2014 9th IEEE International Conference on Networking, Architecture, and Storage

MCRTREE: A Mutually Cooperative Recovery Scheme for Multiple Losses in Distributed Storage Systems Based on Tree Structure

Xiaoqiang Pei, Yijie Wang, Xingkong Ma, Yongquan Fu, Fangliang Xu
Science and Technology on Parallel and Distributed Processing Laboratory
College of Computer, National University of Defense Technology
Changsha, Hunan, P. R. China, 410073
Email: {xiaoqiangpei, wangyijie, maxingkong, yongquanf, xufangliang} @nudt.edu.cn

***Abstract*—To guarantee the reliability of distributed storage systems, erasure coding, as a redundant scheme, has received increasingly attention because it can greatly improve the space efficiency compared with the replica schemes. However, it takes a long time and consumes a lot of network bandwidth for erasure coding to repair the lost data on failed nodes. The state-of-art studies focus on the repairing optimization for the single-node-failure context. Real-world experiments have clearly shown that multi-node failures indeed happen in cloud storage systems. Borrowing single-node repairing techniques to the multi-node setting faces challenges on the efficiency.**

We propose a mutually cooperative recovery scheme MCRTREE based on the tree structure for multiple node failures. MCRTREE improves the bandwidth utilization and reduces the repair time by the construction of regeneration trees between each new node (denoted as newcomers) and alive nodes (denoted as providers). Further, MCRTREE reduces the size of the data volumes to be transmitted for the repair process. Numerical experiments show that MCRTREE consumes less storage cost and the maintenance bandwidth compared with other redundancy recovery schemes. Trace-driven simulation results reveal that the MCRTREE reduces the regeneration time by $30\% - 50\%$, improves the successful regeneration probability by $10\% - 20\%$ and the data availability by $10\% - 20\%$ compared with the typical repair schemes.

***Keywords*-Distributed Storage System; Erasure Codes; Replica; Regeneration Tree**

I. Introduction

Distributed storage systems store data in a large number of storage nodes, either in the context of data centers in cloud computing systems, or in the context of peer-assisted online storage systems e.g.,[1] [2] [3], such as GFS [4] and HDFS [5]. Due to the inherent lack of reliability caused by node departures and hardware failures, data may become temporarily or permanently unavailable in such systems. In fact, large data centers are designed to treat storage node failures as rule, not the exception [6] [7] [8].

In order to provide high data reliability, distributed systems usually adopt redundant schemes, mainly including replica and erasure codes. The replication scheme stores replicas of the data in different nodes. While the erasure codes firstly divide the original data into k blocks, and then encode them into n blocks, where $n > k$, referred to as (n, k). During the reading process, the system with the replication scheme only needs to read any one of the replicas, while the system with erasure codes needs to read any k blocks out of n blocks. Compared with replica, erasure codes provide higher data availability with the same storage cost. However, erasure codes incur more network traffic and cause longer time than replica when repairing the failed data blocks [9] [10].

To reduce the repair cost, Dimakis et al. [11] [12] introduce regenerating codes, which combines the network coding and erasure coding. There are two typical regenerating codes, one is minimum-storage regenerating (MSR) codes, which minimizes the repair bandwidth while keeping the same storage cost with erasure codes. The other is minimum-bandwidth regenerating (MBR) codes, which minimizes the repair bandwidth at the cost of larger storage cost.

The regeneration schemes mentioned above focused on how to generate redundant data to reduce the regeneration traffic, but the bandwidth capacity between nodes has not been taken into account. Jun Li etc.[13] propose the Tree-structured Data Regeneration with Regenerating Codes (RCTREE for short), where the data is transferred from providers to the newcomer through a regeneration tree to accelerate the data transmission and improve the repair efficiency.

To the best of our knowledge, the above redundancy recovery mechanisms are designed for one node failure. However, there are many situations with more than one node failures. For example, some systems like Total Recall [14] reconstruct fragments with lazy repair policy, where a recovery is triggered only when the total amount of losses reaches a given threshold. Besides lazy repair policy, there are many other situations where multiple nodes fail at the same time in real storage systems, such as churn or break down of a cluster, i.e., a large percent of nodes often join and/or leave the network simultaneously.

Borrowing single-node repairing techniques to the multi-node setting faces challenges on the efficiency. For solving the multiple node failure situation, Yuchong Hu etc. [15] design a mutually cooperative recovery (MCR) mechanism. MCR reduces the repair bandwidth cost by exchanging data between newcomers. However, the star-structure MCR adopts could not make good use of the network bandwidth,

978-1-4799-4087-5/14 $31.00 © 2014 IEEE
DOI 10.1109/NAS.2014.33

and prolongs the repair time.

In this paper we present a mutually cooperative the recovery scheme based on tree structure (named MCRTREE) mechanism for multiple losses. Different from RCTREE where all the newcomers repair the lost data one by one, MCRTREE repairs the lost data simultaneously on multiple nodes cooperatively and allows the data exchanges among multiple newcomers. Suppose there are r node failures, RCTREE could only access $n-r$ surviving nodes, while each newcomer in MCRTREE could both access data from $n-r$ surviving nodes and other $r-1$ newcomers. Different from MCR, MCRTREE constructs a spanning tree for each newcomer, and repairs r failed nodes simultaneously, which makes good use of the higher bandwidth of network and markedly reduce the repair time. With the extensive analysis and quantitative evaluations based on the traces collected on the PlanetLab [16], we are able to show that MCRTREE reduces the recovery time for multiple losses and improves the reliability of the system.

The reminder of the paper is organized as follows. We describe the related works of repair in Section II and introduce the basic principle of MCRTREE with an illustrative example in Section III. Further, we introduce network model, and present detailed analysis of MCRTREE in Section IV. While Section V analyzes the performance of MCRTREE through theoretical and numerical analysis. Finally, Section VI concludes this paper.

II. Related Work

Reduction of the repair cost of erasure coding has been attracted much attentions. The typical recovery schemes mainly include the regenerating codes proposed by Dimakis et al. [11] [12], and the optimization of the bandwidth capacity utilization between nodes from Jun li etc. [17] [13].

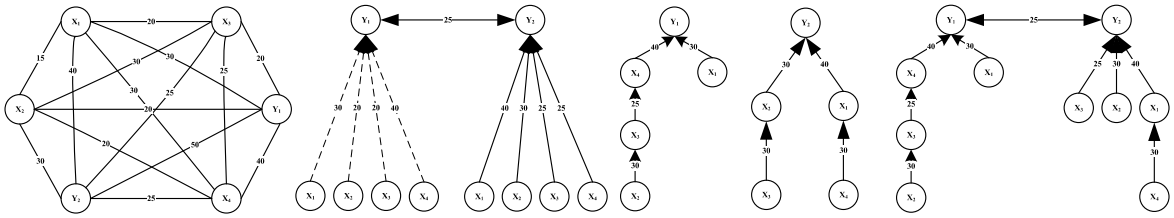
Dimakis et al. [11] [12] prove that the file reconstruction problem in distributed storage systems is equivalent to the multicasting problem. They also show that it will obviously reduce the recovery bandwidth overhead if more than k providers participate into the repair process. For example, they propose one symmetric mechanism, named minimum-storage regenerating (MSR) codes. In the special case of the $(n,k)=(14,7)$, if the newcomer accesses $k=7$ providers, the recovery overhead is M, the same as the original data. However, if the newcomer could access more than $k=7$ providers, such as $n-1=13$ providers, the newcomer only needs to download $0.625M$ to repair a new fragment. They generally call it Regenerating codes (RC for short). Wu et al. [18] showed further analysis of the tradeoff between storage overhead and repair bandwidth, and proposed a construction method of Regenerating Codes.

Yuchong Hu etc. [15] adopt the regenerating codes and design a mutually cooperative recovery (MCR) mechanism. In MCR all the newcomers repair the lost data cooperatively and simultaneously. Not only there are data flows between providers and newcomer, but there are data flows between multiple newcomers. MCR significantly reduces the data volumes to be transmitted, which decreases the recovery time a lot. However, in MCR newcomers and the providers compose a star structure, and the recovery time is restricted by the bottleneck bandwidth between newcomer and providers. Additionally, multiple newcomers come into a full connected network, where there is one link between any two newcomers. MCR could not make good use of the higher available bandwidth in the network.

Traditionally, the erasure-code storage systems use the star-structure recovery scheme, where the newcomer receives data from the providers directly, and constitute a star structure with providers. However, the star-structure recovery scheme can not make good use of the bandwidth even there are links with higher bandwidth in the network, and the recovery time is restricted by the bottleneck bandwidth between the newcomer and providers. For better using the higher bandwidth in the network, Jun Li etc.[17] introduce a tree-structure recovery scheme, where the newcomer and the providers come into a spanning tree, with newcomer as the root and providers as the child nodes. During the recovery process in a tree-structure recovery scheme, the leaf nodes transmit stored data to their parent node, and the parent nodes encode received data with their own data into new blocks, which are transmitted to their parent nodes, until the encoded data reaches the root newcomer. Based on this study, they propose the Tree-structured Data Regeneration with Regenerating Codes (RCTREE for short)[13]. RCTREE combined the advantage of regenerating codes with a tree-structured regeneration topology. By accessing more than k providers, RCTREE constructs a spanning tree with the newcomer as root and $d(d \geq k)$ providers as child nodes. However, the focus of this recovery scheme is repairing the single node failure instead of multiple losses in distributed storage systems.

III. A Motivating Example

We now introduce an illustrative example of data regeneration in the distributed storage system in Fig. 1. Fig. 1(a) shows the network model, which consists of six storage nodes, denoted by X_1, X_2, X_3, X_4, Y_1 and Y_2. We set varying the bandwidth capacity of the link. We assume that the redundancy is coded by a $(6,3)$ MDS code, stored in X_1, X_2, X_3, X_4, and two storage nodes that failed. Each storage node stores a coded block of $\frac{M}{3}$ bits, if the size of the original data is M bits. In order to regenerate the lost redundant nodes, Y_1 and Y_2 are selected to be the newcomers. Since $(6,3)$ MDS code is used, Y_1 and Y_2 need to receive data from at least three providers. If more than k providers are used as providers in the regeneration, regenerating codes [11][18] provide a way to reduce the bandwidth usage in the regeneration.



(a) network model with two newcomers and four storage nodes

(b) MCR with X_1, X_2, X_3 , X_4 and Y_i as providers

(c) RCTREE with X_1, X_2, X_3 and X_4 as providers

(d) MCRTREE with X_1, X_2, X_3 , X_4 and Y_i as providers

Figure 1: Examples of Three Regeneration Schemes: MCR, RCTREE and MCRTREE

Fig. 1(b) - Fig .1(d) show illustrations of three regeneration schemes. For the mutually cooperative recovery (MCR) scheme in Fig. 1(b), the two newcomers Y_1 and Y_2 receive data directly from the four providers X_1, X_2, X_3 and X_4. Additionally, in MCR scheme newcomers not only receive data from providers, but also exchange data with each other, illustrated by the edges between Y_1 and Y_2 in Fig. 1(b). For an (n, k) MCR code, each storage node stores $\frac{M}{k}$ bits and only $\frac{M}{k(n-k)}$ bits are transmitted on each link in the regeneration. The time spent on regenerating r new coded blocks is $\frac{\frac{M}{k(n-k)}}{\min_{1\leq i\leq r} B_i}$, when there are r failed nodes. MCR costs $\frac{M/9}{20Mbps}$ seconds to accomplish the regeneration for Y_1 and Y_2. We ignore the encoding time of MDS code because the processors usually perform encoding operations much faster than the network transmission, and the encoding can be performed simultaneously with the transmission.

MCR, however, suffers from the bottleneck links between newcomer and providers, such as the links of (X_2, Y_1) and (X_3, Y_2). If we consider the links between providers, we can utilize links with higher bandwidth to bypass the slow bottleneck links in MCR. In Fig. 1(c), we show an example of the tree-structured regeneration with regenerating codes(RCTREE), which could make good use of the links with higher bandwidth. For there are r node failures, there are at most $n - r$ surviving nodes in the system, so the data transmitted between nodes equals $\frac{M}{k(n-r-k+1)}$. During the regeneration process, RCTREE will construct a spanning tree for each newcomer with X_1, X_2, X_3 and X_4 as its child nodes. For the construction of Y_1, X_3 receives data from X_2, encodes the received data with the data it stores, and sends the encoded data to X_4. After receiving the data from X_3, X_4 does the same instruction as X_3, Finally, Y_1 receives data from both X_4 and X_1 and then encodes into on new redundant block. By streamlining the relay on X_3 and X_4, X_3 and X_4 encode the data byte-by-byte rather than after receiving the whole block. The regeneration time of Y_1 will be bottlenecked by the link between X_3 and X_4, and thus the regeneration time of Y_1 is $\frac{M/6}{25}$. However, in the situation with multiple node failures, RCTREE will regenerate the failed nodes one by one, therefore the time for regenerating Y_1 and Y_2 is $\frac{M/6}{25} + \frac{M/6}{30}$.

RCTREE constructs spanning trees for newcomers to lower construct time by utilizing the higher bandwidth of the network. However, RCTREE extends the regeneration time for two reasons. Firstly, RCTREE could at most visit $n-r$ providers when there are r node failures, which limits the data volume to be transmitted, thus the data transmitted between nodes is $\frac{M}{k(n-r-k+1)}$. Secondly, RCTREE constructs spanning trees for multiple newcomers one by one, which could not make good use of the parallelism between spanning trees. How could we find a way to efficiently utilize the bandwidth between multiple newcomers? How could we consider the parallelism between multiple spanning trees?

In Fig. 1(d), we present MCRTREE, which constructs two regeneration trees with X_1, X_2, X_3 and X_4 as providers for Y_1 and Y_2. MCRTREE combines the advantages of regeneration codes and tree-structured regeneration. During the reconstruction process, MCRTREE accesses more than k providers, which reduces the data transmitted during the recovery. While MCRTREE constructs one spanning tree for each newcomer, and repairs multiple newcomers simultaneously, which could efficiently utilize the parallelism between spanning trees. As result, the regeneration time can be further reduced to $\frac{M/9}{25Mbps}$ seconds for constructing Y_1 and Y_2. Compared with MCR in Fig. 1(b), the regeneration time is reduced by 20%, while compared with RCTREE in Fig. 1(c), the regeneration time is reduced by 63.64%.

IV. Mutually Cooperative Recovery Scheme Based on Tree Structure

In this section, we present an in-depth analysis of MCRTREE. We first introduce our network model for the regeneration process in distributed storage systems. We then will analyze how to construct the regeneration trees for multiple newcomers. Finally, we will present the assumptions of MCRTREE, and analyze the bottleneck bandwidth of multiple regeneration trees.

A. Network Model for MCRTREE

The redundant data is produced by an (n, k) MDS code, which divides the original file into k blocks $F_1, F_2, ..., F_k$,

and encodes them into n coded blocks $X_1, X_2, ..., X_n$. These n coded blocks are discretely stored in n storage nodes $V_1, V_2, ..., V_n$, with one coded block in one node. Without loss of generality, we assume X_i is stored in V_i, and r nodes(referred to be $X_{n-r-1}, X_{n-r}, ..., X_n$ get lost. To ensure the redundancy of the system, another r newcomers should be regenerated. Assume there are d active nodes for regeneration, and to maintain the MDS property, the newcomer should access at least k active nodes, so we have $k \leq d \leq n-r$. Assume (V_i, V_j) be the undirected link between two nodes V_i and V_j, and $\omega(V_i, V_j)$ be the bandwidth capacity of V_i and V_j. We denote the above presentation as $MCRTREE(n, k, r, d)$.

To be specific, the model *MCRTREE*(n, k, r, d) contains three detailed steps, illustrated in Fig. 2, [19] for reference.

- At the beginning, the original data in the source(noted as Src in Fig. 2) is divided into k blocks, which are encoded into n blocks. And an initial set of n nodes $X_1, X_2, ..., X_n$ are chosen to store the n blocks. The destination (noted as Des in Fig. 2) could download any k blocks to construct the original data.
- Assume there are r node failures in the system, referred to be $X_{n-r-1}, X_{n-r}, ..., X_n$), leaving $n-r$ active nodes in the system, in order to maintain the normal service ability of the system, the reconstruction of the failed nodes should be triggered when $r \geq (n-k)$. The middle of Fig. 2 illustrates the above situation.
- In order to maintain the same level of redundancy, the system selects another r new nodes (denoted as $Y_1, Y_2, ..., Y_r$) to reconstruct the lost nodes. After repairing the lost nodes, the Des could connect to any k nodes out of $X_1, X_2, ..., X_{n-r}, Y_1, Y_2, ..., Y_r$ to download k blocks for the data reconstruction, which is illustrated in the right of Fig. 2.

We next describe how to construct r parallel regeneration trees for the lost nodes. And then we will analyze the assumptions of the MCRTREE and how to obtain the bottleneck bandwidth of the r regeneration trees based on MCRTREE.

B. Construction of Regeneration Trees

In this section, we present MCRTREE regeneration algorithm, based on the network model above. Firstly, we show how the MCRTREE regenerates spanning trees for newcomers. And then we give the encoding scheme for linear coding.

1) Regeneration Algorithm:

Definition 1: In *MCRTREE*(n, k, r, d), an regeneration tree (noted as T) is a spanning tree, where the root is $Y_i, i = 1, 2, ..., r$, and it covers $n-r$ providers as its child nodes.

Lemma 1: For each regeneration tree T in *MCRTREE*(n, k, r, d), the regeneration time depends on the edge with the minimal bandwidth. For r regeneration trees $T_1, T_2, ..., T_r$, the regeneration time depends on the minimal bandwidth in all over the r regeneration trees.

Proof: We have known that the weight of each edge in T, $\omega(V_i, V_j), i, j = 0, 1, 2, ..., n-1, i < j$, denotes the bandwidth capacity between V_i and V_j. The traffic on each edge in T is uniform. If the node encodes and sends data after it receives all the data from its children, it will waste a substantial amount of time for waiting. The optimal transmission method is to use the principle of pipelining. The node encodes and sends data to its parent node immediately

Algorithm 1 Construct r optimal regeneration trees in *MCRTREE*(n, k, r, d), $0 \leq r \leq (n-k)$

Require:
T_i: *the ith regeneration tree*
n: *the total node number*
r: *the failed node number*
$root_{ij}$: *the edge between root of ith regeneration tree and jth node the root connects to*
$edge(i, j)$: *the edge between* $node_i$ *and* $node_j$

1: **for** every T_i in T **do**
2: $T_i \leftarrow \phi$
3: **end for**
4: **for** $i \leftarrow 1$ to r **do**
5: **for** $j \leftarrow 1$ to $n-r$ **do**
6: $e_{ij} \leftarrow$ the edge with maximum bandwidth in $root_i$
7: $T_i \leftarrow T_i \bigcup e_{ij}$
8: **end for**
9: **end for**
10: **for** $i \leftarrow 1$ to r **do**
11: **for** $j \leftarrow 2$ to $n-r-1$ **do**
12: **for** $g \leftarrow j+1$ to $n-r$ **do**
13: **if** $root_{ij}$ not in T_i && $Max(root_i) >$ *bandwidth of* $edge(j, g)$ **then**
14: $T_i \leftarrow T_i \bigcup root_{ij}$
15: **else**
16: search next $edge(j, g)$ with maximum bandwidth
17: $T_i \leftarrow T_i \bigcup edge(j, g)$
18: **end if**
19: **end for**
20: **end for**
21: Shuffle the bandwidth of network
22: **end for**
23: **for** $i \leftarrow 1$ to r **do**
24: **for** $j \leftarrow 1$ to r **do**
25: **if** i != j **then**
26: $e_{ij} \leftarrow$ edge between root of ith regeneration tree and root of jth regeneration tree
27: $T_i \leftarrow T_i \bigcup edge(i, j)$
28: **end if**
29: **end for**
30: **end for**

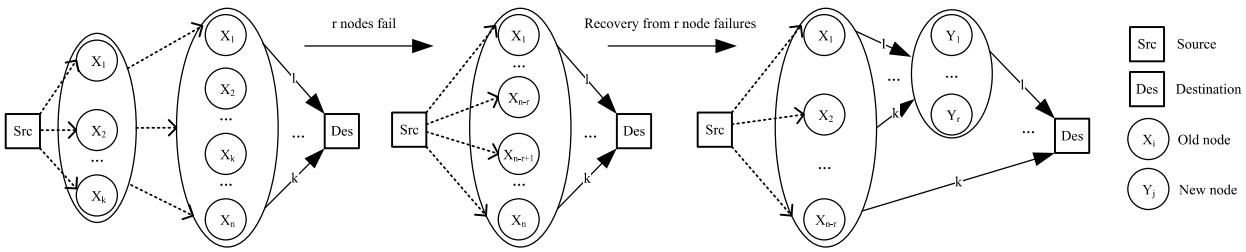


Figure 2: Network Model for MCRTREE

after it has received one byte/packet from all of its children. So the bandwidth bottleneck is the minimal edge in the regeneration tree.

Assume the bottleneck bandwidth of each regeneration tree T_i is ω_i, the regeneration time of r regeneration trees depends on the slowest regeneration tree, namely the smallest ω_i. So the regeneration time for constructing r newcomers depends on the edge with the minimal bandwidth in all over the r regeneration trees. ■

Lemma 2: Let $B(T_i), i = 1, 2, ..., r$ be the bottleneck bandwidth of the ith regeneration tree T_i in *MCRTREE*(n, k, r, d). Then the regeneration time of r newcomers is $\frac{\frac{M}{k(n-k)}}{\min_{1\leq i\leq r} B(T\_i)}$, where M is the size of the original file.

Proof: The amount of data to transferred on each edge in T_i is $\frac{M}{k(n-k)}$ bytes. From Lemma 1, we know that the regeneration time depends on the edge with the minimal bandwidth in all over the r regeneration trees. According to the definition of $B(T_i)$, the regeneration time is $\frac{\frac{M}{k(n-k)}}{\min_{1\leq i\leq r} B(T\_i)}$. ■

According to the Lemmas above, we know the data volume to be transmitted during the regeneration process amounts to $\frac{M}{k(n-k)}$ bytes, and the regeneration time depends on the bottleneck bandwidth of the r regeneration trees. Algorithm 1 shows how to construct r optimal regeneration trees, which contains three parts.

- At the beginning, the algorithm constructs r empty spanning trees, and then finds r newcomers as the roots of the regeneration trees, which will receive data from their children, and encodes the received data with its stored data into a new redundant block to replace the failed one.
- After finding roots for r regeneration trees, the algorithm begins to construct regeneration trees one by one. Firstly, it will traverse the network to find edges with the maximum available bandwidth between root and providers. Secondly, it will set the root and the new node joining in the tree as the new start to find new edges with the maximum available bandwidth between root and the other providers. The algorithm will continue to find the edges for constructing spanning tree until the construction of this tree completes. Thirdly, the algorithm will construct other $r-1$ regeneration trees in the same way until all the construction of r regeneration trees finishes.
- Finally, based on the r regeneration trees constructed above, the algorithm will link the roots of the r regeneration trees. At the end, each regeneration tree contains $n-1$ providers, including $n-r$ active nodes and other $r-1$ newcomers.

MCRTREE reduces regeneration time from two aspects, as illustrated in Algorithm 1. Firstly, any newcomer in MCRTREE could not only visit the other $n-r$ active nodes when there are r node failures, but visit the other $r-1$ newcomers, which could markedly reduce the data volume to be transmitted during the regeneration process. The other one is that MCRTREE constructs trees for each newcomer, which could bypass the low bandwidth between newcomer and providers, while make good use of the edges with high bandwidth. Therefore, MCRTREE not only could reduce the data volume to be transmitted, but could utilize the high bandwidth in the network to reduce the regeneration time.

Fig. 3 shows the construction process of the example showed in Fig. 1. The six steps correspond to the optimal regeneration tree in Algorithm 1. In Fig. 3, the solid lines present the construction process of newcomer Y_1, while the dotted lines correspond to the construction process of newcomer Y_2. After constructing two regeneration trees for Y_1 and Y_2 respectively, then the algorithm will add the link between newcomer Y_1 and newcomer Y_2, illustrated in Fig. 3(f). And the corresponding constructed regeneration trees are showed in Fig. 1(d).

2) Encoding Scheme: During the regeneration, the newcomers must receive data from one or more existing storage nodes to construct a new storage node. For (n, k) -MDS codes, the newcomers need to receive data from at least k providers. In *MCRTREE*(n, k, r, d), the newcomers receive data from $n-r$ providers and other $r-1$ newcomers. Assume that the data block stored in X_i is $B_i, i = 1, 2, ..., (n-r)$, and the corresponding coefficients are $c_i, i = 1, 2, ..., (n-r)$. While the data block combined in Y_j is $B'_j, j = 1, 2, ..., r$ and the corresponding coefficients are $c'_j, i = 1, 2, ..., r$.

In one regeneration tree, if node X_i does not receive data from other nodes, it sends $c_i B_i$ to its parent node. If X_i receives data from $X_{i_1}, X_{i_2}, ..., X_{i_i n(X_i)}$, where $in(X_i)$ is

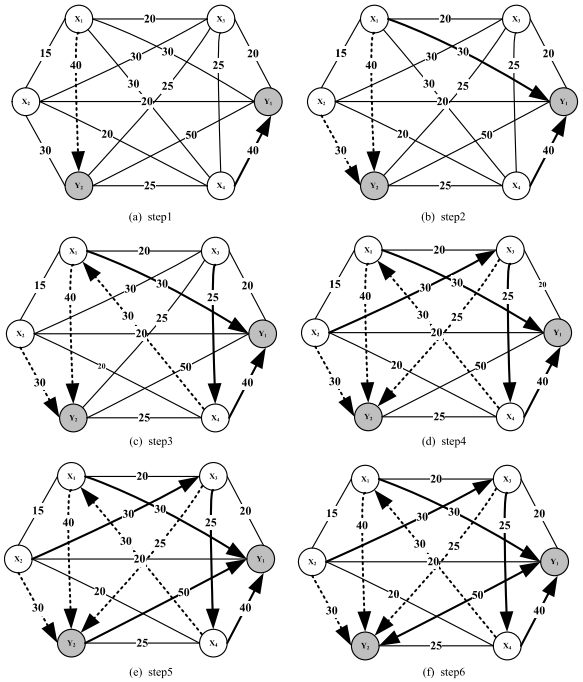


Figure 3: Regeneration Trees after 1st - 6th steps of Algorithm 1 on the network model in Fig. 1(a)

the indegree of X_i, assuming the data received from X_{i_j} is B_{i_j}, it will encode the received and its stored data into a new data block and then sends it to its parent node. Therefore, the data this node sends is $c_iB_i + \sum_{j=1}^{in(X_i)} B_{i_j}$. Other nodes in the regeneration tree follow the same way to encode and send data to its parent node, until reaches the root. As a result, the root could get $\sum_{i=1}^{n-r} c_iB_i$ from its children . After that, newcomers start to exchange data. The newcomer Y_j sends its encoded data block $c'_jB'_j$ to other newcomers. Finally, each newcomer encodes the data received both from its providers and other newcomers with its stored data. Therefore, the data each newcomer finally gets equals to $\sum_{i=1}^{n-r} c_iB_i + \sum_{j=1}^{r-1} c'_jB'_j$.

Fig. 4 illustrates the encoding scheme when there are two node failures Y_1 and Y_2. In Fig. 4, the bottom nodes represent the leaf nodes, which will send data to its parent nodes. While the non-leaf nodes showed in the middle of Fig. 4 firstly receive data from their children, and then encode the received data with its own data into a new data block, and sends it to their parent nodes until reaching the root, illustrated in the top of Fig. 4. Finally, the two newcomers exchange data and encode into a new data block.

C. Assumptions in MCRTREE

In Section IV-B2, we know that the data block stored in X_i is $B_i, i = 1, 2, ..., (n-r)$, and the data block combined in Y_j is $B'_j, j = 1, 2, ..., r$. Assume the data transmitted between newcomer and providers is $B_{ij}, i = 1, 2, ...r, j = 1, 2, ..., (n-r)$, the data transmitted between newcomers is $B'_{ij}, i = 1, 2, ...r, j = 1, \ldots, r$, and the data transmitted between providers is $B''_{ij}, i = 1, 2, ...(n-r), j = 1, 2, ..., (n-r)$.

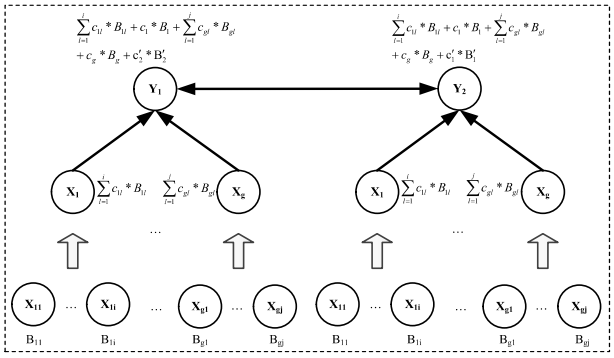


Figure 4: Encoding Scheme of Two Node Failures

From a practical view, a distributed storage network will very difficult to be implemented and managed if the data B_{ij}, B'_{ij} and B''_{ij} are not equal. So we assume that B_{ij}, B'_{ij} and B''_{ij} are the same as β. We know $\beta = \frac{M}{k(n-k)}$, where M is the size of original file.

D. Bottleneck Bandwidth of the r Regeneration Trees

From Lemma 1, we know that for r regeneration trees $T_1, T_2, ..., T_r$, the regeneration time depends on the edge with the minimal bandwidth in all over the r regeneration trees. In this section, we analyze the how to get the bottleneck bandwidth in r regeneration trees.

In a regeneration tree $T_i, i = 1, 2, ..., r$, there are n nodes, including $n-r$ providers and $r-1$ newcomers. There are $\frac{n(n-1)}{2}$ edges, and let $E = \{e_1, e_2, ..., \}$ be the edges in T_i, where $\omega(e_1) > \omega(e_2) > \cdots > \omega(\frac{n(n-1)}{2})$. The bandwidth each edge is assumed to be different from each other, as it realistically reflects real-world networks with high probability.

According to the order statistics lemmas in [17], [13], assume $X_1, X_2, ..., X_n$ are n independent random variables, for each of which the cumulative distribution function is $F(x)$ and the probability dendsity function is $f(x)$. Let $f_{(j:n)}(x)$ denote the probability density function of the jth variable $X_{(j:n)}, X_{(1:n)} \geq X_{(2:n)} \geq \cdots \geq X_{(n:n)}$. If X_i is with continuous distribution [17], [13],

$$f_{(j:n)}(x) = \frac{n!F^{n-r}(x)[1-F(x)]^{r-1}f(x)}{(n-r)!(r-1)!} \qquad (1)$$

Lemma 3: The edge with bottleneck bandwidth is the jth edge in T_i if and only if the minimal edge in maximum spanning tree T_i is the rth maximal edge of E.

We know that the lower bound of the data to be transferred during the regeneration process is equal to $\frac{M}{k(n-k)}$, when the

Table I: End-to-End bandwidth distribution in [16]

Capacity(C)(Mb/s)	Number of paths	Percentage(%)
$0.3 \leq C \leq 20$	6733	30.8
$20 \leq C \leq 50$	1910	8.74
$50 \leq C \leq 80$	1303	5096
$80 \leq C \leq 120$	11744	53.72
$120 \leq C \leq 200$	139	0.64
$200 \leq C \leq 500$	21	0.096
$500 \leq C \leq 682.9$	11	0.05

newcomer receives data from $n-r$ surviving providers and $r-1$ other newcomers. To be specific, there are $\frac{n(n-1)}{2}$ edges in T_i of *MCRTREE*(n,k,r,d). According to the Lemma 3, the edge with bottleneck bandwidth in T_i is the $(n-1)th$ maximum edge of T_i. Assume the edge with bottleneck bandwidth in T_i is B_i, the edge with bottleneck bandwidth in $T(T_1,T_2,...,T_r)$ is the edge with minimum bandwidth among B_i, namely $\min_{1\leq i\leq r} B_i$.

V. Evaluation

In this section, we will compare MCRTREE with other existing symmetric redundancy recovery mechanisms RCTREE and MCR in scenarios both from the real environment and the simulation that emulates the real-world distributed storage systems, based on the availability trace of PlanetLab and the bandwidth distribution measured in PlanetLab network.

Firstly, we compare the storage overhead and maintenance bandwidth of RCTREE, MCR and MCRTREE, which represent the storage cost and the bandwidth cost to repair the lost blocks; Secondly, we test the transfer time of RCTREE, MCR and MCRTREE when transmitting the same volume data in the real environment; Finally, we compare the three regeneration schemes from three aspects: (1) regeneration time: how much time it will take from the start of a regeneration to the end? (2) probability of the successful regeneration: the probability that a regeneration finishes successfully, not interrupted by the node departures; (3) data availability: the probability that a file is available.

Our event-driven simulator simulates the nodes' events based on the trace file of PlanetLab and bandwidth distribution measured in PlanetLab network (Measuring bandwidth between PlanetLab nodes). Lee et al. [16] showed the bandwidth distribution of PlanetLab, illustrated in Table I.

A. Storage Overhead And Maintenance Bandwidth

Consider the following situation. The original file (size M) is divided into k blocks, and then encoded into n blocks, which are stored in n storage nodes separately. Each node stores α. As the time goes by, there are r storage nodes failures, and then the multi-loss recovery scheme is triggered. During the regeneration, each of the r newcomers receive β bytes data from d providers. Therefore, the maintenance bandwidth $\gamma = d\beta$ bytes. Based on the scenario above, we analyze the storage cost and maintenance bandwidth of RCTREE, MCR and MCRTREE.

Table II: Storage Cost and Bandwidth Comparison of RCTREE, MCR and MCRTREE

	Total node storage cost	Maintenance bandwidth
RCTREE	$(\frac{M}{k})\cdot n$	$[\frac{n-r}{n-r-k+1}]\cdot(\frac{M}{k})\cdot r$
MCR	$(\frac{M}{k})\cdot n$	$[\frac{n-1}{n-k}]\cdot(\frac{M}{k})\cdot r$
MCRTREE	$(\frac{M}{k})\cdot n$	$[\frac{n-1}{n-k}]\cdot(\frac{M}{k})\cdot r$

Table III: $n=7,k=4,r=3$

	Total node storage cost	Maintenance bandwidth
RCTREE	1.75M	3M
MCR	1.75M	1.5M
MCRTREE	1.75M	1.5M

RCTREE: In [13], when using RCTREE, if a newcomer is allowed to access to d active providers, it needs to store $\frac{M}{k}$ and cost a traffic of $[\frac{d}{d-k+1}]\cdot(\frac{M}{k})$ to repair the data. So the storage cost is $(\frac{M}{k})\cdot n$, and the maintenance bandwidth is $[\frac{d}{d-k+1}]\cdot(\frac{M}{k})\cdot r$.

MCR: In [15], when using MCR, if a newcomer is allowed to access to d active providers, it needs to store $\frac{M}{k}$ and cost a traffic of $[\frac{d}{d-k+1}]\cdot(\frac{M}{k})$ to repair the data. So the storage cost is $(\frac{M}{k})\cdot n$, and the maintenance bandwidth is $[\frac{d}{d-k+1}]\cdot(\frac{M}{k})\cdot r$.

MCRTREE: From the above analysis in this paper, when using MCRTREE with $\alpha_{MCRTREE}=\frac{M}{k}, \gamma_{MCRTREE}=(n-1)r\beta$ and $\beta=\frac{M}{[k(n-k)]}$, the storage cost is $(\frac{M}{k})\cdot n$ and the maintenance bandwidth is $[\frac{n-1}{n-k}]\cdot(\frac{M}{k})\cdot r$.

It is proved that [18] we could get better tradeoff of storage cost and maintenance bandwidth if d increases. In RCTREE, when there are r storage node failures, the maximum d could be $n-r$, so we set d as $n-r$ for RCTREE. While for MCR, the maximum d could be $n-1$ for the newcomer could download data from both $n-r$ providers and other $r-1$ newcomers. So we set d as $n-1$ for MCR. Table II shows the storage costs and maintenance bandwidths of different redundancy recovery.

Compared with RCTREE, MCR and MCRTREE reduce 50% maintenance bandwidth while keeping the same storage cost with $n=7,k=4,r=3$, illustrated in Table III. Compared with RCTREE, MCR and MCRTREE reduce 58.3% maintenance bandwidth while keeping the same storage cost with $n=11,k=6,r=3$, illustrated in Table IV. While MCR and MCRTREE consume the same storage cost and maintenance bandwidth.

Table IV: $n=11,k=6,r=3$

	Total node storage cost	Maintenance bandwidth
RCTREE	1.375M	3M
MCR	1.375M	1.25M
MCRTREE	1.375M	1.25M

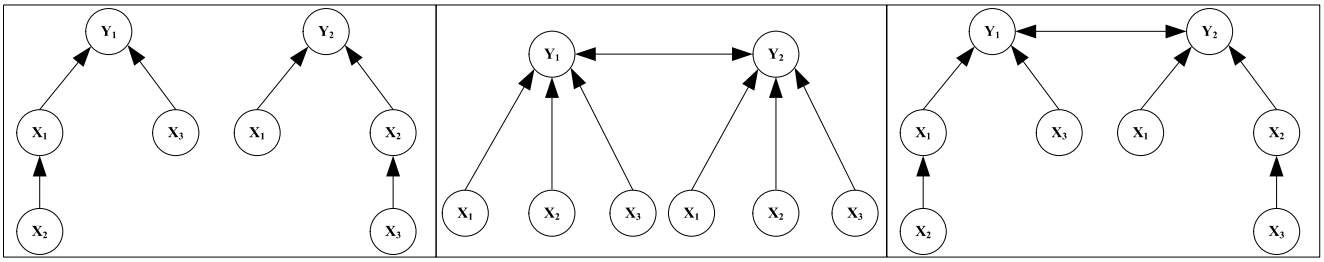


Figure 5: Transmission Time Example

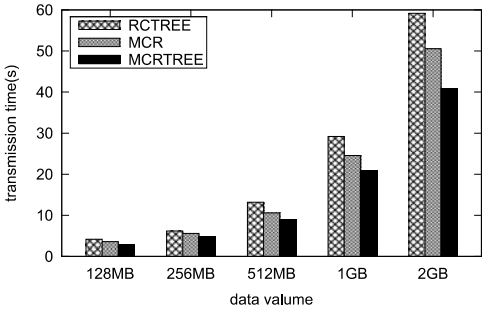


Figure 6: Transmission Time Comparisons of RCTREE, MCR and MCRTREE

B. Transmission Time

In order to test the transmission time of RCTREE, MCR and MCRTREE, we deployed 19 machines for experimentation, and each represents middle-range commodity hardware. Each server contains a 64-bit Intel processor with 24 cores, 16GB of RAM, and the servers are connected with 1GB switches. The operating system is Cent-OS, version 6.2. During the test, we transmit different size of data along with the topology for two newcomers with three providers. For RCTREE, we construct two regenerating trees and send the data along the tree from children to root serially, illustrated in Fig. 5(a), while for MCR, we construct the structure and transmit the data in a parallel way, illustrated in Fig. 5(b), and for MCRTREE, we construct two regenerating trees and transmit the data simultaneously, illustrated in Fig. 5(c).

The transmission time comparisons of RCTREE, MCR and MCRTREE under different data volume is illustrated Fig. 6, which shows that the transmission time becomes longer as the data volume increases, and RCTREE shows the longest transmission time and MCRTREE costs the least, while the transmission time of MCR is between them. The results reveal that MCRTREE performs the highest transmission efficiency.

C. Simulation

In this section, we compare MCRTREE with RCTREE and MCR from there aspects: regeneration time, probability of successful regeneration and data availability, which are analyzed in [20] and [21]. Our event-based simulator simulates the nodes' join and leave activities based on the PlanetLab trace, including the network bandwidth distribution between nodes, as illustrated in Table I. And the simulator lasts for a period of $Time$ days, and deals with the node-join/leave events. Each node in the simulator joins the network Ave_{up} times and leaves Ave_{down} times on average per day. We assume the data object is denoted by its size M and the simulator repeated for Rep_{num} times on $Node_{num}$ nodes.

Table V: Simulation parameters

$Time$	Ave_{down}	Ave_{up}	$Node_{num}$	Rep_{num}	M
1000	0.137	0.135	1000	100	$1GB$

1) ***Regeneration Time****:* How much time is spent from the start of a regeneration to the end? Which is one of the main metrics measuring the performance of recovery schemes.

2) ***Probability of Successful Regeneration****:* Probability of the successful regeneration. The probability that a regeneration finishes successfully, not interrupted by the node departures. The probability that data are available is the ratio of the available time to the total simulation time.

3) ***Data Availability****:* The probability that a file is available. For $(n,k)-$linear coding, data are available when there are at least k active storage nodes.

The parameters of the simulator are showed in Table V, where we configure 1000 nodes for simulation, lasts for 1000s and repeats 100 times. We adopt the $(n,k) = (k+5,k)$-random linear coding and set k as $2,4,\ldots,10$ and r from 2 and 3.

Regeneration time is defines as β/ω, where β is the data volume to be transmitted during the repair, and ω is the available bandwidth for repair. For RCTREE, $\beta = \frac{M}{k(n-r-k+1)}$, while for MCR and MCRTREE, $\beta = \frac{M}{k(n-k)}$. The available bandwidth ω of RCTREE is determined by the link with the minimum bandwidth in the spanning tree, and the available bandwidth ω of MCR is determined by the link with the minimum bandwidth between the providers and newcomer or between the newcomers, while the available bandwidth ω of MCRTREE is determined by the link with the minimum

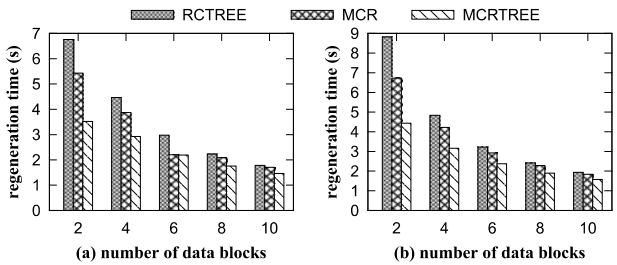


Figure 7: Regeneration Time of RCTREE, MCR and MCRTREE

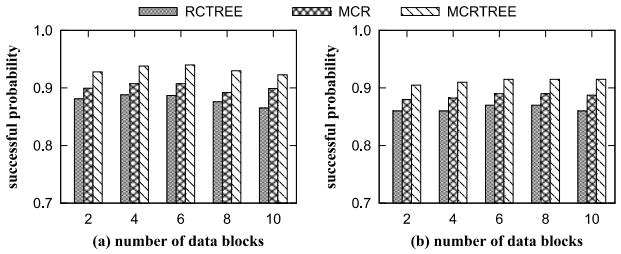


Figure 8: Probability of Successful Regeneration of RCTREE, MCR and MCRTREE

bandwidth of the spanning tree or between the newcomers.

Fig. 7 shows the regeneration time comparisons of the RCTREE, MCR and MCRTREE, where Fig. 7 (a) shows the regeneration time comparisons of 2 failures while Fig. 7 (b) shows the regeneration time comparisons of 3 failures. As the number of data blocks k increases, the regeneration time of the three repair schemes decrease, for bigger k reduces the data volume to be transmitted for repairing. Compared with RCTREE and MCR, MCRTREE reduces the regeneration time significantly, and reduces about 49% and 35% respectively when the failure number equals to 2. In fact, as the number of failures increases, the regeneration time increases for RCTREE, MCR and MCRTREE, as illustrated in Fig. 7 (b), while compared with RCTREE and MCR, the regeneration time of MCRTREE reduces 50% and 34% respectively.

The probability of successful regeneration is the ratio of the number of blocks regenerated successfully and the total number of failed blocks. Different repair schemes brings different probabilities of the regeneration. The repair scheme with higher probability brings better repair performance, so the higher the probability is, the quicker the repair scheme is. Fig. 8 illustrates the probability comparisons of RCTREE, MCR and MCRTREE, and Fig. 8(a) shows the successful regeneration probability when there are 2 failed blocks, while Fig. 8(b) shows the the successful regeneration probability when there are 3 failed blocks. Fig. 8 tells us that as the number of data blocks k increases, the regeneration time decreases, while more providers participating into the

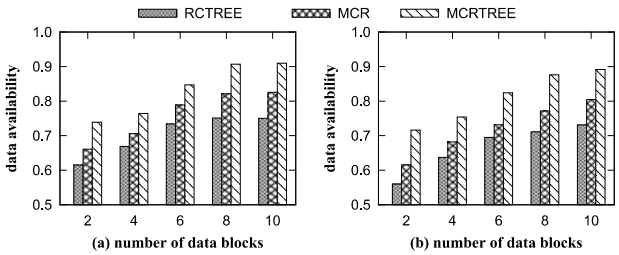


Figure 9: Data Availability of RCTREE, MCR and MCRTREE

regeneration will bring higher probability of failure. Thus, the probabilities of successful regeneration do not change a lot.

For a $(n, k)-$linear code, data are available when there are at least k active storage nodes. As the nodes show the same probabilities to loss, the regeneration could be interrupted by node failures. So it is possible that a data object becomes invalid if the active block number is smaller than k. Fig. 9 (a) and Fig. 9(b) show the data availabilities for constructing 2 and 3 newcomers respectively.

Fig. 9 shows that the data availabilities increase slightly with the growing of number of data blocks. For the decreased regeneration time will improve the data availability, while the increased failure probabilities will reduce the data availabilities. Fig. 9(a) shows that, when the number of providers is less than 6, MCRTREE could increase about 30% and 20% data availabilities respectively compared with RCTREE and MCR when constructing 2 newcomers. MCRTREE could still show larger data availabilities when the number of providers is larger than 6. Fig. 9(b) shows the similar performance with Fig. 9(a) when constructing 3 newcomers.

VI. Conclusion

In this paper, we address the challenges in minimizing the maintenance bandwidth and repair time. We analyze a motivating example to show the characteristics of MCRTREE and then construct a network model for MCRTREE. Based on this model, we discuss the MCRTREE constructing algorithm and then analyze the bottleneck bandwidth of regeneration trees. Considering the fluctuation of nodes in distributed storage system, we analyze the storage cost and maintenance bandwidth with numerical analysis, and then discuss the regeneration time, probability of successful regeneration and data availability with experiment comparisons. The numerical comparison and simulation results show that MCRTREE reduces repair bandwidth while keeping the same storage cost, and improves the repair efficiency and data availability.

ACKNOWLEDGMENT

This work was supported by the National Grand Fundamental Research 973 Program of China (Grant No.2011CB302601), the National Natural Science Foundation of China (Grant No.61379052), the National High Technology Research and Development 863 Program of China (Grant No.2013AA01A213), the Natural Science Foundation for Distinguished Young Scholars of Hunan Province (Grant No.S2010J5050), Specialized Research Fund for the Doctoral Program of Higher Education (Grant No.20124307110015).

REFERENCES

[1] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer *et al.*, "Oceanstore: An architecture for global-scale persistent storage," *ACM Sigplan Notices*, vol. 35, no. 11, pp. 190–201, 2000.

[2] Y. Wang and S. Li, "Research and performance evaluation of data replication technology in distributed storage systems," *International Journal of Computers and Mathematics with Applications*, vol. 51, no. 11, pp. 1625–1632, 2006.

[3] X. Lu, H. Wang, J. Wang, J. Xu, and D. Li, "Internet-based virtual computing environment: Beyond the data center as a computer," *Future Generation Computer Systems*, vol. 29, pp. 309–322, 2011.

[4] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[5] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Project Website*, vol. 11, p. 21, 2007.

[6] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*. USENIX Association, 2010, pp. 1–7.

[7] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: Novel erasure codes for big data," in *Proceedings of the 39th international conference on Very Large Data Bases*. VLDB Endowment, 2013, pp. 325–336.

[8] Y. Wang, X. Li, X. Li, and Y. Wang, "A survey of queries over uncertain data," *Knowledge and information systems*, vol. 37, no. 3, pp. 485–530, 2013.

[9] H. Weatherspoon and J. Kubiatowicz, "Erasure coding vs. replication: A quantitative comparison," *Peer-to-Peer Systems*, pp. 328–337, 2002.

[10] R. Rodrigues and B. Liskov, "High availability in dhts: Erasure coding vs. replication," *Peer-to-Peer Systems IV*, pp. 226–239, 2005.

[11] A. Dimakis and P. Godfrey, "Network coding for distributed storage systems," *Information Theory, IEEE Transactions on*, vol. 56, no. 9, pp. 4539–4551, 2010.

[12] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 476–489, 2011.

[13] J. Li, S. Yang, X. Wang, and B. Li, "Tree-structured data regeneration in distributed storage systems with regenerating codes," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[14] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. Voelker, "Total recall: System support for automated availability management," in *Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*, vol. 1, 2004, pp. 25–25.

[15] Y. Hu, Y. Xu, X. Wang, C. Zhan, and P. Li, "Cooperative recovery of distributed storage systems from multiple losses with network coding," *Selected Areas in Communications, IEEE Journal on*, vol. 28, no. 2, pp. 268–276, 2010.

[16] S.-J. Lee, P. Sharma, S. Banerjee, S. Basu, and R. Fonseca, "Measuring bandwidth between planetlab nodes," in *PAM*, 2005, pp. 292–305.

[17] J. Li, S. Yang, X. Wang, X. Xue, and B. Li, "Tree-structured data regeneration with network coding in distributed storage systems," in *Quality of Service, 2009. IWQoS. 17th International Workshop on*. IEEE, 2009, pp. 1–9.

[18] Y. Wu, A. Dimakis, and K. Ramchandran, "Deterministic regenerating codes for distributed storage," in *Allerton Conference on Control, Computing, and Communication*. Citeseer, 2007.

[19] K. W. Shum, "Cooperative regenerating codes for distributed storage systems," in *Communications (ICC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–5.

[20] Z. Huang, Y. Lin, and Y. Peng, "Robust redundancy scheme for the repair process: Hierarchical codes in the bandwidth-limited systems," *Journal of Grid Computing*, vol. 10, no. 3, pp. 579–597, 2012.

[21] S. Weidong, W. Yijie, and P. Xiaoqiang, "Tree-structured parallel regeneration for multiple data losses in distributed storage systems based on erasure codes," *Communications, China*, vol. 10, no. 4, pp. 113–125, 2013.