IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received September 14, 2020, accepted October 12, 2020, date of publication October 22, 2020, date of current version December 31, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3033024

Erasure Coding-Oriented Data Update for Cloud Storage: A Survey

YIFEI XIAO, SHIJIE ZHOU, AND LINPENG ZHONG
School of Information and Software Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China
Corresponding author: Shijie Zhou (sjzhou@uestc.edu.cn)

ABSTRACT Erasure coding is the leading technique to achieve resilient redundancy in cloud storage systems. However, it introduces two prominent issues: data repair and data update. Compare to data repair, data update is much more common. A variety of update schemes based on erasure coding have been proposed in the literature to optimize data update, such as computation optimization, network traffic overhead reduction, IO overhead reduction, and modern hardware acceleration. However, all of these techniques were proposed individually previously. In this work, we seek to summarize them systematically and group them in a new form. First, we generalize the state-of-the-art researches and introduce existing classifications. Moreover, based on our observation, we propose two classifications: resource-based classification and tier-based classification. In resource-based classification, we group these techniques according to the resource they optimize and introduce them in detail. In tier-based classification, we propose a novel hybrid technique framework with five tiers and conduct a comprehensive comparison between these techniques. We make a conjecture that most techniques in different tiers can be used jointly. Finally, we conclude the research challenges and potential future works.

INDEX TERMS Data update, cloud storage, erasure coding, survey.

I. INTRODUCTION

Currently, it is estimated that approximately 3.6 billion users utilize cloud storage services in 2018 [53], with Dropbox alone claiming 500 million users in 2016 [20]. YouTube needs 1 PB of storage to save new videos every day [7]. Thus, cloud storage systems (CSS) are facing tremendous storage pressure. At the same time, large-scale CSS are constantly facing the risk of data loss, which is caused by either software failures or hardware failures [36]. The common way to improve data reliability and data availability is data redundancy, which can be achieved by *replication* or *erasure coding*. Replication is the conventional technique which repeats the stored data at multiple geo-distributed locations. But this mechanism incurs considerably large storage overhead (e.g., 200% storage overhead for 3-replication). Compare to replication, erasure coding can dramatically reduce storage overhead. Taking QFS (Quantcast File System) [38] as an example, the implementation of QFS with erasure coding can save 50% of storage overhead over the original HDFS which uses 3-replication, while tolerating the same number of failures. The property of low storage overhead on erasure coding is so attractive for many cloud computing providers that erasure coding is gradually employed by many leading players in cloud storage, such as Amazon S3 [5], Google cloud [23], Microsoft Azure [28], Facebook cluster [43] and Alibaba Cloud [62].

However, erasure coding brings some new issues to CSS, such as *data repair (DR)* and *data update (DU)*. In DR, when a data block is lost, replication can simply replace the data link with another one. While erasure coding has to call the regeneration paradigms to recover the data block, which will inevitably consume a large number of resources (e.g., computation, network and IO). To improve DR efficiency, the main body of the literature on erasure coding has made great contributions on designing excellent erasure codes or improving existing erasure codes [9], [27], [29], [41], [52], [64]. However, an additional significant, and arguably overlooked problem for CSS is DU, which is much more common than DR for many real-world workloads in enterprise servers and network file systems [14]. Especially, DU is quite common for big data applications which have rapid data changing. DU is a process of maintaining data consistency between the data blocks and the corresponding parity blocks, which is

The associate editor coordinating the review of this manuscript and approving it for publication was Lorenzo Ciani.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

triggered by data changing. In other words, once a data block is changed, the corresponding parity blocks should be regenerated with the updated data block. For replication, DU is a simple process of sending the modified info to the copies. While for erasure coding, DU is much more complicated than replication for the following reasons:

❶ **Efficiency:** Unlike the simple replication, DU in erasure coding is much more time-consuming. To achieve data consistency without disturbing the normal operations of other applications, the update process should achieve high efficiency.

❷ **Compatibility:** There exist various erasure codes adopted in current storage systems [9], [18], [22], [40], [42], [60]. Once the code is changed, a data transformation process will be conducted inevitably. It is typically time-consuming to change the code since a massive volume of heterogeneous and rapidly changing data in the storage system has to transform to adapt to the new code [14], [28], [49]. What's more, the analysis [14] of MSR Cambridge traces [37] and Harvard NFS traces [21], infers updates are small, since most of the traces have more than 60% of updates smaller than 4KB. While Shen *et al.* [50] have a different observation that update requests with large update sizes are quite common in existing distributed storage systems, especially for online applications [16], [32], [57]. These different observations show that DU is mutable, thus update schemes should be compatible with small updates and large updates.

❸ **Adaptivity:** It is known that the failure is a norm in CSS. Once a failure occurs during an update, an efficient update scheme with rollback-based strategy [59] or similar strategies should be considered.

According to these reasons (requirements) of DU in erasure coding, a variety of update schemes based on erasure coding have been proposed in the literature to optimize DU, including optimizing computation schedule [27], [41], traffic overhead reduction [39], [59], IO overhead reduction [50] and modern hardware acceleration [58]. All of these techniques were proposed individually previously. However, there no exist survey work to summarize them systematically. In this paper, we seek to fill this gap in the literature. Our work mainly makes the following contributions:

- We summarize the state of the art on DU and introduce the existing classifications of DU in the literature.
- According to the optimization goals (computation, network, IO) of different update schemes, we propose the resource-based classification and divide these schemes into 3 types (computation optimization, network optimization, and IO optimization).
- To use different schemes jointly, we design a new hybrid technique framework with 5 tiers. According to their characteristics, we propose the tier-based classification and place these schemes into different tiers.
- We discuss the current challenges and explore the potential future works on DU.

The rest of this paper is organized as follows. In Section II, we introduce the related work on DU. In Section III, we describe the development of DU from 2 perspectives: application scope and resource optimization. In Section IV, we summarize the existing classifications of DU. In Section V, we propose a resource-based classification and describe the state-of-the-art researches involved in DU. In Section VI, we propose a novel hybrid technique framework with 5 operation tiers and conduct a comprehensive comparison between different techniques of DU. In Section VII, we discuss current challenges of DU. Finally, in Section VIII, we conclude this paper and point out the potential future works.

II. RELATED WORK

Although there no exist relevant survey work on DU, several categories of DU have been proposed in the literature.

Chan *et al.* [14] described 3 typical approaches of DU that can be easily found in RAID systems: ① Reconstruct writes (RCW) [56], ② Read-modify writes (RMW) [56], ③ Full-segment writes (FSW). Besides, they introduced another classical class of DU, called the *delta-based update*, which only transfer the modified data range to the corresponding nodes, instead of transferring the entire data block. In the delta-based update, they describe 3 typical approaches: ① Full-overwrite (FO), ② Full-logging (FL), ③ Parity-logging (PL). The details of them will be described in Section V.

Based on Chan's work, Pei *et al.* [39] gave an overview of these approaches on DU. That is, ❶ according to whether transferring the whole data block, DU can be classified into 2 types: *RAID-based update* [39], [54], [59] and *delta-based update*. ❷ According to whether the data is overwritten, DU can be divided into 3 types: *in-place update*, *log-based update* and *hybrid update* [39], [59]. In Section V, we will re-examine them in detail.

III. STATE OF THE ART

As Figure 1 depicted, in this section, we discuss the development of DU from 2 perspectives: application scope and resource optimization.

A. APPLICATION SCOPE

Initially, erasure coding is introduced to RAID [24], such as RAID5, RAID6. There exist 3 typical approaches of DU in RAID: ① Reconstruct writes (RCW) [56], ② Read-modify writes (RMW) [56], ③ Full-segment writes (FSW) [35]. But the constraint of FSW is too strict (it won't perform updating until all data blocks in a stripe are updated), which makes the other two become 2 in fact common update approaches.

With the rapid growth of data variety, volume, and velocity in data storage systems, erasure coding is gradually popular in various data storage systems, ranging from RAID [15], P2P storage systems [61], to distributed storage systems [31], [34], and CSS [11]. Among them, except RAID, there exist multiple geo-distributed nodes in other systems, leading that DU is not only sensitive to CPU and IO, but also sensitive to network.

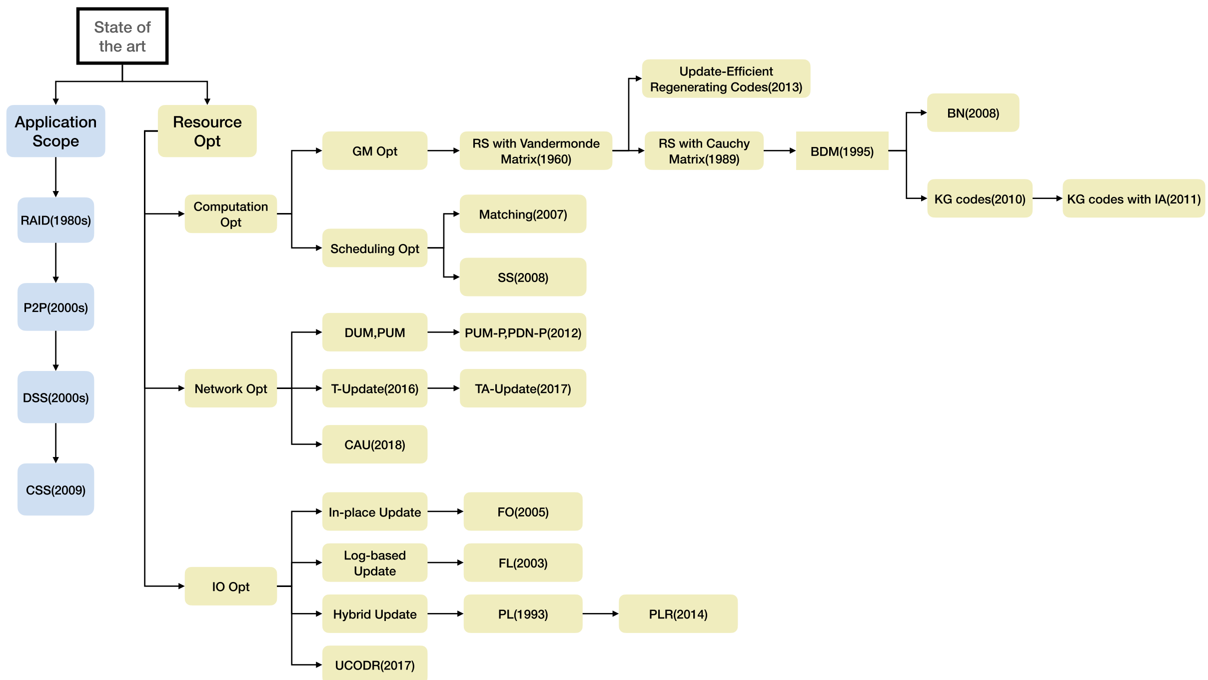


FIGURE 1. The figure depicts the development of DU from 2 perspectives: application scope and resource optimization.

B. RESOURCE OPTIMIZATION

In this subsection, we will discuss the development of DU from the perspective of resource optimization (computation, network and IO). Although many works do not concentrate on CSS, even not for DU, their ideas are significant for DU and can be applied to CSS.

Computation Optimization: Initially, DU directly calls encoding process (e.g., RCW), so it is helpful for DU when researchers are devoted to improving the encoding. The main idea is either reducing the computational overhead or accelerating the computation process, thus we divide them into 2 types: ① *Generator Matrix Optimization (GM Opt)* and ② *Scheduling Optimization (Scheduling Opt)*.

In GM Opt, it is known that the redundancy info (the parity blocks in CSS) is the linear combination of the data blocks, which is accomplished by encoding, with the help of *generator matrix*. The well-known RS codes or RS-based codes use Vandermonde matrix [46] or Cauchy matrix [10], [47] as the genenrator matrix, where all elements are in GF(2^w). However, this will incur a large number of multiplications, thus degrading the performance of DU. To optimize the generator matrix, in 1995, *Binary Distribution Matrix (BDM)* was employed in [9], [10], where elements of encoding were only bits (0 or 1). Thus, all the multiplications and additions can be converted to XORs in GF(2), which can greatly improve the encoding efficiency. To optimize BDM, Plank *et al.* [41] proposed *Bitmatrix Normalization (BN)* in 2008, the main idea of which is reducing the ones in BDM, thus improving the encoding efficiency. In 2010, Anthapadmanabhan *et al.* employed *randomization* to optimize the BDM [6], in which *Update Complexity* was defined and the update-efficient *KG code* was proposed. In 2011, Rawat *et al.* [45] combined the KG code with interference-alignment (IA), which achieved both update efficiency and repair efficiency mathematically. In 2013, Han *et al.* [26] constructed 2 update-efficient regenerating codes based on MSR [26] and MBR [25], respectively, and proved their efficiency mathematically.

In Scheduling Opt, Huang *et al.* [27] recognized that we can adjust the scheduling order of encoding to reduce computational overhead. In 2007, they proposed Matching (*Cardinality Matching* and *Weighted Matching*) to optimize DU, with the idea of computing the common part first, which achieved a great improvement of DU. Similarly, Plank *et al.* [41] proposed *Smart Scheduling (SS)* in 2008.

Network Optimization: Although it is useful for DU to optimize CPU, Agarwal *et al.* [1] noted that the bottleneck of the performance of DU is network, especially the rack-across network [51]. Therefore, some researchers focus on network optimization. DUM and PUM are 2 typical network-aware approaches [63] in DU. In fact, DUM and PUM are the implement of RCW and RMW in distributed storage systems, respectively. The difference is, DUM and PUM use a special node called *Update Manager (UM)* to compute the update info of the parity nodes. Based on PUM, PUM-P and PDN-P were proposed in 2012 [63]. Besides, in 2016, Pei *et al.* [39] recognized that the conventional update transmission path employs star structure, which is easy to cause a single point of bottleneck. To end this, they proposed a new update scheme called T-Update with a tree-structured transmission

path, dispatching the burden of the single data node to the corresponding parity nodes. In 2017, based on T-Update, Wang *et al.* [59] proposed TA-Update to handle the node failure while updating. In 2018, Shen and Lee [51] proposed CAU to mitigate the burden of the rack-across network.

IO Optimization: In IO optimization, as stated before, according to whether the data is overwritten, DU can be divided into 3 types: *in-place update*, *log-based update* and *hybrid update* [39], [59]. These approaches are devoted to accelerating IO access while updating. Besides, Shen *et al.* [50] proposed UCODR in 2017, which is a heuristic of scheduling. They defined the distance between 2 parity nodes and started to update from the closest nodes, which can reduce the number of disk read.

In order to fully demonstrate the main ideas of these approaches, A detailed description of all above approaches will be conducted in Section V.

IV. EXISTING CLASSIFICATION

A. AN EXAMPLE OF DU

Figure 2 depicts a typical erasure-based CSS, in which there exist $k = 3$ data nodes and $m = 2$ parity nodes (i.e., $n = 5$). All files in the CSS are splited into equal-sized data blocks ($d_{i,j}$) which are encoded into n blocks (including k data blocks and m parity blocks). These n blocks group into a *stripe*, scattering in different nodes. The blocks in the same node group into a *strip*. Let $d_{i,j}$ represent the jth data block in the ith data node and $p_{i,j}$ represent the jth parity block in the ith parity node.

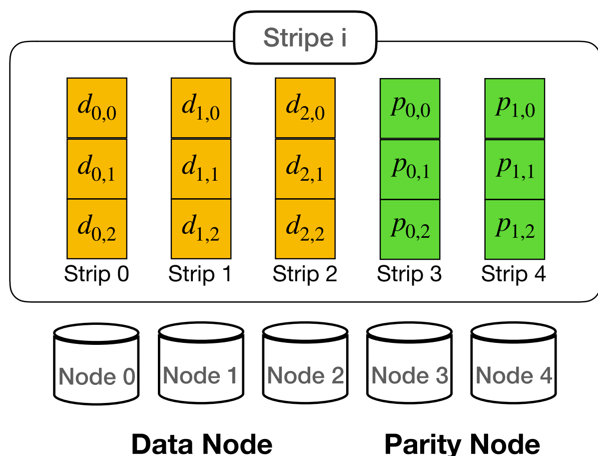


FIGURE 2. A typical CSS with RS(5, 3), where $k = 3$, $m = 2$, and $w = 3$. The data blocks are yellow and parity blocks are green.

It is known that popular erasure codes can be divided into 2 classes: RS-based codes and XOR-based codes [62]. Accordingly, we will discuss RS-based DU and XOR-based DU.

1) RS-BASED DU

Figure 3 shows a typical RS-based encoding about Figure 2, where the leftmost matrix is called *generator matrix*. As mentioned above, generator matrix can be generated from Vandermonde matrix or Cauchy matrix. The top k rows of the generator matrix compose a $k \times k$ *identity matrix* (here $k = 3$). Compare to Figure 2, d_0 represents $(d_{0,0}, d_{0,1}, d_{0,2})$, d_1 represents $(d_{1,0}, d_{1,1}, d_{1,2})$ and d_2 represents $(d_{2,0}, d_{2,1}, d_{2,2})$. In other words, a strip in Figure 2 is denoted by a big data block (d_i, $i = 0, 1, 2$) in Figure 3. The remaining m rows are called *coding matrix* [41] (here $m = 2$). Similarly, p_0 represents $(p_{0,0}, p_{0,1}, p_{0,2})$ and p_1 represents $(p_{1,0}, p_{1,1}, p_{1,2})$. In Figure 3, the generator matrix encodes the data blocks (denoted by d_0, d_1, d_2) into a *codeword* $(d_0, d_1, d_2, p_0, p_1)$. Each block can refer to one symbol in the codeword. After that, data blocks (d_0, d_1, d_2) will be sent to the corresponding data nodes and the parity blocks (p_0, p_1) will be sent to the corresponding parity nodes. From Figure 3 we can infer that, in a (n, k) RS-based CSS, each parity block could be represented by the linear combination of the k data blocks with the following equation,

$$p_i = \sum_{j=0}^{k-1} \alpha_{i,j} d_j, \quad i \in [0, m-1] \tag{1}$$

where all elements are numbers in GF(2^w) for some value of w. Suppose $d_{0,0}$ is changed, Eq.(1) can be called for DU. However, in this way, all parity blocks (p_0, p_1) are related to $d_{0,0}$, thus both p_0 $(p_{0,0}, p_{0,1}, p_{0,2})$ and p_1 $(p_{1,0}, p_{1,1}, p_{1,2})$ are required to be updated.

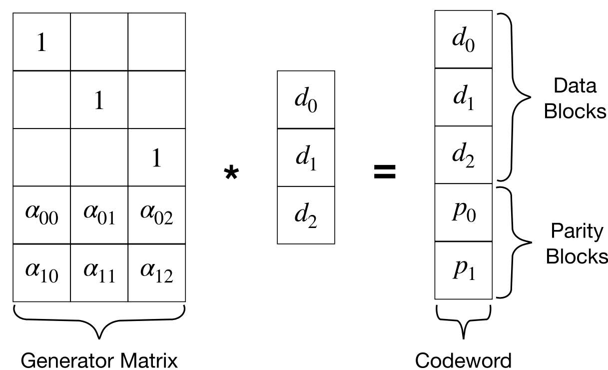


FIGURE 3. Encoding with RS(5, 3): the leftmost *generator matrix* encodes data blocks (d_0, d_1, d_2) into the rightmost codeword $(d_0, d_1, d_2, p_0, p_1)$.

On the other hand, we can apply another class of genenrator matrix which is based on the delta info. The idea is to fully use the old blocks, which can be represented by the following equations,

$$p_i^{r+1} = \sum_{j=0}^{k-1} \alpha_{ij} \delta_j^{r+1} + p_i^r \tag{2}$$

$$\delta_j^{r+1} = d_j^{r+1} - d_j^r \tag{3}$$

where p_i^{r+1} denotes the updated value of p_i in round $r + 1$, and p_i^r denotes the old value of p_i in round r. δ_j^{r+1} represents the delta related to d_j from round r to round $r + 1$. Similarly, suppose $d_{0,0}$ is changed ($j = 0$), although all parity blocks are

still required to be updated, the number of blocks involved in the calculation can be reduced (the comparison between RCW and RMW in Figure 5 will prove this).

2) XOR-BASED DU

As shown in Eq.(1) and Eq.(2), a considerable number of multiplications are generated, which will significantly impede the performance of DU. To end this, XOR-based encoding is proposed (as shown in Figure 4), where BDM is used. Using BDM representation, each element e in GF(2^w) can be denoted by a matrix M(e) of $w \times w$ or a vector V(e) of $1 \times w$, thus the generator matrix of size $k \times m$ can be converted to a new generator matrix of size $wk \times wm$ in GF(2) [64]. On this light, we can use the smaller block as the basic element in encoding (here we use $d_{i,j}$ and $p_{i,j}$, instead of using d_i and p_i). Thus, according to Figure 4, the parity blocks can be computed by the following equations,

$$p_{0,0} = d_{0,0} \oplus d_{1,0} \oplus d_{2,0} \oplus d_{2,2} \tag{4}$$

$$p_{0,1} = d_{0,1} \oplus d_{1,1} \oplus d_{2,0} \tag{5}$$

$$p_{0,2} = d_{0,2} \oplus d_{1,2} \oplus d_{2,1} \tag{6}$$

$$p_{1,0} = d_{0,0} \oplus d_{1,0} \oplus d_{1,2} \oplus d_{2,0} \tag{7}$$

$$p_{1,1} = d_{0,1} \oplus d_{1,0} \oplus d_{2,1} \tag{8}$$

$$p_{1,2} = d_{0,2} \oplus d_{1,1} \oplus d_{2,2} \tag{9}$$

where the matrix multiplications are now converted to XORs of data bits corresponding to the ones in BDM. Similarly, if $d_{0,0}$ is changed, it can be found that only $p_{0,0}$ and $p_{1,0}$ are required to be updated, thus DU with BDM significantly reduces the computation overhead. It is noted that the smaller block makes the less data updates, because more accurate relations between the data blocks and parity blocks can be established.

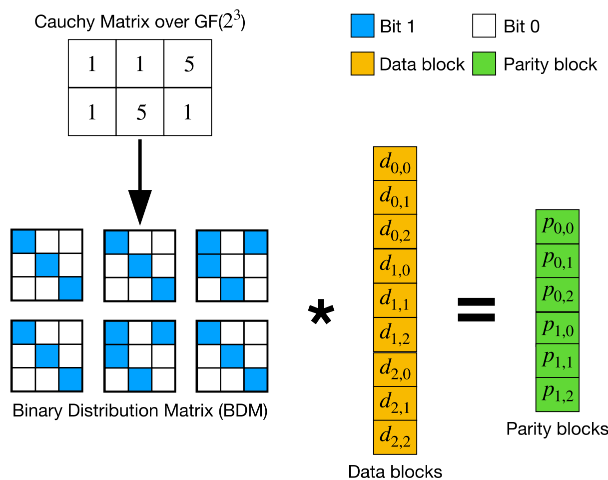


FIGURE 4. Encoding with BDM: the Cauchy matrix is converted to BDM, where the blue block denotes bit 1 and the white block denotes bit 0, identically, $k = 3, m = 2, w = 3$.

Similarly, XOR-based DU also has the delta style. Without loss of generality, let $S_{i,j}$ denote the set of data blocks for updating $p_{i,j}$, and S_u^r denotes the set of updated data blocks in update round r. To simplicity, we use $\oplus_{d_{m,n} \in S_{i,j}}$ to represent the computational result of all the elements in $S_{i,j}$ performing XOR operations. So we have,

$$p_{i,j}^{r+1} = p_{i,j}^r \oplus (\oplus_{d_{m,n} \in (S_{i,j} \cap S_u^{r+1})}(d_{m,n}^{r+1} \oplus d_{m,n}^r)) \tag{10}$$

where $d_{m,n}^{r+1} \oplus d_{m,n}^r$ is the delta related to $d_{m,n}$ from round r to round $r+1$.

B. DATA TRANSMISSION APPROACHES

Before computing the newest parity blocks with RS-based DU or XOR-based DU, we have to transfer relevant data blocks to the corresponding parity nodes. According to whether transferring the whole data block, Pei *et al.* [39] divide update schemes into 2 types: 1) *RAID-based update* and 2) *delta-based update*.

1) RAID-BASED UPDATE

The *RAID-based update* transfers the whole data block. As mentioned above, RCW and RMW are 2 classical approaches of RAID-based update.

● **Reconstructed write (RCW):** when DU is required, RCW reads all data blocks in a stripe and transmit them to the corresponding parity nodes to reconstruct the parity blocks. The updated value p_i^{r+1} of the parity block p_i in update round $r+1$ is either based on RS-based DU(Eq.(1)) or XOR-based DU (Eq.(4) to Eq.(9)).

An example of RCW is shown in Figure 5(1), where the leftmost 3 green blocks are updated while the 3 red blocks are not. For updating the rightmost parity block, RCW sends all these 6 blocks to the parity node. In this round, RCW needs to read 3 blocks and write 4 blocks (as 3 green data blocks stay in memory and do not need to be read).

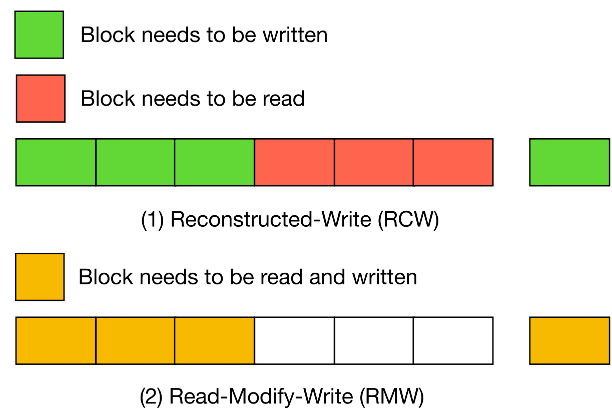


FIGURE 5. An example of RCW and RMW [50]. Assuming there are $n = 7$ nodes and $k = 6$ data nodes.

● **Read-modify write (RMW):** Unlike RCW, RMW computes the deltas by reading the old blocks in data nodes and transfering them to the corresponding parity nodes. Similarly, RMW can employ RS-based DU (Eq.(2) and Eq.(3)) or XOR-based DU (Eq.(10)). An example of RMW is shown

in Figure 5(2), where the leftmost 3 yellow blocks are updated and will be sent to the parity node. The rightmost yellow block is the parity block to be updated. In this round, RMW needs to read 4 blocks and write 4 blocks.

In a word, the difference between RCW and RMW is the accessing range of data blocks. In RCW, all data blocks in a stripe have to be read, while in RMW, disk read only involves updated data blocks.

2) DELTA-BASED UPDATE

In the RAID-based update, DU transfers the entire block from the data nodes to the corresponding parity nodes. But such mechanism will cost an incredibly large network bandwidth for data transmission, thus may have a negative effect on the normal running of applications. A typical class of approaches to reduce traffic cost is called *delta-based update*, which can eliminate redundant network traffic by only transferring a parity delta which is of the same size as the modified data range [13], [55]. The delta-based update fully employs the old blocks to compute the deltas based on Eq.(3) or Eq.(10).

A variety of delta-based schemes have been proposed in the literature, such as PDN-P, T-Update and CAU. All of these schemes will be discussed in Section V.

C. DATA STORAGE APPROACHES

Data storage approaches discuss how to store the data blocks and parity blocks, the goal of which is to reduce IO overhead. According to whether the data is overwritten, Pei *et al.* [39] summarized 3 data storage approaches in DU: ❶ *In-place Update*, such as Full-overwrite (FO) [2], ❷ *Log-based Update*, such as Full-logging (FL) and ❸ *Hybrid Update*, such as Parity-logging (PL) [30], [54] and Parity Logging with Reserved Space (PLR) [14].

1) IN-PLACE UPDATE

In-place update is adopted widely in CSS, since it guarantees strong data consistency. In the in-place update, once a data block is changed, the primitive data block will be overwritten immediately, which triggers the parity block be updated [39]. For example, in FO, clients perform write operations by in-place overwriting on data nodes and compute the deltas which will be transferred to the corresponding parity nodes. After that, these parity nodes compute the newest parity blocks with the deltas and overwrite the old parity blocks immediately. Such mechanism ensures the data blocks and parity blocks are always the newest.

On the downside, however, computing the deltas and parity updates both need extra disk read of the old blocks, which may aggravate the update time. What's more, noting that DU is common in CSS, which will easily degrade the performance of CSS.

2) LOG-BASED UPDATE

Noting that the in-place update needs extra disk read, which incurs a large amount of IO overhead. To end this, another class of DU called *log-based update* is proposed. Log-based update stores update information as logs. For instance, FL is a typical approach of log-based update which mitigates IO overhead by appending all data and parity updates to the old blocks. Specifically, the deltas of data blocks and parity blocks are simply recorded to logs. If clients need to read blocks, the *merge* operation of the primitive data and their updates in logs must be applied in advance. Such mechanism is faster than in-place update, thus it is employed by GFS [23] and Azure [12].

On the other hand, although log-based update (such as FL) is fast for DU, it aggravates the access time in common cases because of merging. Thus, it is probably unacceptable for cloud applications with many read requests [39].

3) HYBRID UPDATE

To solve the problem of log-based update, another class of update called *hybrid update* is proposed. *Parity-logging (PL)* is a typical approach of hybrid update, which combines the real-time performance of FO with the smooth IO usage of FL. In hybrid update, considering data in data nodes will be accessed with much higher probability than that in parity nodes, FO is applied to data nodes and FL is applied to parity nodes. In this way, not only it can save IO overhead of DU, but also it can guarantee the read performance of data blocks.

To accelerate IO further, Chan *et al.* [14] proposed a novel scheme based on PL, called *Parity Logging with Reserved Space (PLR)*, which preserved some extra space in each parity block to save the deltas, so that the parity block and its deltas can be saved in continuous physical locations on disk. Experimental results show PLR shortens the update time further.

In order to visually compare the various approaches of data storage, Figure 6 illustrates the differences between them, where RS(3, 2) is employed and the incoming data stream indicates the sequence of operations: ① write data blocks a and b, ② update part of a with a', ③ write data blocks c and d, and finally ④ update parts of b and c with b' and c', respectively. Figure 6 shows FO performs in-place overwriting for both data and parity blocks; FL performs appending for both data nodes and parity nodes; PL employs in-place overwriting for data nodes while employs appending for parity nodes; PLR does the same thing with PL, but it appends parity deltas in reserved space.

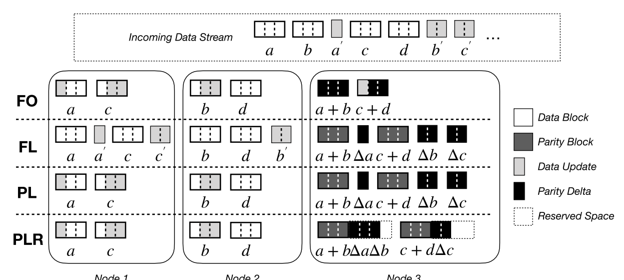


FIGURE 6. Illustration on different parity update schemes [14].

In this section, We re-examine existing schemes that fall into two classes: 1) data transmission approaches and 2) data storage approaches. We recognize that, various excellent schemes have been proposed in the literature either to directly reduce one resource overhead (such as computation, network and IO overhead), or to accelerate the update process by different techniques. Accordingly, we propose a resource-based classification to classify them into 3 types in terms of their goals: ❶ computation optimization, ❷ network optimization, ❸ IO optimization.

V. RESOURCE-BASED CLASSIFICATION

In this section, we begin to introduce our classifications (resource-based classification and tier-based classification). Table 1 clearly shows the comparison between existing classifications and our classifications.

TABLE 1. The comparison between existing classifications and our classifications.

Classification Method	Basis	Goal
Data Transmission Approaches	According to whether transferring the whole data block, divide the update schemes into 2 types: 1) *RAID-based update* and 2) *delta-based update*.	Network Optimization
Data Storage Approaches	According to whether the data is overwritten, divide the update schemes into 3 types: 1) *In-place Update*, 2) *Log-based Update*, and 3) *Hybrid Update*.	IO Optimization
Resource-based Classification	According to the resource they optimize, divide the update schemes into 3 types: 1) *Computation Optimization*, 2) *Network Optimization*, and 3) *IO Optimization*.	Resource Optimization
Tier-based Classification	According to the characteristics of different schemes, divide the update schemes into 5 tiers: 1) *Coding Tier*, 2) *Scheduling Tier*, 3) *Network Tier*, 4) *IO Tier*, and 5) *Hardware Tier*.	Hybrid Technique Optimization

A. COMPUTATION OPTIMIZATION

The computational reduction is the most straightforward way to achieve DU efficiency. As stated before, DU depends on encoding or its delta style. Several excellent techniques have been proposed in the literature, which significantly improve encoding efficiency, such as *Bitmatrix Normalization (BN)* [41], *Smart Scheduling (SS)* [41], *Randomization* [6], *Interference Alignment* [45], *Update-Efficient Regenerating Codes (UERC)* [26], *Matching* [27], *RAPID* [4]. While *Vectorization* [64] is a typical technique of accelerating the computation process.

1) BITMATRIX NORMALIZATION

A simple procedure for computational reduction is using Bitmatrix Normalization (BN), the main idea of which is to reduce blue ones in BDM (Figure 4), thus make computation convenient. Here is the simple heuristic to create BDM from [41],

❶ Create a Cauchy matrix P such that $P_{i,j} = \frac{1}{i \oplus (m+j)}$, where $m = n - k$ and division is over GF(2^w).

❷ For each column-j, to make each element in row 0 to one, division is performed on each element by $P_{0,j}$.

❸ For each row-i except the first,

(a) Collect the number of ones.

(b) Divide row-i by $P_{i,j}$ for each j, and collect the number of ones.

(c) Employing the $P_{i,j}$ which generates the minimum number of ones from previous two steps, replace the values with new row-i after division of $P_{i,j}$. That is, row-i is normalized with the element in the row which induces the minimum number of ones in the P.

Finally, we got a new matrix P', as shown in Figure 4, the M(e) representation of P' is BDM.

Here is the example given by Plank *et al.* [41], where $n = 6, k = 3, w = 3$. The initial P is as follows,

$$\begin{bmatrix} 6 & 7 & 2 \\ 5 & 2 & 7 \\ 1 & 3 & 4 \end{bmatrix}$$

First of all, column 0 is divided by 6, column 1 is divided by 7 and column 2 is divided by 2, for instance, we can get $P_{0,0} = 1, P_{1,0} = 4, P_{2,0} = 3$ from Table 2, to yield,

$$\begin{bmatrix} 1 & 1 & 1 \\ 4 & 3 & 6 \\ 3 & 7 & 2 \end{bmatrix}$$

TABLE 2. Multiplication table for GF(2^3).

i \ j	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	3	1	7	5
3	0	3	6	5	7	4	1	2
4	0	4	3	7	6	2	5	1
5	0	5	1	4	2	7	3	6
6	0	6	7	1	5	3	2	4
7	0	7	5	2	1	6	4	3

Now, we can count the ones in every row according to Figure 7. We concentrate on the second row, in which bitmatrix representation has $5 + 7 + 7 = 19$ ones. If we divide it by $P_{1,0}/P_{1,1}/P_{1,2}$, we can get the bitmatrix has 12/11/16 ones. Thus, we use row 1 divided by $P_{1,1} = 3$ as the new row 1, which generates the minimum number of ones.

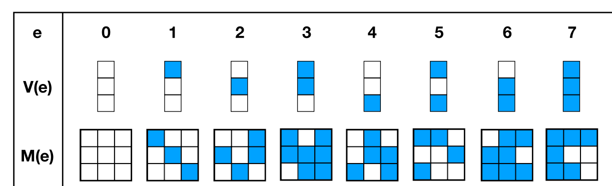


FIGURE 7. Vector and matrix representation of the elements of GF(2^3).

We do the same with the third row and notice $P_{2,0} = 3$ is optimal. Thus, the new matrix P' is,

$$\begin{bmatrix} 1 & 1 & 1 \\ 5 & 1 & 2 \\ 1 & 4 & 7 \end{bmatrix}$$

Replacing every number (e) with M(e) representation (Figure 7), we can get BDM. This example shows BN can reduce 12 ones in the BDM (from 46 to 34). This approach is very simple and useful.

2) SMART SCHEDULING

The idea of Smart Scheduling (SS) is to reuse some parity computation to reduce the overall computation, which is also proposed by Plank *et al.* [41]. For example, if we update $p_{0,0}, p_{1,0}$ by Eq.(4) and Eq.(7),

$$\begin{aligned} 10p_{0,0} &= d_{0,0} \oplus d_{1,0} \oplus d_{2,0} \oplus d_{2,2} \\ p_{1,0} &= d_{0,0} \oplus d_{1,0} \oplus d_{1,2} \oplus d_{2,0} \end{aligned}$$

which requires 6 XOR operations. But if we update $p_{0,0}$ first, $p_{1,0}$ can be updated with the following equation,

$$p_{1,0} = p_{0,0} \oplus d_{1,2} \oplus d_{2,2}$$

which requires only 5 XORs. Compare to BN, SS needs slightly more effort to implement and optimize because searching for the optimal schedule to update parity blocks is proved to be the NP-Complete problem [27], [50]. Fortunately, the computation schedule can be in fact generated offline [64] or be resorted to a heuristic algorithm that can quickly find an excellent schedule [50].

3) RANDOMIZATION

Similar to Anthapadmanabhan *et al.* [6] were also interested in constructing an efficient BDM to optimize DU. They took advantage of *randomization* to generate BDM and proved that, a code can be designed that requires at most $O(\log n)$ updates per change in a single data block.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} \tag{11}$$

Eq.(11) shows an example of BDM with [7, 4, 3] Hamming code [6], which has update complexity of 4. That is, if any single block is changed, 4 blocks are required to be updated at most (e.g., if d_4 is changed, the corresponding p_1, p_2, p_3 are required to be updated). In general, the $[2^r - 1, 2^r - 1 - r, 3]$ Hamming code has an update complexity of $r + 1$ (here $r = 3$). Thus, the update complexity scales logarithemically with the code length. Based on randomization, they constructed an update-efficient *KG code* and proved it has update complexity of $O(\log n)$ in mathematically (n still represents the number of nodes in CSS). Besides, they also proved that RS code is not update efficient because RS code is a MDS code which has the minimum distance $n - k + 1$, leading to update complexity of $n - k + 1$ at least.

4) INTERFERENCE ALIGNMENT

Based on KG code, Rawat *et al.* [45] used a combination of KG code for update efficiency with *interference-alignment (IA)* strategy for distributed storage systems and proved that, as long as we have large enough alphabet, there exist an encoding matrix $G \in \mathcal{C}$, thus, *the repair bandwidth per symbol of repaired data* for the proposed code with IA tends to $\frac{n-1}{n-k_n}$. In other words, it can achieve simultaneously repair efficiency and update efficiency.

However, both the KG code and KG-based codes stay at the theoretical level, experiments are required to prove their efficiency.

$$\overline{G} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ a^0 & a^1 & \cdots & a^{n-1} \\ (a^0)^2 & (a^1)^2 & \cdots & (a^{n-1})^2 \\ (a^0)^3 & (a^1)^3 & \cdots & (a^{n-1})^3 \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ (a^0)^{\alpha-1} & (a^1)^{\alpha-1} & \cdots & (a^{n-1})^{\alpha-1} \end{bmatrix},$$
$$\Delta = \begin{bmatrix} (a^0)^{\alpha} & 0 & \cdots & 0 \\ 0 & (a^1)^{\alpha} & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & \cdots & (a^{n-1})^{\alpha} \end{bmatrix} \tag{12}$$

5) UPDATE-EFFICIENT REGENERATING CODES

It is well-known that regenerating codes [19] represent a class of erasure codes which can achieve the minimum bandwidth cost or the minimum storage cost in DR, or the balance between bandwidth and storage in per node. To minimize storage overhead, Minimum Storage Regenerating codes (MSR) are proposed, which first minimize the amount of data stored per node, and then the repair bandwidth. While Minimum Bandwidth Regenerating (MBR) codes pursue minimization in the reverse order. MSR and MBR represent the two ends of optimal curve of regenerating codes, respectively.

Han *et al.* [26] constructs an update-efficient MSR regenerating codes (UERC) with error correction capability based on RS codes, and specifies the conditions need to satisfy:

1) $G = \begin{bmatrix} \overline{G} \\ \overline{G}\Delta \end{bmatrix}$, $\overline{G}$ contains the first α rows in G and Δ is a diagonal matrix.

2) $\overline{G}$ is a generator matrix of the RS(n, α) and G is a generator matrix of RS(n, $d = 2\alpha$) code C.

From Eq.(12) we can see, a is the generator of GF(2^w), as long as $w \geq \lceil \log_2 n\alpha \rceil$, every element in $\overline{G}$, Δ is distinct. Thus, all elements in G are distinct. Noting that

RS(n, $d = 2\alpha$) code C is a cyclic code which can be arranged as a systematic code. Thus, G can be transformed into G=[D I] style:

$$G = [D\ I] = \begin{bmatrix} b_{00} & b_{01} & \dots & b_{0(n-\alpha-1)} & 1 & 0 & \dots & 0 \\ b_{10} & b_{11} & \dots & b_{1(n-\alpha-1)} & 0 & 1 & \dots & 0 \\ b_{20} & b_{21} & \dots & b_{2(n-\alpha-1)} & 0 & 0 & \dots & 0 \\ \cdot \\ \cdot \\ \cdot \\ b_{(k-1)0} & b_{(k-1)1} & \dots & b_{(k-1)(n-\alpha-1)} & 0 & 0 & \dots & 1 \end{bmatrix} \tag{13}$$

As G is the generator matrix of RS(n, k), the MDS property causes the Hamming weight of G is $n - \alpha + 1$ at least. As shown in Eq.(13), each row of G is nonzero codeword with the minimum Hamming weight $n - \alpha + 1$. In other words, it achieves the minimum Hamming weight while keeping the MDS property. Therefore, it is update-efficient. Similarly, Han *et al.* [25] construct update-efficient MBR codes.

However, similar with KG codes, their efficiency needs to be validated in experiments.

6) MATCHING

In fact, the idea of SS has a related form. To optimize XOR-based codes (OXC), Huang *et al.* [27] proposed 2 greedy algorithms (*Cardinality Catching* and *Weighted Matching*) in terms of COF (computing common operations first) rule. Instead of reusing computed parity bits, they referred common XOR operations as intermediate results which can be reused for others. They related the OXC problem to a graph where all inputs are as nodes in the graph and connected two nodes with an edge whenever there is a potential XOR computation.

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \end{bmatrix} = \begin{bmatrix} o_1 \\ o_2 \\ o_3 \\ o_4 \end{bmatrix} \tag{14}$$

For example, The inputs (data blocks) are i_1, i_2, i_3, i_4, i_5 and i_6. The outputs (parity blocks) are o_1, o_2, o_3 and o_4. Based on Eq.(14), we can get the graph like Figure 8(a), where each edge has a counter shows the number of sharing between different outputs for a specific XOR. A matching is a set of edges in a graph, in which there no exist two edges share the same node. As illustrated in Figure 8(b), the cardinality matching is the matching with the maximum number of edges. To compute such edges first, all edges with fewer counter values are removed.

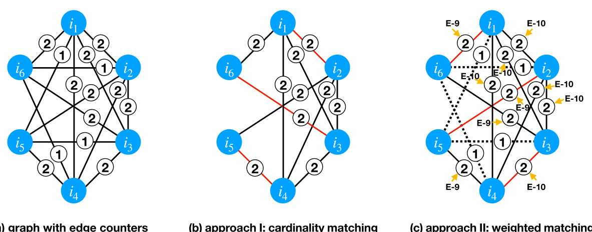


FIGURE 8. **Illustration of two greedy approaches: *Cardinality Matching* and *Weighted Matching* [27].**

Based on the cardinality matching, Figure 8(c) depicts the weighted matching which takes density (degree) into consideration. For each edge with the maximum counter value, its weight is set to be a large constant (i.e., E) minus the degrees of its both end nodes. In every round, the edge with maximum matching and the minimum density should be computed first such that it's probably to contain more matchings for the next round. Huang made a conjecture about OXC's NP-completeness, and proved that the two algorithms proposed can be solvable in polynomial time.

Matching is a heuristic of searching for optimal update scheduling, which can be accomplished in polynomial time. The experimental results show its great improvement for DU.

7) RAPID

Another approach for optimizing DU is to reduce the total number of parity updates, which is called *RAPID* [4]. Akash *et al.* recognized that it is not necessary to update parity nodes if no failures occur in this update round. RAPID is a protocol, which draws an update window in each round and rules that, only parity nodes within this window can be updated. Roughly speaking, RAPID sacrifices part of data consistency to accelerate the process of DU, which may lead to permanent data loss in the face of bulk failures. While according to the failure model in [48] and statistics of failure in Facebook cluster (Table 3), the probability of bulk failures is rather low, which motivates the design of RAPID. To leverage the failure handling and data consistency, Akash *et al.* proposed a dynamic window strategy as follows,

$$T_i = P * N_i + (1 - P) * T_{i-1}$$

where T_i denotes the window size of round i, and P denotes the failure probability. N_i represents the number of fail nodes in round i. Therefore, if N_i is large, the window size will become larger in the next round to avoid data loss.

TABLE 3. **Failure statistics in Facebook [44].**

Type of Failures	# Failures
Single Block Failure	98.08%
Double Block Failure	1.87%
Three or More	0.05%

RAPID carefully combines locking and buffering technologies to implement this protocol, where locking is designed for data consistency and buffering is designed for concurrency. To broaden concurrency, buffering has a higher priority than locking, but a threshold is used to restrict too much concurrency.

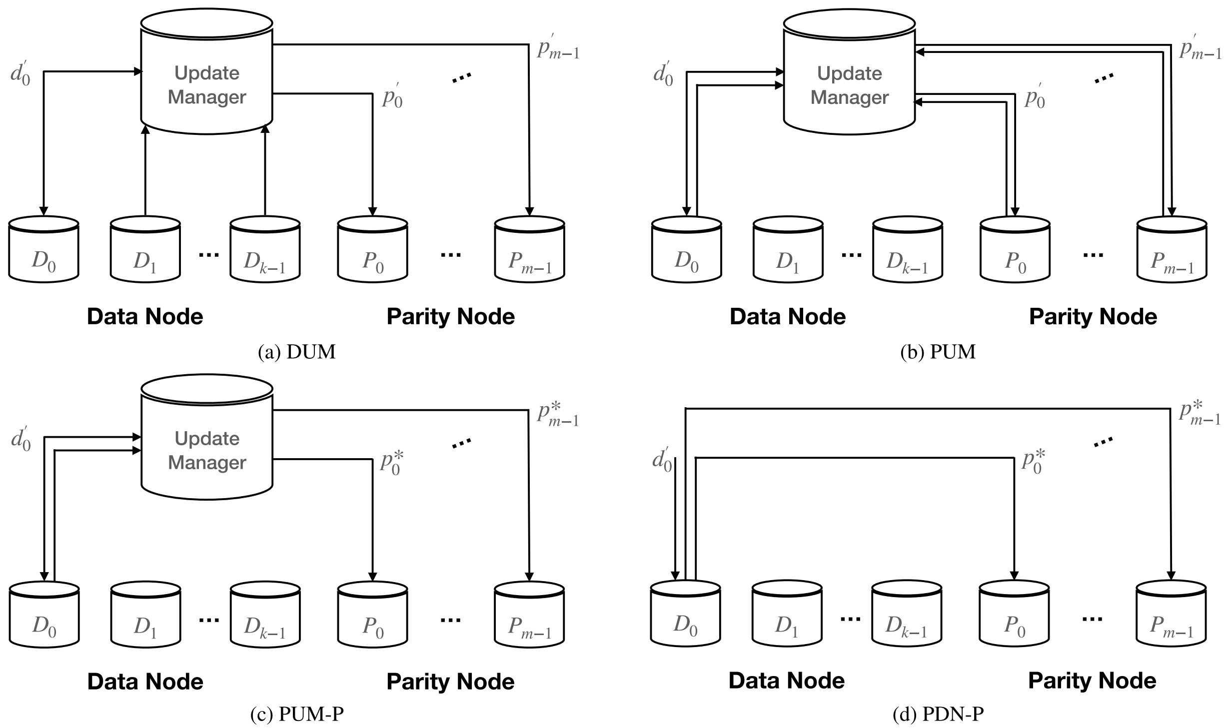


FIGURE 9. The models of DUM, PUM, PUM-P, and PDN-P.

RAPID is a high-level strategy, it fully uses the low probability of bulk failures. We argue that any other approach can combine with it to make a deeper improvement for DU.

8) VECTORIZATION

Modern CPUs are typical with SIMD capability, which can significantly accelerate the computation process of DU. SIMD can be used in XOR operations and more general finite field operations.

Vectorization is a typical technique for DR [64], which was implemented by invoking the 128-bit SSE vectorization instruction set for INTEL and AMD CPUs. Recent 256-bit AVX2 and 512-bit AVX-512 vectorization instructions are becoming more and more common in newer generations of CPUs.

B. NETWORK OPTIMIZATION

While the computation optimization is helpful for DU, the network bandwidth often becomes the bottleneck of update performance in CSS [51]. To mitigate the negative effect of network traffic on the performance of DU, many works make contributions for designing new transmission schemes to reduce the network traffic overhead or increase the bandwidth utilization among multiple storage nodes.

1) BASIC APPROACHES

As mentioned earlier, DUM and PUM are 2 typical network-aware approaches [63], both of which use a special manage node to accomplish the updates.

DUM: DUM is the implement of RCW in distributed storage systems. Figure 9a illustrates how it works: according to the update equations (e.g., Eq.(4) to Eq.(9)), the new parity blocks $p_0, \ldots, p_{m-1}$ are regenerated with all data blocks in the same stripe by a special node called Update Manager (UM).

PUM: Similar to RMW, PUM uses the deltas of updated data blocks and the old parity blocks by UM to compute the new parity blocks. Specifically, as shown in Figure 9b, the updated data blocks (here is d_0') and their old data blocks will be sent to UM to compute the deltas, after that the new parity blocks will be regenerated with the deltas and the old parity blocks according to Eq.(3) or Eq.(10). If the update is small, generally PUM performs better than DUM, since it does not have to fetch all irrelevant data blocks as DUM does.

Zhang *et al.* [63] focused on small updates, and proposed 2 novel PUM-based schemes called PUM-P and PDN-P to optimize the small updates. The underlying idea is to utilize the computational capability of storage nodes to shorten the access chain length of DU and alleviate the update traffic over network.

PUM-P: As depicted in Figure 9c, PUM-P computes the new parity blocks with the parity blocks which are generated by UM and the parity nodes. In PUM-P, UM only computes the deltas ($p_i^*, i \in [0, m-1]$) of updated data blocks. Subsequently, UM sends the deltas to the corresponding parity nodes to accomplish DU.

PDN-P: PDN-P computes the new parity blocks with the parity blocks that are directly generated by the data nodes

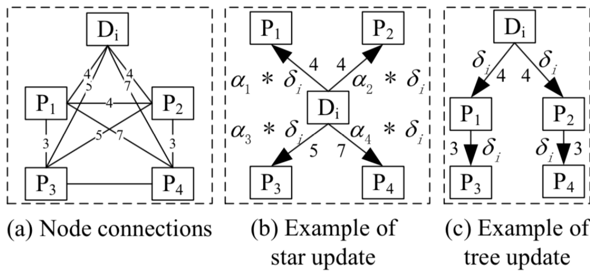


FIGURE 10. **A comparision between star-structured and tree-structured transmission path, where D_i denotes the updated data node and P_1, P_2, P_3, P_4 denote the corresponding parity nodes [39].**

and parity nodes (as depicted in Figure 9d), the difference between PDN-P and PUM-P is that, PDN-P simply utilizes data nodes to compute p_i^*, while PUM-P gets p_i^* by UM. Therefore, compare to PUM-P, PDN-P can slightly save storage overhead, but the burden of DU is partially transferred to data nodes.

The experimental results show that, for small updates, PUM-P and PDN-P can achieve greater performance than DUM and PUM. But for large updates, DUM is optimal.

2) DATA TRANSMISSION OPTIMIZATION

Pei *et al.* [39] noticed that it has long been assumed that the update process is performed on a star structure. In other words, the updated data node sends its delta to the corresponding parity nodes directly when DU is required. However, the traditional star-structured update schemes (e.g., PDN-P) put all the burden of data computing and forwarding tasks of parity information on the data node, increasing the risk of the performance bottleneck [39]. To end this, Pei *et al.* proposed a tree-structured update scheme called *T-Update* [39], which constructs an update tree to organize the data connections. In T-Update, each updated data node is handled separately. Specifically, each updated data node is the root node and the corresponding parity nodes are the intermediate nodes and the leaf nodes. When the root node is updated, it transfers the delta to its children, until to the leaf nodes. Every parity node updates its parity block with the delta and forwards it to its children. In this way, the data transmission and computation can be accomplished in parallel to optimize DU, where the non-leaf parity nodes compute parity updates and forward the delta at the same time.

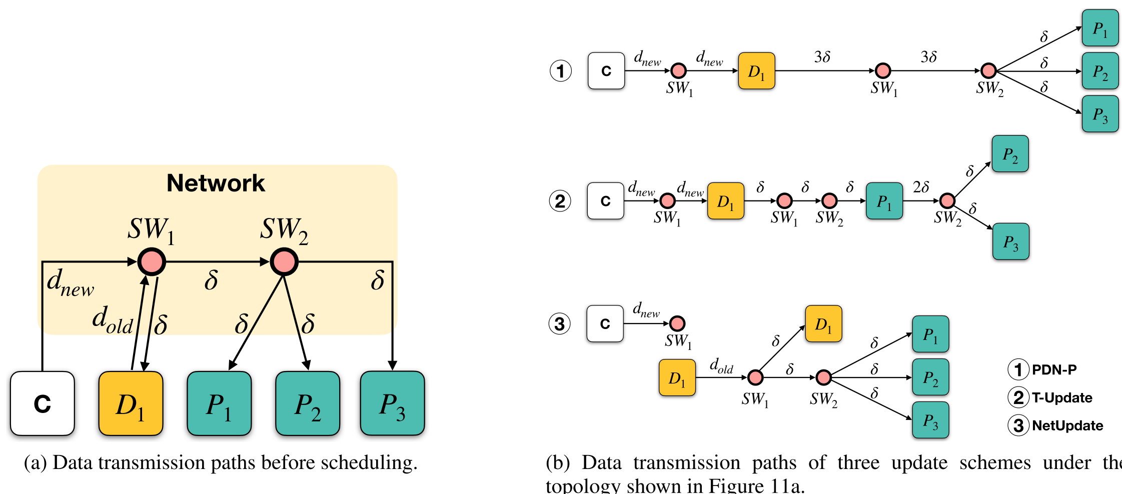
T-Update in fact constructs an MST (Minimum Spanning Tree) based on network distance. If two nodes connect with each other in one switch, the network distance is 1. Figure 10(a) depicts an example of the node connections, where D_i is the updated data node and P_1, P_2, P_3, P_4 are the relevant parity nodes. The weights of edges between different nodes represent different network distances. Figure 10(b) and Figure 10(c) show the corresponding star structure and tree structure, respectively.

The core idea of T-Update is derived from [33], which constructs another MST (Maximum Spanning Tree) based on available bandwidth capability (e.g., 40KB/s). It in fact represents the same meaning with network distance, but the bandwidth capability changes with time, which may complicate DU.

Based on T-Update, Wang and Pei recognized the significance of adaptivity in DU, thus proposed a novel scheme called TA-Update [59], which introduces a rollback-based approach to handle the node failure during DU.

Both T-Update and TA-Update are efficient and compatible. We argue they can also combine with other approaches.

It is well-known that CSS organizes nodes in racks, while the cross-rack bandwidth is often oversubscribed, and much more scarce than the inner-rack bandwidth [3], [8], [17]. To end this, Shen and Lee [51] proposed CAU, which builds on 3 design elements: ① selective parity updates, which select the appropriate DU approach (data-delta commit or parity-delta commit) to mitigate the cross-rack network



(a) Data transmission paths before scheduling.

(b) Data transmission paths of three update schemes under the topology shown in Figure 11a.

FIGURE 11. **Figure 7(a) shows the network topology before scheduling and figure 7(b) shows the comparision of scheduling between *delta-based*, *T-Update* and *XORInc*.**

overhead, ② data grouping, which groups and relocates the updated data blocks in one rack by exchanging data to mitigate the cross-rack bandwidth, ③ interim replication, which creates a transient copy of updated data blocks in another parity node to avoid data loss in face of failures.

CAU divides the DU into 2 phases: ① Append Phase, which is similar to FL, employs logs to save the updated information of data blocks, ② Commit Phase, when a threshold is reached (e.g., 50 blocks), the merge operation of primitive blocks and appending logs can be triggered. It sounds like a kind of batch update. In the commit phase, CAU has 2 choice: 1) data-delta commit, which means each updated node sends its delta to one parity node of a rack, as shown in Figure 12(a), 2) parity-delta commit, which means the deltas of multiple data blocks aggregates into one data node i, and node i sends the mix of deltas to the parity nodes in other racks, as depicted in Figure 12(b).

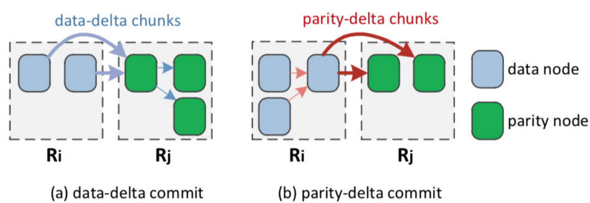


FIGURE 12. Selective parity updates: (a) data-delta commit and (b) parity-delta commit. In (a), CAU sends $i' = 2$ data-delta chunks from R_i to R_j; in (b), CAU sends $j' = 2$ parity-delta chunks from R_i to R_j [51].

To mitigate cross-rack network overhead further, data grouping, which is illustrated in Figure 13, groups updated blocks into one rack, which is implemented by swapping blocks (as long as the swapping network overhead is lower than the orignal cross-rack network overhead).

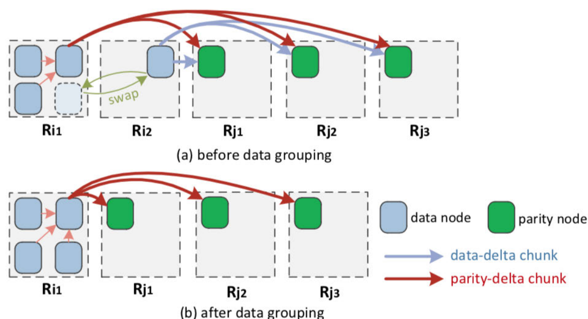


FIGURE 13. Data grouping: we can swap the updated data chunk in R_{i2} with one of the chunks in R_{i1}, such that the four updated data chunks are now stored in R_{i1} [51].

The main features of CAU are batch update and data grouping. It is noting that CAU is generic and can be applied to any practical erasure code.

3) NETWORK DEVICE ACCELERATION

With the emergence of programmable network devices, the concept of in-network computation has been proposed [58]. The key idea is to offload compute operations onto intermediate network devices. Inspired by this idea, Wang *et al.* [58] noted that many works focusing on designing new transmission schemes (e.g., T-Update and CAU) to improve bandwidth utilization among multiple storage nodes, but they do not actually reduce network traffic. Besides, they figured out that all network traffic is required to pass through the switches, no matter which node computes the deltas. Therefore, they proposed XORInc, a framework based on programmable network devices (i.e., modern switches, with XOR computation capability and sufficient buffers to save intermediate results). Based on XORInc, they proposed a new scheme called *NetUpdate* to optimize DU.

Here is an example to illuminate the idea of XORInc, as shown in Figure 11a, the client C updates the data block with d_{new} and sends it to the storage system, and then D_1 receives d_{new} from switch SW_1 and sends its old block d_{old} back to SW_1, where the delta δ will be computed. Obviously, to achieve data consistency, the parity nodes (P_1, P_2, P_3) are required to be updated with δ.

To update P_1, P_2, P_3, three update schemes (PDN-P, T-Update and NetUpdate) are compared in Figure 11b, where they found the length of the data transmission path in NetUpdate is the shortest, the worst one is in PDN-P. The main reasons are as follows: ① Modern switches has the computation capability, thus they do not have to forward updated data to other nodes to compute the deltas. ② Switches have sufficient links, thus multiple data transmissions can be accomplished simultaneously.

Of course, with programmable network devices, we can achieve better performance in DU. However, not every existing CSS has this environment.

C. IO OPTIMIZATION

Many works towards reducing the IO overhead in erasure-coded distributed storage systems have been discussed in Section IV, such as in-place update (e.g., FO), log-based update (e.g., FL) and hybrid update (e.g., PL and PLR). These ideas can also be applied to CSS.

Here we discuss a novel scheme called *UCODR* [50] for IO optimization from another point of view.

1) UCODR

Similar to *Matching*, UCODR is another heuristic to search for a good update sequence, but its update priority depends on the distance between different parity blocks. The definition of the distance between parity block p_i and parity block p_j is as follows,

$$dis_{p_i,p_j} = \{||d_l||, d_l \in (p_i \oplus p_j) \cap d_l \in S_d\}$$

where $|| * ||$ is the number of elements, and d_l is the data block only stored in the disks (i.e., $d_l \in S_d$). For example, the distance between $p_{0,0}$ (Eq.(4)) and $p_{1,0}$ (Eq.(7)) is 1, since by utilizing $p_{0,0}$, $d_{1,2}$ is required to be read for computing $p_{1,0}$ (as $d_{2,2}$ has been read into memory when $p_{0,0}$ is updated, it is unnecessary to read $d_{2,2}$ twice).

The distance shows the similarity of parity blocks, thus UCODR tends to give priority to those similar blocks. Specifically, UCODR starts from the parity blocks with the minimum distance, and for each parity block, UCODR employs RCW or RMW to select the minimum read times (as shown in Figure 5, RCW and RMW have different read times for updating the same block). An example is shown in Figure 14, where $p_{0,0}, p_{0,1}, p_{1,0}, p_{1,1}, p_{1,2}, p_{1,3}$ are parity blocks need to be updated. E1 and E2 represents RCW and RMW, respectively. The green ones represent the data blocks need to be read. For $p_{0,0}$, the number of reading data blocks by RCW and RMW are 0 and 5, respectively, if all the data blocks are not cached in memory, RCW is the better choice for updating $p_{0,0}$. The example shows the main idea of UCODR for IO optimization.

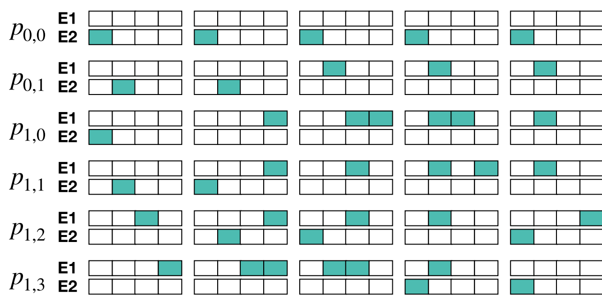


FIGURE 14. Two target elements of update approaches, E1 denotes RCW and E2 denotes RMW, the green ones represent data blocks to be read [50].

In a conclusion, UCODR is a fast update scheme for DU scheduling, thus it is helpful for reducing computational overhead, at the same time, it has great benefits for IO optimization.

VI. TIER-BASED CLASSIFICATION

In this section, we introduce our another classification: tier-based classification.

A. HYBRID TECHNIQUE FRAMEWORK

All of these techniques were proposed individually previously, and most of them mainly focus on optimizing one certain resource (computation, network or IO), which motivates us to think: if we exploit multi-resource optimization, can we use them jointly? Inspired by Zhou's idea of operation tiers [64], we propose a new hybrid technique framework which divide these techniques into 5 tiers: *Coding Tier, Scheduling Tier, Network Tier, IO Tier and Hardware Tier* (As depicted in Figure 15), We use the different color to differentiate the main goals of these techniques.

❶ *Coding Tier:* which mainly seeks to construct the generator matrix (BDM) or the update equation (e.g., RCW and RMW), thus, this tier is devoted to improving the coding theory of DU.

❷ *Scheduling Tier:* once the update equations are established, if multiple parity blocks are required to be updated, searching for the optimal scheduling is naturally required (such as matching, which selects the common parts to compute first). Thus, in this tier, researchers focus on scheduling optimization.

❸ *Network Tier:* scheduling tier offers the schedule order of DU, but before we conduct scheduling, we have to collect data involved in the computing, which leads to data transmission. The techniques in network tier mainly solve the problem of data transmission.

❹ *IO Tier:* as mentioned in Section IV, the IO tier mainly consider data storage to accelerate disk read/write.

❺ *Hardware Tier:* which mainly considers modern hardware techniques to accelerate DU (e.g., CPU with SIMD).

According to the characteristics of different schemes, Figure 15 shows these schemes with distinct goals of resource optimization, except the pink ones (RCW, DUM and RAPID). As stated before, RCW is a simple and straightforward approach, which goal is simply completing the update, and DUM is the implement of RCW in CSS. While RAPID is a protocol that sacrifices data consistency to reduce the total number of parity updates. As we argue that RAPID is a high-level strategy which can be combined with any other schemes, thus we place it on the right side. After classification, most of these techniques in different tiers can be applied in tandem theoretically, such as *combination 1* and *combination 2* in Table 4.

TABLE 4. An example of combinations of individual techniques.

Type of Combinations	Combination 1	Combination 2
Coding Tier	BN	Randomization
Scheduling Tier	Matching	UCODR
Network Tier	T-Update	PDN-P
IO Tier	PLR	PL
Hardware Tier	Vectorization	Vectorization
Other	RAPID	-

B. COMPARISON

According to the main features of these techniques, we make a comparison between these schemes systematically in Table 5. The main goals (e.g., CPU, network and IO) of these techniques are depicted in bold. Here, we first introduce several concepts:

● *Multiple Updates*: which checks whether the scheme supports multiple data updates optimization or not. For example, T-Update only considers DU optimization separately, while UCODR considers how to improve multiple data updates.

● *Parallelism Opt* (Parallelism Optimization): which checks whether the scheme is aware of parallelism acceleration or not. For example, vectorization is a typical technique for parallelism acceleration.

● *Improvement*: which is mainly derived from the results of their experiments. We set 4 types to indicate their improvement (*low* = 0%-30%, *middle* = 30%-50%, *high* = 50%-100%, and *very high* = over 100%)

● *Extra Requirement*: which describes the extra requirements for deploying the scheme.

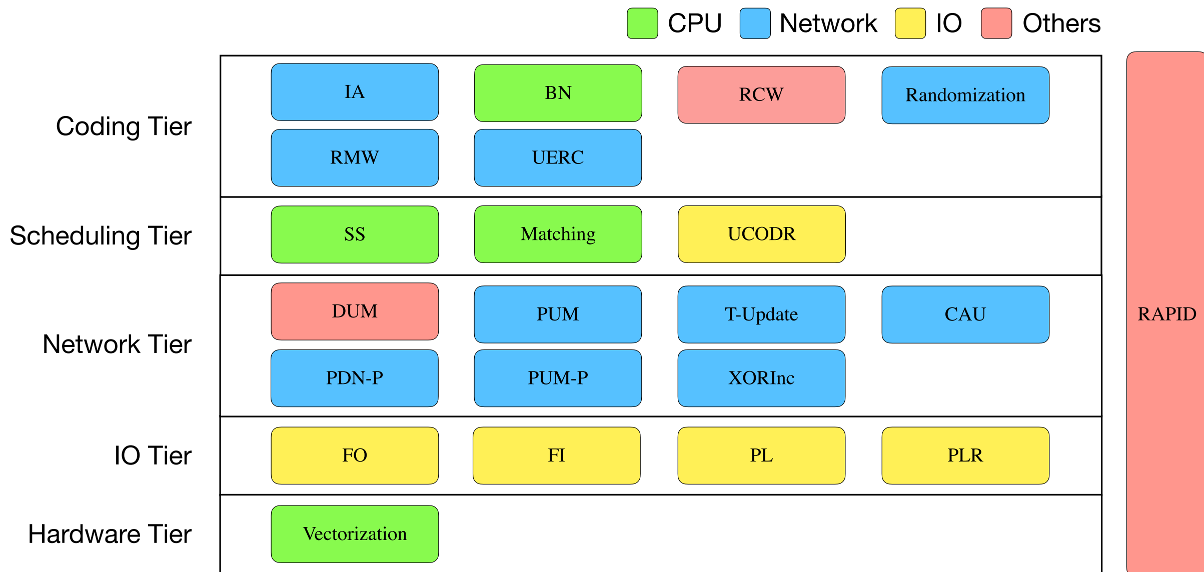


FIGURE 15. **The different techniques are divided into 5 tiers (*Coding Tier, Scheduling Tier, Network Tier, IO Tier and Hardware Tier*). We show the main goals of these techniques with different color (green, blue, yellow and pink represent CPU, Network, IO and others, respectively).**

TABLE 5. **Comparision on typical update schemes.**

Schemes \ Features	CPU	Network	IO	Multiple Updates	Parallelism Opt	Improvement	Extra Requirement	Implement Complexity	Optimal Environment
IA [45]	**Yes**	Yes	Yes	No	No	-	No	medium	large n
Randomization [6]	**Yes**	Yes	Yes	No	No	-	No	low	large n
UERC [26]	**Yes**	Yes	Yes	No	No	-	No	low	large n
BN [41]	**Yes**	Yes	Yes	No	No	low(10%-17%)	No	low	all
RCW [56]	No	No	No	No	No	No	No	low	large update
RMW [56]	No	**Yes**	No	No	No	low(10%-17%)	No	low	small update
SS [41]	**Yes**	Yes	Yes	Yes	No	medium	No	high	large m
Matching [27]	**Yes**	Yes	Yes	Yes	No	medium(50%)	No	medium	large m
UCODR [50]	Yes	Yes	**Yes**	Yes	No	medium(50%)	No	medium	large m
DUM [63]	No	No	No	No	No	-	update manager	low	all
PUM [63]	No	**Yes**	No	No	No	low	update manager	low	all
PUM-P [63]	No	**Yes**	No	No	No	high(>50%)	update manager	low	small update
PDN-P [63]	No	**Yes**	No	No	No	very high(>100%)	No	low	small update
T-Update [39]	No	**Yes**	No	No	Yes	medium(>30%)	No	low	all
XORInc [58]	No	**Yes**	No	No	Yes	very high(>300%)	modern switches	high	all
CAU [51]	No	**Yes**	No	Yes	Yes	medium(>40%)	No	medium	all
FO [2]	No	No	No	Yes	No	No	No	low	all
FI [12], [23]	No	No	**Yes**	No	No	high(>50%)	No	low	less read
PL [30], [54]	No	No	**Yes**	No	No	high(>50%)	No	low	all
PLR [14]	No	No	**Yes**	No	No	high(>50%)	No	medium	all
Vectorization [64]	**Yes**	No	No	No	Yes	-	SIMD	low	all
RAPID [4]	Yes	Yes	Yes	**Yes**	No	medium(30%)	consistency degradation	medium	large m

● *Implement Complexity*: which is mainly inferred from the algorithm complexity and deployment complexity.

● *Optimal Environment*: which shows the optimal environment for the scheme, but it is not the constraint for the scheme.

As shown in Table 5, in coding tier, optimizing CPU (especially reducing ones in BDM) can not only reduce computation overhead of DU, but also can benefit the network and IO, because a sparse BDM involves less blocks, leading to less disk I/Os and transmission. As IA, Randomization and UERC are only proved theoretically, not verified by experiments, thus their improvement of optimization is unclear.

In scheduling tier, the three schemes (SS, Matching and UCODR) are aware of CPU, Network and IO, since they all have the similar idea of reusing common parts. Among them, SS has much higher implement complexity, since it is not heuristic.

In network tier, some schemes need extra requirements, such as update manager (DUM, PUM and PUM-P) and

modern switches (XORInc), we argue that the implement complexity of XORInc is high, since modern switches are not very common in existing CSS.

In hardware tier, vectorization needs modern CPU with SIMD, it is a typical technique in DR, while not yet considered in DU.

VII. CHALLENGES

There are many open challenges in the field of DU. In this section, we examine the challenges of optimizing DU based on the 3 prominent requirements (efficiency, compatibility, and adaptivity).

A. RESEARCH CHALLENGES

1) COMPUTATION EFFICIENCY

a: CONSTRUCT OPTIMAL BDM IS HARD

As mentioned above, BDM is significantly important for computation efficiency. While IA notes that the requirements of DR and DU can be seen as counteracting one another, as the latter needs a sparse BDM and the former needs a dense one. Therefore, we have to make a trade-off between them, but the optimal leverage point is hard to be found.

b: SEEK OPTIMAL SCHEDULE IS HARD

As mentioned above, searching for the optimal scheduling order is proved to be the NP-Complete problem [27], [50]. Although some excellent heuristic schemes (e.g., Matching and UCODR) have been proposed, the potential improvement space is large.

2) NETWORK EFFICIENCY

It is known that network bandwidth, especially rack-across bandwidth is always a scarce resource in CSS. Bandwidth usage is directly proportional to the amount of data transferred in the distributed storage. Therefore, we should speed up network transferring by increasing bandwidth utilization or reducing network traffic while keeping adequate bandwidth for other applications. However, available bandwidth changes with time, which complicates the network transmission optimization. Although, T-Update improves the data transmission path by tree structure, leading to load balance and parallelism acceleration, it is still unknown which structure is optimal. Therefore, improving bandwidth efficiency without sacrificing performance is one of the most important challenges [36].

3) COMPATIBILITY

As mentioned earlier, there exist diverse erasure codes adopted in existing CSS, we should consider how to adapt to them with a little change, which may lead to a trade-off between efficiency and compatibility.

Besides, as stated before, DU is mutable. Therefore, a compatible scheme is required for DU.

4) ADAPTIVITY

It is well-known that the failure is a norm in CSS, which may occur at any time, thus it is dramatically important for DU to consider dealing with failures while updating. Rollback-based schemes are excellent, however, they sacrifice performance to correct failures. Therefore, improving adaptivity without sacrificing performance is also one of the most important challenges in DU.

VIII. CONCLUSION

In this survey, we performed a comprehensive study of the state-of-the-art update techniques which can be applied in CSS. A set of introductions were conducted to understand their ideas, improvements and constraints. A systematic comparison between them was performed to show their differences and relations. According to our observation of these techniques, we proposed 2 classifications: resource-based classification and tier-based classification. In resource-based classification, based on their optimizing goals, we divide them into 3 parts: computation optimization, network optimization, and IO optimization. In tier-based classification, we proposed a novel hybrid technique framework with 5 tiers to group these techniques in a new form. We argued that most of these techniques in different tiers can be used jointly.

DU is common, mutable, and full of challenges, based on which, the following research directions are listed for future:

❶ Construct a proper BDM to achieve update efficiency while keeping repair efficiency.

❷ Propose new techniques or hybrid techniques to reduce network traffic overhead.

❸ Among these update schemes in hybrid technique framework, a set of tests can be conducted to select the best combination to achieve update efficiency.

❹ Propose new compatible schemes or combinations of schemes.

❺ Against failures, propose new techniques to improve adaptivity.

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for their insightful feedback. The authors also appreciate Jingwei Li and Hu Xiong for their sincere help.

REFERENCES

[1] S. Agarwal, S. Kandula, N. Bruno, M.-C. Wu, I. Stoica, and J. Zhou, "Reoptimizing data parallel computing," in *Proc. 9th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2012, pp. 281–294.

[2] M. K. Aguilera, R. Janakiraman, and L. Xu, "Using erasure codes efficiently for storage in a distributed system," in *Proc. Int. Conf. Dependable Syst. Netw. (DSN)*, 2005, pp. 336–345.

[3] F. Ahmad, S. Chakradhar, A. Raghunathan, and T. N. Vijaykumar, "Shufflewatcher: Shuffle-aware scheduling in multi-tenant mapreduce clusters," in *Proc. USENIX Annu. Tech. Conf.*, 2014, pp. 1–12.

[4] G. J. Akash, O. T. Lee, S. D. Madhu Kumar, P. Chandran, and A. Cuzzocrea, "RAPID: A fast data update protocol in erasure coded storage systems for big data," in *Proc. 17th IEEE/ACM Int. Symp. Cluster, Cloud Grid Comput. (CCGRID)*, May 2017, pp. 890–897.

[5] Amazon. *Amazon S3*. Accessed: Jul. 23, 2020. [Online]. Available: https://aws.amazon.com/cn/s3/

[6] N. P. Anthapadmanabhan, E. Soljanin, and S. Vishwanath, "Update-efficient codes for erasure correction," in *Proc. 48th Annu. Allerton Conf. Commun., Control, Comput. (Allerton)*, Sep. 2010, pp. 376–382.

[7] B. Baesens, *Analytics in a Big Data World: The Essential Guide to Data Science and Its Applications*. Hoboken, NJ, USA: Wiley, 2014.

[8] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th Annu. Conf. Internet Meas. (IMC)*, Jan. 2010, pp. 267–280.

[9] M. Blaum, J. Brady, J. Bruck, and J. Menon, "EVENODD: An efficient scheme for tolerating double disk failures in RAID architectures," *IEEE Trans. Comput.*, vol. 44, no. 2, pp. 192–202, 1995.

[10] J. Blömer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman, "An XOR-based erasure-resilient coding scheme," Tech. Rep., 1995.

[11] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: A high-availability and integrity layer for cloud storage," in *Proc. 16th ACM Conf. Comput. Commun. Secur. (CCS)*, 2009, pp. 187–198.

[12] B. Calder *et al.*, "Windows azure storage: A highly available cloud storage service with strong consistency," in *Proc. 23rd ACM Symp. Oper. Syst. Princ.*, 2011, pp. 143–157.

[13] P. Cao, S. B. Lin, S. Venkataraman, and J. Wilkes, "The TickerTAIP parallel RAID architecture," *ACM Trans. Comput. Syst.*, vol. 12, no. 3, pp. 236–269, Aug. 1994.

[14] J. C. W. Chan, Q. Ding, P. P. C. Lee, and H. H. W. Chan, "Parity logging with reserved space: Towards efficient updates and recovery in erasure-coded clustered storage," in *Proc. 12th USENIX Conf. File Storage Technol. (FAST)*, 2014, pp. 163–176.

[15] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "RAID: High-performance, reliable secondary storage," *ACM Comput. Surv.*, vol. 26, no. 2, pp. 145–185, Jun. 1994.

[16] Q. Chen, L. Liang, Y. Xia, H. Chen, and H. Kim, "Mitigating sync amplification for copy-on-write virtual disk," in *Proc. 14th USENIX Conf. File Storage Technol.*, 2016, pp. 241–247.

[17] M. Chowdhury, S. Kandula, and I. Stoica, "Leveraging endpoint flexibility in data-intensive clusters," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 231–242, Sep. 2013.

[18] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar, "Row-diagonal parity for double disk failure correction," in *Proc. 3rd USENIX Conf. File Storage Technol.*, San Francisco, CA, USA, 2004, pp. 1–14.

[19] A. G. Dimakis, P. B. Godfrey, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," in *Proc. 26th IEEE Int. Conf. Comput. Commun. (INFOCOM)*, May 2007, pp. 2000–2008.

[20] Drew and Arash. *Celebrating Half a Billion Users*. Accessed: Jul. 23, 2020. [Online]. Available: https://blog.dropbox.com/topics/company/500-million/

[21] D. J. Ellard, "Trace-based analyses and optimizations for network storage servers," Tech. Rep., 2004.

[22] Y. Fu and J. Shu, "D-code: An efficient RAID-6 code to optimize I/O loads and read performance," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2015, pp. 603–612.

[23] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proc. 19th ACM Symp. Oper. Syst. Princ. (SOSP)*, 2003, pp. 29–43.

[24] L. Hellerstein, G. A. Gibson, R. M. Karp, R. H. Katz, and D. A. Patterson, "Coding techniques for handling failures in large disk arrays," *Algorithmica*, vol. 12, nos. 2–3, pp. 182–208, Sep. 1994.

[25] Y. Han, H.-T. Pai, R. Zheng, and P. K. Varshney, "Update-efficient error-correcting product-matrix codes," 2013, *arXiv:1301.4620*. [Online]. Available: http://arxiv.org/abs/1301.4620

[26] Y. S. Han, H.-T. Pai, R. Zheng, and P. K. Varshney, "Update-efficient regenerating codes with minimum per-node storage," in *Proc. IEEE Int. Symp. Inf. Theory*, Jul. 2013, pp. 1436–1440.

[27] C. Huang, J. Li, and M. Chen, "On optimizing XOR-based codes for fault-tolerant storage applications," in *Proc. IEEE Inf. Theory Workshop*, Sep. 2007, pp. 218–223.

[28] C. Huang, H. Simitci, Y. Xu, A. W. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *Proc. USENIX Annu. Tech. Conf.*, 2012, p. 2.

[29] C. Huang and L. Xu, "STAR : An efficient coding scheme for correcting triple storage node failures," *IEEE Trans. Comput.*, vol. 57, no. 7, pp. 889–901, Jul. 2008.

[30] C. Jin, D. Feng, H. Jiang, and L. Tian, "RAID6L: A log-assisted RAID6 storage architecture with improved write performance," in *Proc. IEEE 27th Symp. Mass Storage Syst. Technol. (MSST)*, May 2011, pp. 1–6.

[31] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "OceanStore: An architecture for global-scale persistent storage," *ACM SIGARCH Comput. Archit. News*, vol. 28, no. 5, pp. 190–201, Dec. 2000.

[32] D. Le, H. Huang, and H. Wang, "Understanding performance implications of nested file systems in a virtualized environment," in *Proc. FAST*, 2012, p. 8.

[33] J. Li, S. Yang, X. Wang, X. Xue, and B. Li, "Tree-structured data regeneration with network coding in distributed storage systems," in *Proc. 17th Int. Workshop Qual. Service*, Jul. 2009, pp. 1–9.

[34] W. Litwin and T. Schwarz, "LH*RS: A high-availability scalable distributed data structure using reed solomon codes," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2000, pp. 237–248.

[35] J. Menon, "A performance comparison of RAID-5 and log-structured arrays," in *Proc. 4th IEEE Int. Symp. High Perform. Distrib. Comput.*, 1995, pp. 167–178.

[36] R. Nachiappan, B. Javadi, R. N. Calheiros, and K. M. Matawie, "Cloud storage reliability for big data applications: A state of the art survey," *J. Netw. Comput. Appl.*, vol. 97, pp. 35–47, Nov. 2017.

[37] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Trans. Storage*, vol. 4, no. 3, p. 10, 2008.

[38] M. Ovsiannikov, S. Rus, D. Reeves, P. Sutter, S. Rao, and J. Kelly, "The quantcast file system," *Proc. VLDB Endowment*, vol. 6, no. 11, pp. 1092–1101, Aug. 2013.

[39] X. Pei, Y. Wang, X. Ma, and F. Xu, "T-update: A tree-structured update scheme with top-down transmission in erasure-coded systems," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[40] J. S. Plank, "The raid-6 liberation code," *Int. J. High Perform. Comput. Appl.*, vol. 23, no. 3, pp. 242–251, 2009.

[41] J. S. Plank, S. Simmerman, and C. D. Schuman, "Jerasure: A library in C/C++ facilitating erasure coding for storage applications-version 1.2," Univ. Tennessee, Knoxville, TN, USA, Tech. Rep. CS-08-627, 2008, vol. 23.

[42] L. X. James S. Plank, "Optimizing cauchy Reed–Solomon codes for fault-tolerant network storage applications," in *Proc. 5th IEEE Int. Symp. Netw. Comput. Appl. (NCA)*, Jul. 2006, pp. 173–180.

[43] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, 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 *Proc. 5th USENIX Workshop Hot Topics Storage File Syst. (HotStorage)*, 2013, pp. 1–5.

[44] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A," hitchhiker's" guide to fast and efficient data reconstruction in erasure-coded data centers," in *Proc. 2014 ACM Conf. SIGCOMM*, pp. 331–342, 2014.

[45] A. Singh Rawat, S. Vishwanath, A. Bhowmick, and E. Soljanin, "Update efficient codes for distributed storage," in *Proc. IEEE Int. Symp. Inf. Theory Proc.*, Jul. 2011, pp. 1457–1461.

[46] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, Jun. 1960.

[47] R. M. Roth and A. Lempel, "On MDS codes via cauchy matrices," *IEEE Trans. Inf. Theory*, vol. 35, no. 6, pp. 1314–1319, Nov. 1989.

[48] B. Schroeder and G. A. Gibson, "Disk failures in the real world: What does an mttf of 1, 000, 000 hours mean to you?" in *Proc. FAST*, vol. 7, 2007, pp. 1–16.

[49] J. Shen, J. Gu, Y. Zhou, and X. Wang, "Cloud-of-Clouds storage made efficient: A pipeline-based approach," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jun. 2016, pp. 724–727.

[50] J. Shen, K. Zhang, J. Gu, Y. Zhou, and X. Wang, "Efficient scheduling for multi-block updates in erasure coding based storage systems," *IEEE Trans. Comput.*, vol. 67, no. 4, pp. 573–581, Apr. 2018.

[51] Z. Shen and P. P. C. Lee, "Cross-rack-aware updates in erasure-coded data centers," in *Proc. 47th Int. Conf. Parallel Process.*, Aug. 2018, p. 80.

[52] Z. Shen, X. Li, and P. P. C. Lee, "Fast predictive repair in erasure-coded storage," in *Proc. 49th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2019, pp. 556–567.

[53] Statista. *Number of Consumer Cloud-Based Service Users Worldwide in 2013 and 2018*. Accessed: Jul. 23, 2020. [Online]. Available: https://www.statista.com/statistics/321215/global-consumer-cloud-computing-users/

[54] D. Stodolsky, G. Gibson, and M. Holland, "Parity logging overcoming the small write problem in redundant disk arrays," *ACM SIGARCH Comput. Archit. News*, vol. 21, no. 2, pp. 64–75, May 1993.
[55] M. W. Storer, K. M. Greenan, E. L. Miller, and K. Voruganti, "Pergamum: Replacing tape with energy efficient, reliable, disk-based archival storage," in *Proc. 6th USENIX Conf. File Storage Technol.*, 2008, p. 1.
[56] A. Thomasian, "Reconstruct versus read-modify writes in RAID," *Inf. Process. Lett.*, vol. 93, no. 4, pp. 163–168, Feb. 2005.
[57] R. Verma, A. A. Mendez, S. Park, S. S. Mannarswamy, T. P. Kelly, and C. B. Morrey, III, "Failure-atomic updates of application data in a Linux file system," in *Proc. 13th USENIX Conf. File Storage Technol. (FAST)*, 2015, pp. 203–211.
[58] F. Wang, Y. Tang, Y. Xie, and X. Tang, "XORInc: Optimizing data repair and update for erasure-coded systems with XOR-based in-network computation," in *Proc. 35th Symp. Mass Storage Syst. Technol. (MSST)*, May 2019, pp. 244–256.
[59] Y. Wang, X. Pei, X. Ma, and F. Xu, "Ta-update: An adaptive update scheme with tree-structured transmission in erasure-coded storage systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 8, pp. 1893–1906, Jun. 2017.
[60] Y. Wang, X. Yin, and X. Wang, "MDR codes: A new class of RAID-6 codes with optimal rebuilding and encoding," *IEEE J. Sel. Areas Commun.*, vol. 32, no. 5, pp. 1008–1018, May 2014.
[61] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: The least-authority filesystem," in *Proc. 4th ACM Int. Workshop Storage Secur. Survivability*, 2008, pp. 21–26.
[62] X. Xie, C. Wu, J. Gu, H. Qiu, J. Li, M. Guo, X. He, Y. Dong, and Y. Zhao, "AZ-code: An efficient availability zone level erasure code to provide high fault tolerance in cloud storage systems," in *Proc. 35th Symp. Mass Storage Syst. Technol. (MSST)*, May 2019, pp. 230–243.
[63] F. Zhang, J. Huang, and C. Xie, "Two efficient partial-updating schemes for erasure-coded storage clusters," in *Proc. IEEE 7th Int. Conf. Netw., Archit., Storage*, Jun. 2012, pp. 21–30.
[64] T. Zhou and C. Tian, "Fast erasure coding for data storage: A comprehensive study of the acceleration techniques," *ACM Trans. Storage*, vol. 16, no. 1, pp. 1–24, Apr. 2020.

YIFEI XIAO was born in Chengdu, Sichuan, China, in 1989. He received the B.S. and M.S. degrees in computer science and technology from Sichuan University, Chengdu, in 2012 and 2015, respectively. He is currently pursuing the Ph.D. degree in software engineering with the University of Electronic Science and Technology of China (UESTC), China.

From 2015 to 2019, he was an Engineer with the School of Computer Science and Engineering, UESTC. His research interests include IaaS in cloud computing and erasure coding in data storage.

SHIJIE ZHOU received the Ph.D. degree in computer science and technology from the University of Electronic Science and Technology of China (UESTC), in 2004. He is currently a Professor with the School of Information and Software Engineering, UESTC. His research interests include communication and security in computer networks, peer-to-peer networks, sensor networks, cloud security, and big data.

LINPENG ZHONG was born in Chengdu, Sichuan, China, in 1988. He received the B.S. and M.S. degrees in information and communication engineering from the University of Electronic Science and Technology of China (UESTC), China, in 2010 and 2013, respectively, where he is currently pursuing the Ph.D. degree in information and communication engineering.

From 2016 to 2019, he was a Lecturer with the School of Information and Software Engineering, UESTC. His research interests include mobile computing and speech signal processing.