each node have a storage space of α symbols, where each symbol is assumed to belong to some finite field $\mathbb{F}_q$ of size q. Clearly, for the reconstruction property to hold, we need

$$\alpha \geq \alpha_{\min} = B/k \tag{1}$$

When a storage node fails, it has to be replaced by a new storage node, in order to maintain the same level of reliability. The new node is created by downloading data from a subset of the existing nodes such that, along with the existing $n-1$ nodes, it satisfies the reconstruction property. This operation of replacing a failed node by a new node is termed *regeneration*. The amount of data to be downloaded for regeneration is termed as *repair bandwidth* denoted by γ. The system is depicted in Figure 1.

A simple way of achieving the above specified storage system is to use a (n, k) MDS code where each node stores one fragment of the MDS code. Here repair bandwidth is the entire file B, though the amount of storage in the failed node is just (B/k).

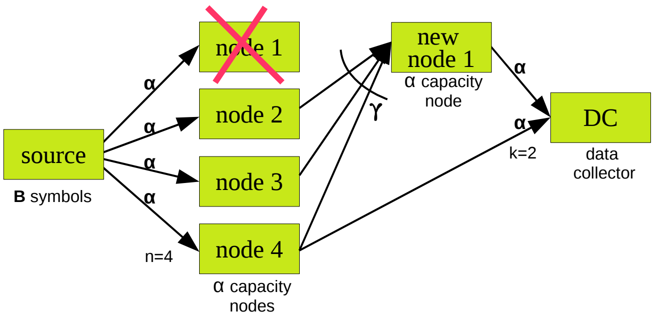

Fig. 1. An illustration of reconstruction and regeneration: On failure of node 1, data from nodes 2, 3 and 4 is used for regeneration

In the pioneering paper in this area [1], authors consider a setting in which each failed node is regenerated by downloading a uniform amount of data from d of the existing nodes. They introduce a scheme called *regenerating codes* wherein each node stores slightly larger amount of data than a MDS based code in order to reduce the repair bandwidth. The authors also establish a tradeoff between the repair bandwidth and amount of storage at each node. The two extreme points of the tradeoff curve are minimum storage regeneration point (MSR) and minimum bandwidth regeneration point (MBR) corresponding to minimum storage at each node and minimum repair bandwidth respectively. Although the existence of regenerating codes has been proved in [1], the suggested code construction is not explicit and has high complexity and demands high field size.

In our previous work [2] we give an explicit code for optimal regeneration at the MBR point where the regenerated node downloads data from all existing $n-1$ nodes, and is constrained to store data identical to the failed node. We also give an optimal explicit construction at the MSR point where regeneration is done using only $k+1$ of the existing nodes.

In [3] and [4] authors consider a set of k nodes as systematic, which store data in uncoded form to decrease computational complexity of reconstruction. In [3] authors present a scheme which reduces the repair bandwidth of the systematic nodes at the MSR point. However this construction is not explicit and also requires a high field size. In [4] we provide a scheme which minimizes the repair bandwidth of the systematic nodes at the MSR point. However in both the schemes mentioned above, emphasis is given to reducing repair bandwidth of only the systematic nodes, whereas the decrease in repair bandwidth of the non-systematic nodes is marginal. Hence the average repair bandwidth across all the nodes will be high if n is large as compared to k and this is usually the case when high reliability is required.

In this paper, we provide an explicit code which reduces repair bandwidth to approximately half the file size for all the nodes. The code construction provided has low complexity and requires a low field size. We provide codes for all values of the parameters B, α, n and k which satisfy equation (1).

The rest of the paper is organized as follows. Section II provides a code construction for the trivial case of $\alpha = 1$. Construction for $\alpha = 2$ is provided in Section III. Using the constructions for $\alpha = 1$ and 2 as building blocks, a construction for any value of α is given in Section IV. Finally, in Section V we compare our construction to other schemes in the literature.

II. Construction for the case $\alpha = 1$

$\alpha = 1$ is a trivial case in which any $(n,\ k)$ MDS code minimizes the repair bandwidth. However, code construction for this case is provided here as this will be used in the construction for the general case.

Let $\underline{f}$ be a vector with B source symbols as its elements. Let l denote the length of this vector. Hence $l = B$, and from equation (1) we have

$$l \leq k \tag{2}$$

Let $\underline{p}^{(i)}$ $(i = 1, \ldots, n)$ be vectors of length l forming a l-dimensional MDS code over $\mathbb{F}_q$. In the rest of the paper, we will use superscripts to denote the node number corresponding to any symbol.

Code: Node i stores $\underline{f}^t \underline{p}^{(i)}$, $i = 1, \ldots, n$.

A. Reconstruction

Lemma 1: This code for $\alpha = 1$ can achieve successful reconstruction for a DC connecting to any k nodes.

Proof: The DC will obtain $\underline{f}^t \underline{p}^{(i)}$ evaluated at k different values. Since $\underline{p}^{(i)}$'s form a $l-$dimensional MDS code and $k \geq l$, the DC can solve for the values of the B source symbols. ■

B. Regeneration

For regeneration of a failed node, we need $d \geq l$.

Lemma 2: Any failed node can be regenerated by downloading one symbol each from any d existing nodes.

Proof: Since $l = B$, the entire file can be reconstructed, using which the symbol stored in the failed node can be regenerated. ■

C. Repair bandwidth

The repair bandwidth for any node is

$$\gamma_1(l) = l = B \tag{3}$$

where the subscript 1 indicates that there is only one symbol to be regenerated in the failed node. This is the minimum possible repair bandwidth for $\alpha = 1$. Note that the repair bandwidth is a function of the length l of the source vector.

III. Construction for the case $\alpha = 2$

The code construction in this case is based on the optimal code construction given in our previous work [2].

Partition the source symbols into two sets S and S' having sizes l and l' respectively such that

$$0 \leq l, l' \leq k \tag{4}$$

and

$$(l + l') = B \tag{5}$$

Let $\underline{f},\ \underline{g}$ be two vectors with their elements as the constituents of the sets S and S' respectively. Hence these vectors have lengths l and l' respectively. For $i = 1 \ldots, n$ let $\underline{p}^{(i)}$ be vectors of length l forming a l-dimensional MDS code over $\mathbb{F}_q$ and $\underline{r}^{(i)}$ be vectors of length l' forming a l'-dimensional MDS code over $\mathbb{F}_q$.

Code: Node i $(i = 1, \ldots, n)$ stores $(\underline{f}^t \underline{p}^{(i)},\ \underline{g}^t \underline{r}^{(i)} + \underline{f}^t \underline{u}^{(i)})$ as its two symbols. The vectors $\underline{p}^{(i)}$ and $\underline{r}^{(i)}$ will be referred to as the *main vectors* and $\underline{u}^{(i)}$ as the *auxiliary vector* of the node i. The elements of $\underline{u}^{(i)}$ can be initialized to any arbitrary values from $\mathbb{F}_q$.

For example, consider $k = 3$ and $B = 5$. Let $b_0,\ b_1,\ b_2,\ b_3,\ b_4$ be the source symbols. Let $l = 3$ and $l' = 2$. Set $\underline{f} = (b_0,\ b_1,\ b_2)^t$ and $\underline{g} = (b_3,\ b_4)^t$. For $i = 1, \ldots, n$ let the main vectors $\underline{p}^{(i)}$ and $\underline{r}^{(i)}$ form a Reed-Solomon code with $\underline{p}^{(i)} = (1\ \theta_i\ \theta_i^2)^t$ and $\underline{r}^{(i)} = (1\ \theta_i)$. θ_i $(i = 1, \ldots, n)$ take distinct values from $\mathbb{F}_q (q \geq n)$. Elements of $\underline{u}^{(i)}$ can be initialized to arbitrary values from $\mathbb{F}_q$.

A. Reconstruction:

Lemma 3: The code given for $\alpha = 2$ can achieve successful reconstruction for a DC connecting to any k nodes.

Proof: The first symbols of some l out these k nodes provide $\underline{f}^t \underline{p}^{(i)}$ at l different values of i. To solve for $\underline{f}$, we have l linear equations in l unknowns. Since $\underline{p}^{(i)}$'s form a $l-$dimensional MDS code, these equations are linearly independent. As $l \leq k$, they can be solved to obtain values of the elements of $\underline{f}$.

Now, as $\underline{f}$ and $\underline{u}^{(i)}$ are known, $\underline{f}^t \underline{u}^{(i)}$ can be subtracted out from the second symbols of some l' out of the k nodes. This gives $\underline{g}^t \underline{r}^{(i)}$ evaluated at l' different values of i. As $l' \leq k$, this can be used to recover the elements of $\underline{g}$. ■

Thus the B symbols can be recovered by a DC which connects to any k nodes. We also see that reconstruction can be performed irrespective of the values of the auxiliary vectors $\underline{u}^{(i)}$.

B. Regeneration:

In this construction, when a node fails, the main vectors of the regenerated node will be identical to that of the failed node and the auxiliary vector is allowed to be different.

Suppose node j fails. The node replacing it would contain $(\underline{f}^t\underline{p}^{(j)},\ \underline{g}^t\underline{r}^{(j)} + \underline{f}^t\tilde{\underline{u}}^{(j)})$ where elements of $\tilde{\underline{u}}_j$ can take arbitrary values from $\mathbb{F}_q$ and are not constrained to be equal to those of $\underline{u}_j$. As the reconstruction property holds irrespective of the values of $\underline{u}_j$, the regenerated node along with the existing nodes has all the desired properties.

For regeneration of the failed node we need

$$d \geq \max(l, l'+1) \tag{6}$$

Lemma 4: A failed node can be successfully regenerated by downloading one symbol each from any d existing nodes.

Proof: The node replacing the failed node downloads one symbol each from some d of the $n-1$ existing nodes. Consider failure of node Λ_{d+1}, where nodes $\Lambda_1, \ldots, \Lambda_d$ participate in regeneration. Here the set $\{\Lambda_1, \ldots, \Lambda_{d+1}\}$ is some subset of $\{1, \ldots, n\}$.

For $i = 1, \ldots, d$, node Λ_i passes a symbol which is a linear combination of the two symbols stored in it. Let a_i and b_i be the coefficients of this linear combination. Thus $\pi_i = a_i(\underline{f}^t\underline{p}^{(\Lambda_i)}) + b_i(\underline{g}^t\underline{r}^{(\Lambda_i)} + \underline{f}^t\underline{u}^{(\Lambda_i)})$ is the symbol passed by node Λ_i.

The two symbols stored in the new node will be linear combinations of these downloaded symbols. Let δ_i and ρ_i be the coefficients of these linear combinations. Thus the regenerated node will store

$$\left(\sum_{i=1}^{d}\delta_i\pi_i\ ,\ \sum_{i=1}^{d}\rho_i\pi_i\right) \tag{7}$$

Choose

$$b_i = \begin{cases} 1 & \text{for } i = 1, \ldots, l'+1 \\ 0 & \text{for } i = l'+2, \ldots, d \end{cases} \tag{8}$$

Now choose $\rho_i\ (i = 1, \ldots, l'+1)$ such that

$$\sum_{i=1}^{l'+1}\rho_i\underline{r}^{(\Lambda_i)} = \underline{r}^{(\Lambda_{d+1})} \tag{9}$$

and $\rho_i = 0 \quad \text{for } i = l'+2, \ldots, d$

Equation (9) is a set of l' non-homogeneous linear equations in $l'+1$ unknowns. Since $\underline{r}^{(\Lambda_i)}$'s form a $l'-$dimensional MDS code, a solution is guaranteed and can be easily obtained.

Choose $\delta_i\ (i = 1, \ldots, l'+1)$ such that

$$\sum_{i=1}^{d}\delta_i\underline{r}^{(\Lambda_i)} = \underline{0} \tag{10}$$

and $\delta_i = 1 \quad \text{for } i = l'+2, \ldots, d.$

Equation (10) is a set of l' homogeneous linear equations in $l'+1$ unknowns. Since $\underline{r}^{(\Lambda_i)}$'s form a $l'-$dimensional MDS code, a solution with all $\delta_i\ (i = 1, \ldots, l'+1)$ non-zero can be obtained in $\mathbb{F}_q$.

Now, choose $a_i\ (i = 1, \ldots, d)$ such that

$$\sum_{i=1}^{d}\delta_i(a_i\underline{p}^{(\Lambda_i)} + b_i\underline{u}^{(\Lambda_i)}) = \underline{p}^{(\Lambda_{d+1})} \tag{11}$$

i.e

$$\sum_{i=1}^{d}\delta_i a_i\underline{p}^{(\Lambda_i)} = \underline{p}^{(\Lambda_{d+1})} - \sum_{i=1}^{d}\delta_i b_i\underline{u}^{(\Lambda_i)} \tag{12}$$

Equation (12) is a set of l linear equations in d unknowns. Since $d \geq l$, none of the $\delta_i\ (i = 1, \ldots, d)$ are zero, and $\underline{p}^{(\Lambda_i)}$'s form a $l-$dimensional MDS code, it can be solved to obtain values for $a_i\ (i = 1, \ldots, d)$. ∎

C. Optimum partition size and repair bandwidth

The repair bandwidth for any node is

$$\gamma_2(l, l') = \max(l, l'+1) \tag{13}$$

Thus, to minimize the repair bandwidth the partition sizes should be

$$l = \left\lceil\frac{B}{2}\right\rceil \text{ and } l' = \left\lfloor\frac{B}{2}\right\rfloor \tag{14}$$

IV. Construction for any α

This section gives a code construction for the general case using constructions for $\alpha = 1$ and $\alpha = 2$ as building blocks.

Let

$$\tau = \left\lfloor\frac{\alpha}{2}\right\rfloor \tag{15}$$

Partition the B source symbols into $2\tau + 1$ sets $S_1, S'_1, \ldots, S_\tau, S'_\tau, S_{\tau+1}$ having sizes $l_1, l'_1, \ldots, l_\tau, l'_\tau, l_{\tau+1}$ respectively satisfying the following conditions

$$0 \leq l_j,\ l'_j \leq k \quad \forall j \in \{1, \ldots, \tau\} \tag{16}$$

$$l_{\tau+1} = 0 \text{ for } \alpha \text{ even} \tag{17}$$

$$0 \leq l_{\tau+1} \leq k \text{ for } \alpha \text{ odd} \tag{18}$$

and

$$\sum_{j=1}^{\tau}(l_j + l'_j) + l_{\tau+1} = B \tag{19}$$

Let $\underline{f}_1, \underline{g}_1, \ldots, \underline{f}_\tau, \underline{g}_\tau, \underline{f}_{\tau+1}$ be $2\tau + 1$ vectors with their elements as the constituents of the sets $S_1, S'_1, \ldots, S_\tau, S'_\tau, S_{\tau+1}$ respectively. Hence the lengths of these vectors are $l_1, l'_1, \ldots, l_\tau, l'_\tau, l_{\tau+1}$ respectively.

For $j = 1, \ldots, \tau+1$ let $\underline{p}_j^{(i)}\ (i = 1, \ldots, n)$ be vectors of length l_j forming a l_j-dimensional MDS code over $\mathbb{F}_q$. For $j = 1, \ldots, \tau$ let $\underline{r}_j^{(i)}\ (i = 1, \ldots, n)$ be vectors of length l'_j forming a l'_j-dimensional MDS code over $\mathbb{F}_q$.

Code: For every pair of vectors $(\underline{f}_j,\ \underline{g}_j),\ j = 1, \ldots, \tau$ apply the construction given for $\alpha = 2$ in Section III by taking $\underline{f}_j$ as $\underline{f}$ and $\underline{g}_j$ as $\underline{g}$. Each pair of vectors determines two symbols to be stored in that node. When α is odd, the construction given for $\alpha = 1$ in Section II is applied on $\underline{f}_{\tau+1}$ to obtain one symbol. The symbols so obtained constitute the α symbols stored at each node.

Hence node $i\ (\in \{1, \ldots, n\})$ stores

$$\left[\{\ \underline{f}_j^t\underline{p}_j^{(i)},\ \underline{g}_j^t\underline{r}_j^{(i)} + \underline{f}_j^t\underline{u}_j^{(i)}\ \}_{j=1}^{\tau},\ \underline{f}_{\tau+1}^t\underline{p}_{\tau+1}^{(i)}\right] \tag{20}$$

as its α symbols (as shown in Figure 2), where the symbol corresponding to $\underline{f}_{\tau+1}$ is present only when α is odd.

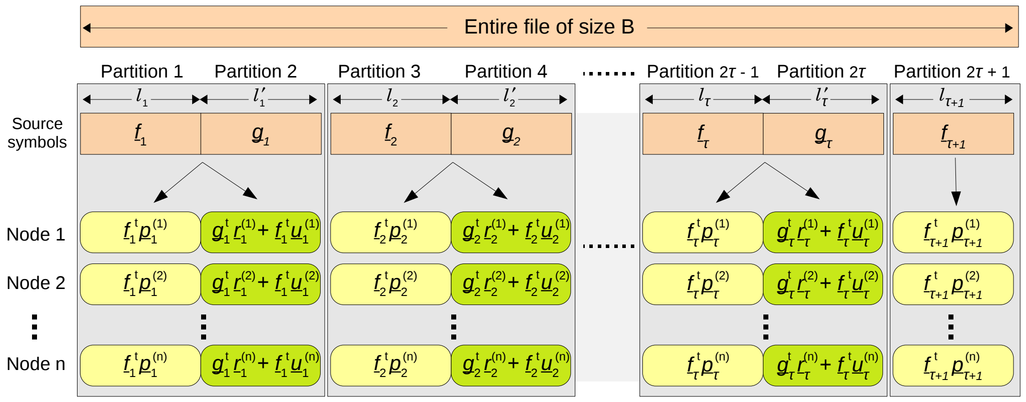

Fig. 2. Partitioning of the source file in the code construction for any α.

A. Reconstruction

Theorem 5: The code given can achieve successful reconstruction for a DC connecting to any k nodes.

Proof: Each of the first τ pair of symbols stored in any node is separately constructed using the code given for $\alpha = 2$ in Section III. Apply Lemma 3 separately on each pair of symbols stored in the k nodes to reconstruct $\{\underline{f}_i,\ \underline{g}_i\}_{i=1}^{\tau}$.

When α is odd, the last symbol stored is constructed using the code for $\alpha = 1$ given in Section II. Apply Lemma 1 on the last symbol stored in the k nodes to reconstruct $\underline{f}_{\tau+1}$. Thus all the B source symbols can be reconstructed by the DC. ∎

B. Regeneration

Theorem 6: The code given can perform successful regeneration of any failed node.

Proof: The symbols to be stored in the new node replacing the failed node are regenerated in the following manner. Each of the first τ pairs of symbols are regenerated separately as described in Lemma 4. The amount of download required for the regeneration of the j^{th} pair of symbols is $\max(l_j, l'_j + 1)$.

When α is odd, the last symbol to be stored in the new node is regenerated as described in Lemma 2 by downloading $l_{\tau+1}$ symbols. ∎

Encoding, reconstruction and regeneration is performed on each pair of vectors separately, thereby immensely reducing the complexity of each of these operations.

C. Optimum partition size and repair bandwidth

The repair bandwidth is dependent on the partition sizes. By the method of regeneration described in Theorem 6, the repair bandwidth for any node is given by

$$\gamma = \sum_{j=1}^{\tau} \gamma_2(l_j,\ l'_j + 1) + \gamma_1(l_{\tau+1}) \tag{21}$$

$$= \sum_{j=1}^{\tau} \max(l_j,\ l'_j + 1) + l_{\tau+1} \tag{22}$$

where equation (22) follows from equations (3) and (13).

Thus the optimum size of the partitions is the solution to the following optimization problem:

$$\min \left[\sum_{j=1}^{\tau} \max(l_j,\ l'_j + 1) + l_{\tau+1} \right] \tag{23}$$

subject to the conditions (16), (17), (18), (19) and $l_j, l'_j, l_{\tau+1}\ \forall j \in \{1, \ldots, \tau\}$ being integers.

The following theorem provides a method to pick the partition sizes in order to minimize the repair bandwidth.

Theorem 7: The bandwidth required to repair a failed node is upper bounded by

$$\gamma \leq \frac{B}{2} + \frac{\alpha}{2} + k - 1 \tag{24}$$

Proof: The following is an intuitive explanation of the strategy to optimally allocate sizes of the partitions. Consider the B source symbols as balls and the α partitions as buckets. The capacity of each bucket is k. We need to distribute all the balls in the buckets in a manner which satisfies the optimization problem given in (23).

Choose a pair of empty buckets. Put k balls in the first bucket and $k - 1$ in the second. Continue picking more empty pairs of buckets and filling them in this manner, until you cannot proceed. Each such pair of buckets consumes $2k - 1$ balls and contributes k to the download bandwidth. The number of buckets used will be $2\min(\lfloor \frac{B}{2k-1} \rfloor, \lfloor \frac{\alpha}{2} \rfloor)$ If there are any more balls left, then one of the two cases must arise:

Case 1: At least one pair of empty buckets is available and the number of balls remaining is less than $2k - 1$ i.e. $\lfloor \frac{B}{2k-1} \rfloor < \lfloor \frac{\alpha}{2} \rfloor$.

The number of balls left will be $B \bmod (2k - 1)$. If this number is even, then distribute the balls equally in the two buckets. If it is odd, then distribute the remaining balls in the two buckets such that the first bucket gets one more than the second. This step contributes $\left\lfloor \frac{B \bmod (2k-1)}{2} \right\rfloor + 1$ to the download bandwidth.

Case 2: The number of empty buckets remaining is at most 1 i.e. $\lfloor \frac{B}{2k-1} \rfloor \geq \lfloor \frac{\alpha}{2} \rfloor$.

The number of balls left will be $B - (2k - 1)\lfloor \frac{\alpha}{2} \rfloor$. Consider the set of all buckets, and put each remaining ball in any bucket which is not yet full. Each such ball will contribute 1 to the download bandwidth.

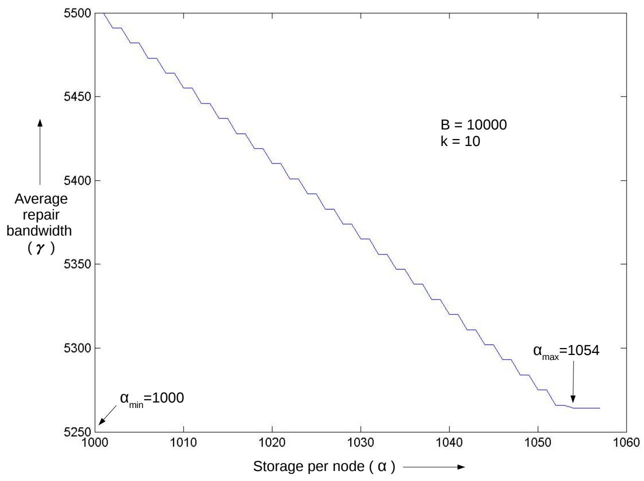

Fig. 3. Repair bandwidth (γ) for regeneration of any failed node for parameters $B = 10000$ and $k = 10$.

The repair bandwidth for any node is given by

$$\gamma = \begin{cases} k\left\lfloor\frac{B}{2k-1}\right\rfloor + \left\lceil\frac{B \bmod (2k-1)}{2}\right\rceil + 1 & \text{for } \left\lfloor\frac{B}{2k-1}\right\rfloor < \left\lfloor\frac{\alpha}{2}\right\rfloor \\ B-(k-1)\left\lfloor\frac{\alpha}{2}\right\rfloor & \text{for } \left\lfloor\frac{B}{2k-1}\right\rfloor \geq \left\lfloor\frac{\alpha}{2}\right\rfloor \end{cases} \quad (25)$$

This expression can be simplified to obtain an upper bound on the repair bandwidth as given by (24). ■

Thus the download bandwidth for any node is approximately half the file size.

The repair bandwidths required for regeneration of a failed node for parameters $B = 10000$ and $k = 10$ are plotted for various values of α in Figure 3. The graph is a downward step since the repair bandwidth decreases when α increases from odd to even, but remains constant when α increases from even to odd.

It follows from equation (25) that if the storage per node is increased beyond a certain threshold, the repair bandwidth does not reduce any further. This threshold is given by

$$\alpha_{\max} = 2\left\lceil\frac{B}{2k-1}\right\rceil \quad (26)$$

V. Comparison with existing schemes

A. Repair bandwidth

The construction provided in the present paper reduces the repair bandwidth uniformly for all the nodes in the system to approximately half the file size. To the best of our knowledge, this is the first explicit code to do so for all feasible values of the system parameters.

In [1] authors use a more general set up where the symbols stored in a regenerated node need not be related to the symbols stored in the failed node. The only constraint on the new node is that it along with the existing nodes should satisfy the reconstruction and regeneration properties. Codes introduced here are optimum in the sense that they achieve the minimum possible repair bandwidth. However no explicit constructions are provided.

In [3] and [4] authors consider the MSR point with the constraint that a failed node is replaced by an identical node. Schemes provided reduce the repair bandwidth of systematic nodes alone. Non-systematic nodes are regenerated by downloading close to entire file size. As the repair bandwidth for the non-systematic nodes is quite high in both these schemes, the average repair bandwidth will be high if n is much larger than k.

B. Complexity

As the code is explicit, the construction is immediate provided the field size is greater than n.

In our construction, the main vectors of the regenerated node are identical to the main vectors of the failed node. However the auxiliary vector is permitted to be different. We term this as an *approximately exact repair*. Since the matrix inversions performed for solving the linear equations during reconstruction and regeneration depend only on the main vectors, these matrix inversions need to be computed just once. Hence, system maintenance has a low complexity.

In [1] the authors suggest the usage of a general network code construction algorithm by Jaggi et al. [5] to obtain the code. This algorithm has complexity higher than the order of the number of sinks and and requires field size of the order of number of sinks. The number of sinks in the network representation of the distributed storage system is large which leads to high complexity and high field size requirements.

The construction given in [4] is explicit and hence immediate given the field size. The construction given in [3] is not explicit, and has a high code construction complexity.

C. Field size required

If we use a Reed-Solomon code as the MDS code in the construction, the minimum field size required is

$$q \geq n \quad (27)$$

On the other hand, constructions provided in [1] rely on the algorithm given in [5] which requires field size of the order of number of sinks, which is high for the distributed storage setup. The scheme provided in [3] also the drawback of high field size requirement. The code given in [4] requires a low field size.

References

[1] Y. Wu, A. G. Dimakis and K. Ramchandran, "Deterministic regenerating codes for distributed storage," in *Proc. Allerton Conference on Control, Computing and Communication*, Urbana-Champaign, September 2007.

[2] K. V. Rashmi, N. B. Shah, P. V. Kumar and K. Ramchandran, "Explicit construction of optimal exact regenerating codes for distributed storage," in *Proc. Allerton Conference on Control, Computing and Communication,* Urbana-Champaign, September 2009.

[3] Y. Wu and A. Dimakis, "Reducing repair traffic for erasure coding-based storage via interference alignment," in *Proc. IEEE Intl Symp. on Information Theory,* Seoul, Korea, July 2009.

[4] N. B. Shah, K. V. Rashmi, P. V. Kumar and K. Ramchandran,"Explicit codes minimizing repair bandwidth for distributed storage," to appear in *Proc. Information Theory Workshop,*, Cairo, January 2010.

[5] S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen, "Polynomial time algorithms for network code construction," *IEEE Transactions on Information Theory*, vol.51, pp.782-795.