BDR: A Balanced Data Redistribution Scheme to Accelerate the Scaling Process of XOR-based Triple Disk Failure Tolerant Arrays*

Yanbing Jiang[1], Chentao Wu[1†], Jie Li[1,2], and Minyi Guo[1]
[1]Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
[2]Department of Computer Science, University of Tsukuba, Tsukuba, Ibaraki, 205-8577, Japan
Corresponding author: †wuct@cs.sjtu.edu.cn

***Abstract*—In large scale data centers, with the increasing amount of user data, Triple Disk Failure Tolerant arrays (3DFTs) gain much popularity due to their high reliability and low monetary cost. With the development of cloud computing, scalability becomes a challenging issue for disk arrays like 3DFTs. Although previous solutions improves the efficiency of RAID scaling, they suffer many problems (high I/O overhead and long migration time) in 3DFTs. It is because that existing approaches have to cost plenty of migration I/Os on balancing the data distribution according to the complex layout of erasure codes.**

To address this problem, we propose a novel Balanced Data Redistribution scheme (BDR) to accelerate the scaling process, which can be applied on XOR-based 3DFTs. BDR migrates proper data blocks according to a global point of view on a stripe set, which guarantees uniform data distribution and a small number of data movements. To demonstrate the effectiveness of BDR, we conduct several evaluations and simulations. The results show that, compared to typical RAID scaling approaches like Round-Robin (RR), SDM and RS6, BDR reduces the scaling I/Os by up to 77.45%, which speeds up the scaling process of 3DFTs by up to $4.17\times$, $3.31\times$, $3.88\times$, respectively.

***Index Terms*—RAID; Erasure Code; Triple Disk Failures; Scalability; Reliability; Evaluation**

I. Introduction

In large scale data centers, massive data are stored in large amounts of redundant disks to facilitate users to access them in parallel. These redundant disks are called **R**edundant **A**rrays of **I**nexpensive (or **I**ndependent) **D**isks (**RAID**) [19]. Recently, disk arrays for correcting triple disk failures (3DFTs) become one of the most popular choices for large scale data centers because they can provide high reliability with low monetary cost [31][23][27][6][7][4].

In the last two decades, many erasure codes are proposed to correct triple disk failures of disk arrays, including **M**aximum **D**istance **S**eparable (**MDS**) codes [21][3][20][14][27][26][35][16] and non-MDS codes [25][10][11][12][13][2][23]. MDS codes supply the maximum protection with a given amount of redundancy [14]. Several popular codes like STAR code [14], Triple-Star code [27], TIP-code [35] and EH-Code [16] are encoded with three different parities (horizontal, diagonal and anti-diagonal parities) to provide high reliability. Non-MDS codes achieve higher performance or reliability by sacrificing the storage efficiency [13][24].

In recent years, with the development of cloud computing, scalability of disk arrays [1] is highly desired due to following reasons. (1) Adding extra disks into 3DFTs based on MDS codes can increase storage efficiency to save monetary cost. (2) A disk array with new added disks provides higher I/O throughput and larger capacity [32][29][30]. More data can be accessed in parallel which decreases I/O cost and improves the entire performance. (3) 3DFTs are widely used in large scale data centers for high reliability, where scalability plays a significant role [15][22].

Unfortunately, existing scaling methods are insufficient for XOR-based 3DFTs. Some methods (e.g., Round-Robin (RR) [9][18][33] and Semi-RR [8]) have been demonstrated that they have high I/O cost of data migration and parity modification on scaling [36][28][29]. Semi-RR suffers unbalanced data distribution after scaling as well [28][29]. Other approaches spend a large amount of I/Os on balancing the data distribution in the complex layout of erasure codes, such as SDM [29] and RS6 [34]. SDM sacrifices a few stripes to achieve uniform data distribution in a stripe set, and deals with the codes encoded by two different parities (there are three different parities in XOR-based 3DFTs), which results in high parity modification and I/O cost. RS6 is designed to accelerate RAID-6 scaling only for RDP code, which leads to additional data migration I/Os. It suffers uneven data distribution for several other codes as well, such as STAR code [14] and TIP-code [35]. Some other approaches like FastScale [36], GSR [28], CRAID [17] and POS [30] are designed for RAID-0 or RAID-5 arrays, and which cannot be applied for 3DFTs.

To address the above problem, we propose a novel **B**alanced **D**ata **R**edistribution scheme (BDR), which can be applied on XOR-based 3DFTs based on MDS codes with three different parities (horizontal, diagonal and anti-diagonal parities). BDR migrates proper data blocks based on the relationships between different parities to achieve minimal data migration and parity modification. Then BDR merges the stripes with empty data blocks to decrease the I/O cost of parity modification. Therefore, BDR can guarantee a uniform data distribution and a small amount of total I/O operations.

We make the following contributions in this paper,

- We propose a **B**alanced **D**ata **R**edistribution scheme (BDR), which speeds up the scaling process of 3DFTs

*This work is partially sponsored by the National 973 Program of China (No.2015CB352403), the National 863 Program of China (No. 2015AA015302), the National Natural Science Foundation of China (NSFC) (No. 61332001, No. 61303012, and No. 61572323), the Scientific Research Foundation for the Returned Overseas Chinese Scholars, and the CCF-Tencent Open Fund.

978-1-5090-5142-7/16/$31.00 ©2016 IEEE

TABLE I
A LIST OF SYMBOLS IN THIS PAPER

Symbols	Description
n	number of disks in a disk array before scaling
m	number of extended disk(s)
i, i'	row ID before/after scaling
j, j'	disk ID before/after scaling
$C_{i,j}$	a block at the ith row and jth disk (column)
B	total number of data blocks
n_d	number of data disks in a disk array before scaling
n_r	number of rows in a stripe before scaling
n_s	number of stripes in a stripe set
n_e	total number of empty rows in a stripe set
L	the smallest number of empty rows in a stripe

based on MDS codes. It offers low I/O overhead on data migration and parity modification.

- Compared with several general schemes, we conduct evaluations and simulations to demonstrate the effectiveness of BDR in 3DFTs.

This paper is organized as follows, Section II briefly overviews the background and related work. BDR is illustrated in detail in Section III. The evaluations are given in Section IV. Finally we conclude this paper in Section V.

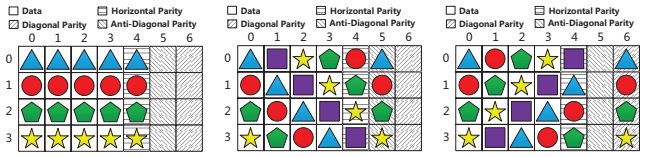
II. RELATED WORK AND OUR MOTIVATION

In this section, we discuss the related work of the existing scaling schemes and the scaling problems in XOR-based 3DFTs based on MDS codes. To facilitate our discussion, we summarize the symbols used in this paper in Table I.

A. Popular MDS Codes in 3DFTs

Several popular MDS codes in 3DFTs, such as STAR code [14], Triple-Star code [27], TIP-code [35] and EH-Code [16], use three different parities (horizontal, diagonal and anti-diagonal parities) and XOR operations to provide enough reliability for 3DFTs.

In this paper, we use Triple-Star code in Figure 1 as an example to describe the scaling process, and compare merits and weakness among various approaches. Figures 1(a)1(b)1(c) show the encodings of horizontal, anti-diagonal and diagonal parities in Triple-Star code, respectively. Different parity blocks and their corresponding data blocks are distinguished by different shapes. The value of each parity block is the sum of its corresponding data blocks through XOR operations.



(a) Horizontal parity coding. For example, $C_{0,4} = C_{0,0} \oplus C_{0,1} \oplus C_{0,2} \oplus C_{0,3}$.

(b) Anti-diagonal parity coding. For example, $C_{0,5} = C_{0,0} \oplus C_{1,1} \oplus C_{2,2} \oplus C_{3,3}$.

(c) Diagonal parity coding. For example, $C_{0,6} = C_{0,0} \oplus C_{3,2} \oplus C_{2,3} \oplus C_{1,4}$.

Fig. 1. Triple-Star Code ($p = 5$).

B. Desired Scaling Features in 3DFTs

To scale a disk array, some data need to be migrated to the new disk(s) to achieve an even data distribution. During data migration, we prefer to keep an evenly distributed workload and minimize the data/parity movement. Four features are typically desired [29][28]. They are (1) Uniform Data Distribution, (2) Minimal Data Migration, (3) Fast Data Addressing, (4) Minimal Parity Computation & Modification.

C. Existing Scaling Methods

(1) Round-Robin (RR): For a traditional RR scaling approach (as shown in Figure 2), all data blocks are migrated and all parities need to be modified and recalculated after data migration. Although RR is a simple approach to be integrated in 3DFTs, it brings high I/O cost during the scaling process.

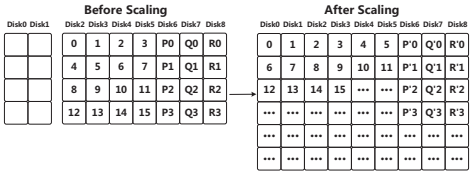


Fig. 2. The scaling of 3DFTs using Triple-Star Code from 7 to 9 disks using RR approach (All data blocks are migrated and all parities are modified).

(2) SDM: SDM scheme is designed for RAID-6 codes shown in Figure 3, which sharply decreases the migration/modification cost. It achieves uniform data distribution in a stripe set (several stripes) by sacrificing a small portion of stripes, but for each stripe, data distribution is not uniform. In the scaling process of 3DFTs, "phantom" rows (the last two rows of the stripe after scaling in Figure 3) increase the number of parity modification because all anti-diagonal parities in Disk 7 in Figure 3 need to be modified.

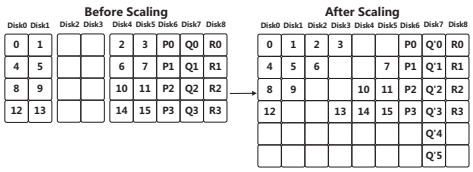


Fig. 3. The scaling of 3DFTs using Triple-Star Code from 7 to 9 disks using SDM scheme (Each stripe suffers unbalanced data distribution and many anti-diagonal parities need to be modified, which increases the parity modification).

(3) RS6: RS6 is designed to accelerate RAID-6 scaling only for RDP code, which suffers unbalanced data distribution for some codes (such as STAR code [14] and TIP-code [35]). Moreover, it uses Round-Robin method to fill the diagonal empty part shown in Figure 4, which leads to additional data migration and high I/O cost.

(4) Semi-RR: Semi-RR [8] is proposed to decrease high migration cost in RR scaling, but it still has the problem of high I/O cost of data migration and parity modification. It suffers unbalanced data distribution after scaling as well, which dissatisfies the feature of Uniform Data Distribution [28][29].

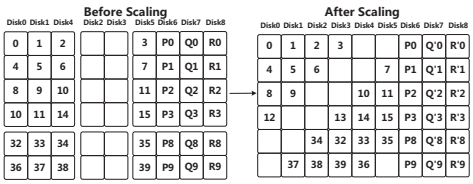


Fig. 4. The scaling of 3DFTs using Triple-Star Code from 7 to 9 disks using RS6 scheme (Filling the diagonal part in a round-robin order leads to extra data migration and high I/O cost).

TABLE II
SUMMARY ON VARIOUS SCALING APPROACHES

Name	Total amount of I/Os	Uniform data distribution?	Used in 3DFTs?
RR	large	√	√
Semi-RR	large	×	√
SDM	medium	√	√
RS6	medium	conditionally	√
FastScale	medium	√	×
GSR	medium	√	×
CRAID	medium	√	×
POS	medium	√	×
BDR	small	√	√

(5) Other approaches: FastScale [36], GSR [28], CRAID [17] and POS [30] are designed for RAID-0 or RAID-5 systems and cannot be used in RAID-6 or 3DFTs.

D. Our Motivation

We summarize the existing scaling methods in Table II, which shows that they are not efficient in 3DFTs. First, the I/O cost of Round-Robin and Semi-RR methods is extremely high. Semi-RR suffers unbalanced data distribution after scaling as well. Second, SDM achieves uniform data distribution with one stripe set, but the data distribution in each stripe is unbalanced. On the other hand, additional parities (like anti-diagonal parities in Triple-Star code) need to be modified used in 3DFTs. Third, RS6 method increases the number of data migration because it fills the empty blocks in the stripes in a round-robin order. In some codes like STAR code and TIP-code, RS6 results in unbalanced data distribution due to odd number of data disks.

In summary, existing scaling approaches are insufficient for the scaling process in 3DFTs, which motivates us to propose a new approach on 3DFTs scaling.

III. BALANCED DATA REDISTRIBUTION SCHEME (BDR)

In this section, a **B**alanced **D**ata **R**edistribution scheme (BDR) is designed to accelerate the scaling process of XOR-based 3DFTs. The purpose of BDR is to minimize the number of I/O operations by decreasing the modification of parity chains. BDR keeps one stripe set in perspective, not limited to a migration on single data/parity block as Round-Robin.

The main steps of BDR can be described as the following three steps which is shown in Figure 5.

- **Step 1 (Preparation Step):** BDR collects the parameters from the storage systems, and calculates the necessary parameters to prepare for the next two steps.
- **Step 2 (Migration Step):** BDR selects a proper number of stripes as a stripe set, and picks up several rows in each stripe to migrate to new disks based on the parameters in Step 1. In the migration process, the data movements are based on the relevances between different parities, which guarantees uniform data distribution.
- **Step 3 (Merging Step):** BDR aggregates empty stripes to decrease the number of parity modification and reduce I/O cost.

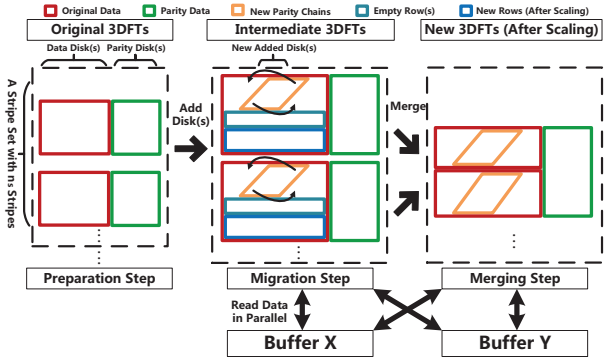


Fig. 5. Overview of BDR (including the Preparation Step, the Migration Step and the Merging Step).

A. Preparation Step

In this preparation step, BDR calculates three parameters to prepare for the next two steps.

According to the layout of an MDS code and its parameters which are used in 3DFTs, we can calculate n_s, n_e and L which are defined in Table I by the following equations. As shown in Figure 5, one stripe set contains several stripes. In each stripe, some rows are set to "empty rows", which are migrated to other disks for even data distribution.

$$n_s = \frac{lcm\{m \times n_r, n_d + m\}}{m \times n_r} \tag{1}$$

$$n_e = \frac{lcm\{m \times n_r, n_d + m\}}{n_d + m} \tag{2}$$

$$L = \begin{cases} 1 & m \times n_r \leq n_d + m \\ floor(\frac{m \times n_r}{n_d + m}) & m \times n_r > n_d + m \end{cases} \tag{3}$$

For example, the scaling of 3DFTs using Triple-Star Code from 7 to 9 disks is shown in Figure 6 where $n_d = 4$, $n_r = 4$ and $m = 2$. According to Equations 1 to 3, $n_s = 3, n_e = 4, L = 1$. It means that 3 stripes and in total 4 empty rows are in one stripe set. Each stripe has at least one empty row. Thus, after scaling in Figure 6, the number of empty rows in the stripe set is 1, 1, 2 ($1 + 1 + 2 = 4 = n_e$), respectively.

B. Migration Step

This step migrates proper data blocks in a stripe set after acquiring the parameters from the above preparation step. It can be described as the following three procedures.

(1) Read the stripe set into the buffers. To reduce the number of read I/O operations in scaling process, BDR reads the data/parity blocks of the entire stripe set into Buffer X in memory at once. The selected empty rows of the stripe set are put into Buffer Y. In a stripe set, reading data in a stripe on a disk is one sequence read which corresponds to one read I/O.

(2) Migrate data blocks. BDR migrates proper data blocks based on the minimum modifications of parity blocks. According to the migration cost of different blocks, BDR preferentially changes the position of data blocks with low migration overhead, which satisfies fast addressing feature in Section II. Small migration cost of blocks means low overhead of modification and computation involved in the migration process, which requires that blocks are resident in their original parity chains. It insures that BDR modifies the fewest parity blocks. Due to the page limit, we only give Algorithm 1, which shows the migration algorithm based on the horizontal and diagonal parities in STAR code and Triple-Star code. Row $r_b = \lfloor \{n_r - (L+1)\}/2 \rfloor$ is regarded as the base row, which does not migrate any data block in the horizontal direction.

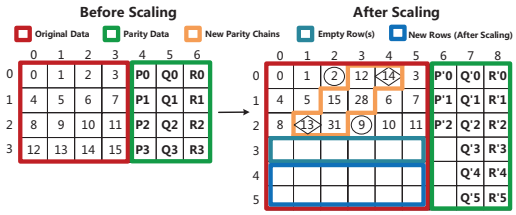
Then BDR changes the mapping relationships between the empty data blocks and the blocks in the empty rows in a same disk. The remaining data blocks in the empty rows are written into the empty blocks in a round-robin order.

Algorithm 1: BDR Migration Algorithm Based on the Horizontal and Diagonal Parities for STAR code and Triple-Star code.

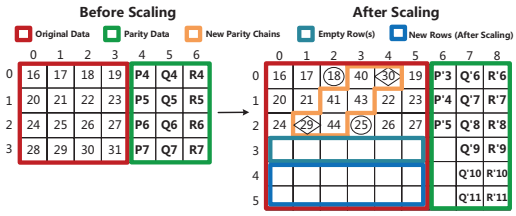
Get or calculate the n**,** m**,** n_r**,** n_d **and** L **value, then label the new disks with column IDs from** $\lceil \frac{n_d}{2} \rceil$ **to** $\lceil \frac{n_d}{2} \rceil + m - 1$
set $r_b = \lfloor \{n_r - (L+1)\}/2 \rfloor$
if $r_b + \lceil \frac{n_d}{2} \rceil \leq i + j \leq r_b + (\lceil \frac{n_d}{2} \rceil + m - 1)$
and $(j < \lceil \frac{n_d}{2} \rceil || j > \lceil \frac{n_d}{2} \rceil + m - 1)$ *and* i *is not empty row ID* **then**
  if $i < r_b$ **then**
    $i' = i, j' = j - m$;
  end
  else if $i > r_b$ **then**
    $i' = i, j' = j + m$.
  end
end
Change the mapping relationships between blocks in the same disk. Write the remaining ones into empty blocks in a round-robin order.

For example, in Figure 6, $r_b = \lfloor \{n_r - (L+1)\}/2 \rfloor = 1$, which means row 1 does not migrate any data block in each stripe. New disks are labeled with column IDs from $\lceil \frac{n_d}{2} \rceil = 2$ to $\lceil \frac{n_d}{2} \rceil + m - 1 = 3$. After inserting new disks, block 2 ($C_{0,2}$) are represented by $C_{0,4}$. Since block 2 ($C_{0,4}$, $i = 0, j = 4$) satisfies the condition in Algorithm 1, BDR migrates it to $C_{0,2}$ ($i' = 0$, $j' = 2$). Similarly, BDR moves data blocks in circular marks (e.g., blocks 9, 18 and 25, etc) to different disks, which share their original horizontal/diagonal parity chains as well. Next, those data blocks with diamond label (such as block 14, block 13 and block 30, etc) are chosen from Buffer Y to fill in the empty blocks. Then, the remaining data blocks (e.g, blocks 12, 15, 28 and 31 in Figure 6) in Buffer Y are written into the empty blocks in a round-robin order.

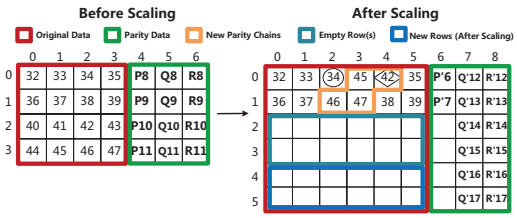
(3) Update and recalculate Parities. Three kinds of parities need to be updated or recalculated. Horizontal parities are written into parity disk at once and the other two parities are sent to Buffer X.



(a) Stripe 0 in the stripe set before/after scaling.



(b) Stripe 1 in the stripe set before/after scaling.



(c) Stripe 2 in the stripe set before/after scaling.

Fig. 6. An example of migration process of 3DFTs within a stripe set based on Triple-Star code using BDR (Scaling from 7 disks to 9 disks, Stripe 0, 1 and 2 are in a stripe set). According to Algorithm 1, blocks 2, 9, 18, 25 and 34 are migrated in rows with minimum parity modifications. The mappings of blocks 13, 14, 29, 30 and 42 are changed to reduce I/O cost. Other blocks in new added disks are migrated from Buffer Y in a round-robin order.

Update Horizontal parities: We calculate the sum of XOR operations of the original horizontal parities and new added data block to update the horizontal parities. Then we write them into the horizontal parity disk at once. For example, $P'_0 = P_0 \oplus V_{12} \oplus V_{14}$, V_{12} and V_{14} mean the value of blocks 12 and 14 in Figure 6.

Update Diagonal parities: Updating the new diagonal parities needs to compute the old diagonal parities, the corresponding data blocks in the empty rows and the corresponding updated horizontal parities through XOR operations. New diagonal parities of the diagonal part need to be calculated by the data in Buffer X. For example, $R'_6 = R_4 \oplus V_{30} \oplus P_5 \oplus P'_4$. Next, put the diagonal parities into Buffer X.

Recalculate Anti-diagonal parities: According to the encoding equations of erasure codes, we recalculate the anti-diagonal parities using the data blocks in Buffer X and send the diagonal parities to Buffer X.

So far, the scaling process in a stripe set is already done and if some data are lost during the scaling process, they can be recovered by using Buffers X and Y.

C. Merging Step

Merging step is typically desired because it can decrease the parity modification I/Os. In BDR, a stripe with empty rows (or empty blocks) still needs the same amount of parities as one full stripe. Therefore, we merge stripes to fill in the empty blocks in order to make the best of all parities and reduce parity modification and I/O cost.

The procedures of merging stripes is shown as follows,

- BDR first determines which stripes are merged or separated to keep the computational overhead small.
- BDR merges stripes within each stripe set.
- BDR merges the remaining stripes (rows) with the stripes (rows) in other stripe sets.

1) Determine which stripes are merged or separated: After the migration step, the row numbers of the stripes in one stripe set is known. BDR selects several stripes which are matched to be merged into a full stripe exactly. If there is no suitable stripe, BDR separates stripes, which computation cost is affordable according to the reduced I/O cost.

In the example of the scaling of 3DFTs using Triple-Star Code from 7 to 9 disks, stripes 0 and 1 are merged into a full stripe as shown in Figure 7. Stripe 2 needs to merged with two stripes 2 in other stripe sets (Due to the page limit, the figure is not shown here).

2) Merge stripes within or among stripe sets: The process of merging stripes within or among stripe sets can be divided into two parts:

- Merging data blocks and horizontal parities.
- Merging diagonal and anti-diagonal parities and write them into parity disks at once.

In the first part, the operation of merging data blocks and horizontal parities only changes the row labels of data blocks and horizontal parities, which does not bear additional I/O cost. Block $C_{i,j}$ (stripe a) is mapped to block $C_{i+x,j}$ (stripe b) and x is the number of rows (excluding empty ones) in stripe b. As shown in Figures 6 7, block $C_{1,2}$ in stripe 1 are mapped to the position of block $C_{4,2}$ in stripe 0 and $x = 3$.

In the second part, we merge the diagonal and anti-diagonal parities and write them into parity disks at once. We simply calculate two parities through XOR operations. Diagonal parity block $C_{i,p+1}$ (stripe a) adds the value of $C_{\langle i-x\rangle_p,p+1}$ (stripe b) to get a new value of $C_{i,p+1}$ and x is the total number of rows (excluding empty ones) in stripe a. That is $C_{i,p+1} = C_{i,p+1} \oplus C_{\langle i-x\rangle_p,p+1}$. For example in Figures 6 7, $R'_0 = V_0 \oplus V_{11} \oplus P'_1 = C_{0,8}$ in stripe 0 and $R'_{10} = V_{44} \oplus V_{43} \oplus V_{30} = C_{4,8}$ in stripe 1. After the first part of merging parts, $R'_0 = (V_0 \oplus V_{11} \oplus P'_1) \oplus (V_{44} \oplus V_{43} \oplus V_{30})$. Therefore, $C_{0,8} = C_{0,8} \oplus C_{4,8}$, $x = 3$, $p = 7$. The calculation of anti-diagonal parities is similar to that of diagonal parities. Then BDR sends them into parity disks and frees the buffers. These above operation are done in parallel, which speeds up the scaling process.

IV. Evaluation

We evaluate the scalability of popular MDS codes used in 3DFTs by using different approaches in this section.

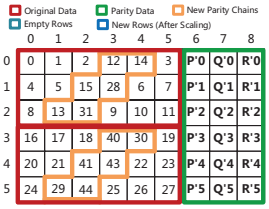


Fig. 7. An example of merging stripe of 3DFTs using Triple-Star code under BDR (Scaling from 7 disks to 9 disks, Stripes 0 and 1 are merged).

A. Evaluation Methodology

We compare the general methods Round-Robin (RR), SDM and RS6 in the evaluation. FastScale, GSR, CRAID and POS cannot be applied in 3DFTs, so that they are excluded from this evaluation. Semi-RR is not evaluated in this paper because the data distribution is not uniform after scaling.

Several popular MDS codes in 3DFTs are selected for comparison,

(1) Codes for $p+1$ disks: TIP-code [35];

(2) Codes for $p+2$ disks: Triple-Star code [27] and EH-Code [16];

(3) Codes for $p+3$ disks: STAR code [14].

We selects the following metrics in our evaluation,

(1) Data Migration Ratio (R_d): the ratio between the number of migrated data/parity blocks and the total number of data blocks.

(2) Parity Modification Ratio (R_p): the ratio between the number of modified parity blocks and the total number of data blocks.

(3) Computation Cost: the ratio between the number of XOR operations and the total number of data blocks.

(4) Scaling I/O Ratio: the ratio between the total number of scaling I/Os and the total number of data blocks.

(5) Spatial Overhead: the memory size utilized as buffer in the scaling process.

For example, the scaling of 3DFTs using Triple-Star Code from 7 to 9 disks ($p = 5$) migrates 16 data blocks in the stripe set as shown in Figure 6. So $R_d = \frac{16}{4\times4\times3} = \frac{1}{3}$. According to features in Section II and Theory 1 in SDM [29], $m \times B/(m+n_d) = 16$, hence BDR achieves the minimal data migration ratio. After the merging step, the number of parity modification is $6 \times 3 = 18$ and $R_p = \frac{18}{6\times6} = \frac{1}{2}$. In 3DFTs, each data migration and parity modification need two I/Os, respectively (one read I/O and one write I/O). Each parity recalculation needs one write I/O. The total number of I/O operations is the total number of I/O operations of data migration and parity modification or recalculation. In the example of the scaling of 3DFTs using Triple-Star Code from 7 to 9 disks ($p = 5$), total number of I/O operations ratio is $\frac{16+6\times3+\frac{9}{4}\times(4+2)}{6\times6} = 1.319$.

B. Numerical Results

In this subsection, we give the numerical results of BDR compared to other scaling approaches using above metrics. SDM cannot apply to the scenarios in Triple-Star code and

EH-Code (p=7, m=6), which are impracticable according to the equations in SDM.

(1) **Data Migration Ratio:** We calculate the data migration ratio (R_d) among various scaling approaches as shown in Figure 9. It is obvious that BDR reduces data migration by up to 84.6% and has the approximate migration ratio compared to SDM [29], which has the minimal data migration ratio.

(2) **Parity Modification Ratio:** Parity modification ratio (R_p) among the scaling approaches is presented in Figure 10. Compared to other schemes with the same p and m, BDR remains the same parity modification ratio. In some cases, the parity modification ratio of SDM is extremely high because the number of corresponding parities is large when the disk array scales from a small size to a large one. Compared to other popular scaling approaches, BDR reduces the parity modification ratio by up to 75%.

(3) **Computation Cost:** We compute the total number of XOR operations per data block in Figure 11. BDR needs more number of XOR operations than SDM scheme. It is reasonable because BDR merges stripes to get the smallest possible parity modification and I/O operations. Compared to SDM and RS6, BDR spends 32.86% and 10.56% additional computation cost on average, respectively. This downside can be covered by the reduced I/O cost, which has been demonstrated that it has slight effect on the scaling process [28][29] (The scaling efficiency is shown in Figure 13).

(4) **Scaling I/Os Ratio:** Scaling I/Os Ratio is presented in Figure 12. BDR has the smallest total number of I/O operations, because BDR reads stripes into buffers in parallel to reduce while other scaling schemes read data/parity blocks one by one. So that large amount of I/O operations are reduced. Compared to popular Round-Robin (RR), SDM and RS6, BDR reduces up to 77.45% I/Os. Table III shows the improvement of BDR compared to popular scaling approaches in terms of scaling I/Os ratio using various MDS codes.

(5) **Spatial Overhead:** We compute the spatial overhead of BDR in the scaling process of 3DFTs based on different block sizes, as shown in Figure 8. The increasing block size leads to an increment in the spatial overhead. When the block size is set to 4KB to 128KB, BDR utilizes 17.16MB of memory on average as the buffer, which is only 0.21% of 8GB memory.

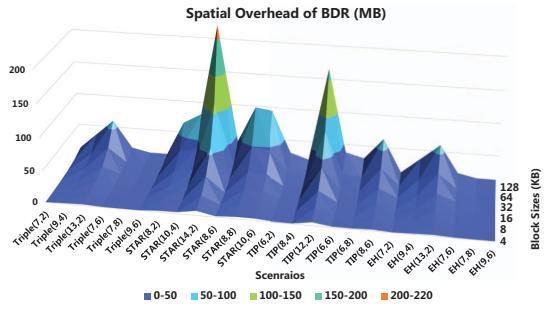


Fig. 8. Spatial overhead of BDR (MB) using various MDS codes based on different block sizes.

TABLE III
IMPROVEMENT OF BDR OVER POPULAR SCALING APPROACHES IN TERMS OF SCALING I/O RATIO AND MIGRATION SPEED USING VARIOUS MDS CODES.

code	p	m	Improvement of Scaling I/O			Improvement of Migration Speed		
			RR	SDM	RS6	RR	SDM	RS6
Triple-Star	5	2	1.47×	1.40×	1.44×	1.93×	1.45×	1.98×
	7	4	1.46×	1.39×	1.41×	2.19×	1.89×	2.13×
	11	2	1.66×	1.49×	1.56×	3.28×	1.23×	2.64×
	5	6	1.51×	1.65×	1.52×	1.71×	2.90×	1.83×
	5	8	1.62×	1.76×	1.63×	1.74×	3.21×	1.86×
	7	6	1.73×	– –	1.72×	2.26×	– –	2.25×
STAR	5	2	1.34×	1.14×	1.27×	2.14×	1.50×	2.01×
	7	4	1.62×	1.54×	1.58×	2.42×	1.91×	2.22×
	11	2	1.77×	1.61×	1.69×	3.56×	1.16×	2.75×
	5	6	1.48×	1.54×	1.47×	1.95×	2.77×	1.93×
	5	8	1.46×	1.59×	1.45×	1.85×	3.29×	1.89×
	7	6	1.59×	1.54×	1.56×	2.35×	2.09×	2.23×
TIP-code	5	2	1.45×	1.33×	1.41×	2.30×	1.29×	2.18×
	7	4	1.54×	1.38×	1.49×	2.47×	1.94×	2.52×
	11	2	1.76×	1.62×	1.72×	3.66×	1.37×	3.39×
	5	6	1.30×	1.36×	1.27×	1.95×	2.93×	2.03×
	5	8	1.24×	1.38×	1.21×	1.83×	2.54×	1.86×
	7	6	1.49×	1.45×	1.46×	2.34×	1.98×	2.21×
EH-Code	5	2	1.06×	1.04×	1.03×	2.85×	1.70×	2.72×
	7	4	1.60×	1.49×	1.54×	2.90×	2.10×	3.24×
	11	2	1.76×	1.66×	1.70×	4.17×	1.32×	3.88×
	5	6	1.62×	1.69×	1.59×	2.32×	3.16×	2.14×
	5	8	1.44×	1.58×	1.42×	2.32×	3.31×	1.98×
	7	6	1.56×	– –	1.52×	2.91×	– –	3.05×

C. Simulation Results

In this subsection, we give the simulation results of the migration speed, which is measured by the number of stripes that finish the scaling process per second. This metric reflects a combination of the metrics **"Computation Cost"** and **"Scaling I/O Ratio"** in Section IV-B. Low computation cost and small number of I/O operations can accelerate the scaling process and increase the migration speed.

We use Disksim [5] as the simulator in our evaluation. It is an efficient and highly-configurable disk simulator for a typical storage system. We deploy DiskSim on an Intel i5-4430 3.00GHz PC with 8GB DRAM. In Disksim, the stripe unit size is set to 8KB, which means the block size is 8KB. According to BDR and other schemes, we simulate the traces of the scaling process among popular MDS codes with three parities and measure the migration speed by using them.

In Figure 13, BDR has the highest migration speed. Compared to RR, SDM and RS6 approaches, BDR speeds up the scaling process by up to $4.17\times$, $3.31\times$, $3.88\times$, respectively. The improvement of BDR over popular scaling approaches on migration speed is shown in detail in Table III.

D. Analysis

The results in Section IV-B and IV-C shows that compared to RS6, SDM and RR methods, BDR has great advantages. First, BDR implements a global management on multiple stripes, which reduces the data migration overhead, parity modification cost and total number of I/O operations. Second, BDR achieves the uniform data distribution in each stripe, which accelerates the scaling process in parallel. Third, BDR merges stripes to decrease the parity modification I/O cost. Therefore, BDR increases the migration speed and accelerates the scaling process as an combination of the above reasons.

V. CONCLUSIONS

In this paper, we propose a Balanced Data Redistribution scheme (BDR), which can be applied on XOR-based 3DFTs

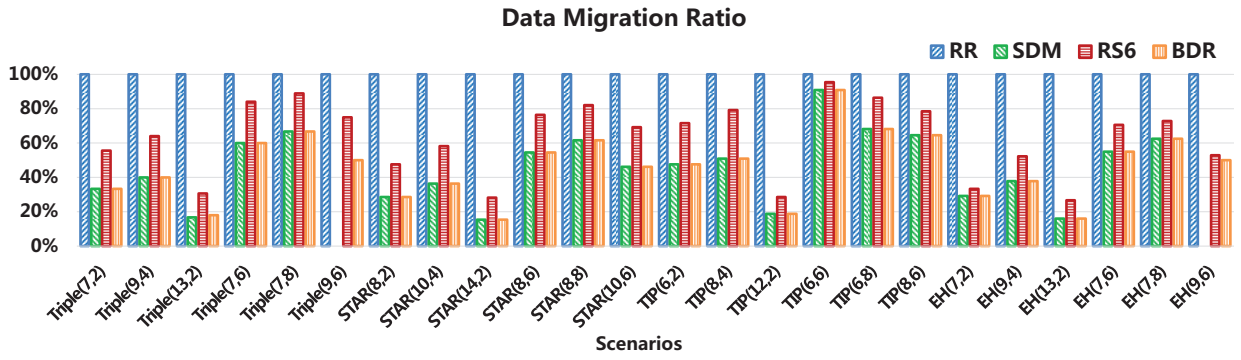


Fig. 9. Comparison on Data Migration Ratio on Various 3DFTs Scaling Approaches Using Different Codes.

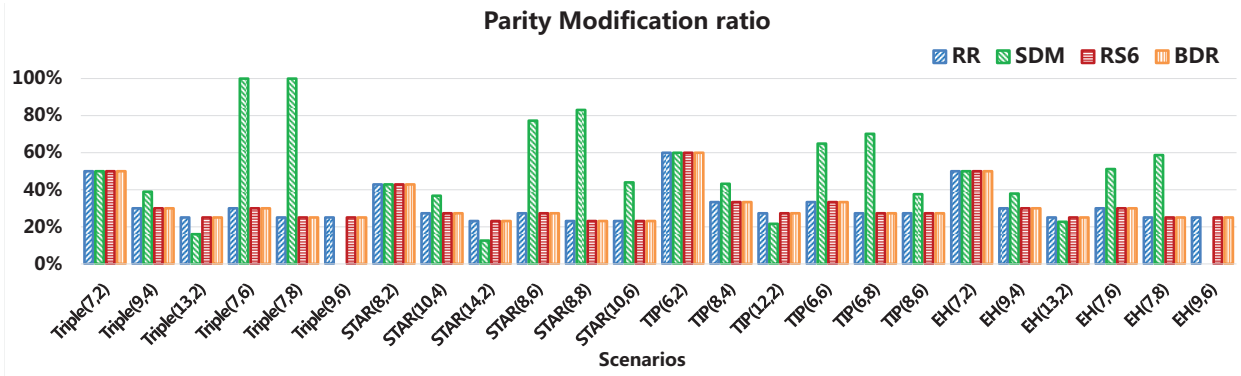


Fig. 10. Comparison on Parity Modification Ratio on Various 3DFTs Scaling Approaches Using Different Codes.

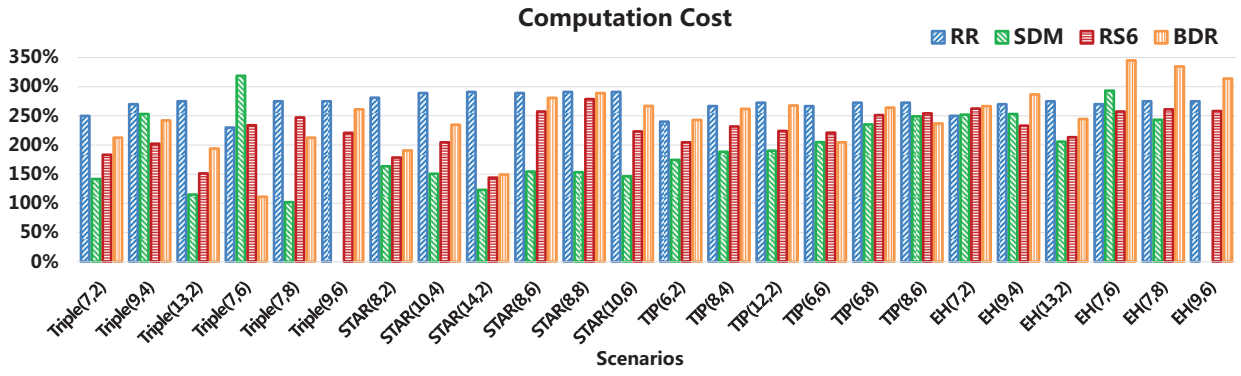


Fig. 11. Comparison on Computation Cost on Various 3DFTs Scaling Approaches Using Different Codes.

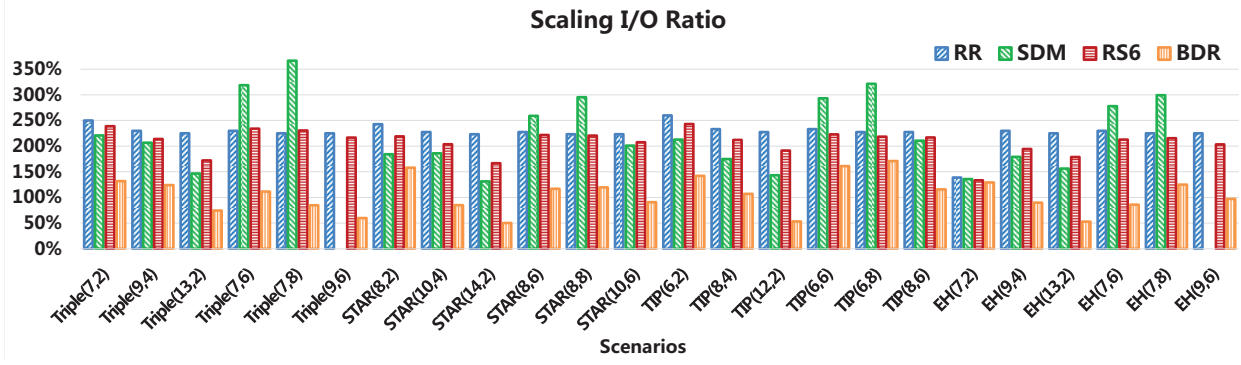


Fig. 12. Comparison on Scaling I/O Ratio on Various 3DFTs Scaling Approaches Using Different Codes.

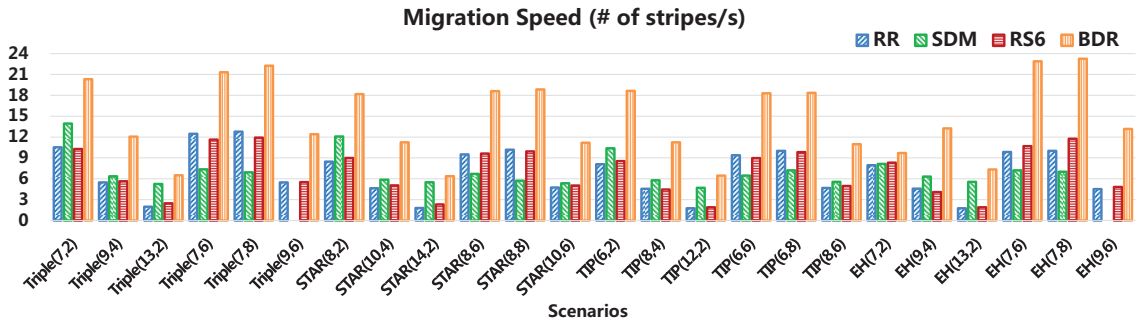


Fig. 13. Comparison on Migration Speed (number of stripes/s) on Various 3DFTs Scaling Approaches Using Different Codes.

based on MDS codes. The main idea of BDR is that, it migrates proper data blocks based on the relations between three parities, which guarantees uniform data distribution and small number of data movements. Through evaluations and simulations, BDR has the following advantages. (1) BDR achieves the balanced data distribution. (2) BDR has the minimal data migration ratio and parity modification ratio. (3) BDR reduces up to 77.45% I/Os by comparison with other popular scaling methods. (4) Compared to RR, SDM and RS6 approaches, BDR speeds up the scaling process by up to $4.17\times$, $3.31\times$, $3.88\times$, respectively.

REFERENCES

[1] M. Armbrust et al. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, UC Berkeley, 2009.

[2] M. Blaum, J. Hafner, and S. Hetzler. Partial-mds codes and their application to raid type of architectures. *IEEE Transactions on Information Theory*, 59(7):4510–4519, 2013.

[3] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. An XOR-based Erasure-Resilient coding scheme. Technical Report TR-95-048, International Computer Science Institute, 1995.

[4] D. Borthakur, R. Schmidt, R. Vadali, S. Chen, and P. Kling. Hdfs raid. In *Hadoop User Group Meeting*, 2010.

[5] J. Bucy, J. Schindler, S. Schlosser, and G. Ganger. The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101). *Parallel Data Laboratory*, 2008.

[6] B. Calder et al. Windows azure storage: a highly available cloud storage service with strong consistency. In *Proc. of the ACM SOSP'11*, 2011.

[7] D. Ford, F. Labelle, F. Popovici, M. Stokely, V. Truong, L. Barroso, C. Grimes, and S. Quinlan. Availability in globally distributed storage systems. In *Proc. of the USENIX OSDI'10*, 2010.

[8] A. Goel et al. SCADDAR: An efficient randomized technique to reorganize continuous media blocks. In *Proc. of the ICDE'02*, 2002.

[9] J. Gonzalez and T. Cortes. Increasing the capacity of RAID5 by online gradual assimilation. In *Proc. of the SNAPI'04*, 2004.

[10] J. Hafner. WEAVER codes: Highly fault tolerant erasure codes for storage systems. In *Proc. of the USENIX FAST'05*, 2005.

[11] J. Hafner. HoVer erasure codes for disk arrays. In *Proc. of the IEEE/IFIP DSN'06*, 2006.

[12] C. Huang, M. Chen, and J. Li. Pyramid Codes: Flexible schemes to trade space for access efficiency in reliable data storage systems. In *Proc. of the IEEE NCA'07*, 2007.

[13] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, S. Yekhanin, et al. Erasure coding in windows azure storage. In *Proc. of the USENIX ATC'12*, 2012.

[14] C. Huang and L. Xu. STAR: An efficient coding scheme for correcting triple storage node failures. *IEEE Transactions on Computers*, 57(7):889–901, 2008.

[15] K. Hwang, H. Jin, and R. Ho. Raid-x: A new distributed disk array for i/o-centric cluster computing. In *Proc. of the HPDC'00*, 2000.

[16] Y. Jiang, C. Wu, J. Li, and M. Guo. EH-Code: An extended mds code to improve single write performance of disk arrays for correcting triple disk failures. In *Proc. of the ICA3PP'15*, 2015.

[17] A. Miranda and T. Cortes. Craid: online raid upgrades using dynamic hot data reorganization. In *Proc. of the USENIX FAST'14*, 2014.

[18] N. Brown. Online RAID-5 Resizing. drivers/md/raid5.c in the source code of Linux Kernel 2.6.18. http://www.kernel.org/, 2006.

[19] D. Patterson, G. Gibson, and R. Katz. A case for Redundant Arrays of Inexpensive Disks (RAID). In *Proc. of the ACM SIGMOD'88*, 1988.

[20] J. Plank and L. Xu. Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications. In *Proc. of the IEEE NCA'06*, 2006.

[21] I. Reed and G.Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[22] Y. Saito, S. Frølund, A. Veitch, A. Merchant, and S. Spence. Fab: building distributed enterprise disk arrays from commodity components. In *Proc. of the ASPLOS04*, 2004.

[23] W. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. Dimakis, R. Vadali, S. Chen, and D. Borthakur. Xoring elephants: Novel erasure codes for big data. In *Proc. of the VLDB'13*, 2013.

[24] P. Subedi and X. He. A comprehensive analysis of XOR-Based erasure codes tolerating 3 or more concurrent failures. In *Proc. of the IPDPSW'13*, 2013.

[25] D. Tang, X. Wang, S. Cao, and Z. Chen. A new class of highly fault tolerant erasure code for the disk array. In *Proc. of the PEITS'08*, 2008.

[26] C. Tau and T. Wang. Efficient parity placement schemes for tolerating triple disk failures in RAID architectures. In *Proc. of the AINA'03*, 2003.

[27] Y. Wang, G. Li, and X. Zhong. Triple-Star: A coding scheme with optimal encoding complexity for tolerating triple disk failures in RAID. *International Journal of Innovative Computing, Information and Control*, 8(3):1731–1472, 2012.

[28] C. Wu and X. He. GSR: A global stripe-based redistribution approach to accelerate raid-5 scaling. In *Proc. of the ICPP'12*, 2012.

[29] C. Wu, X. He, J. Han, H. Tan, and C. Xie. SDM: A stripe-based data migration scheme to improve the scalability of raid-6. In *Proc. of the CLUSTER'12*, 2012.

[30] S. Wu, Y. Xu, Y. Li, and Y. Zhu. POS: A popularity-based online scaling scheme for raid-structured storage systems. In *Proc. of the ICCD'15*, 2015.

[31] Mingyuan Xia, Mohit Saxena, Mario Blaum, and David A Pease. A tale of two erasure codes in hdfs. In *Proc. of the USENIX FAST'15*, 2015.

[32] X. Yu, B. Gum, Y. Chen, R. Wang, K. Li, A. Krishnamurthy, and T. Anderson. Trading capacity for performance in a disk array. In *Proc. of the USENIX OSDI'00*, 2000.

[33] G. Zhang et al. SLAS: An efficient approach to scaling round-robin striped volumes. *ACM Trans. on Storage*, 3(1):1–39, 2007.

[34] G. Zhang, K. Li, J. Wang, and W. Zheng. Accelerate rdp raid-6 scaling by reducing disk i/os and xor operations. *IEEE Transactions on Computers*, 64(1):32–44, 2015.

[35] Y. Zhang, C. Wu, J. Li, and M. Guo. TIP-code: A three independent parity code to tolerate triple disk failures with optimal update complexity. In *Proc. of the IEEE/IFIP DSN'15*, 2015.

[36] W. Zheng and G. Zhang. Fastscale: Accelerate raid scaling by minimizing data migration. In *Proc. of the USENIX FAST'11*, 2011.