SCALING UP OF E-MSR CODES BASED DISTRIBUTED STORAGE SYSTEMS WITH FIXED NUMBER OF REDUNDANCY NODES

Haotian Zhao, Yinlong Xu and Liping Xiang

School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui, China
zhaoht@mail.ustc.edu.cn, ylxu@ustc.edu.cn, xlping@mail.ustc.edu.cn

ABSTRACT

Distributed storage systems are becoming more and more popular with the rapidly increasing demand for large-scale data storage. To increase the capacity and I/O performance of a distributed storage system, scaling it up is a common method. Regenerating Codes are a class of distributed storage codes that offer good reliability through encoding and provide good bandwidth cost on failed nodes repairing. This paper studies the scaling problem of E-MSR codes based distributed storage systems with fixed number of redundancy codes. We generate the encoding matrices of an (n, k) storage system carefully from the encoding matrices of an $(n + t, k + t)$ storage system to minimize the changes of encoded blocks when scaling. Therefore the system can be scaled up with relatively low bandwidth cost and computation cost.

KEYWORDS

Distributed storage, network coding

1. INTRODUCTION

Recent years, distributed storage systems are becoming more and more popular with the rapidly increasing demand for large-scale data storage. The storage nodes in these systems are often individually unreliable. So redundancy is introduced into these systems to improve reliability with the cost of increased storage. The simplest redundancy scheme is replication, where data is straightforwardly replicated in multiple storage nodes. Erasure coding schemes generate redundancy with encoding and such schemes can achieve the same reliability with replication using much less redundant storage cost. However, the bandwidth cost of repairing failed nodes is increased [9]. Regenerating codes are a kind of encoding schemes which significantly decrease the bandwidth cost of failed nodes repairing. Dimakis et al. [3] derived the lower bounds on the repair communication of regenerating codes. Optimal storage efficiency codes are called minimum-storage regenerating (MSR) codes [2] [4] while optimal bandwidth efficiency codes are called minimum-bandwidth regenerating (MBR) codes [7] [8] [5].

To meet the rapidly increasing demand of capacity and I/O performance, scaling up distributed storage systems is a common operation. Replication schemes based systems are relatively easy to scale up because they are quite simple. But the scaling up of regenerating codes is much more complicate. There are some previous researches on RAID scaling problem [11] [10] [12]. But as far as we know, there are few previous works focused on scaling of regenerating codes.

This paper works on the scaling problems of systematic E-MSR codes based distributed storage systems which have fixed number of redundancy nodes. An (n, k) MSR based distributed storage system is a system having n nodes and can tolerant any $n - k$ nodes failure. Such a system is able to provide the same reliability with replication schemes using much less redundancy storage, while the bandwidth cost of failed nodes repairing is much less than erasure

DOI : 10.5121/ijdps.2012.3501

codes based systems. E-MSR is short for Exact-MSR, which means that in the repairing process of a failed node, the replacement node is constrained to store exactly the same data as the corresponding failed node. A "systematic" code means that the system stores systematic parts, i.e., uncoded original file data blocks. Keeping a code in systematic form is important in practice since most of the time a user would simply want to access a part of the stored information. If the code is systematic, accessing part of a file is very simple. On the contrary if every node in the system stores encoded blocks, to decode one piece of the information would require to decoding the whole file, which brings high bandwidth cost and computation cost [1]. This paper focuses on the expansion from an (n,k) E-MSR code to an $(n+t,k+t)$ E-MSR code, ensuring load balancing. Meanwhile by carefully designing the encoding matrices of the (n,k) E-MSR code from the encoding matrices of the $(n+t,k+t)$ E-MSR code, only part of original file data is needed to update the encoded redundancy in the scaling process, thereby the bandwidth cost and computation cost are decreased.

2. PROBLEM FORMULATION

This paper focus on the problem of scaling an (n,k) E-MSR code based distributed storage system up to an $(n+t,k+t)$ E-MSR code based distributed storage system. An (n,k) E-MSR code based distributed storage system is a system achieving the minimum storage point in the storage-repair tradeoff curve in [3] and satisfying the following properties: (1) *Reconstruction property*: An user connecting to any k of the n nodes is able to reconstruct the original file. (2) *Exact repair property*: After repairing process of a failed node, the data stored in the replacement node is exactly the same with the failed one. We want to ensure load balancing after scaling up and decrease the bandwidth cost and computation cost in the scaling process.

2.1. E-MSR codes

Table 1. Matrices.

A_i	$(n-k)\times k(n-k)$	N_i stores A_iD
$B_{i,j}$	$1\times(n-k)$	N_i sends $B_{i,j}A_iD$ to N_0 when N_j failed
C_i	$(n-k)\times(n-1)$	matrix used to rebuild storage node N_i
D	$k(n-k)\times x$	matrix of original file data

Let $N_1,\ldots,N_n$ denote the n storage nodes of the (n,k) E-MSR code based distributed storage system. The original file data of size M is divided into several blocks, constructing data matrix D which has $k(n-k)$ rows. Each node N_i stores A_iD, data of size $\frac{M}{k}$, where A_i is the $(n-k)\times k(n-k)$ encoding matrix. As a systematic code, $\begin{pmatrix}A_1^T & \cdots & A_k^T\end{pmatrix}^T = I_{k(n-k)}$. E-MSR codes satisfy reconstruction property. Choosing any k nodes $N_{c_1},N_{c_2},\ldots,N_{c_k}$, an user connecting to these k nodes gets encoded blocks $\begin{pmatrix}A_{c_1}^T & \cdots & A_{c_k}^T\end{pmatrix}^T D$. If and only if the square matrix $\begin{pmatrix}A_{c_1}^T & \cdots & A_{c_k}^T\end{pmatrix}^T$ of order $k(n-k)$ is full ranked, the original file data D can be decoded.

When a node N_j failed, the repair process is evoked with a new node N_0 to replace the failed node. N_0 connects to all $n-1$ remaining nodes and downloads $\frac{M}{k(n-k)}$ size of data from each of these $n-1$ nodes. For node N_i, it encodes the blocks it stores using a $1\times(n-k)$ vector $B_{i,j}$ and transfers $B_{i,j}A_iD$ to N_0. Then N_0 encodes the blocks it received using an $(n-k)\times(n-1)$ matrix C_j and stores the encoded blocks. The numbers of rows and columns of the encoding matrices $A_i, B_{i,j}, C_i$ here are designed to ensure the system to achieve the minimum storage point in the storage-repair tradeoff curve. E-MSR codes satisfy exact repair

property. After the repair process, the blocks N_0 stores are exactly the same with the blocks stored at the failed node N_j. In other words,

$$C_j \begin{pmatrix} B_{1,j}A_1D \\ \vdots \\ B_{j-1,j}A_{j-1}D \\ B_{j+1,j}A_{j+1}D \\ \vdots \\ B_{n,j}A_nD \end{pmatrix} = A_jD.$$

Fig. 1 is an example of a $(5,3)$ E-MSR code over GF(3), with encoding matrices

$$\begin{pmatrix} A_1 \\ \vdots \\ A_5 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 2 & 2 & 0 & 1 & 1 \\ 1 & 0 & 1 & 2 & 1 & 2 \\ 2 & 2 & 2 & 2 & 0 & 2 \\ 2 & 1 & 0 & 2 & 1 & 1 \end{pmatrix}.$$

Here the encoding matrices of any 3 nodes form a nonsingular square matrix of order 6. For example, the encoding matrices of nodes N_1, N_3, N_4 form an encoding matrix as

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 2 & 2 & 0 & 1 & 1 \\ 1 & 0 & 1 & 2 & 1 & 2 \end{pmatrix}.$$

It is easy to see that this matrix is full ranked. So a user can decode the original file data by connecting to these 3 nodes.

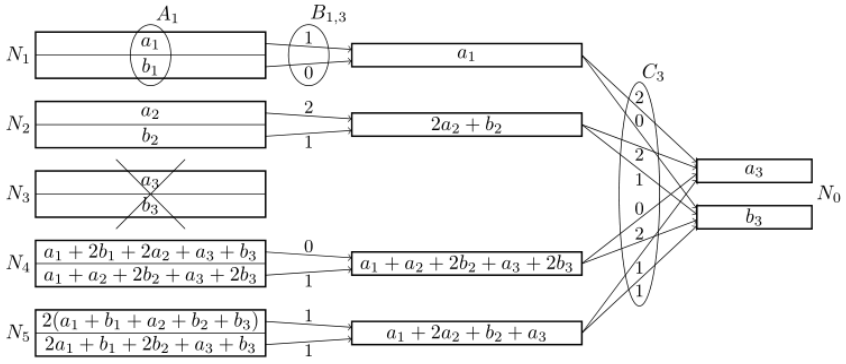

Figure 1. An example of a $(5,3)$ E-MSR code over GF(3)

When a node failed (in this example the failed node is N_3), the repair process is evoked with a new node N_0 to replace the failed node. Each $N_i (i = 1, 2, 4, 5)$ sends encoded blocks to N_0. For example N_1 sends $B_{1,3}A_1D$ to N_0 where $B_{1,3} = \begin{pmatrix} 1 & 0 \end{pmatrix}$, and N_2 sends $B_{2,3}A_1D$ to N_0 where $B_{2,3} = \begin{pmatrix} 2 & 1 \end{pmatrix}$. Then N_0 encodes the encoded blocks received using an encoding matrix $C_3 = \begin{pmatrix} 2 & 2 & 0 & 1 \\ 0 & 1 & 2 & 1 \end{pmatrix}$. After the repair process, N_0 stores exactly the same data with the failed node N_3.

2.2. The Scaling Problem

Suppose we have an (n, k) E-MSR based distributed storage system and we want to scale it up to a system of $n + t$ nodes. This system is $(n + t, k + t)$ E-MSR based, i.e., it also achieves the minimum storage point, meanwhile satisfies reconstruction property and exact repair property. For the purpose of better I/O performance, the $(n + t, k + t)$ system after scaling needs to be load balancing. And we hope to minimize the bandwidth cost in the scaling up process.

Use $N_1^{(t)}, \ldots, N_{n+t}^{(t)}$ to denote the $n + t$ nodes of the $(n + t, k + t)$ E-MSR code based distributed storage system, where $N_1^{(t)}, \ldots, N_t^{(t)}$ are t new nodes and $N_{i+t}^{(t)} (1 \le i \le n)$ is N_i of the (n, k) system. $A_i^{(t)}, B_{i,j}^{(t)}, C_i^{(t)}$ $(1 \le i, j \le n + t, j \ne i)$ are the encoding matrices of the $(n + t, k + t)$ system, just like $A_i, B_{i,j}, C_i$ of the (n, k) one. $D^{(t)}$ is the original file data matrix of the $(n + t, k + t)$ system, which stores the same elements as D, but in a different form. $D^{(t)}$ has $(k + t)(n - k)$ columns. Since the $(n + t, k + t)$ system is load balanced, so each node $N_i^{(t)}$ stores $\frac{M}{k+t}$ size of data, while each node N_i of the (n, k) system stores $\frac{M}{k}$ size of data. In order to the convenience of data transfer in the scaling process, let D has $k + t$ columns denoted as $D = \begin{pmatrix} D_1 & \cdots & D_{k+t} \end{pmatrix}$, so $D^{(t)}$ has k columns denoted as $D^{(t)} = \begin{pmatrix} D_1^{(t)} & \cdots & D_k^{(t)} \end{pmatrix}$. We call $n - k$ blocks A_iD_j to be a strip. So N_i stores $k + t$ strips while $N_i^{(t)}$ stores k strips.

In the scaling process, for the systematic part, $\frac{t}{k+t}M$ size of data are transferred from $N_1, \ldots, N_k$ to $N_1^{(t)}, \ldots, N_t^{(t)}$. For the encoded data blocks stored in the $n - k$ redundancy nodes, a simple way is that the encoding matrices of the (n, k) system and the $(n + t, k + t)$ system are chosen independently so that these blocks need to be totally rebuilt, which brings high bandwidth cost and high computation cost.

Fig. 2 is an example of scaling a $(4, 2)$ E-MSR code up to a $(5, 3)$ E-MSR code, where the encoding matrices of these two E-MSR codes are chosen independently. The left is the $(4, 2)$ E-MSR code. Nodes N_1, N_2 store the systematic blocks where node N_3, N_4 are redundancy nodes storing encoded blocks. The right is the $(5, 3)$ E-MSR code after expansion. Each of N_1, N_2 sends two blocks to $N_1^{(t)}$ to keep load balancing. While nodes $N_4^{(t)}, N_5^{(t)}$ need to rebuild the encoded blocks they store so the whole 12 blocks of original file data are needed. We can see the total bandwidth cost is 28 blocks. And the computation cost is also high because the whole file needs to be encoded when rebuilding $N_4^{(t)}, N_5^{(t)}$. One way to reduce bandwidth cost is that only $N_4^{(t)}$ downloads the 12 blocks of original file data and besides generating the encoded blocks itself needs, $N_4^{(t)}$ also generates the 4 encoded blocks which $N_5^{(t)}$ needs. So $N_5^{(t)}$ may download only 4 blocks from $N_4^{(t)}$ instead of 12 blocks. The total bandwidth cost in this way is 20 blocks.

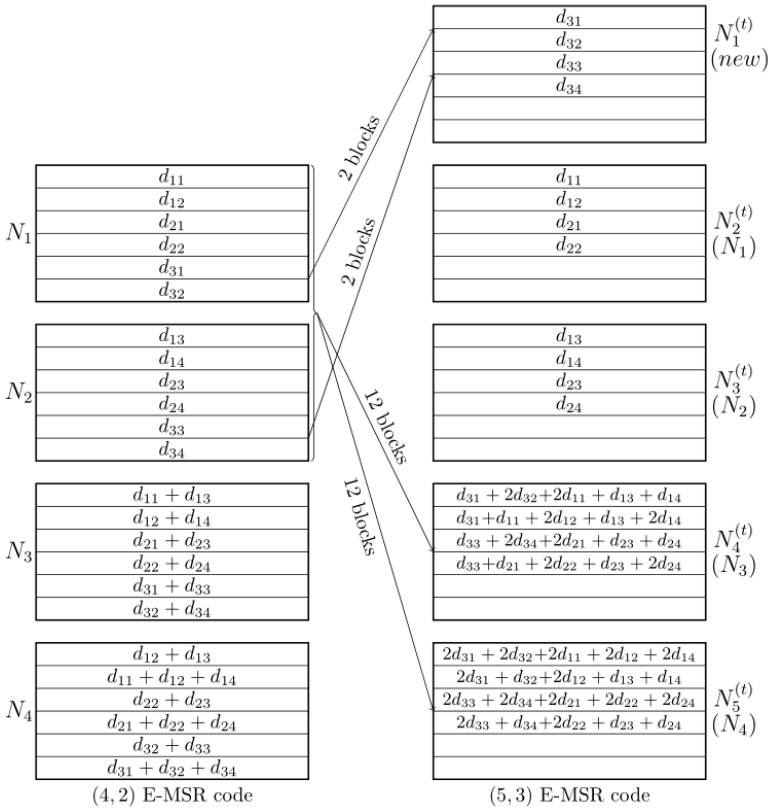

Figure 2. An example of scaling a $(4, 2)$ E-MSR code based system up to a $(5, 3)$ system where the encoding matrices of the two systems are chosen independently.

It can be seen that the bandwidth cost of rebuilding encoded blocks is very high. And since the encoded blocks are encoded from the whole original file data, so the computation cost is also high. In order to reduce the bandwidth cost and computation cost, we need to carefully design the encoding matrices of the (n, k) E-MSR code and the $(n + t, k + t)$ E-MSR code such that in the scaling process, the redundancy nodes need only parts of the original file data to update their encoded blocks instead of needing the whole original file data to rebuild the encoded blocks.

3. DESIGN OF ENCODING MATRICES

Suppose it is known that it will be later scaled up to an $(n + t, k + t)$ E-MSR code when constructing the (n, k) E-MSR code. We construct the encoding matrices of the (n, k) E-MSR code using the encoding matrices of the $(n + t, k + t)$ E-MSR code to make them having some same features. In the scaling process, the encoded data blocks of the (n, k) E-MSR code can still be used to generate the encoded data blocks of the $(n + t, k + t)$ E-MSR code. Instead of downloading the whole original file data, the redundancy nodes downloading only parts of the systematic data and encoding these data with the encoded data blocks they originally stored, are able to accomplish the scaling process. So the bandwidth cost and computation cost are reduced.

Some previous works focused on the constructions of E-MSR codes. For example Rashmi *et al.* [6] presented explicit constructions of E-MSR codes for all $(n, k, d \geq 2k - 2)$. We first get an $(n + t, k + t)$ E-MSR code just like in the previous works. Then we construct an (n, k) E-MSR

code using this $(n+t, k+t)$ E-MSR code. Such constructed (n, k) E-MSR code can be scaled up to $(n+t, k+t)$ sometime later with a relatively low bandwidth cost and computation cost.

An $(n+t, k+t)$ E-MSR code satisfies reconstruction property and exact repair property. Choosing any $k+t$ nodes $N_{c_1}^{(t)}, \ldots, N_{c_{k+t}}^{(t)}$, the matrix $\begin{pmatrix} A_{c_1}^{(t)^T} & \cdots & A_{c_{k+t}}^{(t)^T} \end{pmatrix}^T$ is full ranked. And

$$A_j^{(t)} = C_j^{(t)} \begin{pmatrix} B_{1,j}^{(t)} A_1^{(t)} \\ \vdots \\ B_{j-1,j}^{(t)} A_{j-1}^{(t)} \\ B_{j+1,j}^{(t)} A_{j+1}^{(t)} \\ \vdots \\ B_{n+t,j}^{(t)} A_{n+t}^{(t)} \end{pmatrix}$$

for any $j (1 \le j \le n+t)$. It is able to construct an (n, k) E-MSR code from any $(n+t, k+t)$ E-MSR code. The encoding matrices $A_i, B_{i,j}, C_i \, (1 \le i, j \le n, j \le i)$ of the (n, k) E-MSR code can be constructed like this. Let

$$A_i = A_{i+t}^{(t)} \begin{pmatrix} O \\ I_{k(n-k)} \end{pmatrix}_{(k+t)(n-k) \times k(n-k)}, \tag{1}$$

$$B_{i,j} = B_{i+t,j+t}^{(t)}, \tag{2}$$

$$C_i = C_{i+t}^{(t)} \begin{pmatrix} O \\ I_{n-1} \end{pmatrix}_{(n+t-1) \times (n-1)}. \tag{3}$$

Such constructed (n, k) E-MSR code also satisfy reconstruction property and exact repair property.

Proof of reconstruction property: Choosing any k nodes $N_{c_1}, \ldots, N_{c_k}$ from $N_1, \ldots, N_n$, we get an encoding matrix $A_c = \begin{pmatrix} A_{c_1}^T & \cdots & A_{c_k}^T \end{pmatrix}^T$. If A_c is not full ranked, for a $k+t$ nodes choice $N_1^{(t)}, \ldots, N_t^{(t)}, N_{c_1+t}^{(t)}, \ldots N_{c_k+t}^{(t)}$ in the $(n+t, k+t)$ E-MSR code, the encoding matrix is $A_c^{(t)} = \begin{pmatrix} A_1^{(t)^T} & \cdots & A_t^{(t)^T} & A_{c_1+t}^{(t)^T} & \cdots & A_{c_k+t}^{(t)^T} \end{pmatrix}^T$. Since it's a systematic code, so $\begin{pmatrix} A_1^{(t)^T} & \cdots & A_l^{(t)^T} \end{pmatrix}^T = \begin{pmatrix} I_{t(n-k)} & O \end{pmatrix}$. $A_c^{(t)}$ can be converted to $\begin{pmatrix} I_{t(n-k)} & O \\ O & A_c \end{pmatrix}$ by elementary row transformation. This matrix is not full ranked because A_c is not full ranked, and this is contradicted to the reconstruction property of the $(n+t, k+t)$ E-MSR code.

Proof of exact repair property: For the $(n+t, k+t)$ E-MSR code, we have

$$A_j^{(t)} = C_j^{(t)} \begin{pmatrix} B_{i,j}^{(t)} A_1^{(t)} \\ \vdots \\ B_{j-1,j}^{(t)} A_{j-1}^{(t)} \\ B_{j+1,j}^{(t)} A_{j+1}^{(t)} \\ \vdots \\ B_{n+t,j}^{(t)} A_{n+t}^{(t)} \end{pmatrix}.$$

So for the (n, k) E-MSR code we constructed,

$$A_j = A_{j+t}^{(t)} \begin{pmatrix} O \\ I_{k(n-k)} \end{pmatrix} = C_{j+t}^{(t)} \begin{pmatrix} B_{1,j+t}^{(t)} A_1^{(t)} \\ \vdots \\ B_{j+t-1,j+t}^{(t)} A_{j+t-1}^{(t)} \\ B_{j+t+1,j+t}^{(t)} A_{j+t+1}^{(t)} \\ \vdots \\ B_{n+t,j+t}^{(t)} A_{n+t}^{(t)} \end{pmatrix} \begin{pmatrix} O \\ I_{k(n-k)} \end{pmatrix} = C_{j+t}^{(t)} \begin{pmatrix} B_{1,j+t}^{(t)} O \\ \vdots \\ B_{t,j+t}^{(t)} O \\ B_{1+t,j+t}^{(t)} A_1 \\ \vdots \\ B_{j-1+t,j+t}^{(t)} A_{j-1} \\ B_{j+1+t,j+t}^{(t)} A_{j+1} \\ \vdots \\ B_{n+t,j+t}^{(t)} A_n \end{pmatrix}$$

$$= C_{j+t}^{(t)} \begin{pmatrix} O \\ B_{1,j} A_1 \\ \vdots \\ B_{j-1,j} A_{j-1} \\ B_{j+1,j} A_{j+1} \\ \vdots \\ B_{n,j} A_n \end{pmatrix} = C_{j+t}^{(t)} \begin{pmatrix} O \\ I_{n-1} \end{pmatrix} \begin{pmatrix} B_{1,j} A_1 \\ \vdots \\ B_{j-1,j} A_{j-1} \\ B_{j+1,j} A_{j+1} \\ \vdots \\ B_{n,j} A_n \end{pmatrix} = C_j \begin{pmatrix} B_{1,j} A_1 \\ \vdots \\ B_{j-1,j} A_{j-1} \\ B_{j+1,j} A_{j+1} \\ \vdots \\ B_{n,j} A_n \end{pmatrix}$$

4. THE SCALING PROCESS

The scaling process contains two parts: (1) systematic data transfer for the sake of load balancing; (2) updating of encoded blocks stored in redundancy nodes in order to make the system satisfy reconstruction property and exact repair property.

In part one, the systematic data stored in k nodes $N_1, \ldots, N_k$ are distributed evenly into $k + t$ nodes $N_1^{(t)}, \ldots, N_{k+t}^{(t)}$. Node N_i, i.e., node $N_{t+i}^{(t)}$ $(1 \le i \le k)$ transfers $A_i D_{k+j}$ to node $N_j^{(t)}$ $(1 \le j \le t)$. Marking $\begin{pmatrix} D_1 & \cdots & D_k \end{pmatrix}$ as D', so $D^{(t)}$ will be in the form of $D^{(t)} = \begin{pmatrix} D'' \\ D' \end{pmatrix}$, where D'' is a $t(n-k) \times k$ matrix transformed from $k(n-k) \times t$ matrix $\begin{pmatrix} D_{k+1} & \cdots & D_{k+t} \end{pmatrix}$. Before data transfer each node N_i $(1 \le i \le k)$ stores $k + t$ strips of data, while after data transfer each node $N_i^{(t)}$ $(1 \le i \le k + t)$ stores k strips. The total data transferred of this part is kt strips, i.e., $\frac{t}{k+t} M$.

In part two, each node $N_{t+k+1}^{(t)},\ldots,N_{t+n}^{(t)}$ needs to update the encoded blocks it stores. After the scaling process, for the $(n+t,k+t)$ E-MSR code, node $N_{i+k+t}^{(t)}$ $(1\le i\le n-k)$ stores encoded blocks $A_{i+k+t}^{(t)}D^{(t)}=A_{i+k+t}^{(t)}\begin{pmatrix}D''\\D'\end{pmatrix}$. While before the scaling process, node $N_{i+k+t}^{(t)}$, i.e., node N_{i+k} stores $A_{i+k}D$. The relations between encoding matrices $A_i,B_{i,j},C_i$ and $A_i^{(t)},B_{i,j}^{(t)},C_i^{(t)}$ are like which shown in equations (1), (2), and (3) above. Node $N_{i+k+t}^{(t)}$ stores $A_{i+k}D$, including $A_{i+k}D'=A_{i+k+t}^{(t)}\begin{pmatrix}O\\I_{k(n-k)}\end{pmatrix}D'=A_{i+k+t}^{(t)}\begin{pmatrix}O\\D'\end{pmatrix}$ which are parts of $A_{i+k+t}^{(t)}\begin{pmatrix}D''\\D'\end{pmatrix}$. So $N_{i+k+t}^{(t)}$ needs D'', i.e., $\begin{pmatrix}D_{k+1}&\cdots&D_{k+t}\end{pmatrix}$, or $A_{i+k+t}^{(t)}\begin{pmatrix}D''\\O\end{pmatrix}$ to accomplish the update.

Node N_{i+k+t} downloads kt strips of data $\begin{pmatrix}D_{k+1}&\cdots&D_{k+t}\end{pmatrix}$ from $N_{t+1}^{(t)},\ldots,N_{t+k}^{(t)}$ or from the t new nodes $N_1^{(t)},\ldots,N_t^{(t)}$ to get D''. Notice that N_{i+k+t} itself stores t encoded strips $A_{i+k+t}^{(t)}\begin{pmatrix}D_{k+1}&\cdots&D_{k+t}\end{pmatrix}$, so actually N_{i+k+t} needs to download only $(k-1)t$ strips to decode D''. So the total data transferred in this part is $\frac{(k-1)t(n-k)}{k(k+t)}M$. Total data transferred in both parts is $\frac{kt+(k-1)t(n-k)}{k(k+t)}M$.

When $t(k-1)>k$, the data transferred in part two can be reduced like this. First a node N_{i+k+t} downloads the data it needs and get D'' just like before. Then it can generate the blocks N_{j+k+t} $(j\ne i)$ needs, i.e., $A_{j+k+t}^{(t)}\begin{pmatrix}D''\\O\end{pmatrix}$. Each N_{j+k+t} can download k strips $A_{j+k+t}^{(t)}\begin{pmatrix}D''\\O\end{pmatrix}$ from N_{i+k+t} to update their encoded blocks instead of $(k-1)t$ strips. The data transferred in this method is $\frac{t(k-1)+k(n-k-1)}{k(k+t)}M$. Together with the first part is $\frac{kt+t(k-1)+k(n-k-1)}{k(k+t)}M$.

5. EXAMPLE

Now let's see an example of scaling a $(4,2)$ E-MSR code up to a $(5,3)$ E-MSR code. Here the encoding matrices of the $(5,3)$ E-MSR code are just like shown in Fig. 1. Mark the nodes and encoding matrices of the $(5,3)$ E-MSR as $N_i^{(t)},A_i^{(t)},B_{i,j}^{(t)},C_i^{(t)}$. The encoding matrices of the $(4,2)$ E-MSR code, i.e., $A_i,B_{i,j},C_i$ are constructed like the following.

$$\begin{pmatrix}A_1\\A_2\\A_3\\A_4\end{pmatrix}=\begin{pmatrix}A_1^{(t)}\\\vdots\\A_5^{(t)}\end{pmatrix}\begin{pmatrix}O\\I_4\end{pmatrix}_{6\times4}=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\\2&0&1&1\\1&2&1&2\\2&2&0&2\\0&2&1&1\end{pmatrix},B_{1,2}=B_{2,3}^{(t)}=\begin{pmatrix}2&1\end{pmatrix},$$

$$B_{3,2}=B_{4,3}^{(t)}=\begin{pmatrix}0&1\end{pmatrix},B_{4,2}=B_{5,3}^{(t)}=\begin{pmatrix}1&1\end{pmatrix},C_2=C_3^{(t)}\begin{pmatrix}O\\I_3\end{pmatrix}_{4\times3}=\begin{pmatrix}2&0&1\\1&2&1\end{pmatrix}.$$

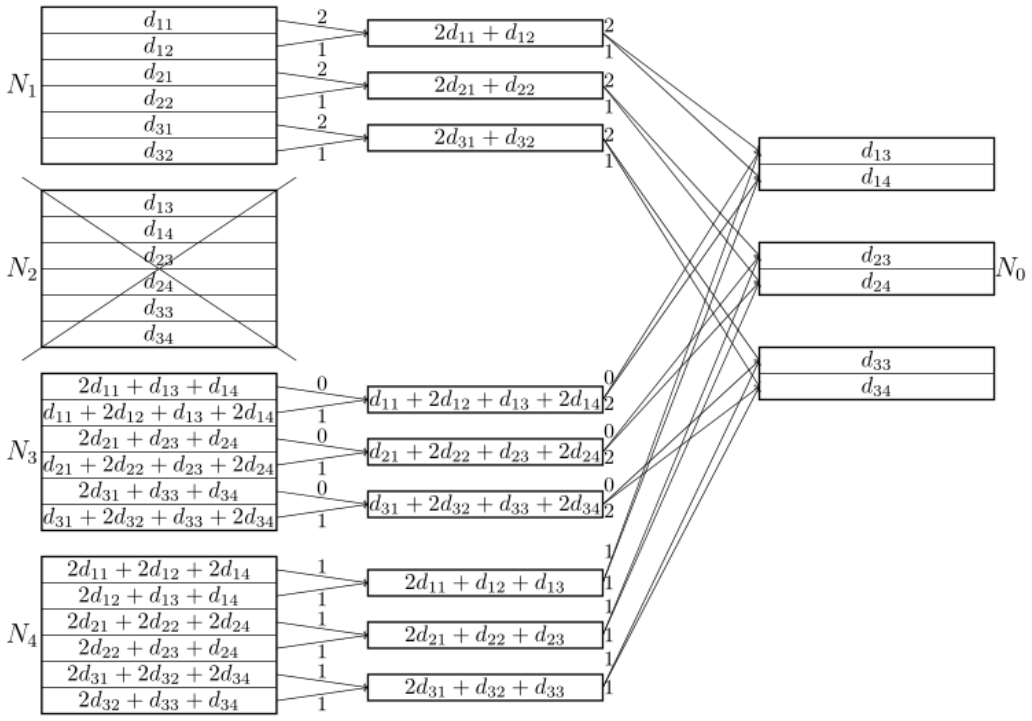

Figure 3. A $(4, 2)$ E-MSR code constructed from a $(5, 3)$ E-MSR code.

The original file data is divided into 12 blocks $d_{11}, \ldots, d_{34}$, constituting the data matrix

$$D = \begin{pmatrix} D_1 & D_2 & D_3 \end{pmatrix} = \begin{pmatrix} d_{11} & d_{21} & d_{31} \\ d_{12} & d_{22} & d_{32} \\ d_{13} & d_{23} & d_{33} \\ d_{14} & d_{24} & d_{34} \end{pmatrix}.$$

As is shown in Fig. 3, each node N_i stores 3 strips of data A_iD_1, A_iD_2, A_iD_3. A user connecting to any 2 of these 4 nodes can decode the 12 blocks of original file data. When a node failed (in the figure it is N_2), a new node N_0 is used to replace the failed one. Each node $N_i (i = 1, 3, 4)$ generates one encoded block per strip using $B_{i,2}$ and transfers these blocks to N_0. Then N_0 encodes the blocks received using C_2 to get exactly the data N_2 stored.

Fig. 4 is the $(5, 3)$ E-MSR code after scaling. Denote the 4 old nodes $N_1, \ldots, N_4$ as $N_2^{(t)}, \ldots, N_5^{(t)}$, and the new node as $N_1^{(t)}$. $N_1^{(t)}$ downloads d_{31}, d_{32} from $N_2^{(t)}$ and downloads d_{33}, d_{34} from $N_3^{(t)}$. So the data matrix of the $(5, 3)$ E-MSR code is

$$D^{(t)} = \begin{pmatrix} d_{31} & d_{33} \\ d_{32} & d_{34} \\ d_{11} & d_{21} \\ d_{12} & d_{22} \\ d_{13} & d_{23} \\ d_{14} & d_{24} \end{pmatrix} = \begin{pmatrix} D'' \\ D' \end{pmatrix}.$$

$N_4^{(t)}$ needs $D'' = \begin{pmatrix} d_{31} & d_{33} \\ d_{32} & d_{34} \end{pmatrix}$ to update the encoded blocks it stores. It downloads d_{31}, d_{32} from $N_2^{(t)}$ and decode d_{33}, d_{34} using the 2 encoded blocks $2d_{31} + d_{33} + d_{34}, d_{31} + 2d_{32} + d_{33} + 2d_{34}$ it originally stores. Then $N_4^{(t)}$ can update the encoded blocks it stores. $N_5^{(t)}$ is similar with $N_4^{(t)}$. The total data transferred in both parts is 8 blocks, i.e., $\frac{2}{3}M$, much less than the example in Fig. 2.

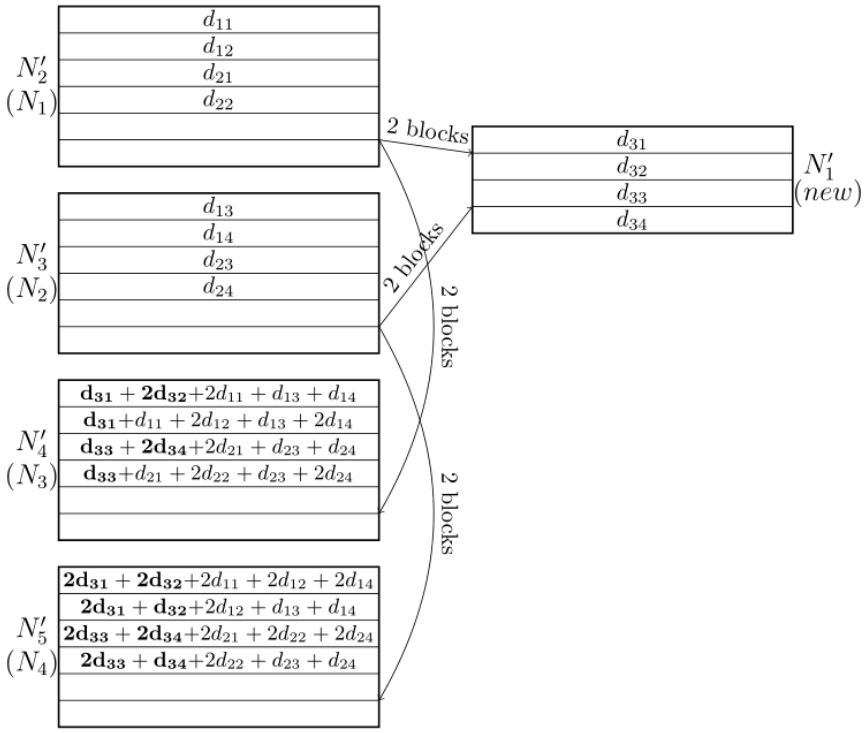

Figure 4. $(5, 3)$ E-MSR code scaling up from the $(4, 2)$ E-MSR code.

Another way of data transfer is after decoding d_{33}, d_{34}, $N_4^{(t)}$ generates the 4 blocks which $N_5^{(t)}$ needs, i.e., $2d_{31} + 2d_{32}, 2d_{31} + d_{32}, 2d_{33} + 2d_{34}, 2d_{33} + d_{34}$, and transfers these 4 blocks to $N_5^{(t)}$. When $t(k-1) > k$, this method is with lower bandwidth cost.

6. SCALING UP TO OTHER SCALES

In section 3, when designing the encoding matrices, we suppose it is known that the (n, k) E-MSR code will be later scaled up to an $(n + t, k + t)$ E-MSR code. Such condition may be hard to achieve in a real system. In this section we show that the (n, k) E-MSR code can be scaled up to any $(n + s, k + s)$ E-MSR code where $0 < s < t$. So when constructing (n, k) from $(n + t, k + t)$ E-MSR code, if it is not known exactly what scale will scaling up to in the future, we can just use a big t. Then when the (n, k) E-MSR code needs scaling, any $(n + s, k + s)$ E-MSR code that $0 < s < t$ can be easily achieved.

When constructing (n, k) E-MSR code from $(n + t, k + t)$ E-MSR code, the only constraint is $t > 0$. So we can construct any $(n + t - i, k + t - i)$ E-MSR code from $(n + t, k + t)$ E-MSR code where $0 < i < k + t$. Considering $n' = n + t - i$, $k' = k + t - i$, and $t' = i$, the problem

is converted to a problem of constructing (n', k') E-MSR code from $(n'+t', k'+t')$ E-MSR code, as is discussed before. Since $0 < s < t$, we can construct an $(n+s, k+s)$ E-MSR code from $(n+t, k+t)$ E-MSR code. Then we can construct an (n, k) E-MSR code from this $(n+s, k+s)$ E-MSR code. Such constructed (n, k) E-MSR code can be scaled up to the $(n+s, k+s)$ E-MSR code, just like scaling up to $(n+t, k+t)$ E-MSR code, which discussed before.

Denote the encoding matrices of the $(n+s, k+s)$ E-MSR code as $A_i^{(s)}, B_{i,j}^{(s)}, C_i^{(s)}$, and encoding matrices of the (n, k) E-MSR code constructed from $(n+s, k+s)$ as $A'_i, B'_{i,j}, C'_i$. $A'_i = A_{i+s}^{(s)} \begin{pmatrix} O \\ I_{k(n-k)} \end{pmatrix}_{(k+s)(n-k)\times k(n-k)} = A_{i+t}^{(t)} \begin{pmatrix} O \\ I_{(k+s)(n-k)} \end{pmatrix}_{(k+t)(n-k)\times(k+s)(n-k)} \begin{pmatrix} O \\ I_{k(n-k)} \end{pmatrix}$ $= A_{i+t}^{(t)} \begin{pmatrix} O \\ I_{k(n-k)} \end{pmatrix} = A_i$. Similarly we can see that $B'_{i,j} = B_{i,j}$ and $C'_i = C_i$. That is to say, no matter what s is, the $(n+s, k+s)$ E-MSR code can be scaled up from the original (n, k) E-MSR code constructed from the $(n+t, k+t)$ E-MSR code. In other words, the (n, k) E-MSR code can be scaled up to $(n+s, k+s)$ for any s that $0 < s < t$.

7. CONCLUSION

This paper studied the scaling up problem of an E-MSR code based distributed storage system with fixed number of redundancy nodes. There are few previous works on scaling problem of regenerating codes based distributed storage systems. Our works bring forward a scheme of scaling an (n, k) E-MSR code up to an $(n+t, k+t)$ E-MSR code or any $(n+s, k+s)$ E-MSR code that $0 < s < t$. We generate the encoding matrices of an (n, k) E-MSR code from the encoding matrices of an $(n+t, k+t)$ E-MSR code. Through carefully designing of encoding matrices, the changes of encoded blocks when scaling are minimized, so the distributed storage system can be scaled up with relatively low bandwidth cost and computation cost.

REFERENCES

[1] Erasure coding for distributed storage wiki. URL: csi.usc.edu/?dimakis/StorageWiki/doku.php?id=start.

[2] Daniel Cullina, Alexandros G. Dimakis, and Tracey Ho. Searching for minimum storage regenerating codes. arXiv:0910.2245, October 2009. IEEE Intl Symp. on Information Theory (ISIT), Seoul, Korea, June 2009.

[3] A.G. Dimakis, P.B. Godfrey, Yunnan Wu, M.J. Wainwright, and K. Ramchandran. Network coding for distributed storage system- s. Information Theory, IEEE Transactions on, 56(9):4539 –4551, September 2010.

[4] Yuchong Hu, Yinlong Xu, Xiaozhao Wang, Cheng Zhan, and Pei Li. Cooperative recovery of distributed storage systems from multiple losses with network coding. Selected Areas in Communications, IEEE Journal on, 28(2):268 –276, February 2010.

[5] S. Pawar, N. Noorshams, S. El Rouayheb, and K. Ramchandran. DRESS codes for the storage cloud: Simple randomized constructions. In Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on, pages 2338 –2342, August 2011.

[6] K. V. Rashmi, Nihar B. Shah, and P. Vijay Kumar. Optimal Exact- Regenerating codes for distributed storage at the MSR and MBR points via a Product-Matrix construction. arXiv:1005.4178, May 2010. IEEE Transactions on Information Theory, vol. 57, no. 8, pp. 5227 - 5239, August 2011.

[7] K.V. Rashmi, N.B. Shah, P.V. Kumar, and K. Ramchandran. Explicit construction of optimal exact regenerating codes for distributed stor- age. In Communication, Control, and Computing, 2009. Allerton 2009. 47th Annual Allerton Conference on, pages 1243 –1249, October 2009.

[8] Kenneth W. Shum and Yuchong Hu. Exact Minimum-Repair- Bandwidth cooperative regenerating codes for distributed storage systems. arXiv:1102.1609, February 2011. Presented in IEEE Int. Symp. on Inform. Theory (ISIT) 2011.

[9] Yunnan Wu and A.G. Dimakis. Reducing repair traffic for erasure coding-based storage via interference alignment. In Information Theory, 2009. ISIT 2009. IEEE International Symposium on, pages 2276 –2280, July 2009.

[10] Guangyan Zhang, Jiwu Shu, Wei Xue, and Weimin Zheng. SLAS: an efficient approach to scaling round-robin striped volumes. Trans. Storage, 3(1), 2007.

[11] Guangyan Zhang, Weiman Zheng, and Jiwu Shu. ALV: a new data redistribution approach to RAID-5 scaling. Computers, IEEE Transactions on, 59(3):345 –357, March 2010.

[12] Weimin Zheng and Guangyan Zhang. FastScale: accelerate RAID scaling by minimizing data migration. In Proceedings of the 9th USENIX conference on File and stroage technologies, FAST'11, page 1111, Berkeley, CA, USA, 2011. USENIX Association.