2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)

Topology-aware Efficient Storage Scheme for Fault-tolerant Storage Systems in Data Centers

Junxu Xia*, Deke Guo*, Lailong Luo, Jiangfan Li, Chendie Yao
Science and Technology on Information System Engineering Laboratory,
National University of Defence Technology, Changsha Hunan 410073, P.R. China.

***Abstract*—In data centers, files are stored with the method of erasure code or replication to guarantee data reliability. However, both of the methods are not communication-friendly. An erasure coding system spends vast time to extract k data blocks across the racks during the decoding process. The transmission time contributes up to 94% of the total decoding time. In a multi-replica system, the frequent data writing and updating also lead to non-trivial bandwidth overhead. In this paper, we consider server-centric data centers (such as BCube), in which any pair of nodes are interconnected with multiple parallel paths. In such data centers, the transmissions for file storage can be significantly speed up by utilizing the parallel paths concurrently. With this insight, we first define the *disjoint node* in BCube. Based on this definition, we design the node-disjoint storage strategy (NDSS) and the nested node-disjoint storage strategy (N-NDSS) to improve the transmission between distributed storage nodes. Comprehensive simulations show the performance of our strategies in both erasure coding system and multi-replica system.**

***Index Terms*—Distributed Storage, Data center, Replication, Erasure code**

I. Introduction

Disk, server, and network failures are common in large-scale data centers. For example, Facebook reports on average 50 machine failures every day in its data centers, where each machine generally has a storage capacity of 24-36 TB [1]. These failures may lead to immediate data unavailability and damage the quality of service. To improve the reliability, fault-tolerant storage technology, such as replication and erasure code, are implemented in various file storage systems.

However, the penalty of the fault-tolerant storage technologies is the performance decline of these systems. For example, the source node in the R-replica system has to transmit R replicas to other storage nodes when storing a file. If the time of transferring a replica is t, it takes $R \times t$ time to store a file. Similarly, the update of a file also need $R \times t$ time-consumption. In an erasure coding enabled storage system, the vast number of data blocks further exacerbates the transmission time. Moreover, when a failed data block needs to be repaired from the (k, m) erasure coding system, the repair node has to extract k data blocks from k storage nodes. As a consequence, the repair node may be overwhelmed by the received data blocks thereby leading to congestion (this phenomena is also called the incast problem). When the k data blocks are sent to the repair node simultaneously, the time of network transmission takes up as much as 94% of the whole repair process [2][3][4]. To speed up the transmission of data blocks during repairing, the partial parallel repair (PPR) [5] scheme proactively divides the repair process as multiple steps and then aggregates the data blocks on storage nodes during transmission. But PPR is only designed for data repair and remains inapplicable for file storage or recovery.

We note that, modern data center networks (DCNs) alway provide multiple parallel paths between any pair of node with fancy network topologies, e.g., BCube [6], HFN[7], VLCcube [8], etc. Especially, with the capability of forwarding and packet processing, servers has been employed as networking devices in DCNs. As a typical server-centric topology, BCube [6] recursively interconnects the servers and switches and thus provides parallel paths between arbitrary pair of nodes. BCube has been widely employed by large companies to further improve the connectivity and decrease the hardware cost of DCNs, such as Sun's Modular Data Center [9], HP's POD [10], IBM's Modular Data Center [11], etc.

If the parallel paths in DCNs can be employed to transmit the data blocks in the replication or erasure code system, the transmissions can be significantly speed up without congestion. This insight, however, is not practical in current distributed file systems. The reason is that the replica or data block placement in current distributed storage systems has to relay on overlapped links to transmit replicas or data blocks. Therefore, in this paper, based on the given data center topology, we propose to place the replicas or data blocks on specific nodes, so that the parallel paths can be employed during file storing, recovering and repairing.

Taking BCube as an example, we define the *disjoint nodes* as any pair of nodes which have different identifiers in any topological dimension. With such definition, in a group of *disjoint nodes*, any pair of nodes are able to transmit replicas or data blocks with multiple parallel paths simultaneously. Based on the properties of the disjoint node, we design the node-disjoint storage strategy (NDSS) to realize the one-to-many or many-to-one parallel transmission for storage systems. In addition, we propose the nested node-disjoint storage strategy (N-NDSS) to increase the upper limitation on the number of storage nodes in erasure coding systems. Lastly, an optimized transmission scheme is designed to improve the performance of repairing a failed data block in the erasure coding system. The contributions of this paper are summarized as follows:

- We define the concept of *disjoint nodes* in given DCN and then propose the NDSS model to find parallel trans-

*Deke Guo and Junxu Xia are the corresponding authors

978-1-5386-7308-9/18/$31.00 ©2018 IEEE
DOI 10.1109/ICPADS.2018.00083

mission paths between a single node and multiple disjoint nodes. NDSS significantly resolves the incast problem of the many-to-one transmission and improve the efficiency of the one-to-many transmission.
- For erasure coding system, based on the definition of *partial disjoint nodes*, we further present the N-NDSS model to improve the network transmission efficiency during writing, updating and file recovery.
- Comprehensive evaluations are conducted to compared our proposals with the PPR. The results indicate that both NDSS and N-NDSS outperforms PPR in terms of transmission time.

The reminder of this paper is organized as follows. Section II introduces the related works and background. Section III reports how to design the storage models in fault-tolerant systems, and Section IV evaluates the network performance of NDSS and N-NDSS. Lastly, Section V concludes this paper.

II. Related Work and Background

In this section, we introduce the multi-replica and the erasure coding system, and then analyze the drawbacks of the existing transmission methods.

A. The multi-replica system

In recent years, HDFS [12], Ceph [13], Swift [14], and other systems that handle large amounts of data have embedded multiple replicas to guarantee data reliability. Typically, a R-replica system generates R replicas of any data block and store them on R different storage nodes. For example, in HDFS, the value of R is set as 3 to easily determine whether the stored replica has an error or not. If the replica in a node is different with others, that replica is definitely incorrect and should be updated or recovered.

The co-existence of these replicas guarantees data reliability, however, at the cost of $3\times$ storage space and non-trivial communication overhead. In most clusters, the intra-rack bandwidth is usually greater than the inter-rack bandwidth. Thus, it is better to control the inter-rack transmissions in distributed file systems. With this insight, as shown in Fig.1, most distributed file systems place one replica in the local node, and two other replicas in two randomly selected nodes in an adjacent rack. If more replicas are required, these file systems will randomly chose system-wide host nodes for extra replicas. This strategy eases the transmission overhead of maintaining such a multi-replica system to some extent than random placement of replicas. However, it still falls short to reduce the frequent data transmissions between nodes.

B. The erasure coding system

Although the replication improves the storage performance, it leads to 3 $\times$ storage space. By contrast, the erasure code method reduces the storage redundancy by encoding the original file. Erasure code is first proposed to solve the problem of partial data loss in transmission industry. The basic insight is to divide the transmission signal, and then add some check blocks to establish certain relation between the segments. Even if parts of the signal are lost in the transmission process, the receiver can still get the complete information by algorithm. Erasure code is widely used in distributed systems like DiskReduce [15], Google Colossus [16] and Microsoft Azure [17].

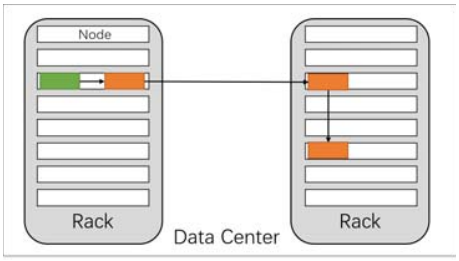


Fig. 1. An improved storage strategy of HDFS.

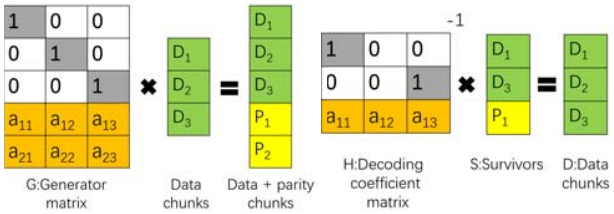


Fig. 2. RS $(3,2)$ encoding.

Fig. 3. The decoding coefficient matrix for RS $(3,2)$ reconstruction.

Especially, the Reed-Solomon (RS) erasure code [18] has been widely used in storage system. For a (k,m) RS code, a file of N bytes is divided into k data blocks, each of which is N/k bytes. In addition, m parity blocks are calculated based on the k data blocks. In this way, a file is decomposed into $k+m$ data blocks. The original file can be reconstructed according to any k of data blocks. Moreover, when a data block fails, the system can repair the failure data block with extract k data blocks. However, unlike the file recovery process, the repair process can be carried out in steps because each block is a linear combination of other k involved blocks.

Encoding. Similar to the binary case, the RS encoding process can be expressed as the matrix vector multiplication, as shown in Fig.2. The matrix G, which is called generator matrix, is computed by the Vandermonde matrix [19]. The elements a_{ij} etc. are calculated according to the Galois Field (GF) arithmetic [18]. Using the matrix G and the data blocks matrix, the matrix of new data blocks and parity blocks can be obtained.

Decoding. The decoding process can be divided into the file recovery and the repair of a failed data block. For the file recovery, it constructs the decoding coefficient matrix H. As shown in Fig.3, H is derived as the inverse of the matrix which deletes the rows of the missing blocks from the coefficient matrix G. Finally, multiply H as the decoding coefficient with the surviving blocks S, then the original file can be restored. When any data blocks fails, it should be repaired.

The repair process can be expressed as follows:

$$C_i = (g_{i1}\ g_{i2}\ \cdots\ g_{ik}) \times (D_1, D_2, \cdots, D_k)^T, \qquad (1)$$

where $g_{ij} \in F_q, i = 1,2,...,k+m, j = 1,2,...,k$, F_q is the galois field of size q.

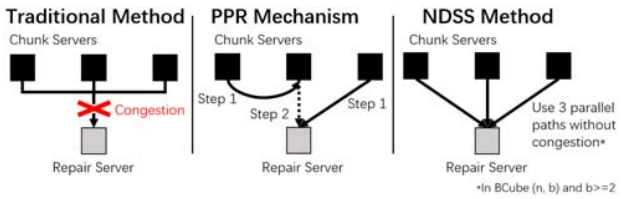


Fig. 4. The comparison between traditional method, PPR method and NDSS method with RS $(3, 2)$.

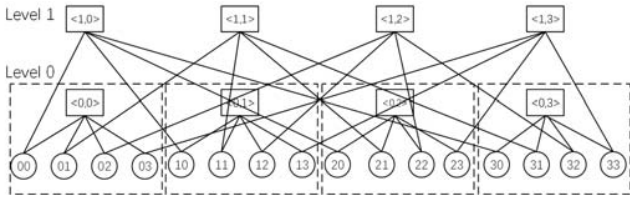


Fig. 5. The BCube $(4, 1)$ topology, consisting of 16 servers and 8 4-port mini-switches.

Therefore, when repairing a failed data block, the repair node needs to extract k data blocks from the remaining storage nodes, then invert the coding coefficient matrix and calculate the corresponding data block. The whole decoding process can be seen as a linear combination of the data blocks:

$$C^{'} = \sum_{i=1}^{k} \beta_i \times C_i^{'}, \tag{2}$$

where $C^{'}$ is the data block need to repair. We record the extracted k data blocks as $C_1^{'}, C_2^{'}, ..., C_k^{'}$, the corresponding decode coefficient as $\beta_1, \beta_2, ..., \beta_k, \beta_i \in F_q$. The repair operation is the process of a linear combination of k data blocks. Hence, the aggregation of multiple data blocks does not increase their size.

C. Analysis of existing transmission methods

For the multi-replica system, although the current strategy (in Fig.1) improves the bandwidth utilization, it still has to sent replicas to different nodes separately to avoid congestion. In addition, files in data centers call for frequent updates which are also time-consuming. In the erasure coding system, besides of file storing and recovering, repairing a failed data block is also time-consuming. It is reported that the inefficient repair of erasure code has become a bottleneck to restrict its wide application [20]. In order to maintain the reliability of data, the system has to spend amount of network overhead repairing the failed data blocks. Besides, according to literatures [2][3][4], the transmission time takes up almost 94% when repairing a failed data block.

The traditional method is a star-structured scheme. Its basic idea is to transfer k involved data blocks to the repair node simultaneously. This will cause the incast problem, decreasing the transmission efficiency, as shown in Fig.4. In order to increase the speed of repairing a failed data block, a partial parallel repair (PPR) mechanism is proposed [5]. PPR builds a binomial reduction tree to optimize the repair process by XOR because the size of a group of data blocks after aggregation is the same as an origin data block. Although PPR gives a solution to the incast problem, it is only applicative to repair a failed data block. In data centers, files need to be stored and updated frequently, this method cannot improve the efficiency of storing or updating a file. Both the traditional method and PPR fail to employ the parallel paths in DCNs. Therefore, in this paper, we propose NDSS and N-NDSS to place the replicas or data blocks in storage systems to enable the using of parallel paths. As a result, the transmission of file storing, recovering and repairing can be significantly speed up.

III. Topology-aware efficient storage Scheme

In this section, taking the BCube topology as an example, we propose the node-disjoint storage strategy (NDSS) and the nested node-disjoint storage strategy (N-NDSS). These strategies speed up the transmission by selecting special storage nodes and thus realizing parallel transmission of data blocks with parallel paths.

A. The NDSS model

The BCube structure is a server-centric structure proposed by Microsoft researchers, which is mainly designed for modular data center container interconnection [6]. BCube uses the commercial rather than customized mini-switch and multi-port server to build large-scale data center network recursively. In the BCube, the server is not only the place for data processing and storage, but also plays the role of forwarding data.

Let n denote the mini-switch port number, k denote the number of BCube network topology level. Then, BCube_0, which is made up of n servers and a mini-switch with n ports, is built as the basic unit of BCube. BCube_1 is composed of n BCube_0 and n n-port mini-switches. BCube_k is an interconnection between n BCube_{k-1} and n^k n-port mini-switches. The server in BCube_k has $k+1$ ports, numbered from 0 to k. Note that switches in BCube are only connected to servers. The BCube $(4, 1)$ topology is shown in Fig.5. It consists of 16 servers and 8 4-port mini-switches.

In BCube (n, b), we denote a server with an address array $s_0 s_1 s_2 ... s_b$ ($s_i \in [0, n-1], i \in [0, b]$) and a switch with $< l, w_0 w_1 ... w_{b-1} >$, where l represents the level of switch, $l \in [0, b]$, $w_i \in [0, n-1]$ and $i \in [0, b-1]$. In BCube, for any two servers that are connected to the l-th level, their address numbers are different in the l-th dimensional digit. Therefore, for servers S_1=$s_{10} s_{11} ... s_{1b}$ and S_2= $s_{20} s_{21} ... s_{2b}$, we can derive out the path between S_1 and S_2 by changing each digit step by step. For example, in BCube $(4, 1)$, if S_1=00 is the source and S_2=13 is the destination. The path from S_1 to S_2 can be constructed as path=$(00, 10, 13)$. Their path is from 00 to the server 13 through the server 10. According to this characteristic, if we want to achieve BCube multi-path transmission of data blocks, we can choose several nodes whose address numbers are mutually exclusive to store data blocks. In this way, for the last step of transmission, the data flows enter the destination node from different levels of

Algorithm 1 Generate a set of disjoint nodes

1: **Initialize** $node() = \phi$;
2: Except the source node, select a node S_1 from the node set randomly, $node(1) = S_1$;
3: **For** $i = 2$ **to** n
4: S_i = Each digit of S_{i-1}'s address number plus 1 (If the digit is equal or greater than $n-1$, set this digit to 0);
5: $node(i) = S_i$;
6: **EndFor**
7: **Return** $node()$

Algorithm 2 Calculate the paths for $b+1$ disjoint nodes

1: Sort the $b+1$ disjoint nodes based on their identifiers;
2: **For** $i = 1$ **to** $b+1$;
3: For the disjoint node in the i-th position, the address number of S_0 is the starting point;
4: Start with the i-th digit of S_0, change each digit until the address number is the same as the i-th disjoint node;
5: Record the address number after each change;
6: The address number sequence is the path from S_0 to the i-th disjoint node;
7: **EndFor**
8: **Return** all the paths

switches, which can effectively avoid the link overlap in the last step. If a corresponding method is used in the previous transmission process to avoid path conflicts, we can transmit multiple data blocks to any a node without congestion.

Definition 1: For any pair of nodes S_1=$s_{10}s_{11}s_{12}...s_{1b}$ *and* S_2=$s_{20}s_{21}s_{22}...s_{2b}$*, we call them* disjoint nodes *if each digit* $s_{1i} \neq s_{2i}$ $(s_{1i}, s_{2i} \in [0, n-1],\ i \in [0, b])$.

Take the BCube (4, 1) (shown in Fig.5) as an example, the servers 21 and 03 are *disjoint nodes* because 2 does not equal 0, and 1 does not equal 3. Similarly, we call the servers 00, 11, 22 and 33 a group of *disjoint nodes*. The servers 12 and 22 are not disjoint nodes since their second digits are the same.

Theorem 1. *In BCube* (n, b)*, there are up to* n *disjoint nodes in a group.*

Proof: For each node $s_0s_1s_2...s_b$, each digit s_i has n kinds of choice, from 0 to $n-1$. There is no effect between the different digit s_i and s_j $(i \neq j)$. Therefore, if the i-th digit, s_i, is different from each other, there are n possibilities at most, which means each group has n disjoint nodes. Algorithm 1 provides a method to find a set of disjoint nodes in BCube. ■

Theorem 2. *The data blocks in any* $b+1$ $(b+1 \leq n)$ *disjoint nodes can be transmitted to any a node via* $b+1$ *parallel links.*

Proof: Each node in BCube (n, b) is connected with $b+1$ links. For a node $s_0s_1s_2...s_b$, it can receive up to $b+1$ data flows at a time. If the $b+1$ links are not reused, there will be a data flow transmitted on each link. We achieve this goal using the sorting operation and an improved routing algorithm, as shown in Algorithm 2. Because each node's address number is mutually exclusive and each node is transformed at most b times, there is no link reused in the intermediate path of transmission. ■

Theorem 3. *The data block on any a disjoint node is transmitted to any a node only through* b *nodes at most.*

Proof: Each disjoint node use a different link to transfer data blocks. From the source node to the destination node, it needs to change $b+1$ digits at most. ■

Theorem 3 points out that compared with the traditional routing method, this method will not bring extra link resource consumption. Therefore, if we choose a group of disjoint nodes to store files, multiple blocks on the disjoint nodes can be written or read concurrently via the parallel paths. In addition, It can be seen from Algorithm 1 that the selection of a set of non-intersecting nodes is random, which is conducive to the load balance of the system. Next, we take the BCube $(4, 1)$ as an example, and introduce the design of the storage model.

According to the features of *disjoint node*, we leverage a group of disjoint nodes to store data blocks. Given the node 21 in BCube $(4, 1)$ as a source or destination node, the system will randomly select a node as an initial disjoint node according to Algorithm 1. If the selected node is 12, add 1 to each digit of its address number and get the nodes 23, 30, and 01 in turn. The nodes 12, 23, 30 and 01 form a group of disjoint nodes. We can store 4 data blocks on the 4 disjoint nodes respectively.

According to Theorem 2, any two data blocks in this group of disjoint nodes can be transmitted to any a node through two parallel paths. If we choose 01 and 12 to calculate their transmission paths, we first need to sort them according to Algorithm 2. Because the digits in the corresponding positions of the first node 01 and the second node 12 are not the same as those in node 21, the order remains the same. Next, the address number is transformed on the basis of the node 21. The path from node 01 to 21 is path=$\{21, 01\}$, and the path from node 12 to 21 is $path = \{21, 22, 12\}$, as shown in Fig.6.

For the writing operation, the node 21 can transmit 2 data blocks to nodes 01 and 12 at a time. When reading multiple data blocks, 2 data blocks from 01 and 12 can be transmitted to node 21 via parallel paths. In addition, the number of parallel paths is related to the hierarchy b of BCube. Hence, as the dimension of BCube increases, the number of parallel paths increases accordingly.

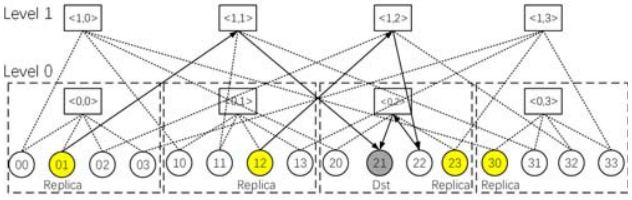


Fig. 6. One possible situation of node-disjoint storage strategy in BCube $(4, 1)$. The yellow parts 01, 12, 23 and 30 represent servers that can store data blocks. The node 21 is a source or destination node.

B. The NDSS in multi-replica system

The NDSS model can effectively reduce the times of transmission in BCube. In the NDSS model, the replicas are stored on the disjoint nodes. In this way, the source node can transmit $b+1$ replicas to $b+1$ disjoint nodes to write or

update a file. When $R \leq b+1$, the R-replica system can write or update all the replicas simultaneously. In fact, the BCube data centers are usually more than 3 layers ($b \geq 2$), hence, 3 replicas can be written or updated simultaneously. In addition, a destination node can simultaneously receive multiple replicas to compare them and avoid errors. This cannot be done by traditional methods.

We can select $b+1$ disjoint nodes for each group to store replicas. If the $b+1$ disjoint nodes are not enough, select another group of $b+1$ disjoint nodes. Because the number of replicas in the multi-replica system is not likely to be excessive, or it will consume a large amount of storage space, it is generally possible to complete the transmission of all replicas in $1 \sim 2$ logical timesteps.

NDSS makes it possible to simultaneously transmit multiple replicas in data centers, improving the efficiency of writing and updating a file. It is important for the storage systems with a large number of replicas. Moreover, the multi-path transmission can be implemented on any a node, meeting the actual needs of data centers.

C. The N-NDSS in erasure coding system

As mentioned above, the traditional transmission methods usually bring high network consumption when i) storing data blocks, ii) updating a file, iii) recovering a file and iv) repairing a failed data block. PPR improves the efficiency of repairing a failed data block through a partial-parallel-repair mechanism. However, there is not a better solution to improve the efficiency of i, ii and iii. In the server-centric data centers, each node is connected with multiple links, but the existing routing methods cannot find the parallel paths between a node and a group of nodes. Hence, it usually causes congestion on some links for a storage node to transfer or receive multiple data blocks.

In the erasure coding system, a file needs more storage nodes to store its data blocks. Hence, a group of n disjoint nodes in the NDSS model are usually not enough. In order to meet the needs of erasure coding system, we extend the NDSS model to N-NDSS.

Definition 2: For the nodes $S_1 = s_{10}s_{11}s_{12}...s_{1b}$ *and* $S_2 = s_{20}s_{21}s_{22}...s_{2b}$*, we call them* partial disjoint nodes *if more than two digits exist* $s_{1i} \neq s_{2i}$ $(s_{1i}, s_{2i} \in [0, n-1],\ i \in [0, b])$*, while other digits* $s_{1j} = s_{2j}$ $(s_{1j}, s_{2j} \in [0, n-1],\ j \in [0, b])$.

According to Definition 2, the nodes such as 001, 012, 023 and 030 in BCube $(4, 2)$ are a group of partial disjoint nodes. They are both 0 in the first digit and different in the other digits. For such a group of nodes, they have some properties similar to *disjoint nodes* .

Theorem 4. *For a group of partial disjoint nodes* $s_{10}s_{11}s_{12}...s_{1b}$, $s_{20}s_{21}s_{22}...s_{2b}$, ..., $s_{j0}s_{j1}s_{j2}...s_{jb}$, ... $(j \leq n)$*, if they have* d $(d \neq b+1)$ *disjoint digits and the rest of the digits is the same, then they have the same properties as the disjoint nodes with* d *digit lengths.*

Proof: When a group of partial disjoint nodes has d disjoint sections and $b+1-d$ common sections, we can view it as the dimension of their BCube topology is reduced. It turns out that a group of partial disjoint nodes of BCube (n, b) is now a group of disjoint nodes in BCube $(n, d-1)$, so they have the same properties. ■

With this feature, we propose the N-NDSS method to increase the number of storage nodes. We learn from the PPR mechanism to improve the transmission scheme of repairing a failed data block. Like PPR, this repair scheme has a basic requirement that the multiplication by the scalar decoding coefficients or a XOR between two items does not increase the size of the data during aggregation, as shown in Equ. 2.

1) Storing a file with N-NDSS: When storing a file, the N-NDSS chooses a group of disjoint nodes first. In BCube (n, b), there are n disjoint nodes. Then, select n groups of partial disjoint nodes according to the n disjoint nodes to form n^2 storage nodes. If they are not enough, continue to select the partial disjoint nodes. In general, even with the RS $(12, 4)$ code, a file only needs 16 storage nodes. Hence, it is enough when the size of BCube is greater than BCube $(4, 2)$. In reality, the scale is much larger than that.

We use an example to introduce the selection of storage nodes. In BCube $(4, 2)$, the first node 231 is randomly obtained. Then, based on the node 231, three disjoint nodes 302, 013 and 120 are obtained according to Algorithm 1. The nodes $(231, 302, 013, 120)$ are called the first group of disjoint nodes. Then, each node in this group is taken as the initial node to calculate n groups of partial disjoint nodes. Keep each disjoint node's first digit constant, add 1 to the other digits in turn, and n groups of partial disjoint nodes are obtained. The first group of n disjoint nodes is included in each group respectively. For the first node 231 in the first group of disjoint nodes, keep its first digit unchanged, add 1 to the last two digits, and get a *partial disjoint node* 202. By analogy, the other two partial disjoint nodes 213 and 220 are obtained.

The first group of partial disjoint nodes is 231, 202, 213 and 220. For the second node 302 in the first group of disjoint nodes, keep its first digit unchanged, add 1 to the last two digits, and get a *partial disjoint node* 313. By analogy, the other two partial disjoint nodes 320 and 331 are obtained. The first group of partial disjoint nodes is 302, 313, 320 and 331. Similarly, the third group of partial disjoint nodes is 013, 020, 031 and 002, and the last group is 120, 131, 102 and 113. According to Theory 4, each group of the partial disjoint nodes can be considered as a group of disjoint nodes in BCube $(4, 1)$. In this way, 4 disjoint nodes can be extended to 16 nodes in four groups. These 16 nodes can form 4 groups of disjoint nodes or 4 groups of partial disjoint nodes. The 16 nodes are 4 groups of disjoint nodes in the vertical view, or 4 groups of partial disjoint nodes in the horizontal view. The process can be described as Algorithm 3.

When storing a file, the source node first stores data blocks on the initial n disjoint nodes. If the nodes are not enough, or $n > 2 \times (b+1)$, the data is stored on their partial disjoint nodes. The data blocks are first stored on the first node of each group of partial disjoint nodes, then, on the second, and so on. In this way, the multi-path feature can be effectively

utilized to transfer more data blocks.

2) The design of transmitting data blocks: In the erasure coding system, when a file need to be stored, updated or recovered, the involved data blocks can be transmitted to each group of disjoint nodes via the parallel paths. In order to further improve the speed of repairing a failed data block, we optimize the transmission scheme referring to PPR.

Different from the previous transmission scheme in NDSS, we first need to determine the paths among partial disjoint nodes, then the paths of $b+1$ disjoint nodes to the repair node. The path selection method of partial disjoint nodes is similar to Algorithm 2. It is equivalent to sorting these partial disjoint nodes in a low-dimension BCube and change each digit in turn. When determining the transport paths from disjoint nodes to the destination node, we should avoid those links that have been occupied by the transmission of the partial disjoint nodes: if the link has been occupied, select other nodes connected to the switch at the same level. Since the transmission of partial disjoint nodes is in the sub-level BCube and occupies at most two links of a switch, there are still available links and nodes.

For the first logical timestep, it is necessary to determine the transmission paths of partial disjoint nodes first. Because these partial disjoint nodes are equivalent to disjoint nodes in BCube $(4, 1)$, two parallel paths can be used to transfer data blocks. Hence, in each rectangular frame, the two latter partial disjoint nodes transmit their data blocks to the first partial disjoint nodes in a sub-level BCube. The aggregation of 3 data blocks is performed on this first *partial disjoint node*. Similarly, in BCube $(4, 2)$, each node is connected with 3 links, so it can transfer or receive 3 data blocks simultaneously. We choose 3 nodes from the remaining group of nodes (the last node in each rectangular frame). It is obvious that those 3 nodes form a group of disjoint nodes. They transmit their 3 data blocks to the repair node, completing the first step of transmission.

If the repair node is 103, the paths based on partial disjoint nodes in Fig.7 include: $202 \rightarrow 201 \rightarrow 231$, $213 \rightarrow 233 \rightarrow 231$; $313 \rightarrow 312 \rightarrow 302$, $320 \rightarrow 300 \rightarrow 302$; and $020 \rightarrow 023 \rightarrow 013$, $031 \rightarrow 011 \rightarrow 013$. Similarly, the paths based on *disjoint nodes* include: $220 \rightarrow 223 \rightarrow 203 \rightarrow 103$; $331 \rightarrow 131 \rightarrow 133 \rightarrow 103$; and $002 \rightarrow 102 \rightarrow 103$.

Algorithm 3 Select a group of storage nodes with N-NDSS

```
Initialize node() = φ;
Select an initial node, node(1,1), from the whole node set randomly;
For i = 2 to n
    Add 1 to each digit of node(1, i−1)'s address number to get node(1, i);
EndFor
For i = 1 to n
    For j = 1 to n − 1
        Add j to the remaining digits to get node(j, i);
    EndFor
EndFor
Return node();
```

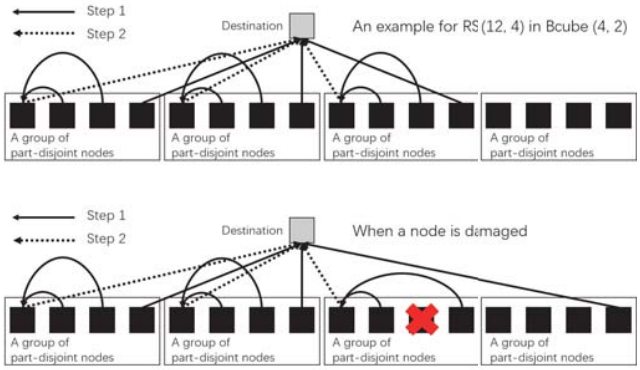


Fig. 7. An example of repairing a failed data block with RS $(12, 4)$.

For the second logical timestep, the 3 aggregated data blocks on the first group of disjoint nodes 231, 302 and 013 are transmitted to the repair node. The paths of the example above are: $231 \rightarrow 233 \rightarrow 203 \rightarrow 103$, $013 \rightarrow 113 \rightarrow 103$ and $302 \rightarrow 102 \rightarrow 103$ (the change of order from 231, 302, 013 to 231, 013, 302 is caused by the sorting operation, see Algorithm 2 for details). Because the size of the aggregated data blocks remains unchanged according to Formula 2, their transmission time does not increase.

In general, it is the failure of some data blocks that triggers the repair operation. Because the position of a failed data block is random, the transmission paths of the repair process may not be exactly the same as the example. The latter half of Fig.7 shows a transmission scheme when another data block is failed. It can be seen that we can still adjust the transmission scheme to minimize the transmission time. When a failure of the data block occurs again, we take the repair node as an initial node, and select its disjoint node to repair the failed data block. This is equivalent to moving the original data blocks to other disjoint nodes and their partial disjoint nodes.

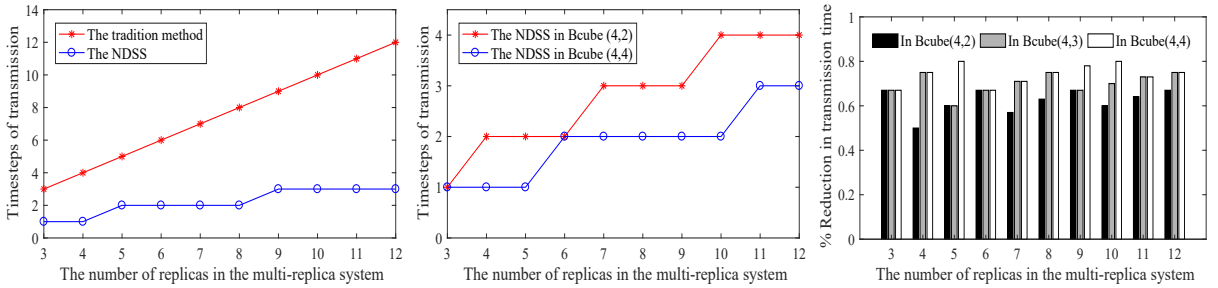
IV. Performance Evaluation

In this section, we evaluate the performance of node-disjoint strategy in different BCube networks.

A. Evaluation settings

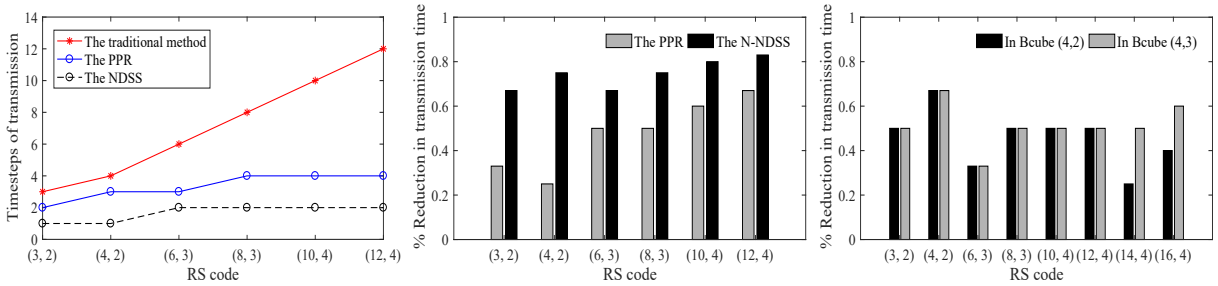
We simulate these methods using an Intel i7-6700HQ laptop with a 2.6GHz main frequency processor and 8GB memory. The number of layers in the BCube network is the main factor affecting the transmission efficiency of NDSS and N-NDSS. The higher the level of the BCube network, the more links are connected to each node, and there are more parallel paths to transport data blocks in parallel. Hence, to make the experiment more general, we keep the number of servers per pod unchanged, only changing the number of layers. The simulation evaluates the performance of NDSS, N-NDSS and other methods in BCube $(4, b)$.

The simulation evaluates the access performance by comparing the number of logical timesteps. We define a *timestep* for a data block to be transmitted from the source node to the destination node. Each timestep takes the same amount of time



(a) # of timesteps for file storing and update. (b) # of timesteps in different BCubes. (c) Transmission efficiency in different BCubes.

Fig. 8. The performance of our methods in multi-replica system with diverse parameter settings.



(a) # of timesteps for data block repairing. (b) Transmission efficiency comparison. (c) Transmission efficiency in different BCubes.

Fig. 9. The performance of our methods in erasure coding system with diverse parameter settings.

because the data blocks transferred over the links are of the same size. In addition, for the sake of comparison, we assume that the bandwidth in data centers is uniform.

In the repair process of a data block, the time is roughly divided into the network transmission time, the disk reading time and the computing time, among which the network transmission time accounts for the largest proportion, up to about 94% [2][3][4]. Therefore, when comparing the repair time, we can generally ignore the computing time, and use the number of timesteps as the indicator.

B. Performance in the multi-replica system

In the multi-replica system, the traditional method requires transferring replicas from the source node to each destination node separately, or it may lead to the congestion on some links. Although the optimized HDFS storage strategy uses the characteristic that bandwidth in a rack is greater than the bandwidth between racks, saving two replicas in the same rack, its improvement is still very limited. Moreover, this method will reduce the reliability of data storage, because the racks still have a certain probability to damage. If a rack fails, it is likely to lead to the loss of 2 replicas.

In BCube $(4, 3)$, the Fig. 8(a) compares the number of timesteps required for storing or updating a file under the traditional method and the NDSS method. It can be seen that the number of timesteps required by the traditional method increases linearly along with replicas increasing. By contrast, the NDSS method is quite different. In BCube $(4, 3)$, there are 4 parallel paths. The source node can transmit up to 4 data blocks to 4 storage nodes simultaneously. Hence, its time consumption grows slowly. In addition, as can be seen from the figure, the performance improvement brought by the NDSS method becomes more and more obvious with the increase of replicas.

In the BCube network, this gain is closely related to the number of BCube levels. Fig. 8(b) shows the comparison of the NDSS method in BCube $(4, 2)$ and BCube $(4, 4)$. It can be seen that the more network levels, the fewer required transmission timesteps. Fig. 8(c) further compares the transmission efficiency of NDSS under BCube networks of different layers with the traditional method. In different multi-replica systems, the time reduction with NDSS is more than a half. In BCube $(4, 4)$, the transmission time can be reduced by up to 80%.

C. Performance in the erasure coding system

When storing or updating a file, the source node transfers the $k + m$ data blocks to each storage node respectively with the traditional method. When a file needs to be recovered, k involved storage nodes send their k data blocks to the destination node simultaneously. The result of this method is the congestion on some links. The N-NDSS realizes the parallel transmission of multiple data blocks by searching the parallel paths between a node and multiple nodes, increasing the transmission efficiency. Its performance improvement can refer to the simulations in the multi-replica system. Therefore, we mainly evaluate its performance of repairing a failed data block in this section.

Like the recovery of a file, the k involved data blocks are transmitted to the repair node simultaneously with the traditional method. By contrast, PPR enables the parallel

transmission of multiple data blocks by directing the data flows to different storage nodes for aggregation. However, compared with the N-NDSS, the PPR still has a lot of room for improvement.

Fig. 9(a) compares the number of timesteps of repairing a failed data block required by the traditional method, the PPR and the N-NDSS. As can be seen from the figure, the number of timesteps to repair a failed data block required by the traditional method is still increased linearly. PPR makes use of the partial-parallel-repair mechanism to reduce the number of timesteps significantly. However, the N-NDSS can repair a failed data block with smaller number of timesteps.

As shown in Fig. 9(b), the PPR improves the transmission efficiency by more than 25% compared with the traditional method, but this performance improvement is still limited compared with the N-NDSS. The N-NDSS reduces the required number of timesteps via the parallel paths. Compared with the traditional method, the N-NDSS improve the efficiency of repairing a failed data block by more than 65%, and the maximum effect is as high as 83%. In addition, it can be seen from the figure that compared with the PPR, the N-NDSS can achieve better results when there are fewer data blocks. This is because the number of parallel paths is $b + 1$. When the number of data blocks is small, the N-NDSS only needs a timestep to finish the transmission of multiple data blocks.

In order further to evaluate the performance improvement, we compared the performance of the N-NDSS in different BCube networks with the PPR, the result is shown in Fig. 9(c). It can be seen that even if the PPR has greatly improved the repair efficiency, the N-NDSS can still increase the transmission efficiency by more than 25% again, with a maximum of 65% in BCube $(4, 3)$.

As a summary, we conclude that the NDSS and N-NDSS outperform the existing methods in terms of timesteps and transmission efficiency.

V. Conclusion

In this paper, we note that the current file storage strategies (including replication and erasure coding) fail to utilize the parallel paths offered by the modern data center topologies. There is a mismatch between the offered parallel paths in DCNs and the file placement strategy which relays on overlapped links during file storing, repairing and recovering. This mismatch, however, results in vast transmission overhead and non-trivial time-consumption in distributed storage systems. Therefore, in this paper, we define the concept of *disjoint node* in BCube (a representative server-centric DCN). Based on that definition, we propose the NDSS model and N-NDSS model for multi-replicas system and erasure coding system, respectively. By storing the replicas or data blocks on disjoint and partial disjoint nodes, NDSS and N-NDSS utilize the parallel paths in BCube when file storing, recovering and repairing are triggered. The simulation shows that the NDSS and N-NDSS outperform the existing methods in terms of timesteps and transmission efficiency.

VI. Acknowledgement

This work is partially supported by National Natural Science Foundation of China under Grant No.61772544, National Basic Research Program (973 program) under Grant No.2014CB347800, the Hunan Provincial Natural Science Fund for Distinguished Young Scholars under Grant No.2016JJ1002, and the Guangxi Cooperative Innovation Center of cloud computing and Big Data under Grant Nos.YD16507 and YD17X11.

References

[1] K. V. Rashmi, D. Borthakur, D. Borthakur, D. Borthakur, D. Borthakur, and K. Ramchandran, "A solution to the network challenges of data recovery in erasure-coded distributed storage systems: a study on the facebook warehouse cluster," in *Usenix Conference on Hot Topics in Storage and File Systems*, 2013, pp. 8–8.

[2] M. Silberstein, L. Ganesh, Y. Wang, L. Alvisi, and M. Dahlin, "Lazy means smart:reducing repair bandwidth costs in erasure-coded distributed storage," 2014, pp. 1–7.

[3] K. V. Rashmi, P. Nakkiran, J. Wang, N. B. Shah, and K. Ramchandran, "Having your cake and eating it too: jointly optimal erasure codes for i/o, storage and network-bandwidth," in *Usenix Conference on File and Storage Technologies*, 2015, pp. 81–94.

[4] J. Li and B. Li, "Beehive: erasure codes for fixing multiple failures in distributed storage systems," in *Usenix Conference on Hot Topics in Storage and File Systems*, 2015, pp. 6–6.

[5] S. Mitra, R. K. Panta, M. Ra, and S. Bagchi, "Partial-parallel-repair (PPR): a distributed technique for repairing erasure coded storage," in *Proceedings of the Eleventh European Conference on Computer Systems*, 2016, pp. 30:1–30:16.

[6] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube:a high performance, server-centric network architecture for modular data centers," 2009, pp. 63–74.

[7] Z. Ding, D. Guo, X. Liu, X. Luo, and G. Chen, "A mapreduce-supported network structure for data centers," *Concurrency & Computation Practice & Experience*, vol. 24, no. 12, pp. 1271–1295, 2012.

[8] L. Luo, D. Guo, J. Wu, T. Qu, T. Chen, and X. Luo, "Vlccube: A vlc enabled hybrid network structure for data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 2088–2102, 2017.

[9] "Sun's modular datacenter," http://docs.oracle.com/cd/E19115-01/mod.dc.s20/index.html.

[10] "Hp performance optimized datacenter (pod)," http://h18000.www1.hp.com/products/servers/solutions/datacentersolutions/pod/.

[11] "Ibm portable modular data center," http://www-935.ibm.com/services/us/en/it-services/data-center/modular-data-center/index.html.

[12] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *IEEE Symposium on MASS Storage Systems and Technologies*, 2010, pp. 1–10.

[13] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, "Ceph: a scalable, high-performance distributed file system," in *Symposium on Operating Systems Design and Implementation*, 2006, pp. 307–320.

[14] "Openstack object storage (swift)," http://ceph.com/.

[15] B. Fan, W. Tantisiriroj, L. Xiao, and G. Gibson, "Diskreduce: Raid for data-intensive scalable computing," in *The Workshop on Petascale Data Storage*, 2009, pp. 6–10.

[16] "Colossus," http://www.quora.com/Colossus-Google-GFS2.

[17] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. Mckelvie, Y. Xu, S. Srivastav, J. Wu, and H. Simitci, "Windows azure storage: a highly available cloud storage service with strong consistency," in *ACM Symposium on Operating Systems Principles*, 2011, pp. 143–157.

[18] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial & Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[19] A. Björck and V. Pereyra, "Solution of vandermonde systems of equations," *Mathematics of Computation*, vol. 24, no. 112, pp. 893–903, 1970.

[20] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Project Website*, vol. 11, no. 11, pp. 1 – 10, 2007.