Cooperative Repair Based on Tree Structure for Multiple Failures in Distributed Storage Systems with Regenerating Codes

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
{xiaoqiangpei, wangyijie, maxingkong, yongquanf, xufl89}@nudt.edu.cn

ABSTRACT

Regenerating codes have been proposed to achieve an optimal trade-off curve between the amount of storage space and the network traffic for repair. However, existing repair schemes based on regenerating codes are inadequate to meet the requirements of small network traffic cost and high efficiency when repairing multiple failures. In this paper, we propose a cooperative repair scheme based on tree structure for multiple failures with regenerating codes, called CTREE. For generality, we propose a two-layer repair framework to support both repairs for single and multiple failures. For high repair efficiency, a parallel tree-structured data transmission technique is proposed to organize the data transmissions between the providers and newcomers. For small network network traffic cost, a core-based data exchange technique is proposed to organize the data exchanges between the coordinator and the other newcomers. To evaluate the performance of CTREE, we conduct experiments on both 30 physical and 200 virtual servers. Numerical analysis and extensive experiments confirm that CTREE can support both single and multiple failure repairs, significantly reduces the network traffic cost and improves the repair efficiency compared with the state-of-the-art approaches under various parameter settings.

Keywords

Regenerating Codes, Multiple Failures, Network Traffic Cost, Repair Efficiency

1. INTRODUCTION

Distributed storage systems aim to provide a reliable storage environment for large scale data over long periods, with applications like social network, document archiving and video sharing. While the storage nodes deployed in distributed storage systems are individually unreliable and node failures become normal, but not the exception as the number of storage nodes increases. To ensure the data reliability, erasure coding has been adopted to provide orders of magnitude more reliability with the same storage cost compared with replication [19] [16] [14]. To maintain the redundancy level, it is necessary to trigger the repair for reconstructing the lost data when failures happen. Each new node (denoted as newcomer) needs to download k blocks from the survival nodes (denoted as providers) and repair the lost block for a (n, k) code. This is very costly because it will download the entire file to create a lost block, with a network traffic of M. To alleviate the network traffic cost, Dimakis et al. [5] propose regenerating codes, which could achieve the optimal tradeoffs between storage efficiency and repair bandwidth consumption.

However, the repair schemes mentioned above mainly focus on single node failure. While there are situations of multiple failures in distributed storage systems. For instance, Total Recall [2] adopts the lazy repair scheme, which triggers the repair when the number of lost blocks reaches a given threshold. Besides the lazy repair, many real systems often face the situations of large percent of node failures from node leaving or power off. Furthermore, a large number of nodes may leave the network at the same time [4] and it is difficult for the system to detect every failure [18] .

These characteristics of failures require the repair of the distributed storage systems to be with high repair efficiency, small network traffic and generality. Firstly, the repair must be with high efficiency to minimize the possibility of further failures during the repair process. The key is to improve the data transmission efficiency and reduce the data volume transmitted between nodes. Secondly, the repair must be with small network traffic cost to support the various kinds of network-intensive applications. With the increasing number of applications in the data center, it is necessary to reduce the total network traffic cost. Otherwise, the repair process will degrade the performance of the applications. Meanwhile, the repairs should be completed at the background, which should not cause performance degradation. Thirdly, it's quite necessary to provide a general framework to support the repair of both single and multiple failures.

To address these challenges, researchers try to improve the repair efficiency by various techniques. However, existing repair approaches are inadequate to satisfy the requirements of high repair efficiency, small network traffic and generality [21]. This mainly steps from the following two reasons. Firstly, most schemes designed for single failure repair with

CF'15 May 18-21, 2015, Ischia, Italy

http://dx.doi.org/10.1145/2742854.2742869.

star structure [3] may suffer from the low data transmission efficiency, where the repair efficiency is limited by the bottleneck of the available bandwidth of the newcomer. Most schemes designed for single failure repair with tree structure [12] suffer from the high network traffic cost as multiple failures are repaired independently with each other. Secondly, most schemes designed for multiple failure repair with star structure [13] [8] suffer from the repair bottleneck as the relayer needs to complete the tasks of data forwarding and data reconstructions. Most schemes designed for multiple failure repair with tree structure [22] suffer from the high network traffic as multiple failures are repaired independently or it consumes much network traffic during the data exchanges between newcomers. [20]

To this end, this paper presents a cooperative repair scheme based on tree structure with regenerating codes, called C-TREE, for both single and multiple failures. Specifically, we mainly focus on three questions: (1) how to construct a general repair framework for both single and multiple failures? (2) how to improve the repair efficiency and reduce the repair time? (3) how to reduce the network traffic cost when repairing the multiple failures? We provide the following contributions in this paper:

1. We propose a general two-layer repair framework to support the repair of both single and multiple failures.

2. To improve the repair efficiency and reduce the repair time, we propose a parallel tree-structured data transmission technique to organize the data transmission between the providers and the newcomers.

3. To reduce the network traffic cost, we propose a core-based data exchange technique to organize the data exchanges between the newcomers. Meanwhile, we adopt the lazy repair within a stripe to further the network traffic cost.

4. We conduct extensive experiments for both single failure repair and multiple failure repairs under different parameter settings to confirm the low repair time and low network traffic cost of our approach.

The remainder of the paper is organized as follows. In Section 2, we introduce a two-layer repair framework, and present how CTREE improves the repair efficiency and reduce the network traffic cost. Section 3 compares CTREE with other typical repair schemes from both numerical and simulation comparisons. In Section 4 we introduce the related work. Finally, Section 5 concludes this paper.

2. COOPERATIVE REPAIRS

Our study focuses on the repair for multiple failures where the newcomers cooperatively complete the repair. In this section, we firstly present a two-layer repair framework to support both single and multiple failure repair. Then we introduce how to construct parallel trees for the failed blocks. Further, we introduce how the parallel tree-structured data transmission technique improves the data transmission efficiency and how core-based data transmission technique between newcomers reduces the network traffic cost. Finally, we introduce the encoding scheme for CTREE. The notations used in this paper are illustrated in Table 1.

2.1 Framework of CTREE

In this section, we propose a general repair framework $GRF(n,k,r)$ to support repairs of both single and multiple failures, illustrated in Figure 1. In $GRF(n,k,r)$, the data object is divided into k blocks and then encoded into n blocks (denoted as $(X_1,\ldots,X_n)$), which are stored at n storage nodes to maximize the data reliability. When there are r nodes fail, the system needs to find another r newcomers (denoted as $(Y_1,\ldots,Y_r)$) to reconstruct the failed blocks to keep the same level of redundancy.

Table 1: Parameter notations.

Symbols	Representations
N	Total number of nodes
M	Data object size
(n,k,r)	Coding parameters
X_i	The ith storage node
Y_i	The ith newcomer
T_i	The ith regeneration tree
e_i	The ith edge
ω_{e_i}	Weight of e_i

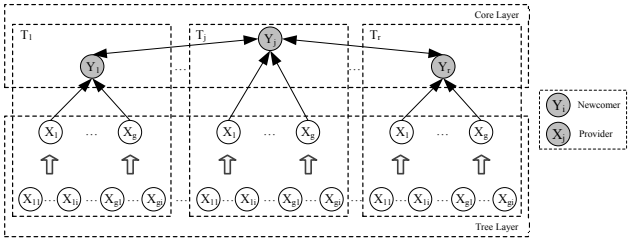

Figure 1: An illustration of $GRF(n,k,r)$.

At a high level, $GRF(n,k,r)$ consists of two layers: the tree layer and the core layer. The tree layer is responsible for data transmission between providers along the regeneration trees. At this layer, there are r regenerating trees $(T_1,\ldots,T_r)$ covering the providers and the newcomers. In each regenerating tree $T_i, 1 \le i \le r$ with the newcomer as the root and providers as the children, each non-leaf node encodes the received block with its stored block into a temp block, which is sent to its parent. The transmission is pipelined and the packet is the unit transmitted between nodes. Each node starts transmitting the packets as it completes the encoding operations. Thus, the bottleneck is the edge with the least available bandwidth in the tree.

The core layer is responsible for data exchanges between newcomers to reduce the network traffic cost. At this layer, there is a coordinator Y_c among all the r newcomers $(Y_1,\ldots,Y_r)$ and the data exchanges consist of three steps. Firstly, each newcomer Y_i receive data from the corresponding regeneration tree T_i and encodes the received data into a temp block t_i, which is sent to the selected coordinator Y_c. Secondly, the coordinator Y_c receives r temp blocks (including itself) $(t_1,\ldots,t_r)$ and encodes these temp blocks into r new partial blocks $(t'_1,\ldots,t'_r)$. These encoded partial blocks are sent to the other $r-1$ newcomers, with t'_i to Y_i. Finally, there are two blocks t_i and t'_i in each newcomer Y_i, and Y_i encodes them into the required block b_i with $b_i = c_i * t_i + c'_i * t'_i$, where c_i and c'_i are the coefficients.

To sum up, the two-layer framework $GRF(n,k,r)$ cooperatively completes repairs of the r lost blocks. The detail techniques related above mainly consist of the selection of newcomers, construction of parallel regeneration trees, transmission scheme of CTREE and encoding scheme of C-TREE. we specify these techniques in the following sections.

2.2 Selection of Newcomers

Newcomers are nodes, which are the roots of the regeneration trees and where the new reconstructed data stores. Different selection of newcomers will bring different repair performance. In $GRF(n,k,r)$, there is one special newcomer, called *coordinator*, which is the core exchanging data with other newcomers. Naturally, it will achieve the best repair performance if we can select the newcomer with the highest available bandwidth. However, it is difficult to detect the highest available bandwidth as the environment changes and the total node number in the data center increases. Furthermore, it may cause the performance skew if we try to select the newcomers with the highest available bandwidth. Thus, assume there are N storage nodes in total, we randomly select r newcomers among $N-n$ nodes, avoiding the possibility that there are blocks stored in the selected newcomers. As the coordinator needs to exchange data with all the other $r-1$ newcomers, we try to find the newcomer with the highest available bandwidth with other newcomer. Assume the available bandwidth between Y_i and Y_j is denoted as $ava_{i,j}$, the average available bandwidth between Y_i and other newcomers could be represented as $ava_i = \frac{\sum_{j=1,j\neq i}^{r} ava_{i,j}}{(r-1)}$. Thus, Y_c is the newcomer with highest average available bandwidth $ava_c = max\{ava_1, \ldots, ava_r\}$.

Meanwhile, each newcomer needs at least k providers among the $n-r$ survival nodes to complete the repair. It is proved that [5] will reduce the repair network traffic cost as the provider number increases for regenerating codes. Thus, it is natural to set all the $n-r$ survival nodes (denoted as $(X_1, \ldots, X_{n-r})$ without loss of generality) as providers.

2.3 Construction of Parallel Regeneration Trees

During the repair process, CTREE needs to construct r parallel regeneration trees. In each regeneration tree T_i, the non-leaf providers receive data from their children nodes, encode the received data with the data they store, and relay the encoded data to their parent nodes packet-by-packet. A regeneration tree (noted as T_i) is a tree, where the root is the newcomer $Y_i, 1 \leq i \leq r$ and covers $n-r$ providers as child nodes [12].

CTREE aims to construct r parallel optimal regeneration trees with the highest available bandwidth and repair the r lost blocks cooperatively and simultaneously. In this section, we propose a edge-shared tree construction technique, called ED-TREE, to construct the r regeneration trees in parallel. Each regeneration tree T_i covers a newcomer and $n-r$ providers. ED-TREE completes the construction of regeneration trees by two steps. Firstly, ED-TREE constructs r regeneration trees with the Kruskal's algorithm to optimize the available bandwidth. Secondly, ED-TREE adjusts some of the edges to maximize the available bandwidth. We specify these two steps as follows.

Regeneration tree construction with Kruskal's algorithm. The Kruskal's algorithm constructs a maximum spanning tree starting from the root inductively to optimize the bottleneck bandwidth. If the root has been selected, in the $(n-r)th$ step of Kruskal's algorithm, there are $n-r+1$ nodes in the regeneration tree, whose bottleneck bandwidth is optimal among all the regeneration trees in $GRF(n,k,r)$.

THEOREM 1. *After $(n-r)th$ step, the regeneration tree T^* constructed by Kruskal's algorithm is an optimal regeneration tree in $GRF(n,k,r)$.*

PROOF. T^* is a regeneration tree of $GRF(n,k,r)$. Assume T is a different regeneration tree with T^*, the definition of T could be defined as follows if T^* is not the optimal regeneration tree of $GFR(n,k,r)$:

$$f(T) = min\{i \backslash e_i \in T\}. \tag{1}$$

Select one optimal regeneration tree T_0 to maximize $f(T_0)$. Assume $f(T_0) = k$, $e_1, e_2, \ldots, e_{k-1}$ are in T_0 and T^* at the same time. However e_k is not in T_0. There is only one circle C in $T_0 + e + k$, and there is at least one edge e'_k in C but not in T^*.

$$T' = (T_0 + e_k) - e'_k \tag{2}$$

Namely, T' is a connected graph with $p(G)-1$ edges. Thus, T' is the regeneration tree of $GRF(n,k,r)$ and we can get Eq. 3.

$$W(T') = W(T_0) + \omega(e_k) - \omega(e'_k) \tag{3}$$

Since the selected edge e_k by Kruskal's algorithm is the edge that makes $(e_1, e_2, \ldots, e_k)$ no circuit graph with the least weight. $(e_1, e_2, \ldots, e_{k-1}, e_k)$ is a subgraph with no circuit. Thus, we can get Eq. 4.

$$\omega(e'_k) \geq \omega(e_k) \tag{4}$$

Combining Eq. 3 and Eq. 4, we can get Eq. 5.

$$\omega(T') \leq \omega(T_0) \tag{5}$$

Thus, T' is also a optimal regeneration tree of $GFR(n,k,r)$. However, as $\{e_1, e_2, \ldots, e_k\} \subseteq E(T')\}$, We can get Eq. 6.

$$f(T') > k = f(T_0) \tag{6}$$

This leads to the contradiction with the selection of T_0. Thus, T^* is a optimal regeneration tree of $GFR(n,k,r)$. □

The bottleneck bandwidth of an optimal regeneration tree T is the largest bottleneck bandwidth. However, as the construction process of r regeneration trees is serial by Kruskal's algorithm, where the former constructed regeneration trees may occupy all the available bandwidth between nodes, it is possible there is little available bandwidth when constructing the following regeneration trees. Thus, we need to adjust the edges of constructed regeneration trees to maximize the available bandwidth.

Edge adjustment. Assume the r constructed regeneration trees are $(T_1, \ldots, T_r)$. ED-TREE tries to figure out the bottleneck edges shared by multiple regeneration trees. We set the sharing threshold as t (default as $r/2$) and edges shared by more than t regeneration trees as $(e_1, \ldots, e_s)$, it may cause performance degradation if we ignore the edges shared by multiple regeneration trees. The available bandwidth $(\omega_1, \ldots, \omega_s)$ of the shared edges $(e_1, \ldots, e_s)$ is diluted to $(\omega_1/t, \ldots, \omega_s/t)$, where the data transmission time is determined by $min(\omega_1/t, \ldots, \omega_s/t)$. In fact, there may be some edges with higher available bandwidth that are not utilized. Thus, ED-TREE tries to adjust the shared edges to edges with less transmission burden to alleviate the burden of the shared edges. It is true that ED-TREE may not transmit the burden of all the shared edges to the edges with the higher available bandwidth, but it is possible to improve the data transmission efficiency by increasing the minimum bottleneck bandwidth $min(\omega_1/t, \ldots, \omega_s/t)$. Namely, the bottleneck available bandwidth $min(\omega'_1/t, \ldots, \omega'_s/t)$ after the adjust may be larger than that before, i.e., $min(\omega'_1/t, \ldots, \omega'_s/t) \geq min(\omega_1/t, \ldots, \omega_s/t)$.

Algorithm 1: Construction of regeneration trees

Input : Y_i: the ith newcomer.
providers: the providers list.
Output: T_i: the ith regeneration;

```
for each T_i, 1 ≤ i ≤ r do
    add Y_i into T_i.
    add T_i into treeList.
for each T_i, 1 ≤ i ≤ r do
    while node number of T_i is smaller than n − r + 1 do
        find the provider_j in providers with the maximum bandwidth with the nodes in T_i.
        add provider_j into T_i.
        delete provider_j from providers
```

Algorithm 1 shows how to construct r optimal regeneration trees. We first construct r optimal regeneration trees by Kruskal's algorithm. Then we complete the adjust by two steps. Firstly, if the degree of the regeneration tree T_i is invalid, we adjust the edges in the tree by adding the edge in E_{root} inductively. Secondly, we adjust the bottleneck edge to the edge with higher available bandwidth if it does not lead to contradiction with the indegree limitation of the root.

2.4 Transmission Scheme of CTREE

During the repair process of CTREE, there are two data flows. One is the data transmission along the regeneration trees, and the other is the data exchanges between the newcomers. In this section, we propose a pipelined data transmission technique, called PTransmission, to organize the data transmission along the regeneration trees, and a cored-based data exchange technique, called, CExchange, to organize the data exchanges between the newcomers. We specify these two techniques in the following sections.

2.4.1 Data Transmission along the Regeneration Tree

In this section, we are concerned about the data transmission along the regeneration trees and propose a pipelined data transmission technique PTransmission. In the tree layer of $GRF(n,k,r)$, there are r parallel regeneration trees. In each regeneration tree T_i, each leaf provider X_i transmits the required data β_i to its parent provider X_j. Each non-leaf provider X_j receives data $(\beta_{j,1},\ldots,\beta_{j,j_{in}})$ (j_{in} is the indegree of X_j) from its children, encodes them with the data β_j it stores with $\beta'_j=\sum_{i=1}^{j_{in}} c_{j,i}*\beta_{j,i}+c_j*\beta_j$, and relay the encoded data β'_j to its parent node byte-by-byte. With the relay of non-leaf providers, each newcomer Y_i (root of T_i) will receive a linear combination of i_{in} coded blocks $(\beta_{i,1},\ldots,\beta_{i,i_{in}})$ from i_{in} child providers. The newcomer Y_i will encode the i_{in} received blocks into a new temp block β_i with $\beta_i=\sum_{j=1}^{i_{in}} c_{i,j}*\beta_{i,j}$.

As we are concerned about the repair for r lost blocks, EDTREE constructs r parallel regeneration trees $(T_1,\ldots,T_r)$. During the repair process, the data transmission between multiple regeneration trees are independent with each other. Thus, the leaf nodes for all the r regeneration trees start to transmit the data simultaneously, which enables the parallel repair for the r lost blocks. The data transmission time is determined by the bottleneck available bandwidth of the regeneration tree, which is the edge with the minimum available bandwidth. The data transmission time for all the r regeneration trees is determined by the regeneration tree with the smallest available bandwidth. Assume the data transmission time for T_i is $time_i$, the data transmission time for all the r regeneration trees could be represented by $max\{time_1,\ldots,time_r\}$, illustrated in Algorithm 2.

Algorithm 2: *Data Transmission*

Input : *newcomer*: the selected newcomer list.
regenerationTrees: the regeneration tree list.

```
Assume the jth newcomer Y_j is the controller for (each newcomer Y[i] in newcomer, 0 ≤ i < r) do
    Receive data from providers along the regenerationTrees[i].
    if (i == j) then
        Encode/Decode the data received from regenerationTrees[j] and get β_{j,j}.
        while ((⋁_{g=0}^{r} signal_g) do
            for (x ∈ [1, r]) do
                β'_{j,x} = Σ_{l=1}^{r} c_x * β_{l,j}.
                Send the data β'_{j,x} to the xth newcomer Y_x.
                if (Data sending completely) then
                    Send the signal_j = false to newcomer_x.
        Store the new generated data β'_{j,j} locally.
    else
        Encode/Decode the data received from regenerationTrees[i] and send the generated data β_{i,j} to the controller Y_j.
        if (Data sending completely) then
            Send the signal_i = false.
        while (signal_j) do
            Receive data β'_{j,i} from the controller Y_j.
            Store the data β'_{j,i} locally.
```

2.4.2 Data Exchanges Between Newcomers

In this section, we are concerned about the data exchanges between the newcomers and propose a core-based data exchange technique CExchange. In the core layer of $GRF(n,k,r)$, there is a coordinator among the r newcomers, which is the core newcomer exchanging data with other newcomers. Based on the analysis of PTransmission, we know that the data volume transmitted along the regeneration trees is equal to $\beta =$. As the newcomers exchange the data received from the regeneration trees, the data volume transferred between newcomers is the same as β.

In each regeneration tree T_i, the newcomer Y_i combines all the received blocks from its children into a new temp block β_i, including the coordinator Y_c. One of the main characteristics of CTREE is that the data exchanges between newcomers could reduce the network traffic cost. Thus, the required blocks for newcomers would be obtained by exchanging the temp blocks. CExchange adopts the core-based data exchange pattern to organize the data exchange. The data exchange could be completed by three steps. Firstly, All

the newcomers transmit the combined blocks from the regeneration trees to the coordinator. Thus, the coordinator will receive r temp blocks $(\beta_1, \ldots, \beta_r)$, $r-1$ of them coming from the other $r-1$ newcomers, and $\beta_c, 1 \leq c \leq r$ is from its regeneration tree. Secondly, the coordinator Y_c will encode the r temp blocks into r partial blocks $(p'_1, \ldots, p'_r)$ with $p'_i = \sum_{j=1}^{r} c_{i,j} * \beta_j$. Thus, the coordinator Y_c will get r partial blocks. Thirdly, the coordinator forwards the $r-1$ partial blocks to the corresponding newcomers, with p'_i for Y_i. Each newcomer Y_i combines the temp block β_i received from the regeneration tree and the partial block p'_i received from the coordinator into the required block p_i by $p_i = c_i * \beta_i + c'_i * p'_i$.

Different from the data exchange between any two newcomers in mutually cooperative recovery (MCR) mechanism, CExchange adopts the core-based data exchanges between the newcomers, where the other newcomers exchange data with the coordinator. Thus, CExchange could save much network traffic cost. It will cost $A_r^2 * \beta = r * (r-1) * \beta$ for MCR to complete the data exchange between newcomers, and $2 * (r-1) * \beta$ for CTREE to complete the data exchange between the coordinator and the other newcomers, where β is the data volume transmitted between nodes. Thus, MCR consumes more network traffic than CTREE as $r*(r-1) \geq 2*(r-1)$ if $r \geq 2$. It is true that it may cause a bottleneck as all the newcomers exchange data with the coordinator. However, the number of data exchange between the coordinator and the newcomers will stay small as r is a relative small number. Furthermore, CExchange adopts the pipelined data transmission scheme to improve the data transmission efficiency, which could alleviate the bottleneck of coordinator. Thus, the relative small newcomer number and the pipelined data transmission ensure that the coordinator will not become the bottleneck.

2.5 Encoding Scheme of CTREE

A coding scheme in CTREE is valid if a data object encoded by this scheme can be repaired after multiple failures based on the CTREE transmission scheme. Inspired from the tree-structured data regeneration with regenerating codes (RCTREE) [12], CTREE adopts the MSR codes to reduce the repair time and the number of providers is variable as the failure number changes. Thus, the encoding scheme of CTREE should adapt to this. For the repair with $d, k \leq d \leq n-r$ providers, the data object should be divided into k blocks, and each block should be divided into at least $(d-k+1)$ symbols to achieve the lower bound of network traffic [7] [12]. For each newcomer in CTREE, it receives data from both the regeneration tree and the coordinator. It seems that there are $n-r$ providers and $r-1$ other newcomers transmitting data to each newcomer. We could take it as there are $n-1$ providers participating in the repair for each lost blocks, namely, $d = n-1$. Thus, the data object is divided into $k(n-k)$ symbols and each storage node stores one block consisting of $(n-k)$ symbols. The $k(n-k)$ symbols are encoded into $n(n-k)$ symbols, and these symbols are stored at n nodes $(X_1, \ldots, X_n)$, each node storing $n-k$ encoded symbols. In a regeneration tree with $n-r$ providers, each provider encodes its $n-k$ symbols into a symbol, and then sends it to its parent nodes. Then each newcomer receives a total of i_{in} symbols from the regeneration trees and one symbol from the coordinator, which combines the information from other $r-1$ newcomers, so the newcomer has to receive data directly from at least $n-k-(r-1)$ providers. The traffic on each link is $\frac{M}{k(n-k)}$.

3. EVALUATION

In this section, we compare CTREE with RCTREE and MCR from both numerical analysis and experimental evaluation. The numerical analysis compares the network traffic cost while the experiments compare the repair time.

3.1 Implementation

To evaluate the repair time of CTREE, we design and implement the prototype of CTREE, RCTREE and MCR based on the Openstack-based testbed [15]. To develop the prototype as modular and portable, we use ICE [1] as the basic communication platform. Through defining the communication interfaces, ICE supports various communication patterns among processes, which allows the developers focusing on the application logic.

The framework of CTREE consists of two layers: core layer and tree layer. Both layers greatly affect the performance of repair. Existing works focus on either core layer or tree layer. For example, RCTREE focuses on the tree layer to improve the data transmission efficiency without any data exchanges between newcomers. MCR focuses on the data exchanges between newcomers without considering the optimization of data transmission between the providers and the newcomers. We design and implement RCTREE and MCR on our testbed to evaluate the performance of them.

RCTREE: It constructs a regeneration tree for each newcomer and its providers on the tree layer, where the data transmits along the providers and reaches the newcomer. Compared with CTREE, there is no data exchange between the newcomers on the core layer and the multiple failed blocks are repaired in a serial way.

MCR: It constructs a star structure for each newcomer and its providers on the tree layer, where all the providers transmit data to the newcomer directly without any relay. Meanwhile, each newcomer needs to exchange data with any other newcomer on the core layer. Compared with CTREE, the star structure on the tree layer degrades the data transmission efficiency and it consumes more network traffic when exchanging data on the core layer.

In the following evaluations, we evaluate the performance of network traffic cost by numerical analysis and repair time by experiments on both physical and virtual machines.

3.2 Parameters and Metrics

To evaluate the performance of the system, the testbed contains two kinds of servers: one is the physical machines (PMs) equipped with two hexa-core Intel Xeon E5-2640 2.5GHz processors, 48GB RAM, 2TB hard disk and a 1Gb/s Ethernet card. The other is the virtual machines (VMs) equipped with a two-core processor, 8GB RAM, 300GB hard disk. All the physical or virtual servers are homogeneous running 64-bit Ubuntu 14.04 with JDK 1.6.0 37. The physical machines that these VMs are attached to are connected using gigabit switches. There are 30 PMs, where each of them contains about 7 VMs. Each VM performs as a degraded PMs, which may degrade the repair performance. However, the experiments are conducted on the same PMs for all the three repair schemes. Thus, it is fair for all of them when executing the repair processes.

It is important to note that we are concerned about that

Table 2: Storage Cost and Bandwidth Comparison of RCTREE, MCR and CTREE

	Total storage cost	Total network traffic cost
RCTREE	$(\frac{M}{k}) \cdot n$	$[\frac{n-r}{n-r-k+1}] \cdot (\frac{M}{k}) \cdot r$
MCR	$(\frac{M}{k}) \cdot n$	$[\frac{n-1}{n-k}] \cdot (\frac{M}{k}) \cdot r$
CTREE	$(\frac{M}{k}) \cdot n$	$\frac{r(n-r+2)-2}{n-k} \cdot \frac{M}{k}$

the network traffic cost and the repair time when repairing multiple failures. We analyze network traffic cost by numerical comparisons with variation of n, k and r. To evaluate the repair time, we conduct the experiments on both PMs and VMs. Total network traffic cost and total repair time are two main metrics we are concerned about. Total network traffic cost is the total data volume transmitted between nodes during the repair process. The total repair time is the duration from the start of the first repair to the end of the last repair. We expect both as small as possible.

3.3 Numerical Comparisons

In this section, we compare CTREE with RCTREE and MCR from numerical comparisons in storage overhead and network traffic cost. Assume the original data object (denoted as its size of M) is divided into k blocks, and encoded into n blocks. Each of the storage nodes stores M/k bytes data. After some time, there are r storage nodes fail, and the repair is triggered. Based on the scenario above, we analyze the storage overhead and repair bandwidth cost of RCTREE, MCR and CTREE .

RCTREE: In [12], 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 repair bandwidth is $[\frac{d}{d-k+1}] \cdot (\frac{M}{k}) \cdot r$.

MCR: In [8], 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 $\gamma_{MCR} = (n-1)r\beta$ and $\beta = \frac{M}{[k(n-k)]}$ to repair the data. So the storage cost is $(\frac{M}{k}) \cdot n$, and the repair bandwidth is $[\frac{n-1}{n-k}] \cdot (\frac{M}{k}) \cdot r$.

CTREE: From the above analysis in this paper, when using CTREE with $\alpha_{CTREE} = \frac{M}{k}, \gamma_{CTREE} = (n-r)r\beta+2\beta$ and $\beta = \frac{M}{[k(n-k)]}$, the storage cost is $(\frac{M}{k}) \cdot n$ and the total network traffic is $\frac{n-r+2}{n-k} \cdot \frac{M}{k} \cdot r$.

It is proved that [5] we could get better tradeoff of storage cost and repair bandwidth if d increases for regenerating codes. When there are r storage node failures, the maximum number of d is $n-r$, so we set $d = n-r$, the data transmitted between nodes is $\beta = \frac{Md}{k(d-k+1)}$, and the total maintenance bandwidth for repair r lost blocks is $[\frac{d}{d-k+1}] \cdot (\frac{M}{k}) \cdot r$ for RCTREE. And MCR could not only access data from $n-r$ providers, but there are data exchanges between newcomers, so the maximum d for each newcomer is $n-1$, which includes $n-r$ providers and $r-1$ newcomers. Thus we set $d = n-1$, the data transmitted between nodes is $\beta = \frac{M}{k(n-k)}$ and the total maintenance bandwidth for repair r lost blocks is $[\frac{n-1}{n-k}] \cdot (\frac{M}{k}) \cdot r$ for MCR. While in CTREE, each newcomer could access data from $d = n-r$ providers, and there are data exchanges between newcomers and the root newcomer, so we set $d = n-1$, the data transmitted between nodes is $\beta = \frac{M}{k(n-k)}$ and the total maintenance bandwidth for repair r lost blocks is $\frac{n-r+2}{n-k} \cdot \frac{M}{k} \cdot r$ for CTREE. The storage costs and total maintenance bandwidths of different redundancy recovery schemes are illustrated in Table 2.

Table 3: Parameter Values

Symbols	Range	Default
n	{10,11,12,13,14}	14
k	$\{3,4,5,6,7,8,9,10\}$	6
r	$\{1,2,3,4\}$	3
$b\ (MB)$	$\{32,64,128,256\}$	64
$pkg\ (KB)$	$\{16,32,48,64\}$	48

3.4 Experimental Results

In this section, we compare CTREE with RCTREE and MCR from both physical machines and virtual machines with the variation of parameters illustrated in Table 3. For RCTREE, we construct r regeneration trees and the data flows along the regeneration tree, until reaching the root. Since RCTREE repairs multiple failures serially, the r regeneration trees are constructed in a serial way. For MCR, we construct the star structure with each newcomer and its providers, and exchange data between any two of the r newcomers. For CTREE, we construct r regeneration trees and exchange data between newcomers, where each newcomer contacts with the coordinator, which receives data both from its regeneration tree and other newcomer. Finally, the coordinator sends the encoded data back to the other newcomers. We specify the experimental results as follows.

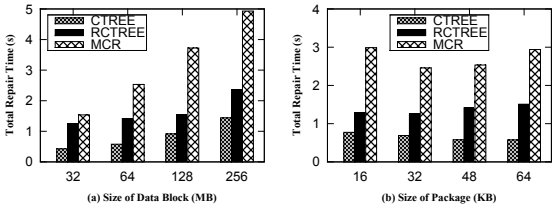

Figure 2: Repair time comparisons of CTREE, RCTREE and MCR with data block size and package size on PMs.

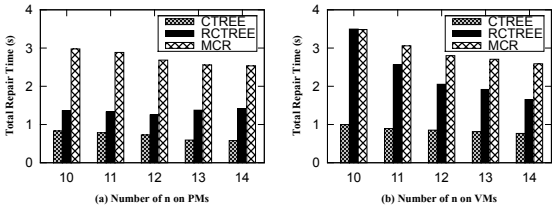

Figure 3: Repair time comparisons of CTREE, RCTREE and MCR with n on PMs and VMs.

3.4.1 Results on PMs

Figure 2(a) shows that the repair time of CTREE, RCTREE and MCR increases as the data block size increases from 32MB to 256MB, since larger block consumes more time to complete the transmission and encoding. Figure 2(b) shows that the repair time decreases as the package size

increases from 16KB to 48KB, but increases as the package size continuously increases to 64KB. Since larger package size reduces the package number, which improves the data transmission efficiency. However, as the data packets are transmitted with TCP connection, where each TCP package is smaller than 64KB. Each TCP package will be divided into two TCP packages as the total volume of data package and the TCP header is larger than 64KB when the data package size is larger than 48KB. Thus, the repair time increases as the package size continuously increases from 48KB to 64KB. Figure 3(a) shows the repair time decreases as n increases from 10 to 14, since larger n increases the providers participating in the repair but reduces the data volume transmitted between nodes during the repair process. Figure 5(a) shows the repair time increases as k increases from 3 to 10, since larger k increases the data volume transmitted between nodes, which consumes more time for repairing the lost blocks. Figure 6(a) shows the repair time increases as r increases from 1 to 4, since the repair for more lost blocks needs to connect more nodes and transmit more data during the repair process.

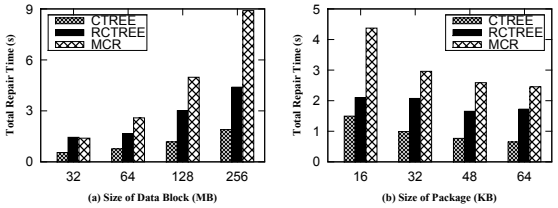

Figure 4: Repair time comparisons of CTREE, RCTREE and MCR with data block size and package size on VMs.

3.4.2 Results on VMs

Figure 4, Figure 3 (b), Figure 5 (b) and Figure 6 (b) show the total repair time comparisons of CTREE, RCTREE and MCR on VMs with the varied parameters illustrated in Table 3. The experimental results tell us that the repair time for all the three repair schemes shows the similar trend with the variation of parameters. However, the repair time on VMs is larger than that on PMs, since the network bandwidth of PMs is shared by the VMs located on it. The shared network bandwidth may degrade the data transmission efficiency between VMs. Meanwhile, all the VMs on a PM share the same hard disk, where the I/O performance of VMs on a PM is limited by the I/O throughput of the hard disk. The more number of I/O operations on the hard disk, the less I/O throughput for the VMs. Thus, the repair time for all the three repair schemes on VMs is larger than that on PMs. However, the degraded performance of VMs does not have impact on the trend of among CTREE, RCTREE and MCR, where we can treat the VMs as the degraded PMs. The experimental results on both PMs and VMs tell us that CTREE shows the least repair time compared with RCTREE and MCR.

4. RELATED WORK

The optimization of repair performance for erasure coding has attracted many attentions. Huang et al. [9] proposed Pyramid codes, which divide the k blocks into groups and

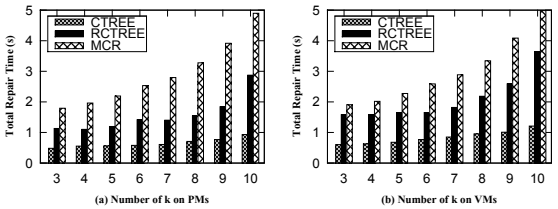

Figure 5: Repair time comparisons of CTREE, RCTREE and MCR with k on PMs and VMs.

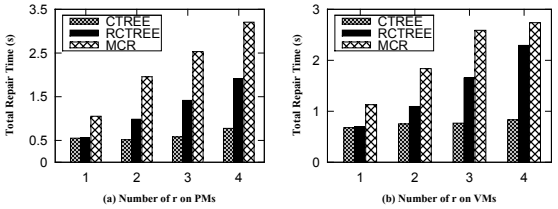

Figure 6: Repair time comparisons of CTREE, RCTREE and MCR with r on PMs and VMs.

create local coded blocks and global coded blocks. The Local Reconstruction Codes (LRC) proposed in [10], Locally Repairable Codes (LRC) presented in [17] and Hierarchical Codes (HC) presented in [6] [11] share the similar idea with pyramid codes. Compared with the MDS codes, codes based on degree restriction reduce the repair bandwidth cost by grouping the blocks at the cost of extra storage cost.

Dimakis et al. [5] introduced network coding into erasure codes and propose Regenerating Codes (RC) which use slightly larger fragments than MDS but have lower overall bandwidth consumption. RC requires that each block in the storage node needs to be subdivided into fragments and allows the newcomer could access more than k providers to download the required data. It is proved that RC could reduce the overall bandwidth consumption for repair. The conventional repair schemes connect providers and the newcomer with a star structure, where the repair time is restricted by the bottleneck bandwidth between providers and the newcomer. Jun Li et al. [12] proposed a tree-structured regeneration scheme and show how the tree-structured scheme regenerates redundant data at the newcomer. It is proved that the tree-structured repair scheme can reduce the regeneration time by improving the transmission rate.

Hu et al. [8] proposed a mutually cooperative recovery (named MCR) mechanism for multi-loss recovery, where all the newcomers repair the lost data cooperatively and simultaneously. Each newcomer in MCR chooses $n-1$ nodes (both all the $n-r$ survival nodes and the other $r-1$ newcomers) for recovery, and needs to download less data from the providers and the other newcomers. Li et al. [13] propose a system CORE to support both single and concurrent failure repair, which aims to minimize the repair network traffic cost by reconstructing all the lost data in one core newcomer and distributing these constructed data to the other newcomers. Compared to these studies, this paper focuses on the optimization of repair time and network traffic cost for multiple failures.

5. CONCLUSIONS

In this paper, we focus on the optimization of repair network traffic and repair time when repairing multiple failures, and propose a a cooperative repair scheme based on tree structure with regenerating codes CTREE. For generality, CTREE presents a two-layer framework to support both the single and multiple failures. Low network traffic cost is attained by CExchange to organize the data exchange between the coordinator and the other newcomers. Low repair time is achieved by ED-TREE and PTransmission to improve the data transmission efficiency by regeneration trees and pipeline data transmission. Numerical analysis and experiments based on the real deployment under various parameter settings revealed that CTREE reduces the repair traffic cost and the repair time compared with the typical repair schemes.

6. ACKNOWLEDGMENTS

This work was supported by 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.14JJ1026), Specialized Research Fund for the Doctoral Program of Higher Education (Grant No.20124307110015).

7. REFERENCES

[1] Zeroc. [online]. available:, http://www.zeroc.com/.

[2] 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*, volume 1, pages 25–25, 2004.

[3] V. R. Cadambe, C. Huang, and J. Li. Permutation code: Optimal exact-repair of a single failed node in MDS code based distributed storage systems. In *IEEE International Symposium on Information Theory Proceedings, ISIT 2011, St. Petersburg, Russia, July 31 - August 5, 2011*, pages 1225–1229, 2011.

[4] O. Dalle, F. Giroire, J. Monteiro, and S. Pérennes. Analysis of failure correlation impact on peer-to-peer storage systems. In *Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on*, pages 184–193. IEEE, 2009.

[5] A. Dimakis and P. Godfrey. Network coding for distributed storage systems. *Information Theory, IEEE Transactions on*, 56(9):4539–4551, 2010.

[6] A. Duminuco and E. Biersack. Hierarchical codes: How to make erasure codes attractive for peer-to-peer storage systems. In *Peer-to-Peer Computing, 2008. P2P'08. Eighth International Conference on*, pages 89–98. IEEE, 2008.

[7] A. Duminuco and E. Biersack. A practical study of regenerating codes for peer-to-peer backup systems. In *Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on*, pages 376–384. IEEE, 2009.

[8] 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*, 28(2):268–276, 2010.

[9] C. Huang, M. Chen, and J. Li. Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. *ACM Transactions on Storage (TOS)*, 9(1):3, 2013.

[10] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin. Erasure coding in windows azure storage. In *USENIX Annual Technical Conference (USENIX ATC)*, 2012.

[11] Z. Huang, E. Biersack, and Y. Peng. Reducing repair traffic in p2p backup systems: Exact regenerating codes on hierarchical codes. *ACM Transactions on Storage (TOS)*, 7(3):10, 2011.

[12] 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*, pages 1–9. IEEE, 2010.

[13] R. Li, J. Lin, and P. P. C. Lee. Core: Augmenting regenerating-coding-based recovery for single and concurrent failures in distributed storage systems. In *IEEE 29th Symposium on Mass Storage Systems and Technologies, MSST 2013, May 6-10, 2013, Long Beach, CA, USA*, pages 1–6, 2013.

[14] 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*, 29:309–322, 2011.

[15] L. OpenStack. Openstack: The open source cloud operating system, 2012.

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

[17] 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*, pages 325–336. VLDB Endowment, 2013.

[18] W. Sun, Y. Wang, Y. Fu, and X. Pei. A discrete data dividing approach for erasure-code-based storage applications. In *Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on*, pages 308–313. IEEE, 2014.

[19] 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*, 51(11):1625–1632, 2006.

[20] Y. Wang, X. Li, X. Li, and Y. Wang. A survey of queries over uncertain data. *Knowledge and information systems*, 37(3):485–530, 2013.

[21] Y. Wang and X. Ma. A general scalable and elastic content-based publish/subscribe service. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2014.

[22] 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*, 10(4):113–125, 2013.